Synthesis of Winning Strategies for Interaction under Partial Information

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen University zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Diplom-Mathematiker
Bernd Puchala
aus
Aachen

Berichter:
Prof. Dr. Erich Grädel
Prof. Dr. Jean-François Raskin

Tag der mündlichen Prüfung:
10. Juni 2013

Preface

Interaction is a fundamental concept in computer science. Besides the interaction between human users and computing systems, many computing systems are inherently interactive themselves. The individual computers in a network, for example, interact with each other via a given communication structure according to certain protocols. In a reactive system, one or more computing devices, called controllers, interact with some kind of environment, trying to guarantee a correct behavior of the system. Logic as one of the foundations of computer science is intimately linked to interaction, demonstrated by various kinds of model checking games. Moreover, semantics of alternating computing devices as well as several graph complexity measures are characterized in terms of games.

Many of these interactive scenarios take place under certain forms of uncertainty. An individual computer in a network, for example, does not necessarily know all the parameters of the other members of the network or the past message transmissions in the joint computation. The same holds for the controllers in reactive systems which often do not have full information about all the internal states of the other components or the history of past events in the whole system. Furthermore, model checking games for certain logics as well as several graph searching games are games with partial information.

This work studies theoretical aspects of interaction under partial information and focuses on controller synthesis for nonterminating reactive systems. Synthesis means that one tries to construct, from a formal model of the system and some formal specification, implementations for certain controllers which guarantee that all behaviors of the system meet the specification. This is in contrast to *verification* where the whole system realization is given so that the interactive aspect of the system is already implemented. While verification algorithms deal with successful inidividual system runs (or trees of system runs, in the branching time paradigm), synthesis deals with *winning strategies* in games.

The major starting point of this theory is the groundbreaking work of Alonso Church [54, 55] where he introduced the *Circuit Synthesis Problem*, followed by many milestone results, including the fundamental work of Büchi and Landweber [44], Rabin [177], Pnueli and Rosner [169, 170, 171] as well as Kupferman and Vardi [127, 129]. Chapter 1 provides a detailed introduction to controller synthesis for nonterminating reactive systems and winning strategies in games with imperfect information in general, and outlines the main contributions of this work.

Related Research. Besides the particular models and algorithmic problems that are studied here, there are a lot of other incarnations and facets of synthesis. Two important extensions of the deterministic and discrete models that are used in this work are stochastic and hybrid systems which incorporate probabilistic and continuous aspects, respectively.

In a stochastic system (cf. [62, 63]), the events in the system as well as the implementations of the controllers may be probabilistic. In general, such an implementation can not guarantee that *all* systems runs are correct. Rather, a possible goal might be to guarantee that *almost all* system runs are correct, that means, the probability that a system run is correct is one. Variants of the problem include nonzero probability or approaching a probability of one by a family of implementations. Hybrid systems (cf. [5, 4]) take into account the fact that computing systems (especially embedded ones) often have continuous aspects which should not be discretized in advance: Design and analysis of the systems should be in terms of real values and only then, the controllers of the system are implemented by finite state devices and finite precision sensors.

All these extensions of Church's basic setting – systems with partial information, stochastic systems and hybrid systems – aim at making the models more applicable for real-life scenarios and the goal is to develop (preferably fast) algorithmic solutions for synthesis over these models as well as certain variants of verification and many other algorithmic problems.

Another area of research that is concernced with control and synthesis of computing systems is the *control theory of discrete event systems* which was introduced by Ramadge and Wonham and is motivated by control theory rather than switching circuits and automata theory [179, 186]. It uses somewhat different models and accentuates other aspects and concepts than the theory of nonterminating reactive systems as it is considered in this work. However, substantially, the two lines of research act in concert. See for example [8, 9] for some work which is based on concepts from the control theory of discrete event systems but incorporates specification formalisms and techniques which rely on automata theoretic methods for fixed-point logics.

Moreover, model checking games are also closely related to the research reported here. For example, parity games with full information are the model checking games for the modal μ-Calculus which in turn is intimately connected to the graph complexity measure of entanglement (cf. Chapter 4). For a detailed discussion of the intimate connection between logic and games see for example [98]. Beyond the connection between logic and games with full information, there are also several logical systems that induce certain forms of imperfect information in their respective model checking games, starting with the fundamental work of Hintikka and Sandu [110] on independence-friendly first order logic.[1] Another approach is the so-called dependence logic of Väänänen [207]. Infinite model checking games with imperfect information go on stage when fixed-point extensions of such logical systems are considered [40, 41].[2]

Other Aspects of Interaction. Aside from interaction in design and analysis of computing systems, with logic and automata as conceptual and methodological core, there are a lot of other aspects of interaction – especially interaction under partial information – that are important in computer science. A diverse and inspiring compilation of several such aspects can be found in the retrospective brochure [70] of the ESF Eurocores Programme *Modelling Intelligent Interaction (LogICCC)*. This quite interdisciplinary project had gathered scientists from various research areas which are all somehow connected to logic and interaction. A distinguishing feature of the project has been the analysis of human interaction and reasoning not only from the viewpoint of computer sience (as *users*) but as a central issue that should be investigated logically and philosophically as well.

Acknowledgement. This dissertation was supervised by Erich Grädel and co-examined by Jean-Francois Raskin. It is partially based on several publications which have mostly been joint work. Section 3.5 is based on [173] and my Diploma Thesis [172] which was also supervised by Erich Grädel. Chapter 4 is based on [174, 175], co-authored by Roman Rabinovich. Chapter 5 is based on [86] which has been co-authored by Wladimir Fridman. Finally, Chapter 6 is based on [31], co-authored by Dietmar Berwanger and Łukasz Kaiser.

I would like to thank all people who have contributed to my work during the last years, in particular all my co-authors.

[1]First incarnations of this concept go back to the work of Henkin [107] where it came in the disguise of *partially ordered quantifiers* which Henkin introduced in the course of his disucssion of infinitely long formulas.

[2]It is probably not quite familiar, a fortiori expedient to note, that independence friendly fixed-point logic *captures* EXPTIME. It is not known, however, whether independence friendly fixed point logic is strictly more expressive than existential second order logic, hence leaving NP $\neq$ EXPTIME to further research.

Contents

Chapter 1

Introduction

A transformational system receives some input and after the computation is finished, it yields an output. During the computation, the behavior of the system is completely determined by its current configuration. In contrast, in a reactive system, the components of the system continuously interact with each other. In each computation step of the system, certain components receive inputs from other components and produce outputs which, in turn, may be inputs to other components in the next step.

In the systems we consider, one or more computing devices, called controllers, interact with some kind of environment and try to guarantee a correct behavior of the system. The systems are nonterminating, that means, each run of the system is infinite. We assume that each component of the system (including the environment) has a finite output alphabet, that is, for each component there are only finitely many possible outputs (actions, events) which it can produce during any step of a run of the system.

The controllers of the system *cooperate*, that means, they try to combine their different information and controls over the system to achieve a correct behavior. This network of computing devices is also called joint controller or simply *controller* of the system. On the other hand, we do not make any a priori assumptions on the behavior of the environment, so *all* possible inputs of the environment to the system have to be taken into account in each step.[1] It has turned out that a coherent and appealing way to view such scenarios is as *games* between the controllers and an antagonistic environment.

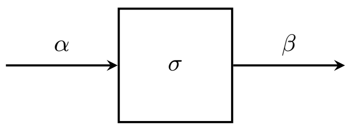
1.1 Church's Synthesis Problem

The ultimate goal of describing such situations by formal models is to be able to construct, from a given model and some suited formal specification of the correct system behaviors, implementations for all controllers of the system which guarantee satisfaction of the specification. In other words, we try to *synthesize* an implementation of the system controller, which is called *controller synthesis*.[2] That means, in the game model we try to find a joint winning strategy for those players that represent the controllers. The coalition of these players is also called the *grand coalition*. The major starting point of this theory is the groundbreaking work of Alonso Church [54, 55] where he introduced the *Circuit Synthesis Problem (1957)*:

Given a requirement which a circuit is to satisfy, we may suppose the requirement expressed in some suitable logistic system which is an extension of restricted recursive arithmetic. The

[1]Such systems are sometimes also called *open* systems [169, 127]. Notice however, that all these terms and notions concerning different kinds of computing systems are not defined precisely, nor is their use completely standardized.

[2]Rather than just *checking* a system that has already been built against some specification, which is called *verification*.



$$\alpha \in \{0,1\}^\omega$$
$$\beta \in \{0,1\}^\omega$$
$$L \subseteq \{0,1\}^\omega \times \{0,1\}^\omega$$

Figure 1.1: Church's Synthesis Problem

synthesis problem is then to find recursion equivalences representing a circuit that satisfies the given requirement (or alternatively, to determine that there is no such circuit). ([54], p.8-9)

In this scenario, in each computation step of the system, the switching circuit receives an input bit from an infinite bit string and has to produce an output bit, so it is a nonterminating reactive system with a single controller (with full information about the history of events in the system!), see Figure 1.1. In our terminology, the problem can be stated as follows:

Given a formula φ of monadic second order logic of one successor, defining a language $L \subseteq \{0,1\}^\omega \times \{0,1\}^\omega$, find a function $\sigma : \{0,1\}^\omega \to \{0,1\}^\omega$ which is computable by a finite state automaton with output M such that $(x, \sigma(x)) \in L$ for all $x \in \{0,1\}^\omega$ (or show, that there is no such function).

Notice that the computability of σ by M means that the i-th bit of $\sigma(x)$ is the output of M after reading the first i bits of x. The first solution to Church's synthesis problem was given by Büchi and Landweber [44]. They reduced the problem to solving a *Muller game on a finite game graph* and provided a solution of such games. In [177] Rabin gave an alternative proof using *tree automata.* For a closer look at the history of controller synthesis for reactive systems and, in particular, Church's problem, we refer the reader to [204, 205].

1.2 Subsequent Development

After these fundamental results, for a while, research on synthesis focused on synthesis of transformational programs [218, 138], using theorem proving techniques, and synthesis of closed reactive systems [56, 139], where the key step is checking satisfiability of the specification. An important foundation for synthesis of closed reactive systems has been the invention of temporal logics for program specification like linear temporal logic [167] and computation tree logic [56]. At the end of the 1980s, the importance of *open* reactive system as in Church's scenario has been rediscovered [169, 1].

Since then, controller synthesis has been an active area of research and many extensions as well as refinements of the problem have been addressed. For example, other specification formalisms have been considered like temporal logics [169, 127] and context-free specification [219, 128]. Moreover, the systems have been extended to systems with partial information where the controller does not necessarily have full information about the events in the system [171, 127]; to distributed systems which consist of an antagonistic environment and several controllers, each of which has different information and control about the events in the system [171, 80]; and to stochastic systems, where the environment as well as the controller may be probabilistic [63, 53, 42]. In [64] and in [178], Church's problem has been considered in more general logical settings.

1.2.1 Partial Information

In the context of the theory of computing systems, the concept of partial information in games can be traced back to the late 1970s. In 1978, Jones [117] considered a special case of reachability games with partial information on finite graphs, so called blindfold games. In 1979, Reif [181] considered general reachability games with partial information on finite graphs, see also [182].

Also in 1979, Peterson and Reif [163] went beyond two-player games and considered multiplayer reachability games on finite graphs with partial information.

However, these investigations were initially completely unrelated to synthesis of reactive systems. They were motivated by recent results on the complexity of certain concrete combinatorial games (like the so called *Shannon switching game*) and the work of Chandra, Kozen and Stockmeyer [49, 48] on alternating computing devices. In particular, Reif generalized the concept of alternating Turing machines to private alternating Turing machines, which correspond to reachability games with partial information. In the same way, Peterson and Reif generalized the computing model to multiple person alternating machines which correspond to multiplayer reachability games with partial information.

Unlike alternating computing machines, the (multiplayer) blind and private alternating Turing machines did not have much impact on computational complexity theory. One reason is that, while alternation is a very natural completion of the fundamental concept of nondeterminism[3], partial information seems a bit arty and random in this context. Moreover, while many problems have very natural solutions in terms of alternating computing devices, alternating Turing machines with partial information are often quite intricate to build.

The great importance of partial information and multiple players in the context of synthesis of reactive systems was noticed only later. In 1990, in their fundamental paper [171], Pnueli and Rosner considered the problem of synthesizing *distributed systems*. They argued, that asking only for single process implementations of controllers is too restrictive:

The limitation is that all the synthesis algorithms produce a program (strategy) for a single module (processor) that receives all the inputs to the system and generates all the outputs. This is particularly embarrassing in cases that the problem we set out to solve is meaningful only in a distributed context, such as the mutual exclusion problem, and a centralized single module solution does not seem very relevant. ([171], p.2)

Clearly, any system that has a distributed controller solution also has a single controller solution, because the network of controllers can be assembles to a single controller, reading all the inputs and producing all the outputs. However, it is often not appropriate to do this a priori, because then we don't have access to the internal (distributed) structure of the controller. In particular, we cannot address the communication structure between the individual computing units which might be necessary, for example, to fix a certain communication protocoll that the components have to use to communicate. So, distributed system synthesis may yield solutions for more complex systems where a single controller solution can not be technically realized.

In the distributed systems setting, partial information comes into play quite naturally, since each two controllers which have exactly the same information about the events in the system can easily be combined to a single controller. In particular, if all controllers have full information about the events in the system, then the problem reduces to the synthesis problem for a single controller with full information: the different powers that the individual controllers have over the events in the system can be easily decomposed.

Subsequently, many authors have argued that, also in the single controller setting, studying the extension of Church's original problem to interaction under partial information is inevitable for modeling realistic scenarios as well as a valuable contribution to the theory of computing systems [221, 127, 52]. In particular, reactive systems are often embedded into more complex systems like trains, cars or certain parts of a traffic control and the controllers of the system acquire certain information using sensors with finite precision, which might make the information imprecise (depending on the underlying discretization of the continuous parts of the system). Moreover, the environment as well as the controllers of the system may encapsulate certain variables which cannot be accessed by other components directly. These variables may be private to the component or may only be broadcasted via some limited communication structure which does not enable full transmission of the values.

[3]Notice that for machines on words, universality is the *dual* of nondeterminism.

1.3 Setting

Our basic abstract model are *infinite multiplayer games with partial information* where the grand coalition of n cooperating players $1, \ldots, n$ plays against the environment player 0. So, if we model a system with n controllers then, in the game, we have $n+1$ players. In each round of the game, each player chooses an *action* from some predefined *finite* set. We consider only *non-stochastic* games[4], so the result is an infinite sequence of actions, called a *play* of the game. A finite prefix of a play is called a *history* of the game. The infinite tree of all possible histories is called the *game tree*. The players have *partial information about the history* in the game, that is, at each point during a play of the game, a player may consider a set of histories to have been actually played so far instead of just the actual history. As is common in classical game theory [160] we call games with this kind of partial information *games with imperfect information*, see also Chapter 2.

A *strategy* for player i is a function that takes the information about the history of a play which player i has and provides the next action for player i to choose. The specification of the system is the *winning condition* of the game, which in our setting is usually given as a set of infinite plays of the game that represent those system behaviors which the controllers want to ensure. This, of course, puts our investigation into the realm of *linear time*. We will, however, make some remarks about branching time specifications. The specifications we consider here range from simple reachability objectives to context-free specifications.

We consider two realizations of our basic abstract model. One model are *game graphs with partial information* where the players move a token along the directed edges of the graph. The information of a player about the positions of the game graph is given in terms of an *equivalence relation*, called indistinguishability relation: equivalent positions are *indistinguishable* for that player. Unless explicitly mentioned otherwise, we consider only *finite* game graphs. Notice that this does not limit our investigation entirely to finite state systems because context-free *specifications* induce an infinite state space.[5] However, a large part of our results does concern finite state systems.

The second model are *distributed systems*, which define a set of players, also called *processes*, and the information flow between these processes by means of a directed graph, also called *architecture*. The edges of the graph are called *channels* and each process has exactly the information available which is *sent to him* via channels of the architecture. The players of the grand coalition are called *controllers* and we speak of the strategy problem for distributed system as *controller problem*.[6] Moreover, we usually keep the term *specification* for the winning condition.

Unless explicitly mentioned otherwise, the information that a player has about a finite history in the resulting game with imperfect information is defined in terms of *synchronous observability* in both models,[7] see Chapter 2. Notice that both realizations provide *finite presentations*.

It is important to make clear, that these models have, in a sense, the same expressive power, that means, each instance of our basic abstract model that can be represented in one model can also be represented in the other model. However, the representation of *certain parameters* is quite different in the models so if we want to analyze the impact of a certain parameter on decidability and complexity of the synthesis problem, one of the models may be more suited than the other one.

[4]We suggest, however, that knowing and understating certain phenomena in stochastic games with partial information is essential for a comprehensive understanding of partial information. In any case, the investigation of stochastic systems is clearly also a very natural and important extension of Church's synthesis problem and a highly successful and active area of research. We refer the reader to [50, 206].

[5]In fact, games on finite graphs with deterministic context-free winning condition can also be viewed as games on infinite game graphs with regular winning condition, cf. Section 2.2.2.

[6]Notice that this is a somewhat inaccurate terminology. Rather, we would have to speak of the controller realizability problem. However, since the term of controller problem is short and common, we stick to it.

[7]In particular, the players have *perfect recall*.

For example, if we consider games played on graphs we can immediately distinguish instances according to the complexity of the game graph with respect to certain graph complexity measures. Architectures do not provide an explicit game graph but the structure provided by the graph has to be encoded in the winning condition. Hence, we cannot immediately distinguish instances with respect to graph complexity. On the other hand, in a distributed systems, the information flow between the processes is explicitly given by means of a communication graph. It has turned out that there are elegant characterizations of decidable architectures (communication graphs) according to certain patterns of information flow [80]. Games on graphs do not provide an explicit communication structure but the information flow between the players has to be encoded in the indistinguishability relations. So we cannot directly distinguish instances with respect to patterns of information flow. We will discuss the two models and their relationship in Chapter 2.

1.4 Outline and Main Contributions

As we have mentioned, we focus on the *strategy problem*, that is, given a game with imperfect information, does the grand coalition of cooperating players have a joint winning strategy? An important characteristic of this problem in our *non-stochastic* setting is that we can always assume that the environment has *full* information about the whole system. We will discuss this in Chapter 2. The main aspect of our investigation will be decidability and complexity of the strategy problem. Notice that this problem is different from Church's problem in that a winning strategy for a (single) controller is only a solution to Church's problem if we can also construct an implementation of the strategy by some kind of computation device. The strategy problem as we have formulated here asks for *any* winning strategy for the grand coalition which, in particular, might be some nonrecursive function.

Of course, once we know that a joint winning strategy exists we would like to be able to synthesize an implementation of such a strategy by some kind of computing device which is called *strategy synthesis.* We will see that our decision procedures also imply techniques for synthesizing winning strategies which do not require significantly more computational effort than the solution of the decision problem. Depending on the specification formalisms, the implementations are given by finite state automata or pushdown automata.

As it turns out, the strategy problem for games with imperfect information is computationally hard. For games on finite graphs it is undecidable for three players [163, 171] and still EXPTIME-hard for two players [181], even for very simple regular objectives like reachability and safety. For deterministic context-free specifications the problem is already undecidable for two players [163]. However, restricting the possible ways of information flow between the players, the amount of uncertainty that a player may have or to which extend the winning condition may involve facts that the players cannot observe, leads to relevant decidable, and even tractable, subcases of the problem. We conduct a detailed analysis of several such subcases along this kind of parameters which concern partial information and information flow in the game.

Aside from decidability and tractability of certain subcases of the general strategy problem, we are also interested in the question in which cases we can apply a specific method, usually referred to as *knowledge tracking.* Knowledge tracking means that we construct, from a certain game structure $\mathcal{G}$ with partial information, a new structure $\mathrm{Tr}(\mathcal{G})$, consisting of epistemic states, which comprises all the possible states of knowledge that the players of the grand coalition may have during some play of $\mathcal{G}$. Moreover, the structure should reflect how moves in plays of $\mathcal{G}$ cause transitions between these knowledge states. So the result of such a knowledge tracking construction is a game with full information such that strategies can be translated from the original game to the new game and vice versa. Since this yields an explicit representation of the possible states of knowledge of the players and the dynamics of this knowledge, analyzing the possibilities and limitations of knowledge tracking is of independent interest and meaningful also in cases which are already known to be decidable by automata based methods.

We will structure our analysis along the relevant parameters of games with imperfect information and their representation in our models: (1) the number of (cooperating) players; (2) the complexity of the winning condition, where we have to distinguish between (2.1) the expressive power and (2.2) the extent to which the winning condition may involve facts that the players cannot observe. Moreover, (3) the complexity of information flow between the players which involves, for example, the structure of the communication graph in distributed systems or the size of the equivalence classes which are induced by the indistinguishability relations in game graphs with partial information and, finally, also in the context of game graphs with partial information (4) the complexity of the game graph with respect to graph complexity measures.

In Chapter 2 we introduce and discuss the models that we consider here. In particular, we compare the models of game graphs with partial information and distributed systems and we introduce relevant notions like winning conditions and finite state strategies. Chapter 3 is devoted to a review of basic methods for solving strategy problems under partial information. We discuss how alternating tree automata can be applied to synthesis under partial information and we study their close relationship to partial information in general. Moreover, we introduce the concept of knowledge tracking for two-player games and we present certain extensions and unifications of known solutions.

Chapter 4 is concerned with *two-player games (1)* on finite game graphs with partial information and *observable parity conditions (2)*. We study the influence of the *complexity of the game graphs (4)* on the complexity of the strategy problem for such games. From Reif's fundamental work [181, 182] it can be easily obtained that this problem is EXPTIME-complete. On the other hand, the strategy problem for parity games with *full* information is known to be in NP ∩ co-NP due to positional determinacy of parity games with full information [75] and it is a major open problem in the theory of synthesis and, especially, verification whether the problem is in PTIME, cf. [118]. However, it has been shown that the problem can be solved in polynomial time on classes of graphs that have a bounded complexity with respect to several graph complexity measures such as tree-width [156], DAG-width [24] or entanglement [26].

We show that these results do *not* carry over to games with imperfect information. More precisely, we prove that the strategy problem is EXPTIME-hard even for reachability games with imperfect information on graphs of DAG-width at most three. Moreover, the problem is still PSPACE-hard on acyclic graphs. A natural restriction of the general case is to bound the *size of the equivalence classes of positions (3)* induced by the indistinguishability relation for player 1. That means, the amount of uncertainty that player 1 may have about the positions of the game graph is fixed, independently of the size of the game graph. We prove that in this case, the strategy problem for two-player games with observable parity conditions can be solved in polynomial time on graphs of bounded DAG-width. For this, we introduce a new measure dw_r of graph complexity which is characterized by a graph searching game with *multiple* robbers that have the capability to *jump* to each other. Our main technical result about this new graph complexity measure is that if k cops monotonously capture a single robber on a directed graph G then $k \cdot r$ cops monotonously capture r jumping robbers on G.

After this intensive analysis of two-player games culminating in a PTIME-result for a rather restricted case of the strategy problem, we turn to more general settings. Chapter 5 and Chapter 6 are devoted to the strategy problem for games with an arbitrary number of players. As we have mentioned, this problem is undecidable in general, even for three players and regular specifications.[8] However, in the setting of distributed systems where a communication graph is given explicitly, restricting the possible ways of information flow between the processes has led to relevant decidable subcases [171, 129] and, ultimately, a complete characterization of the decidable architectures by means of patterns of information flow [80].

[8] In [163] undecidability has already been proved even for reachability condition but for a model with *asynchronous* observability. A first result for synchronous observability has been given in [171], showing that the problem is undecidable for LTL-specifications.

This however, was done for the case of *arbitrary* (regular and branching time) specifications. In Chapter 5, we also restrict *the extent to which the specifications may involve facts that the processes cannot observe (2.2)*. More precisely, we consider *locally decomposable* specifications, that means, specifications which are obtained (by conjunction) from a collection of *local* specifications for the individual processes, each of which involves only facts that the particular process can observe. In [137], a complete characterization of the decidable architectures has been provided for locally decomposable specification, but only for *acyclic* architecture and *regular* specifications. Our main result in Chapter 5 is an extension of the characterization to the case of architectures which may contain *cycles* and to *context-free specifications*.

So far, our view of the uncertainties and communication of the players in a game was rather pragmatic and focused on strategic powers. In Chapter 6 we discuss the epistemic context of our models and, in particular, we turn to the task of developing methods for representing the possible states of mind of the players explicitly. We first provide a broad discussion of knowledge and cooperation in infinite games, in particular, synthesis from epistemic specifications and strategic dependencies. We have a look at the epistemic temporal logic ETL and we prove that any winning condition that can be defined in ETL using both, synchronous and asynchronous, knowledge operators is, in fact, ω-regular. This generalizes the decidability of ETL model checking to asynchronous observability and has also significant consequences on the decidability of synthesis from ETL specifications.

The main result is a knowledge tracking construction for game graphs with partial information and an arbitrary number of players. Moreover, the construction can handle arbitrary winning conditions. The result of the construction is a game graph with full information which is, however, an infinite tree with epistemic models as nodes. To obtain a less extensive representation of the game graph, we identify nodes of the tree that are *homomorphic equivalent*. We prove that for the case of *observable winning conditions (2.2)* this quotient construction is sound, which yields a semi-decision procedure for the strategy problem for such games. Moreover, we show that decidability of the strategy problem for *hierarchical games* [171, 129] can be obtained as a special case of our construction for observable winning conditions and we extend this result to deterministic context-free specifications.

Chapter 2

Modeling Interaction under Partial Information

In this chapter we introduce and discuss the models for interaction under partial information that we will work with and we present a detailed discussion of several special cases of finitely presented winning conditions that we add to the model in the second step. All these winning conditions are context-free languages.

In Section 2.1 we set up the basic framework of games with imperfect information and the two concrete instantiations, game graphs with partial information and distributed systems. We also demonstrate some basic properties that concern, for example, position based strategies in games on graphs and focused strategies in distributed systems. Later on, such properties will be used quite frequently, but rather implicitly. In both these models, imperfect information is defined via synchronous observability. In Section 2.1.4 we briefly discuss more general notions of imperfect information, in particular asynchronous observability. Section 2.1.5 provides some basic definitions concerning finite state devices as memory structures which are needed to make clear what kind of strategy implementations we are looking for.

Section 2.2 is devoted to winning conditions. We discuss ω-regular winning conditions (Section 2.2.1), in particular parity conditions, and the corresponding automata models, as well as context-free specifications (Section 2.2.2). We present all the tools that we will need to deal with these winning conditions in the context of synthesis under partial information. In Section 2.2.3 we introduce the concept of observability which is not about expressiveness but restricts the extent to which a winning conditions may involve facts that the players cannot observe. This concept is of central importance for us: The main results of this work are on observable and locally decomposable winning conditions.

In Section 2.3 we compare game graphs with partial information and distributed systems.

Basic Notation. The set of natural numbers is denoted by $\mathbb{N}$ or sometimes, if we use it as an ordinal number, by ω. For a natural number $k \in \mathbb{N}$, let $[k]$ denote the set $\{0, ..., k-1\}$. The boolean alphabet is denoted by $\mathbb{B}$ and we shall use the notation $\mathbb{B} = \{\bot, \top\}$ as well as $\mathbb{B} = \{0, 1\}$. For a set X, the power set of X is denoted by 2^X. If $\sim \subseteq X \times X$ is an equivalence relation and $x \in X$, the equivalence class of x is denoted by $[x]_\sim$. The set of equivalences classes induced by $\sim$ on X is denoted $X/\sim$ or $[X]_\sim$. We omit the subscripted equivalence relation as long as no confusion arises. For any function f, the domain of f is denoted $\mathrm{dom}(f)$. If $f\colon X \to Y$ and $g\colon Y \to Z$, the composition $f \circ g\colon X \to Z$ is defined by $(f \circ g)(x) = g(f(x))$. Moreover, for $X' \subseteq X$ we denote $f(X') = \{f(x) \,|\, x \in X'\}$.

For a Cartesian product $X = X_0 \times \ldots \times X_{n-1}$ and $I \subseteq [n]$ we denote $X_I = \prod_{i \in I} X_i$. Moreover, for $I, I' \subseteq [n]$, $\mathrm{Pr}_I(x) = (x_i)_{i \in I \cap I'}$ for an element $x \in X_{I'}$, $\mathrm{Pr}_I(\alpha) = \mathrm{Pr}_I(\alpha_0)\mathrm{Pr}_I(\alpha_1)\ldots$ for a word $\alpha \in X^* \cup X^\omega$ and $\mathrm{Pr}_I(L) = \{\mathrm{Pr}_I(\alpha) \,|\, \alpha \in L\}$ for a language $L \subseteq X^* \cup X^\omega$. Notice that we apply a projection operator Pr_I to tuples $x \in X_{I'}$ which do not necessarily contain all the components from X_I, that is $I \subseteq I'$ does not necessarily hold. To improve readability, we often do not refer to an explicit ordering of the components of a Cartesian product and simply write Pr_{X_I} instead of Pr_I. If X has certain identical components, this is ambiguous, but it will be clear from the context to which components we project.

For an alphabet Σ, Σ^* denotes the set of all finite words $[k] \to \Sigma$ over Σ and Σ^ω denotes the set of all infinite words $\omega \to \Sigma$ over Σ. Finite words are usually denoted u, v, w with $u = a_0 a_1 \dots a_{l-1}$ and infinite words are denoted α, β, γ with $\alpha = a_0 a_1 a_2 \dots$. If we refer to a word which may be either finite or infinite we use the notation for infinite words. For some words $\alpha, \beta \in \Sigma^* \cup \Sigma^\omega$ with $\alpha = a_0 a_1 a_2 \dots$ and some $j \in \mathbb{N}$, we denote the $(j+1)$-th letter a_j of α also by α_j or $\alpha(j)$ and we write $\alpha_{<j}$ or $\alpha({<}j)$ for the prefix $\alpha_0 \dots \alpha_{j-1}$ consisting of the first j letters of α. Notice that $\alpha({<}j)$ is the empty word, denoted ε. If $\alpha = a_0 a_1 \dots a_{l-1} \in \Sigma^*$ we denote $|\alpha| := l$ and by $\mathrm{first}(\alpha) = a_0$ and $\mathrm{last}(\alpha) = a_{l-1}$ we denote the first and last letter of α, respectively. Moreover, by $\alpha^{-j} := \alpha({<}|\alpha| - j)$ we denote the word which is obtained from α by pruning the last j letters. We write $\alpha \sqsubseteq \beta$ if α is a prefix of β, that is, if $\beta({<}|\alpha|) = \alpha$. For a set X and a word $\alpha \in X^\omega$, we denote $\mathrm{Inf}(\alpha) = \{a \in X \mid \alpha_i = a \text{ for infinitely many } i\}$.

If $\alpha \in \Sigma^\omega$ and $\beta \in \Gamma^\omega$, $\alpha^\frown\beta \in (\Sigma \times \Gamma)^\omega$ denotes the ω-word with $(\alpha^\frown\beta)_i = (\alpha_i, \beta_i)$ for all $i \in \mathbb{N}$. For finite words $u \in \Sigma^*$ and $v \in \Gamma^*$, the word $u^\frown v$ is defined analogously but, if $|u| \neq |v|$, we have to fill up one component with a default symbol that we denote $\sharp$. So if $|u| = |v|$ then $u^\frown v$ is a word from $(\Sigma \times \Gamma)^*$ and if $|u| \neq |v|$ then $u^\frown v$ is a word from $((\Sigma \cup \{\sharp\}) \times (\Gamma \cup \{\sharp\}))^*$.

For a function $\sigma \colon \Sigma^* \to \Sigma'$, we define the finite iteration $\sigma^* \colon \Sigma^* \to (\Sigma')^*$ of σ by $\sigma^*(u) = \sigma(u_{<0}) \dots \sigma(u_{<|u|})$ and the ω-iteration $\sigma^\omega \colon \Sigma^\omega \to (\Sigma')^\omega$ of σ by $\sigma^\omega(\alpha) = \sigma(\alpha_{<0})\sigma(\alpha_{<1}) \dots$. Notice that $|\sigma^*(u)| = |u| + 1$. The ω-language which is generated by σ over a language $L_{\mathrm{in}} \subseteq \Sigma^\omega$ of (infinite) input words is $\sigma^\omega(L_{\mathrm{in}}) = \{\sigma^\omega(\alpha) \mid \alpha \in L_{\mathrm{in}}\} \subseteq (\Sigma')^\omega$. Moreover, the $*$-language generated by σ over L_{in} is $\sigma^*(L_{\mathrm{in}}) = \{\sigma^*(u) \mid u \sqsubseteq \alpha \text{ for some } \alpha \in L_{in}\}$. Notice that we have $\sigma^*(L_{\mathrm{in}}) = \{v \in (\Sigma')^* \mid v \sqsubseteq \beta \ \text{ for some } \ \beta \in \sigma^\omega(L_{\mathrm{in}})\}$.

2.1 Game Graphs and Distributed Systems

To start with, recall Church's synthesis problem and the representation in Figure 1.1. There, we have a single controller that successively receives input bits from the environment and reacts, in each step, by sending an output bit. So this representation of Church's scenario already specifies a *distributed system*: we have two processes and two channels, as depicted in the figure. Of course, in a setting with only a single controller with full information, we never have to mention the architecture explicitly. On the other hand, as we have mentioned, the solution of Church's problem provided by Büchi and Landweber uses a reduction of this problem to a Muller game, played on a *finite game graph* which is basically the transition graph of a deterministic Muller automaton that recognizes the monadic second order specification of the system.

Partial information came into play in [181] where John H. Reif used games with partial information for his work on generalizations of Chandra and Stockmeyer's alternating Turing machines. Reif called these games *games of incomplete information.*[1] In classical game theory, however, this term is used for games in which a player may be uncertain about the possible strategies and objectives of the other players. There is still no canonic terminology for partial information in the theory of computing systems, but *games with imperfect information* or *games with partial observation* seem to be widely acknowledged for the kind of games we consider here, where the players have partial information about *the history of past events* in the game.

We introduce games with imperfect information as a basic abstract model that encompasses many settings of interaction under partial information considered in the theory of computing systems. We start with a casual description of games with imperfect information in extensive form, that is, games played on infinite trees where the players have partial information about

[1]In [127], Kupferman and Vardi also called the synthesis problem for single controller systems but with possibly *hidden channels* from the environment *synthesis with incomplete information.*

the current position in the tree. We will use this, however, merely as a common framework for the models of game graphs with partial information and distributed systems, allowing for a comprehensive exposition of fundamental notions and characteristics such as the strategy problem and determinacy. All our constructions and results explicitly refer to one of these particular models and we will always highlight the view of interaction under partial information as suggested by this model.

2.1.1 Extensive Games with Imperfect Information

An *extensive game with imperfect information* has the form

$$\mathcal{T} = \big(T, (\sim_i)_{i \in [n+1]}, W\big)$$

where T is a *game tree*, that is, T is a tree with edges labeled by elements from a set A, also called *actions*. We assume that each edge is labeled by exactly one action and that, for each node $t \in T$ in the tree and each action $a \in A$ there is at most one a-successor t' of t. Moreover, an action $a \in A$ is compounded of individual actions of the players, i.e., $a = (a_0, \dots, a_n)$ where action a_i is chosen by player i. A maximal (labeled) path $\pi = t_0 a_1 t_1 \dots$ through T is called a *play* of $\mathcal{T}$. We refer to the occurrence of an action and the subsequent transition to the next level of the game tree also as an *event*. Hence, a play is a sequence of events starting from the root of T. We consider infinite games, so we assume that each play of $\mathcal{T}$ is infinite. A finite prefix π of a play is called a *history* of $\mathcal{T}$. Notice that each history of $\mathcal{T}$ corresponds uniquely to a node of the game tree T, so we shall not distinguish between histories and nodes of T. Moreover, for $i \in [n+1]$, $\sim_i \subseteq T \times T$ is an equivalence relation and W is a set of plays of $\mathcal{T}$. The set W is the winning condition for the *grand coalition* of players $1, \dots, n$ that means, the plays in W are those which are won by this grand coalition, all other plays are won by the opponent player 0.

The equivalence relation $\sim_i$ defines the *imperfect information* of player i, that means, any two histories $s, t \in T$ with $t \sim_i s$ are *indistinguishable* for player i (and any $s, t \in T$ with $t \not\sim_i s$ are distinguishable for player i). The indistinguishability of certain histories means that player i has exactly the same information about all those histories, hence an $\sim_i$-equivalence class of histories is also called an *information set* of player i. So whenever the game is in some node $t \in T$, player i does not necessarily know that history t has been played but he considers exactly the histories $s \in [t]_{\sim_i}$ possible to have been actually played.

A game $\mathcal{T}$ is called a game with *perfect* or *full* information, if the following two conditions are fulfilled. First $\sim_i = \mathrm{id}_T$ for all $i \in [n+1]$, that is, each player can distinguish any two histories. And second, the game is turn-based, that means, for each history $t \in T$ there is a *unique* player to move at t. Clearly this can be realized in the model presented here by making, at each node $t \in T$, the actions of all but one player trivial, that is, they can only choose a default action $\perp$. This second postulation is necessary since the *concurrent* moves in our model implicitly incorporate imperfect information into the game: if two players 0 and 1 are to move simultaneously, then we could equivalently let first make player 0 his move and then let player 1 make his move while hiding the previous move of player 0 from player 1.

For games with full information, we omit the indistinguishability relations in the description and denote such a game as $\mathcal{T} = (T, W)$. Moreover, notice that since we consider the players $1, \dots, n$ as forming a grand coalition of *cooperating* players, under the assumption of full information, the players $1, \dots, n$ can be simulated by a single player which has full information. We will use this observation implicitly, so we assume that games with full information are always two-player games.[2] Such games are also called *Gale-Stewart games* as a tribute to the seminal work [90] where infinite games of perfect information have been studied in the context of topological properties of the winning sets W.

[2]Notice, however, that in other settings where, for example, each player $0, \dots, n$ has his own objective and we ask for a Nash-equilibrium, the multiplayer case is clearly richer than the two-player setting, also under the assumption of full information, see for example [101].

Remark. Notice that the definition of an extensive game with imperfect information in classical game theory somewhat differs from ours [124, 160]. The main differences are of course due to the fact that the situation which the game is supposed to model is different: The games there are usually finite and, rather than having a coalition of cooperating players playing against a single antagonist, each player has his own objective. There are also differences in the presentation. In particular, the games are generally turn-based and the indistinguishability relation $\sim_i$ of player i is only defined on those histories where it is player i's turn. While for defining imperfect information in the *game tree* this is sufficient, in models where imperfect information is defined in terms of observability of moves, $\sim_i$ is intrinsically defined on all histories. This will become apparent in a turn-based variant of the model of game graphs with partial information.

Strategies. The imperfect information of the players comes into effect in the actual course of decision making: In each round of the game, being in some node $t \in T$, each player i chooses an individual action a_i which is available to him in t, that means, there is a compound action $a \in A$ such that there is some edge from t labeled by a and $\text{Pr}_i(a) = a_i$. The resulting collection of individual actions defines a compound action a and the game proceeds to the a-successor of t. (We assume that this is always possible, that means, if $a_0, \dots, a_n$ are available to players $0, \dots, n$ at node t, then t has an a-successor.) We assume that any player always knows which actions are available to him in the current situation: For all i, whenever $t \sim_i s$, the same actions are available to player i in t and in s.

Now the connection between imperfect information and the decisions of the players is that, for deciding on his next move, a player can not use any information on the history that is not available to him. This restricts the possible *strategies* of a player to those which are based on the information of that player, that means, constant over equivalence classes of histories: A *strategy* for player i is an A_i-labeling of T, that means, a function

$$\sigma_i \colon T \to A_i,$$

such that for all histories $t \in T$:

(S1) $\sigma_i(t)$ is an action which is available to player i in t

(S2) for all histories $s \sim_i t$ we have $\sigma_i(s) = \sigma_i(t)$.

Notice that in our setting, strategies are *deterministic* and hence, a strategy for player i completely determines the behavior of player i in the game. Condition (1) ensures that a strategy prescribes a *legal* behavior for each possible situation in the game. Condition (2) ensures that the strategy does not use any information which is not available to player i.

A history $\pi = t_0 a_1 t_1 \dots a_l t_l$ in $\mathcal{T}$ is *consistent* with a strategy σ_i for player i if for all $0 \le j \le l-1$,

$$\text{Pr}_i(a_{j+1}) = \sigma_i(t_j),$$

that means, the action chosen by player i in the joint action a_{j+1} is exactly the action that is prescribed by the strategy σ_i in node t_j. A play is consistent with σ_i if all its histories are consistent with σ_i. We call a history which is consistent with a strategy σ_i also a *σ_i-history*, analogously for plays.

A joint strategy (or simply a strategy) for the grand coalition is a collection

$$\sigma = (\sigma_1, \dots, \sigma_n)$$

of individual strategies for the members of the grand coalition. Or, equivalently, it is a function

$$\sigma : T \to \prod_{i=1}^{n} A_i$$

such that $\mathrm{Pr}_{A_i}(\sigma) : T \to A_i$ fulfills conditions (S1) and (S2) for all i. A history (play) is consistent with a joint strategy σ if it is consistent with each individual strategy. A strategy σ for the grand coalition is a winning strategy for $\mathcal{T}$ if each σ-play of $\mathcal{T}$ is won by the grand coalition, that is, contained in W.

Clearly, for the winning property of a strategy σ the value of σ is important only on those histories, which are (inductively) already consistent with the strategy. Accordingly, as far as winning strategies are concerned, the conditions (S1) and (S2) could also be restricted to histories which are consistent with σ. Although we are mainly interested in winning strategies here, we still prefer to have a strategy defining a value on all possible histories as this is sometimes more convenient. Nevertheless, when we construct a winning strategy it is clearly sufficient to define the value only on histories which are already consistent with this strategy: We can always assume that the value of the strategy on all other histories is arbitrary (but constant over equivalence classes!).

The Strategy Problem. As we have mentioned, we are mainly interested in the *strategy problem for games with imperfect information*:

Given a game $\mathcal{T}$ with imperfect information, does the grand coalition have a joint winning strategy for $\mathcal{T}$?

Of course, for an *algorithmic* treatment of this question, we need finite presentations of extensive games with imperfect information. Game graphs with partial information and distributed systems provide such a finite presentation.

We have already suggested that partial information of player 0 is irrelevant here which is due to the fact that partial information comes into effect only in the notion of a strategy and strategies of player 0 are not involved in this formalization of the strategy problem.

One might consider that as a shortcoming of the formalization and think of the following formulation as being more appropriate:

Given a game $\mathcal{T}$ with imperfect information, does the grand coalition have a joint strategy σ which is winning against all strategies of player 0, that means, *for all strategies* σ_0 *of player* 0, all plays which are consistent with σ and σ_0 are won by the grand coalition?

Of course, this formulation implicitly assumes, that we build a controller against some environment which follows a *predefined* but *unknown* strategy. However, it is not hard to see that under the assumption that player 0 always knows the number of his own past moves in the game (which, in the setting with simultaneous moves, means that he always knows the number of total past moves in the game), these two formulations are *equivalent*:

Proposition 2.1. *If, for all $s, t \in T$, $t \sim_0 s$ implies that t and s are on the same level of T, then the grand coalition has a winning strategy for $\mathcal{T}$ if, and only if, the grand coalition has a strategy for $\mathcal{T}$ which is winning against all strategies of player* 0.

Proof. Clearly, if the grand coalition has a winning strategy σ, then σ is also winning against all strategies of player 0. If, on the other hand, the grand coalition has a strategy σ which is winning against all strategies of player 0, then for any play π which is consistent with σ, it is easy to construct a strategy σ_0 for player 0 such that π is consistent with σ_0: For any history t of π we define $\sigma_0(t)$ to be player 0's next action in the play π. (Notice that due to the assumption that player 0 always knows the number of his past moves there are no two different histories of π which are indistinguishable for player 0.) For all histories s in $\mathcal{T}$ which are not prefixes of π we can define $\sigma_0(s)$ arbitrarily but constant on $\sim_0$-equivalence classes of histories, which is clearly possible. □

In the settings we study, the property that a player knows the number of moves that he has made previously, will always be granted and we shall only consider the strategy problem as stated above. Nevertheless, we like to mention that we could obtain nontrivial variants of the problem which would also involve the information of player 0 if we didn't ask for strategies for the grand coalition which are winning against *all* strategies of player 0 but only against strategies which can be implemented by certain kinds of computing devices. However, in our investigation we do not restrict the possible strategies of the players a priori.

Proposition 2.1 also demonstrates that, as long as the strategy problem is concerned, concurrent moves are noncritical, also for games with full information: We can first let the players of the grand coalition make their joint move and then let player 0 make his move and there is no need to hide the previous move of the grand coalition from player 0. However, concurrent games are not determined in general, even for reachability conditions.

Determinacy and Imperfect Information. One of the most important and fundamental notions for games is that of *determinacy*: A game is called *determined* if either the grand coalition has a joint winning strategy $\sigma = (\sigma_1, \ldots, \sigma_n)$, or, if they do not, player 0 has a winning strategy σ_0. A fundamental result of Martin [140] states that any *full information* two-player game $\mathcal{T} = (T, W)$ where W is a *Borel set* is determined. We do not define Borel sets formally, since we will not be concerned with them here. We note, however, that Borel sets subsume many important classes of omega-languages such as regular and deterministic context-free languages which we will consider as winning conditions.

On the other hand, Martin's Theorem does *not* hold for games with imperfect information. In fact, it is obvious that determinacy already fails in concurrent two-player games with *reachability conditions* where the players only make one concurrent move. Conveniently, meaningful considerations of the strategy problem as stated above are not subject to determinacy, since we are merely interested in winning strategies for the grand coalition and do not care about the existence of winning strategies for player 0. However, the failure of determinacy in games with imperfect information is clearly one of the most important characteristics of interaction under partial information and should be kept in mind whenever partial information comes into play.

Also notice that determinacy plays an important role in the theory of finite automata on infinite trees that we will use for solving games with imperfect information, see Section 3.1.

2.1.2 Game Graphs with Partial Information

A *game graph with partial information and* $n+1$ *players* has the form

$$\mathcal{G} = \big(V, \delta, (\sim_i^V)_{i=1,\ldots,n}, (\sim_i^A)_{i=1,\ldots,n}\big)$$

- V is a finite set of positions
- $\delta : \mathrm{dom}(\delta) \subseteq V \times A \to V$ is the move function where $A = \prod_{i \in [n+1]} A_i$ is the set of joint actions $a = (a_0, \ldots, a_n)$ of the players
- for $i \in \{1, \ldots, n\}$, $\sim_i^V \subseteq V \times V$ and $\sim_i^A \subseteq A \times A$ are equivalence relations.

We usually require that the graph is non-terminating, that means, $\mathrm{act}(v) \neq \emptyset$ for all $v \in V$ where $\mathrm{act}(v) = \{a \in A \mid (v, a) \in \mathrm{dom}(\delta)\}$. As before, we also assume that all the players $1, \ldots, n$ always know which actions they have available, that is, if $u, v \in V$ with $u \sim_i^V v$ then $\mathrm{act}_i(u) = \mathrm{act}_i(v)$ where $\mathrm{act}_i(v) := \{a_i \in A_i \mid \text{ there is some } a \in \mathrm{act}(v) \text{ such that } \mathrm{Pr}_i(a) = a_i\}$. Moreover, any compound action that can be chosen by the players is actually available, that is, if $a_i \in \mathrm{act}_i(v)$ for $i \in [n+1]$ then $(a_0, a_1, \ldots, a_n) \in \mathrm{act}(v)$. Hence, $\mathrm{act}(v) = \mathrm{act}_0(v) \times \mathrm{act}_1(v) \times \ldots \times \mathrm{act}_n(v)$. We emphasize, however, that these assumptions are just technical simplifications which help to streamline the presentation. For providing examples it is sometimes more convenient to drop them, so we shall not be too strict about these assumptions.

Again, for game graphs with *full information*, that means, turn-based games with $\sim_i^V = \mathrm{id}_V$ and $\sim_i^A = \mathrm{id}_A$ for all $i \in [n+1]$, we omit the indistinguishability relations in the description and assume that the grand coalition consists of a single player. Consequently, we denote such a game graph as $\mathcal{G} = (V, \delta)$.

Notice that this model also provides partial information of the players about the *actions* of the game graph. This model is flexible enough, to encompass all settings of games on graphs that we shall be concerned with in this work. For the sake of a clear and simple presentation we will, however, in most concrete settings, consider only restricted versions of the model. In particular, having partial information on the actions as well is convenient for providing certain examples which mostly concerns Chapter 4 where we consider mainly two-player games. We will discuss several properties of this model as well as certain special cases after we have defined the relevant notions like plays and strategies.

Plays, Strategies and Winning Conditions. In each round of the game, being in position v, each player chooses an action $a_i \in A_i$ and the game proceeds to $\delta(v, (a_0, a_1, \dots, a_n))$. In this way, an infinite sequence of positions and actions is established which is called a play of the game: A *play* in $\mathcal{G}$ is an infinite sequence $\pi = v_0 a_1 v_1 a_2 v_2 \dots \in V(\mathrm{AV})^\omega$ such that $v_{j+1} = \delta(v_j, a_{j+1})$ for all $j \in \mathbb{N}$. We denote the set of all plays by Π. A *history* in $\mathcal{G}$ is a finite prefix $\pi \in V(\mathrm{AV})^*$ of a play in $\mathcal{G}$. Notice that a history is a word over the alphabet $V \cup \mathrm{AV}$. However, given such a history $\pi = v_0 a_1 v_1 \dots a_l v_l$, by $\mathrm{last}(\pi)$ we denote v_l instead of $a_l v_l$.

A *strategy* for player $i \in \{1, \dots, n\}$ is a function

$$\sigma_i \colon V(\mathrm{AV})^* \to A_i$$

such that for all histories $\pi = v_0 a_1 v_1 \dots a_l v_l$ and $\rho = w_0 b_1 w_1 \dots b_l w_l$ the following holds:

(S1) $\sigma_i(\pi) \in \mathrm{act}_i(v_l)$

(S2) if $v_j \sim_i^V w_j$ and $a_j \sim_i^A b_j$ for all j then $\sigma_i(\rho) = \sigma_i(\pi)$.

Condition (S2) states that a strategy for player i must yield the same value on any two histories of the same length, in which all positions and actions are indistinguishable for player i. So a strategy for player i is based on exactly the information which player i has about the positions and actions in a play, if we assume *synchronous observability* of events: In each round of the game, each player observes a certain part of the current event that is visible to him which, in the model used here, is exactly the *equivalence class* of this event. In particular, each round of the game is noticed by each player i, even if the game proceeds from a position u to a position v such that $u \sim_i^V v$, which constitutes a situation where player i does not observe any part of the change to the current position in the game graph. So in a sense the game comes with a shared clock to which all players have access.

To make the information that player i has in this model more explicit, we also denote such a strategy as a function

$$\sigma_i \colon [V]_i([V]_i[A]_i)^* \to A_i$$

taking sequences of equivalence classes of positions and actions as inputs. For a history $\pi = v_0 a_1 v_1 \dots a_l v_l$, we call $[v_0]_i[a_1]_i[v_1]_i \dots [a_l]_i[v_l]_i$ also the *local view* of player i on π or the *local history* of player i. It is obvious how to get one from one representation of a strategy to the other.[3] However, for the representation $\sigma_i \colon [V]_i([V]_i[A]_i)^* \to A_i$, condition (S2) is redundant!

A history $\pi = v_0 a_1 v_1 \dots a_l v_l$ is *consistent* with a strategy σ_i for player i if we have $\mathrm{Pr}_{A_i}(a_j) = \sigma(\pi(<j))$ for $j = 1, \dots, l$. A play is consistent with σ_i if all its histories are. A *joint strategy* or simply a strategy for the grand coalition is a tuple $\sigma = (\sigma_1, \dots, \sigma_n)$ where each σ_i is a strategy for player i. A history is consistent with σ if it is consistent with each individual strategy σ_i.

A *winning condition* is a set $W \subseteq \Pi$ of plays in $\mathcal{G}$. A joint strategy σ for the grand coalition is winning for W from a position $v_0 \in V$ if each play $\pi = v_0 a_1 v_1 \dots \in \Pi$ which is consistent with σ is won by the grand coalition, that is, belongs to W. The set Win, called the *winning region* of the grand coalition in $\mathcal{G}$, consists of all vertices $v \in V$ such that the grand coalition has a winning strategy from v for W.

[3]Notice, however, that when it comes to actually *implementing* strategies, the correspondence between the two representations is not so straightforward anymore.

Initial Positions and Extensive Games. It will be comfortable for certain constructions and proofs, to consider only winning strategies from a fixed initial position v_0. To see that this is sufficient, also in the presence of partial information, we argue that we can always assume that all players of the grand coalition *know* the initial position v_0. First notice that for a strategy σ the question whether σ is winning from a set $U \subseteq V$ of initial positions is independent of the value of σ on histories which start in some position from $V \setminus U$. So it suffices to require condition (S2) only for histories which start from some position in U and hence we can assume that any player of the grand coalition knows that the initial position is in U.[4] Now we can add some new position $v_0 \notin V$ to the game graph such that $[v_0]_{\sim_i^V}$ is a singleton for all $i \in \{1, \dots, n\}$ and from which the environment can choose secretly any position from U. That means, player 0 has actions a_u, $u \in U$ available at v_0 with $a_u \sim_i^A a_{u'}$ for all $u, u' \in U$, and the players of the grand coalition have only the trivial action $\perp$ available at v_0. Then the grand coalition has a joint strategy that is winning from all positions in U if, and only if, the grand coalition has a joint strategy that is winning from v_0.

Now, a game graph with partial information together with some winning condition W defines an extensive game with *imperfect information* $\mathcal{T} = \mathcal{T}_{\mathcal{G},W}$. The game tree T of $\mathcal{T}$ simply consists of all histories in $\mathcal{G}$ with action labels defined in the obvious way. As we have just seen, we can restrict our attention to histories and plays from a fixed initial position which is known to all players, so we consider games $\mathcal{T} = \mathcal{T}_{\mathcal{G},W,v_0}$ for a given $v_0 \in V$. The game tree of $\mathcal{T}$ then consists of all histories which start in v_0 and has as root the node v_0. Each infinite path in $\mathcal{T}$ corresponds uniquely to a play in $\mathcal{G}$ so we shall identify plays and histories in $\mathcal{T}$ with those in $\mathcal{G}$.

Imperfect information of player i in $\mathcal{T}$ is given by the equivalence relation $\sim_i^* \subseteq T \times T$ which is defined according to the synchronous observability described above: For two histories $\pi = v_0 a_1 v_1 \dots a_l v_l$ and $\rho = w_0 b_1 w_1 \dots b_l w_l$,

$$\pi \sim_i^* \rho, \text{ if and only if } v_j \sim_i^V w_j \text{ and } a_j \sim_i^A b_j \text{ for all } j$$

Although the game tree of $\mathcal{T}$ is infinite, we call $\mathcal{T}$ a game on a finite graph since the possible plays as well as the imperfect information in this game are determined by the game graph $\mathcal{G}$.

Notice that $\mathcal{T}$ is *not* a game of *perfect recall* in the classical sense (cf. [124, 160]) as long as we don't require that a player can distinguish any two of his actions. In $\mathcal{G}$ this amounts to the claim that for any $i = 1, \dots, n$ and all $a, b \in A$ such that $\mathrm{Pr}_i(a) \neq \mathrm{Pr}_i(b)$ we have $a \not\sim_i^A b$. We do not make this assumption generally (in fact we will consider settings where this does *not* hold) but emphasize that in our *non-stochastic* setting this is merely a technical issue: We can always make all the actions of a player distinguishable for him without affecting the possible strategies of this player. The reason is that even though a player may not remember his own past actions, he has to consider only plays that are consistent with the strategy that he follows: If player i is in a situation (after some σ_i-history π has been played) where he does not know whether in some previous situation (after $\rho \sqsubset \pi$) he has chosen action a or action b ($\rho a v \sqsubseteq \pi$ and $\rho' b w \sqsubseteq \pi'$ for some $\pi' \sim_i^* \pi$ or vice versa) then he can look up his strategy σ_i and if it prescribes to choose a in this previous situation ($\rho a v \sqsubseteq \pi$) then player i can dismiss the possibility that he might have chosen b (because $\sigma_i(\rho) = \sigma_i(\rho')$ and so π' is not consistent with σ_i).

Moreover, notice that a player always remembers anything that he actually has observed previously which also is an interesting notion of perfect recall. (Formally, if π, ρ are histories in $\mathcal{T}$ with $\pi \not\sim_i^* \rho$ and π' and ρ' are any extensions of π and ρ, respectively, then $\pi' \not\sim_i^* \rho'$.)

The Strategy Problem. Having these notions at hand, we are now ready to state the *strategy problem for game graphs with partial information*:

Given a game graph with partial information $\mathcal{G}$, a finite presentation of a winning condition W and some initial position v_0, does the grand coalition have a joint winning strategy for $(\mathcal{G}, W, v_0)$?

[4]Notice, however, that this does *not* mean that we make any assumptions on the equivalence relations $\sim_i^V$.

Of course, a winning strategy for the grand coalition for $(\mathcal{G}, W, v_0)$ is the same as a winning strategy for the grand coalition for $\mathcal{T} = \mathcal{T}_{\mathcal{G},W,v_0}$. So this problem is in fact a special case of the strategy problem for games with imperfect information, the *strategy problem for games with imperfect information on finite graphs.*

As we have shown, under the assumption that player 0 always knows the number of his own past moves, this problem is equivalent to the problem of finding a strategy for the grand coalition which is winning against all strategies of player 0. Clearly, $\sim_0^*$ fulfills this assumption so in this setting, we only consider the above formulation of the strategy problem.

Two-Player Games. We now present and discuss a restricted two-player version of game graphs with partial information that we will use in Section 3.5 and Chapter 4. Two-player games are played by player 0 (environment) against player 1 (controller), so the model of game graphs with partial information reduces to the form

$$\mathcal{G} = (V, \delta, \sim^V, \sim^A)$$

where $\sim^V = \sim_1^V$ and $\sim^A = \sim_1^A$. The indistinguishability relation $\sim_1^*$ of player 1 on histories is denoted $\sim^*$ accordingly. Moreover, for two-player games we usually assume that they are *turn-based*, that is, for each position one of the players has only a default action $\perp$ available. We denote the set of positions from which player 0 has only the action $\perp$ available by V_1 and $V_0 := V \setminus V_1$. For convenience, we denote actions $(a, \perp)$ and $(\perp, a)$ simply by a and we define $A_i = \bigcup\{\mathrm{act}(v) \mid v \in V_i\}$, that is, A_i is the set of actions which are available at some position of player i. (We do not require A_0 and A_1 to be disjoint.) Of course, in order to be able to play the game properly, player 1 needs to know when it is his turn, so we assume that whenever $v \sim^V w$ we have $v, w \in V_1$ or $v, w \notin V_1$. Furthermore, in this setting we usually do assume that player 1 can distinguish any two of his own actions, that is, for all $a, b \in A_1$ with $a \neq b$ we have $a \not\sim^A b$. It is useful to observe that under this assumption, for any strategy σ of player 1 and any two histories π and ρ in $\mathcal{G}$ with $\pi \sim^* \rho$ we have that π is consistent with σ if, and only if, ρ is consistent with σ. Notice that this restricted version of game graphs also encompasses the full information case for which we have assumed that only two players are present and the game is turn-based.

Nondeterministic game graphs. In our investigation it will sometimes be convenient to consider *nondeterministic* game graphs, that is, game graphs $\mathcal{G} = (V, \delta, (\sim_i^V)_i, (\sim_i^A)_i)$ where the move function has the form $\delta : \mathrm{dom}(\delta) \subseteq V \times A \to 2^V$. We describe such a nondeterministic move function also as a move relation $\Delta \subseteq V \times A \times V$. All notions like *plays* and *strategies* are defined just as before. Clearly, nondeterministic games are not determined in general, even under full information and for simple winning conditions like reachability.

However, the strategy problem asks for a joint winning strategy for the grand coalition and a strategy $\sigma = (\sigma_1, \dots, \sigma_n)$ for the grand coalition is a *winning strategy* if *each* play that is consistent with σ is won by grand coalition Hence, in the context of the *strategy problem*, we can assume that player 0 has control over the nondeterministic choices in the game which can also easily be implemented into the game graph.

That is, given a game graph $\mathcal{G}$ and a winning condition W, we can construct a deterministic game graph $\mathcal{G}^d$ with $V^d \supseteq V$ and a winning condition W^d such that for each $v \in V$, the grand coalition has a winning strategy for $(\mathcal{G}, W, v)$ if, and only if, the grand coalition has a winning strategy for $(\mathcal{G}^d, W^d, v)$: We add *intermediate* positions to the game such that from any position v, with action $a \in \mathrm{act}(v)$, the game proceeds to position (v, a). Then from such a position, player 0 chooses some next position $v' \in \delta(v, a)$ by choosing the action v'. Each player of the grand coalition has just the action $\perp$ available in an intermediate position (v, a). Moreover, we let all new positions as well as all new actions that we add to the game graph be indistinguishable for each player of the grand coalition. Formally:

$$\mathcal{G}^d := (V \cup V \times A, \delta^d, (\sim_i^{V,d})_i, (\sim_i^{A,d})_i)$$

- $\delta^d(v,a) = (v,a)$ for all $v \in V$ and all $a \in A$
- $\mathrm{act}_0(v,a) = \{v' \in V \mid (v,a,v') \in \Delta\}$ and $\mathrm{act}_i(v,a) = \{\perp\}$ for $i = 1,\dots,n$
- $\delta^d((v,a),(v',\perp,\dots,\perp)) = v'$ for $(v,a,v') \in \Delta$.
- $v \sim_i^{V,d} w :\Longleftrightarrow v \sim_i^V w$ and $a \sim_i^{A,d} b :\Longleftrightarrow a \sim_i^A b$
- $(v,\perp,\dots,\perp) \sim_i^{A,d} (w,\perp,\dots,\perp)$ for all $v,w \in V$ and $(v,a) \sim_i^{V,d} (w,b)$

Moreover, a play π^d of $\mathcal{G}^d$ is in W' if and only if the play π in $\mathcal{G}$, which is obtained from π^d by deleting all positions and actions, which do not belong to V and A, respectively, is in W.

We will use this observation for the constructions that we present in Section 3.5 and in Chapter 6 where nondeterministic game graphs provide a convenient intermediate step in the construction. There, however, we will apply the determinization construction only to game graphs which have already *full* information (and, consequently, have only two players). Now observe that the construction of $\mathcal{G}^d$ as described above yields a game graph with partial information, even if $\mathcal{G}$ is a game graph with full information. Hence, in case we start with a game graph $\mathcal{G}$ with full information we use a simplified version of the construction where any two new positions (v,a) and (w,b) as well as any two new actions $(v,\perp,\dots,\perp),(w,\perp,\dots,\perp)$ are distinguishable for player 1. Since player 1 has full information in $\mathcal{G}$, this simplification is obviously sound and the result $\mathcal{G}^d$ is then also a game graph with full information.

Position Based Strategies. Infinite games on (finite) graphs with full information have a long tradition in the theory of automata theory and synthesis of nonterminating reactive system, see also Section 2.2. In this context, such a game is often given by a graph of the form $\mathcal{G} = (V,E)$ where $E \subseteq V \times V$, that is, the edges are not labeled. A strategy for player i picks a next position for each history where it is player i's turn, that means, it is a function $\sigma_i : V^*V_i \to V$ such that $(v, \sigma(\pi v)) \in E$ for each $\pi v \in V^*V_i$. Recall that V_i denotes the set of positions where it is player i's turn. However, in the context of *partial information* we have to impose consistency requirements on strategies which guarantee that a strategy for a player does not use information which is not available to that player. A coherent and transparent way to do so is to introduce *actions* to the model and let strategies for a player choose an action instead of a direct successor of the current position. As we shall see, knowledge tracking constructions may require the potential to let winning conditions depend on the actions as well, even if the winning condition of the original game depends only on the positions of the game graphs.

Nevertheless, we are particularly interested in winning conditions and strategies which are only based on the sequence of positions in plays and histories, respectively. In particular, to be able to use results on infinite games on graphs of the form $\mathcal{G} = (V,E)$ with full information in a black box fashion, we have to make sure that for game graphs $\mathcal{G} = (V,\delta)$ with full information, we can always migrate to the position based setting.

Let $\mathcal{G} = (V,\delta,\sim^V,\sim^A)$ be a game graph with partial information and let W be a winning condition for $\mathcal{G}$. We consider only the two-player case here. W is called *position based* if for all plays $\pi = v_0a_1v_1\dots$ and $\rho = w_0b_1w_1\dots$ in $\mathcal{G}$ with $v_j = w_j$ for all i we have $\pi \in W$ if, and only if $\rho \in W$. A strategy $\sigma\colon V(\mathrm{AV})^* \to A_1$ for player 1 is called *position based* if for all histories $\pi = v_0a_1v_1\dots a_lv_l$ and $\rho = w_0b_1w_1\dots b_lw_l$ in with $v_j = w_j$ for all j we have $\sigma(\pi) = \sigma(\rho)$.

One obvious possibility to get to a purely position based setting is to construct a new game graph $\mathcal{G}^{\times A}$ where the set of positions is $V \times A$ and $(v,a) \sim^V (w,b)$ if $v \sim^V w$ and $a \sim^A b$. That is, the action that was chosen in the last step is always encoded in the current position and hence, winning conditions and strategies can clearly be expressed in terms of the positions of the game graphs. However, this requires a transformation of the game graph and involves a blow-up of the set of positions, so it would be desirable to figure out certain scenarios where we can get to a purely position based setting easier.

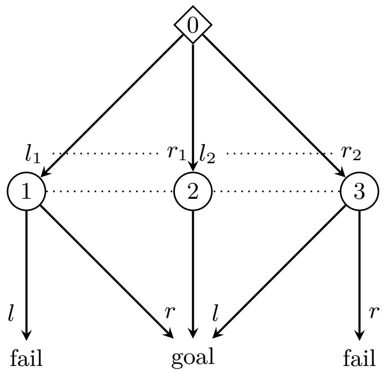


Figure 2.1: Observation based winning conditions do not imply position based winning strategies.

To this end, we first observe that a strategy σ for player 1 never needs to depend on his own actions, even if the winning condition does. The reason is the same as for the assumption that player 1 does not need to observe his own actions: Player 1 needs to concern himself only with histories which are consistent with the strategy σ with which he complies. So whenever player 1 encounters a sequence of his own past actions that is different from the one that he gets by looking up σ, he can behave haphazardly. To make this more precise, consider some σ-history $\pi = v_0 a_1 v_1 \dots a_l v_l$ with $v_l \in V_1$ and consider another history $\rho = w_0 b_1 w_1 \dots b_l w_l$ such that $w_j \sim^V v_j$ for all j and moreover $b_j \sim^A b_j$ for all j with $v_j \in V_0$. Then either $b_j = a_j$ for all j with $v_j \in V_1$ or ρ is not consistent with σ. Clearly, if ρ is not consistent with σ then we can redefine σ on ρ arbitrarily, so in particular we can set $\sigma(\rho) = \sigma(\pi)$. The strategy σ which is obtained in this way depends only on the sequence of positions and the sequence of actions chosen by player 0 in a history.

On the other hand, a strategy σ for player 1 needs to depend on the actions of player 0 in general. This is quite obvious, if the winning condition depends on these actions. But under partial information, this is also the case if the winning condition is position based since the observations about the action may yield information about the events in the game which is not contained in the positions. In Figure 2.1 we give an example of a (turn based) two-player game with a reachability winning condition where player 1 has a winning strategy but he does not have one which is position based. Notice that this requirement is weaker than stipulating that player 1 has a winning strategy which is only based on the information about the positions which he observes. Also notice that the winning condition of the game depends only on the observations about the positions that player 1 makes.

In the picture, diamond positions belong to player 0 while round positions belong to player 1. Dotted lines indicate the indistinguishabilities of player 1 and the goal for player 1 is to reach the position *goal*.

However, in games with full information, position based strategies suffice to win for position based winning conditions. In fact, from a game graph $\mathcal{G} = (V, \delta)$ and a position based winning condition $W \subseteq V(AV)^\omega$ we can easily construct a game graph $\mathcal{G}^p = (V, E)$ and a winning condition $W^p \subseteq V^\omega$ such that winning strategies for player 1 (and in fact also for player 0) are preserved: we just delete the actions from the edges and from the sequences in W and let the players choose successor positions instead of actions. Although the correctness of the construction is intuitively quite evident, we like to give a detailed proof. The reason is that this proof gives an easy example of a kind of reasoning that we shall use quite often when we transfer winning strategies between games.

Proposition 2.2. *Player* 1 *has a winning strategy for* $(\mathcal{G}, W, v_0)$ *if and only if he has a winning strategy for* $(\mathcal{G}^p, W^p, v_0)$.

Proof. First, if player i has a winning strategy for $(\mathcal{G}^p, W^p, v_0)$ then he obviously has a winning strategy for $(\mathcal{G}, W, v_0)$. Now let σ be a winning strategy for player 1 for $(\mathcal{G}, W, v_0)$. By induction on the length of sequences $\pi \in V^*$ we define a strategy $\sigma^p : V^*V_1 \to V$ for player 1 for $(\mathcal{G}^p, W^p, v_0)$ and at the same time, with each history $\pi^p = v_0v_1 \dots v_l \in V^*$ in $\mathcal{G}^p$ (with $l \geq 1$) that is consistent with σ^p we associate a sequence $\zeta(\sigma^p) = a_1 \dots a_l$ of actions such that the following conditions hold.

(1) $\pi = v_0a_1v_1 \dots a_lv_l$ is a σ-history in $\mathcal{G}$

(2) if $\rho^p \sqsubseteq \pi^p$ then $\zeta(\rho^p) \sqsubseteq \zeta(\pi^p)$.

First, consider $\pi^p = v_0$. If $v_0 \in V_1$ then we define $\sigma^p(\pi^p) := v$ for the uniquely determined $v \in V$ with $\delta(v_0, \sigma(v_0)) = v$ and with $\pi^p v$ we associate the sequence $\zeta(\pi^p v) = \sigma(v_0)$. If $v_0 \notin V_1$ then with any history $\pi^p v_1 \in V^*$ in $\mathcal{G}^p$ we associate the sequence $\zeta(\pi^p v_1) = a_1$ for some $a_1 \in A$ such that $\delta(v_0, a_1) = v_1$. By construction, conditions (1) and (2) are fulfilled.

Now consider any σ^p-history $\pi^p = v_0v_1 \dots v_l \in V^*$ with $l \geq 1$ in $\mathcal{G}^p$ and assume that $\zeta(\pi^p) = a_1, \dots, a_l$ has been constructed. Let $\pi = v_0a_1v_1 \dots a_lv_l$. If $v_l \in V_1$ then we define $\sigma^p(\pi^p) := v$ for the uniquely determined $v \in V$ with $\delta(v_l, \sigma(\pi)) = v$ and with πv we associate the sequence $\zeta(\pi v) = a_1, \dots, a_l, \sigma(\pi)$. If $v_1 \notin V_1$ then with any history $\pi^p v_{l+1} \in V^*$ in $\mathcal{G}^p$ we associate the sequence $\zeta(\pi^p v_{l+1}) = a_1, \dots, a_l, a_{l+1}$ for some $a_{l+1} \in A$ such that $\delta(v_l, a_{l+1}) = v_{l+1}$.

Now we have to show that σ^p is a winning strategy for $(\mathcal{G}^p, W^p, v_0)$, so consider any σ^p-play $\pi^p = v_0v_1v_2 \dots$ in $\mathcal{G}^p$. By condition (2) we obtain a sequence $\zeta(\pi^p) = a_1a_2 \dots$ such that $\pi = v_0a_1v_1a_2v_2 \dots$ is a σ-history in $\mathcal{G}$ and since σ is a winning strategy for $(\mathcal{G}, W, v_0)$, $\pi \in W$. But since W is position based, the construction of W^p yields $\pi^p \in W^p$. $\square$

Of course in a symmetric setting where we are also interested in strategies for player 0 (like for example, determinacy results for games with full information), all these notions and observations apply to player 0 completely analogously. Notice that this result also holds for infinite game graphs.

2.1.3 Distributed Systems

A *distributed system with* $n + 1$ *processes* has the form

$$\mathcal{D} = (\mathfrak{A}, (\Sigma_c)_{c \in C}), \text{ with } \mathfrak{A} = (C, r, w)$$

- C is a finite set (of channels)
- $r \colon C \to [n+1]$ assigns to each channel a unique process which *reads* it
- $w \colon C \to [n+1]$ assigns to each channel a unique process which *writes* to it
- $(\Sigma_c)_{c \in C}$ is a collection of finite alphabets, one for each channel.

The structure $\mathfrak{A}$ is also called the *architecture* or *communication graph* of the system and the alphabets Σ_c assign finite sets of *signals* that can be sent along the channels $c \in C$. We denote the set $\{0, 1, \dots, n\}$ of processes also as $P = \{p_0, p_1, \dots, p_n\}$. As before, $p_0 = p_{\text{env}}$ represents the environment of the system and the processes $P_{\text{con}} = \{p_1, \dots, p_n\}$ form the grand coalition of players which represent the controllers.[5]

Hence, $\mathfrak{A}$ is a directed graph with vertices $p_1, \dots, p_n$ and edges $c \in C$ where each c is an edge from vertex $w(c)$ to vertex $r(c)$. If such an edge c exists we say that $w(c)$ *sends information to* $r(c)$. If $w(c) = p \in P_{\text{con}}$ then we assume that $w(c) \neq p$, that means, no controller can send information to itself. (So in the communication graph we do not have selfloops except, possibly, on the environment p_{env}.) Notice that $\mathfrak{A}$ has *multiedges*, that is, a process may send information to another process via multiple channels. However, such multiple channels can always be simulated by a single channel, so whenever this is more convenient, we assume w.l.o.g. that there is at most one such channel. For $p \in P$ we define

[5]As is common, in the context of distributed systems we shall usually stick to the term *processes* for the players and *controllers* for the members of the grand coalition.

- $O_p := \{c \in C \mid w(c) = p\}$
- $I_p := \{c \in C \mid r(c) = p\}$

that is, O_p and I_p are the sets of *input* and *output* channels of process p respectively. Moreover,

- $H_{p_\text{env}} = O_{p_\text{env}} \cap I_{p_\text{env}}$
- $R_{p_\text{env}} = O_{p_\text{env}} \setminus H_{p_\text{env}}$

Channels $c \in O_{p_\text{env}}$ of the environment p_env are also called *external input* channels, $H_{p_\text{env}} \subseteq O_{p_\text{env}}$ are called *hidden input* channels, that is, they cannot be read by any of the controllers, and $R_{p_\text{env}} \subseteq O_{p_\text{env}}$ are called readable input channels. The channels $c \in C$ with $w(c) \in P_\text{con}$ and $r(c) \in P_\text{con}$ are *internal communication* channels and for $p \in P_\text{con}$, the channels $c \in O_p$ with $r(c) = p_\text{env}$ are the *external output* channels of controller p.

As before, the information of the environment about the events in the system is irrelevant, that is, we can assume that process p_env has *full information.* So the channels from processes $p \in P_\text{con}$ to p_env are actually futile, that is, we could equivalently let those channels be selfloops or channels which do not have a particular destination. However, having channels that actually transmit information to the environment resembles physical systems more closely and is also convenient for certain definitions in the context of distributed systems.

Finally, we assume $O_p \neq \emptyset$ for all $p \in P_\text{con}$, that is, each controller has at least one output channel. Notice that with this assumption, the set of processes is uniquely determined by the architecture $\mathfrak{A}$. Although this assumption seems quite natural (an idle controller seems rather redundant), it still is a restriction in that controllers without output channels might increase the expressiveness of *locally decomposable* specification, without making the architectures undecidable. This concept will be introduced in Section 2.2 and in Chapter 5 we will also discuss the issue of processes without output channels in the context of such specifications.

Runs, Strategies and Specifications. For a process $p \in P$, the *output* alphabet of p is

$$\Sigma^p_\text{out} := \prod_{c \in O_p} \Sigma_c$$

and for p_env we also define $\Sigma_\text{hid} := \prod_{c \in H_{p_\text{env}}} \Sigma_c$ and $\Sigma_\text{read} := \prod_{c \in R_{p_\text{env}}} \Sigma_c$ as the hidden and the readable output alphabet, respectively. $\Sigma_\text{out} := \prod_{p \in P_\text{con}} \Sigma^p_\text{out}$ is the joint output alphabet of the controllers. For a controller, $p \in P_\text{con}$, the *input* alphabet of p is

$$\Sigma^p_\text{in} := \prod_{c \in I_p} \Sigma_c$$

and if $I_p = \emptyset$ then we define $\Sigma^p_\text{in} := \{|\}$ instead of the empty set.[6] The reason is that the information of a process is given in terms of *synchronous observability*, so in each step, any process receives some input signal and if $I_p = \emptyset$ this signal is just a *tick* $|$ of the shared clock telling the process that a step of the system has been performed. The (local) alphabet of a controller $p \in P_\text{con}$ is $\Sigma^p = \Sigma^p_\text{in} \times \Sigma^p_\text{out}$ and the (global) *system alphabet* is

$$\Sigma^\mathcal{D} = \prod_{c \in C} \Sigma_c = \prod_{i=0}^{n} \Sigma^{p_i}_\text{out}.$$

At each (discrete) point in time $j \in \mathbb{N}$, each process p sends a signal $\alpha_c(j) \in \Sigma_c$ to each of his output channels $c \in O_p$. Together, these signals define the *action* $\alpha_p(j) \in \Sigma^p_\text{out}$ of process p at step j. In this way, an infinite sequence $\alpha \in (\Sigma^\mathcal{D})^\omega$ is built, called a *system run* of $\mathcal{D}$. A *history* of $\mathcal{D}$ is a finite prefix $w \in (\Sigma^\mathcal{D})^*$ of a run. Such a run is composed of the sequences of actions of the individual processes: $\alpha = \beta_{p_0}{}^\frown\beta_{p_1}{}^\frown \ldots {}^\frown \beta_{p_n}$ where $\beta_{p_i} \in (\Sigma^{p_i}_\text{out})^\omega$ for $i = 0, 1, \ldots, n$. The

[6]Notice that the input alphabet of p_env is not relevant

local run of a controller $p \in P_{\text{con}}$ is $\alpha_p{}^\frown\beta_p$, where the *local input history* $\alpha_p = \Pr_{\Sigma^p_{\text{in}}}(\alpha) \in (\Sigma^p_{\text{in}})^\omega$ of p contains exactly the *information* about the history of events that this controller *observes.*

A *strategy* for a controller $p \in P_{\text{con}}$ takes the local input history of controller p and yields a next action of p, that means, it is a function

$$\sigma_p \colon (\Sigma^p_{\text{in}})^* \to \Sigma^p_{\text{out}}.$$

Notice that we could represent a strategy for p also as a function $\sigma_p \colon (\Sigma^{\mathcal{D}})^* \to \Sigma_{\text{out}}$ which fulfills the consistency condition

(S2) $\sigma(u) = \sigma(u')$ for all $u, u' \in (\Sigma^{\mathcal{D}})^*$ with $\Pr_{\Sigma^p_{\text{in}}}(u) = \Pr_{\Sigma^p_{\text{in}}}(u')$

(Condition (S1) is redundant for distributed systems.) We will sometimes make use of this representation but usually prefer to have a strategy for process p read only the signals which are actual inputs to process p.

A local history $u_p{}^\frown v_p$ of process p is *consistent* with σ if $v_p(i) = \sigma(u_p(< i))$ for all $i = 0, \ldots, |u_p| - 1$. A local run $\alpha_p{}^\frown\beta_p$ of p is consistent with σ if all its histories are. A joint strategy for the grand coalition is a collection $\sigma = (\sigma_{p_1}, \ldots, \sigma_{p_n})$ of individual strategies for the controllers. A system run $\alpha \in (\Sigma^{\mathcal{D}})^\omega$ is consistent with σ if the local run $\Pr_{\Sigma^p}(\alpha)$ of any $p \in P_{\text{con}}$ is consistent with σ_p.

A *system specification* for $\mathcal{D}$ is a language $L \subseteq (\Sigma^{\mathcal{D}})^\omega$ consisting of those system runs which the controllers $p_1, \ldots, p_n$ want to ensure.

For a language $L^p_{\text{in}} \subseteq (\Sigma^p_{\text{in}})^\omega$, a strategy σ_p for a controller $p \in P_{\text{con}}$ is *locally winning* on L^p_{in} if each local run $\alpha_p{}^\frown\beta_p \in (\Sigma^p)^\omega$ of process p with $\alpha_p \in L^p_{\text{in}}$ which is consistent with σ_p is in $L_p := \Pr_{\Sigma^p}(L)$. (L_p is also called the *local specification* of p.) This means that, for each input sequence of the given input language L_{in}, the corresponding local run of process p determined by σ_p must occur in some system run which is in L. The strategy is called locally winning if it is locally winning on $L^p_{\text{in}} = (\Sigma^p_{\text{in}})^\omega$. A joint strategy σ for the grand coalition is *winning* if any system run which is consistent with σ is in L.

Notice that a collection $(\sigma_{p_1}, \ldots, \sigma_{p_n})$ of local winning strategies is *not* necessarily a joint winning strategy for the grand coalition which is obvious if there are hidden channels from the environment since they are not accounted for in any of the local winning conditions L_p. But even in the absence of hidden channels from the environment, although each σ_p guarantees that process p creates only local runs which are correct from his local view on L, not all arbitrary compositions of these local behaviors of the individual processes are necessarily in L. Moreover, if conversely, $\sigma = (\sigma_{p_1}, \ldots, \sigma_{p_n})$ is a joint winning strategy for the grand coalition then an individual strategy σ_p is *not necessarily locally winning.* That is, it is not locally winning on *all* possible inputs for process p but merely on those inputs that it receives from other processes and the outputs of the other controllers are constrained by their own local strategies.

It is now plain to obtain a game $\mathcal{T} = \mathcal{T}_{\mathcal{D},L}$ with imperfect information from a distributed system $\mathcal{D}$ and a specification L: The game tree of $\mathcal{T}$ consists of all finite histories $w \in (\Sigma^p)^*$ of runs of $\mathcal{D}$ and the actions of the players in the game tree are their actions in $\mathcal{D}$. Of course, *imperfect information* again is given in terms of *synchronous observability*, that is, here we also use the indistinguishability relation $\sim_i^*$ that we have defined in Section 2.1.2: for $w, w' \in (\Sigma^p)^*$ we have $w \sim_i w'$ if $\Pr_{\Sigma^{p_i}_{\text{in}}}(w) = \Pr_{\Sigma^{p_i}_{\text{in}}}(w')$, that means, if the local input histories of process p_i in w and in w' are the same.

Remark. Aside from each controller having at least one output channel, another restriction of our setting is the absence of *broadcast channels*, that is, channels for which there is a single process that writes to them but which can be read by several processes. In principle, such channels can be simulated in our setting by requiring in the *specifications* for certain channels that the same signals is written to them in each step. Technically, however, this is a substantial difference. First, this involves reachability conditions in the case of broadcast channels from the environment but safety conditions in the case of broadcast channels from the controllers, so neither safety conditions nor reachability conditions are necessarily preserved by this construction. Moreover, in the case of broadcast channel from the environment, the construction does *not* preserve *locally decomposability* of the specification in general, cf. Section 2.2.3 and Chapter 5. Finally, we are usually interested in characterizations of decidability of architectures just by means of their graph structure without incorporating specifications. In particular, the characterization in Chapter 5 is not valid in the presence of broadcast channels.[7] We do not pursue this further.

The Controller Problem. As we have mentioned, in the context of distributed systems, we speak of the strategy problem also as the *controller problem*:

Given a distributed system $\mathcal{D}$ and a specification L, does the grand coalition P_{con} of controllers have a joint winning strategy?

For a fixed architecture $\mathfrak{A}$, the *controller problem for* $\mathfrak{A}$ asks, given a specification L, whether the grand coalition P_{con} of controllers has a joint winning strategy. We say that $\mathfrak{A}$ is *decidable for a class $\mathcal{L}$ of specifications* if the controller problem for $\mathfrak{A}$ is decidable for all $L \in \mathcal{L}$. Notice that an architecture does not specify a labeling. However, we shall be a bit sloppy about the terms *distributed systems* and *architectures*. In particular, when considering an architecture $\mathfrak{A}$ and some *specification*, we implicitly assume a labeling of the architecture.

Subarchitectures, Informdness and Pipelines. In the following, let $\mathfrak{A} = (C, r, w)$ be an architecture with $n + 1$ processes. $\mathfrak{A}$ is called *connected* if any two controllers are connected via some (not necessarily directed) path that does not go through p_{env}, that is, contains only vertices from P_{con}. In other words, the set P_{con} induces a connected subgraph of $\mathfrak{A}$.

A *subarchitecture* is a subgraph $\mathfrak{A}'$ of $\mathfrak{A}$ which contains p_{env} such that $\mathfrak{A}'$ is an architecture, that means, each controller has at least one output channel. For a set $Q \subseteq P_{\text{con}}$, if the subgraph $\mathfrak{A}'$ of $\mathfrak{A}$ induced by the vertices $Q \cup \{p_{\text{env}}\}$ is an architecture then we denote $\mathfrak{A}' =: \mathfrak{A}(Q)$ and we call $\mathfrak{A}(Q)$ the subarchitecture of $\mathfrak{A}$ induced by Q. A *connected* component of $\mathfrak{A}$ is a maximal connected subarchitecture $\mathfrak{A}'$ of $\mathfrak{A}$, that is, $\mathfrak{A}'$ is connected and there is no connected subarchitecture of $\mathfrak{A}$ that strictly contains $\mathfrak{A}'$.

A process $p \in P_{\text{con}}$ is called *reachable* if there is a directed path from p_{env} to p. Process p is *better informed* than $p' \in P_{\text{con}} \setminus \{p\}$ if p is reachable and each directed path from p_{env} to p' goes through p. If p is better informed than p' we write $p < p'$.[8] Processes p and p' are called *equally informed* if each directed path from p_{env} to p' goes through p and vice versa. Notice that in our setting without broadcast channels, two processes can only be equally informed if they coincide or if both of them are not reachable. If $p < p'$ or p and p' are equally informed, then we write $p \leq p'$. (So $\leq$ is not necessarily antisymmetric.) If $p \not\leq p'$ and $p' \not\leq p$, we also say that p and p' are *incomparably informed.* The set $\{p, p'\}$ is called *information fork*, cf. [80].

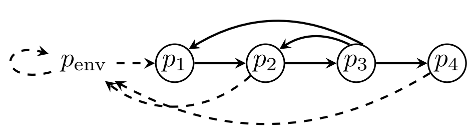
$\mathfrak{A}$ is called *pipeline* with backward channels if

- $\{p_1\} \subseteq r(O_{p_0}) \subseteq \{p_0, p_1\}$ and
- $\{p_{i+1}\} \subseteq r(O_{p_i}) \subseteq \{p_j \mid 0 \leq j \leq i+1\}$ for all $i \in \{1, \ldots, n-1\}$.

That is, the environment sends information only to p_1 and each process p_i sends information to p_{i+1} and possibly to other processes p_j but only if $j < i$, see Figure 2.2. A channel $c \in O_{p_i}$ with $r(c) = p_j$ is called *backward channel* if $0 < j < i$ and it is called forward channel if $j = i + 1$. Moreover, $\mathfrak{A}$ is called a *two-flanked pipeline with backward-channels* if $\{p_1, p_n\} \subseteq r(O_{p_0}) \subseteq \{p_0, p_1, p_n\}$ and $\{p_{i+1}\} \subseteq r(O_{p_i}) \subseteq \{p_j \mid 0 \leq j \leq i+1\}$ for all $i \in \{1, \ldots, n-1\}$, see Figure 2.2. We call $\mathfrak{A}$ a (two-flanked) pipeline, if $\mathfrak{A}$ has no backward channels.

[7]Notice that, while Chapter 5 we use the specification to actually *eliminate* feedback channels, this construction does not eliminate the broadcasting requirement from the architecture.

[8]This notation may seem counterintuitive, but notice that in terms of equivalence relations, $p < p'$ implies $\sim_p^* \subseteq \sim_{p'}^*$. Moreover, in terms on the communication graph, on each path starting in p_{env} which contains p and p', the first occurrence of p is *before* the first occurrence of p'.



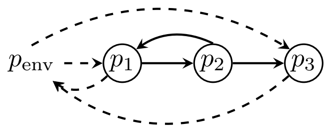


Figure 2.2: Pipeline and two-flanked pipeline with backward-channels

So the only difference between a pipeline (with backward channels) and a two-flanked pipeline (with backward channels) is the additional channel from the environment to process p_n. In particular, while in pipelines (with backward channels) the set of controllers can be ordered linearly according to informdness ($p_1 \leq p_2 \leq \ldots \leq p_n$), in two-flanked pipelines (with backward channels) process p_n and any process p_i with $i \in \{1, \ldots, n-1\}$ form an information fork.

Feedback Channels and Joint Strategies. A strategy for a controller p is a function $\sigma : (\Sigma^p_{\mathrm{in}})^* \to \Sigma^p_{\mathrm{out}}$, so it reads the sequence of all the inputs that p receives and yields an output of p. However, intuitively it is quite clear that if we consider a joint strategy $\sigma = (\sigma_1, \ldots, \sigma_n)$ for the grand coalition, then a strategy σ_i for controller p_i does not need to actually read the inputs that p_i receives from processes p_j that are less informed than p_i ($p_i \leq p_j$): Any paths from p_{env} to any such controller p_j leads through p_i and so, given this fixed joint strategy, p_i can *deduce* the outputs of all the processes $p_j \geq p_i$ from the inputs that it receives from processes that are not less informed than itself.[9]

To prove this formally, we need some notation. For $p \in P_{\mathrm{con}}$, we define

$$F_p := \{c \in I_p \,|\, w(c) \geq p\},$$

that means, F_p contains the input channels of p which originate in a process that is less informed than p (or equally informed). We call such channels *feedback channels.* We also define $S_p := I_p \setminus F_p$ and we call $c \in S_p$ the *significant* input channels of p. The corresponding alphabets are denoted Σ^p_{fb} and Σ^p_{sig} If $F_p = \emptyset$, then $\Sigma^p_{\mathrm{fb}} := \emptyset$ and if $S_p = \emptyset$ then $\Sigma^p_{\mathrm{sig}} = \{|\}$.

Moreover, we define

- $P^{\geq p} := \{p' \in P_{\mathrm{con}} \,|\, p' \geq p\}$ and
- $\Sigma^{\geq p}_{\mathrm{out}} := \prod_{p' \in P^{\geq p}} \Sigma^{p'}_{\mathrm{out}}$

Notice that while there may be channels from arbitrary processes in $P^{\geq p}$ to processes outside $P^{\geq p}$, there are no channels from processes outside $P^{\geq p}$ to processes in $P^{\geq p} \setminus \{p\}$. So any process $p' \in P^{\geq p} \setminus \{p\}$ receives all its inputs from processes $p'' \in P^{\geq p}$, hence $\Sigma^{p'}_{\mathrm{in}}$ is included in $\Sigma^{\geq p}_{\mathrm{out}}$.

A *focused strategy* for process p is a strategy that reads only the significant input of process p, that means, it is a function

$$\tau : (\Sigma^p_{\mathrm{sig}})^* \to \Sigma^p_{\mathrm{out}}.$$

A tuple $\tau = (\tau_1, \ldots, \tau_n)$ of focused strategies for the controllers is called a joint focused strategy for the grand coalition. Notions like consistency and focused winning strategy are defined as for strategies before. The following lemma makes precise what we mean by saying that p_i can deduce the outputs of all the processes $p_j \geq p_i$ from his own inputs and the joint strategy. We will use this observation to construct, from an arbitrary given winning strategy for the grand coalition, a focused winning strategy.

Lemma 2.3. *Let σ be a joint winning strategy for the grand coalition and let $p \in P_{con}$. Then for all $u \in (\Sigma^p_{sig})^*$ there is exactly one $v \in (\Sigma^{\geq p}_{out})^*$ such that $u^\frown v$ is consistent with σ.*

[9] Notice that this is not the case if we consider a local strategy for process p_i individually since then the inputs that p_i receives from other controller are unconstrained.

Proof. We show this by induction on the length of u. For $u = \varepsilon$ this is trivial, so let $u = u'a$ for some $u' \in (\Sigma^p_{\text{sig}})^*$ and some $a \in \Sigma^p_{\text{sig}}$ be a sequence of significant inputs for p. By induction hypothesis, there is some sequence $v' \in (\Sigma^{\geq p}_{\text{out}})^*$ of joint outputs of all processes in $P^{\geq p}$ such that $u'^\frown v'$ is consistent with σ. We define $b \in \Sigma^{\geq p}_{\text{out}}$ as follows. For any process $p' \in P^{\geq p} \setminus \{p\}$, we define

- $\Pr_{\Sigma^{p'}_{\text{out}}}(b) = \sigma_{p'}(\Pr_{\Sigma^{p'}_{\text{in}}}(v'))$ and we define
- $\Pr_{\Sigma^{p}_{\text{out}}}(b) = \sigma_p(u'^\frown \Pr_{\Sigma^p_{\text{in}}}(v'))$.

As we have mentioned, $\Sigma^{p'}_{\text{in}}$ is included in $\Sigma^{\geq p}_{\text{out}}$. By definition, $u^\frown v$ with $v := v'b$ is consistent with σ.

Now let $w = w'c \in (\Sigma^{\geq p}_{\text{out}})^*$ such that $u^\frown w$ is consistent with σ. Then $u'^\frown w'$ is consistent with σ and so, by induction hypothesis, $w' = v'$. Moreover, as $u^\frown w$ is consistent with σ we get

$$\Pr_{\Sigma^{p'}_{\text{out}}}(c) = \sigma_{p'}(\Pr_{\Sigma^{p'}_{\text{in}}}(w')) = \sigma_{p'}(\Pr_{\Sigma^{p'}_{\text{in}}}(v')) = \Pr_{\Sigma^{p'}_{\text{out}}}(b)$$

for all $p' \in P^{\geq p}$ and

$$\Pr_{\Sigma^{p}_{\text{out}}}(c) = \sigma_p(u'^\frown \Pr_{\Sigma^p_{\text{in}}}(w')) = \sigma_p(u'^\frown \Pr_{\Sigma^p_{\text{in}}}(v')) = \Pr_{\Sigma^p_{\text{out}}}(b).$$

Since $b, c \in \Sigma^{\geq p}_{\text{out}}$ this demonstrates that $b = c$ and so $v = w$. □

Proposition 2.4. *There is a winning strategy for the grand coalition if, and only if, there is a focused winning strategy for the grand coalition.*

Proof. The if-direction is obvious: If $\tau = (\tau_1, \ldots, \tau_n)$ is a focused winning strategy for the grand coalition then the joint strategy $\sigma = (\sigma_1, \ldots, \sigma_n)$ for the grand coalition defined by $\sigma_i(u) = \tau_i(\Pr_{\Sigma^{p_i}_{\text{sig}}}(u))$ is obviously a winning strategy. Now let conversely $\sigma = (\sigma_1, \ldots, \sigma_n)$ be a winning strategy for the grand coalition. We define the focused strategy $\tau = (\tau_1, \ldots, \tau_n)$ for the grand coalition as follows. For $i \in \{1, \ldots, n\}$ and $u \in (\Sigma^{p_i}_{\text{sig}})^*$ let $v \in \Sigma^{\geq p_i}_{\text{out}}$ be the uniquely determined joint output of all processes in $P^{\geq i}$ such that $u^\frown v$ is consistent with σ according to Lemma 2.3. We define

$$\tau_i(u) = \sigma_i(u^\frown \Pr_{\Sigma^{p_i}_{\text{in}}}(v)).$$

To prove that τ is winning, consider any system run $\alpha \in (\Sigma^{\mathcal{D}})^\omega$ which is consistent with τ. By induction on k we show that $\alpha_{<k}$ is consistent with σ. For $k = 0$ this is trivial so let $k > 0$ and let $\alpha_{<k} = \alpha_{<k-1}a$ for some $a \in \Sigma^{\mathcal{D}}$. Now consider any $i \in \{1, \ldots, n\}$ and let $u := \Pr_{\Sigma^{p_i}_{\text{sig}}}(\alpha_{<k-1})$, that means, u is the significant input that p_i receives during $\alpha_{<k-1}$.

Since α is consistent with τ we have $\Pr_{\Sigma^{p_i}_{\text{out}}}(a) = \tau_i(u)$ and by definition of τ_i we have $\tau_i(u) = \sigma_i(u^\frown v)$ for the uniquely determined $v \in \Sigma^{\geq p_i}_{\text{out}}$ such that $u^\frown v$ is consistent with σ. Since $\alpha_{<k}$ is consistent with σ by induction hypothesis, $u^\frown \Pr_{\Sigma^{\geq p}_{\text{out}}}(\alpha_{<k-1})$ is consistent with σ and hence, $\Pr_{\Sigma^{\geq p}_{\text{out}}}(\alpha_{<k-1}) = v$. Therefore, $\Pr_{\Sigma^{p_i}_{\text{out}}}(a) = \sigma_i(u^\frown v) = \sigma_i(\Pr_{\Sigma^{p_i}_{\text{sig}}}(\alpha_{<k-1})^\frown \Pr_{\Sigma^{\geq p}_{\text{out}}}(\alpha_{<k-1})) = \sigma_i(\Pr_{\Sigma^{p_i}_{\text{in}}}(\alpha_{<k-1})$, so $\alpha_{<k}$ is consistent with σ_i.

Since i has been chosen arbitrarily, α is consistent with σ which is a winning strategy so α is won by the grand coalition. Therefore, τ is a winning strategy as well. □

Notice that Lemma 2.3 does not only imply the equivalence between winning strategies and focused winning strategies, but it also shows that a controller p could take the role of all processes $p' \in P^{\geq p}$ by means of an extended strategy: An *extended* strategy for process p is a function

$$\sigma_{\geq p} : (\Sigma^p_{\text{in}})^* \to \Sigma^{\geq p}_{\text{out}}.$$

(Analogously, an extended focused strategy for p is a function $\sigma_{\geq p} : (\Sigma^p_{\text{sig}})^* \to \Sigma^{\geq p}_{\text{out}}$.) Of course, too much information may be used by such a strategy $\sigma_{\geq p}$ when it chooses an action of a process

$p' \geq p$. To prohibit this abuse of information, we need to require consistency conditions like (S2) on an extended strategy, one for each process $p' > p$:

$$(\mathrm{S2})_{p'} \quad \mathrm{Pr}_{\Sigma_{\mathrm{in}}^{p'}}(\sigma^*_{\geq p}(u^{-1})) = \mathrm{Pr}_{\Sigma_{\mathrm{in}}^{p'}}(\sigma^*_{\geq p}(v^{-1})) \implies \mathrm{Pr}_{\Sigma_{\mathrm{out}}^{p'}}(\sigma_{\geq p}(u)) = \mathrm{Pr}_{\Sigma_{\mathrm{out}}^{p'}}(\sigma_{\geq p}(v)).$$

For process p we don't have to require such a consistency condition since for a function $\sigma_{\geq p} : (\Sigma_{\mathrm{in}}^{p}) \to \Sigma_{\mathrm{out}}^{\geq p}$, the projection $\mathrm{Pr}_{\Sigma_{\mathrm{out}}^{p}}(\sigma_{\geq p})$ is a strategy for p. If, on the other hand, p' is a process which is less informed than p then both, the input and the out output of p', are included in the extended output of p. Since p' may make his decision only dependent on the history of events up to the previous step, we have to require that the output of p' in each step depends only on the extended output of p that it produces up to the previous step.

We conclude these observations with a remark on joint strategies. So far, a joint strategy for the controllers has been described as a tuple of strategies. This is not quite appropriate though for solutions of controller problems using tree automata since there, we view strategies as trees and a tuple of trees is not a tree per se. Instead, we resort to the representation from Section 2.1.1 where we have mentioned that a joint strategy for the grand coalition can be viewed as a function $\sigma : T \to \prod_{i=1}^{n} A_i$ such that $\mathrm{Pr}_{A_i}(\sigma)$ fulfills the consistency condition (S2) for all $i = 1, \ldots, n$. So a strategy for the grand coalition $\{p_1, \ldots, p_n\}$ in $\mathcal{D}$ would be a function $\sigma : (\Sigma^{\mathcal{D}})^* \to \Sigma_{\mathrm{out}}$ such that $\mathrm{Pr}_{\Sigma_{\mathrm{out}}^{p_i}}(\sigma)$ fulfills (S2) for all $i = 1, \ldots, n$. We will discuss the tree representation of such a function and its consequences in Chapter 3.

Here, we like to comment on the fact that using the observations presented above, a joint strategy for the grand coalition for $\mathcal{D}$ can also be viewed as a function $\sigma : \Sigma_{\mathrm{read}}^* \to \Sigma_{\mathrm{out}}$ that reads only the (readable) inputs from the environment and fulfills the consistency condition $(\mathrm{S2})_p$ for all $p \in P_{\mathrm{con}}$. We can obtain this representation by introducing a virtual controller p_{vir} which receives all readable inputs from the environment directly and has only the task of routing the information to the corresponding processes. Then an extended strategy of this process is a function $\sigma : \Sigma_{\mathrm{read}}^* \to (\Sigma_{\mathrm{out}}^{\geq p_{\mathrm{vir}}})^*$ that generates the outputs of all controllers and fulfills condition $(\mathrm{S2})_p$ for all $p > p_{\mathrm{vir}}$, that is, for all $p \in P_{\mathrm{con}}$. This representation will also be useful for decision procedures based on tree automata, cf. Chapter 3 and Chapter 5.

2.1.4 Beyond Synchronous Observability

For game graphs with partial information and distributed systems, we have defined strategies for the grand coalition with respect to synchronous observability, that is, in *each step*, every member of the grand coalition receives a certain observation about the current event in the system. These observations are predefined by the system: In game graphs with partial information, player i receives the observation $[a_j]_i[v_j]_i$ while in a distributed system a controller p receives the observation $\mathrm{Pr}_{\Sigma_{\mathrm{in}}^{p}}(\alpha(j))$. So every player always takes notice of each move and also, a player never forgets anything that he has observed.

On the other, in extensive games with imperfect information, the information of the players is given by *arbitrary* equivalence relations $\sim_i$. These equivalence relation may cut through different levels of the game tree, they may hide certain moves from the players completely and they may also make the players forget certain facts that they previously knew. In this section we discuss some general possibilities to represent equivalence relation that define the partial information of the players in a game in a finite way. Notice that even if the set of nodes of the game tree is finitely presented like in the case of game graphs or distributed systems, an equivalence relation on this set may still be a complicated object: The set $\sim_i \subseteq V(\mathrm{AV})^* \times V(\mathrm{AV})^*$ may not be regular or not even finitely presentable.

The notion of synchronous observability is a particular way to obtain such a finite representation, where all we have to define on the model are the observations of the players about the events – the observability is specified implicitly. Of course, this is not just some artificial way to impose finiteness on the equivalence relations. Synchronous observability is a very natural and meaningful way to define the information of the players in a game that encompasses many real life scenarios. But it is a special case of several more general concepts that can be used to define equivalence relations like automata and logical formulas.

We consider these two possibilities and we will analyze which formalisms are sufficient to define synchronous observability. We shall also briefly discuss the decision problems that result from using these more general concepts. We just mention some basic results and we do not go into technical details here. Moreover, as far as game graphs $\mathcal{G} = (V, \delta, (\sim_i^V), (\sim_i^A))$ with partial information are concerned, we restrict to the case where $\sim_i^A = A \times A$ for all i, that is, the actions are completely invisible for the players of the grand coalition. This section is based on work presented in [172].

Automata. There are several ways how a finite automaton can recognize a relation $R \subseteq \Sigma^* \times \Gamma^*$. Here we consider relations that are *componentwise recognizable*, *automatic* and *rational*, respectively. Componentwise recognizable relations are those that can be recognized by a finite automaton that reads, for a given pair $(u, v) \in \Sigma^* \times \Gamma^*$, the word $u\sharp v$. A relation is automatic if there is a finite automaton over the alphabet $(\Sigma \cup \{\sharp\}) \times (\Gamma \cup \{\sharp\})$ that reads the word $u^\frown v$. Finally, a relation is called rational if it can be recognized by an asynchronous automaton that is allowed to read in one component while not reading in the other component. It is easy to see that any componentwise recognizable relation is also automatic and that any automatic relation is also rational. The converse statements are, however, not true.

Clearly, none of these automata models is guaranteed to recognize *equivalence* relations, so the corresponding decision problem for games with imperfect information has to be restricted to those automata that recognize equivalence relations. More precisely, we consider the following decision problem:

Given a game graph $\mathcal{G} = (V, \delta)$ and automata $\mathcal{A}_1, \dots, \mathcal{A}_n$, such that for each i the relation $L(\mathcal{A}_i)$ is an equivalence relation, does the grand coalition have a joint winning strategy?

Of course, a joint winning strategy is now a tuple $\sigma = (\sigma_1, \dots, \sigma_n)$ such that each σ_i is a $\sim_i$-strategy where $\sim_i = L(\mathcal{A}_i)$. Of course, $L(\mathcal{A}_i)$ is not uniquely determined by the automaton $\mathcal{A}_i$, but also depends on the recognition mode. However, once the relations $\sim_i$ are fixed, the question for $\sim_i$-strategies does not depend on the way in which the relation $\sim_i$ is recognized. A $\sim_i$-strategy for player i is the obvious generalization of a usual strategy, that is, it has to satisfy $\sigma_i(\pi) \in \mathrm{act}_i(\mathrm{last}(\pi))$ for all π and $\sigma_i(\pi) = \sigma_i(\rho)$ for all $\pi \sim_i \rho$. Notice that we do not need the equivalence relations $\sim_i^V$ and $\sim_i^A$ anymore.

The following proposition states that the usual strategy problem as introduced before is in fact a special case of this decision problem, but only if we consider at least automatic relations.

Proposition 2.5. *The relation $\sim^* \subseteq V(AV)^* \times V(AV)^*$ is automatic but not componentwise recognizable, for any given equivalence relation $\sim^V$.*

The first part of the proposition is trivial. In fact, $\sim^*$ can be recognized by an automaton with only two states. On the other hand, to see that $\sim^*$ is not componentwise recognizable, independent of the equivalence relation $\sim^V$, the usual pumping argument can be applied. This can be also seen, however, from the more general statement that for any componentwise recognizable equivalence relation $\sim \subseteq \Sigma^* \times \Sigma^*$, the index $|\Sigma^*/\!\sim|$ of $\sim$ if *finite*. To see this, consider a deterministic finite automaton recognizing $\sim$ and define, for each pair of states (p, q) of the automaton such that q is accepting, the set $X_{p,q} := \{u \in \Sigma \mid \delta^*(q_{\mathrm{in}}, u) = p \text{ and } \delta^*(q_{\mathrm{in}}, u\sharp u) = q\}$. It is easy to see that the sets $X_{p,q}$ form a partition of Σ^* and that each two words $u, v \in X_{p,q}$ are equivalent. Hence, $|\Sigma^*/\!\sim| \leq |Q|^2$ where Q is the set of states of the automaton.

Although the relation $\sim^*$ is not componentwise recognizable, componentwise recognizable equivalence relation may still be interesting for certain applications. In particular, this includes the case of relations that are defined in the following way: Consider a deterministic finite automaton $\mathcal{A}$ over the alphabet $V \cup \mathrm{AV}$ and for two histories π and ρ, let $\pi \sim \rho$ if, after reading π and ρ, respectively, $\mathcal{A}$ is in the same state. Clearly, the relation $\sim$ is componentwise recognizable: A corresponding automaton $\mathcal{A}'$ just needs to simulate $\mathcal{A}$ on π, memorize the current state and then simulate $\mathcal{A}$ on ρ.

Clearly, such an equivalence relation is not about observability at all. Instead, the automaton $\mathcal{A}$ can be seen as a device that predetermines the way in which a controller processes the history of events in the system. It does not, however, specify a controller, since it does not determine the *actions* of the controller. But it prescribes the way in which the controller processes the information that it receives, which, in this case, is the full history of events. This is interesting for applications where the computing device that is supposed to serve as a controller is already given and the task is to program an appropriate output function.

It is not hard to see that for this special case, the strategy problem for games on finite graphs with an arbitrary number of players is decidable for ω-regular winning conditions, cf. Section 2.2. If we have a game graph with a parity winning condition then we can proceed as follows: We guess, for each automaton $\mathcal{A}_i$, $i = 1, \dots, n$ a labeling of the states of $\mathcal{A}_i$ which, essentially, yields a strategy automaton for player i, cf. Section 3.5.2. Then we can take the product of the game graph and the resulting strategy automata which results in a game graph where only player 0 makes actual moves. It is easy to verify whether player 0 wins this game. We do not go into the details of this construction. For the general notion of the product of an automaton and a game graph, see Section 2.2.

On the other hand, for automatic relations, the strategy problem is undecidable, even for two-player games with safety winning conditions. To see this, we use the fact that for three-player games on finite graphs with partial information the strategy problem is undecidable for safety conditions, cf. Section 3.2. We restrict to turn-based game graphs for which this also holds.

This case can then be easily simulated by two-player games if we use automatic relations to define the information of player 1: Given a turn-based game graph $\mathcal{G} = (V, \delta, (\sim_i^V))$ with partial information and three players, we keep the game graph $\mathcal{H} = (V, \delta)$ as it is, but now, all positions $V_1^{\mathcal{G}} \cup V_2^{\mathcal{G}}$ of player $1^{\mathcal{G}}$ and player $2^{\mathcal{G}}$ belong to player $1^{\mathcal{H}}$. Moreover, we define the relation $\sim_1^{\mathcal{H}}$ as follows: $\pi \sim_1^{\mathcal{H}} \rho$ if $\mathrm{last}(\pi), \mathrm{last}(\rho) \in V_1^{\mathcal{G}}$ and $\pi \sim_1^* \rho$ or if $\mathrm{last}(\pi), \mathrm{last}(\rho) \in V_2^{\mathcal{H}}$ and $\pi \sim_2^* \rho$. That is, if player $1^{\mathcal{H}}$ is supposed to make a move of player $1^{\mathcal{G}}$ then he also has the view of player $1^{\mathcal{G}}$ and accordingly for player $2^{\mathcal{G}}$.[10]

Notice that the relation $\sim_1^{\mathcal{H}}$ imposes a strong form of imperfect recall on player $1^{\mathcal{H}}$: When playing as player $1^{\mathcal{G}}$, he forgets all the information that he has acquired while playing as player $2^{\mathcal{G}}$ and vice versa. Clearly, player $1^{\mathcal{H}}$ has a winning strategy for $(\mathcal{H}, W, v_0)$ if, and only if, the grand coalition has a joint winning strategy for $(\mathcal{G}, W, v_0)$. Moreover, since the relations $\sim_1^*$ and $\sim_2^*$ are both automatic it is easy to see that $\sim_1^{\mathcal{H}}$ is automatic as well (going through nondeterministic automata and using determinization). This reduction is obviously effective, so the strategy problem for two-player games on finite graphs with safety winning conditions is undecidable.

Logics. Another way to define equivalence relations $\sim \subseteq V(\mathrm{AV})^* \times V(\mathrm{AV})*$ are logical systems like first order logic and monadic second order logic. In this case we have to refer to some structure which the formulas should talk about.[11] Of course, the definability of a relation depends highly on the chosen structure.

Here, we choose a possibility that is close to finite automata. For a game graph $\mathcal{G} = (V, \delta)$, we consider the structure $\mathcal{T}_{\mathcal{G}} = (V(\mathrm{AV})^*, E, (P_v)_{v \in V})$ which is the game tree defined by $\mathcal{G}$ where $E \subseteq V(\mathrm{AV})^* \times V(\mathrm{AV})^*$ is the edge relation of $\mathcal{T}_{\mathcal{G}}$, that is, $(\pi, \rho) \in E$ if $\rho = \pi a v$ for some $av \in \mathrm{AV}$. Moreover, we can access the last position of a history: $\pi \in P_v$ if $\mathrm{last}(\pi) = v$.

We say that an equivalence relation $\sim \subseteq V(\mathrm{AV})^* \times V(\mathrm{AV})^*$ is definable in a logic $\mathcal{L}$ if there is a formula $\varphi(x, y) \in \mathcal{L}$ over the signature $\{E, (P_v)_{v \in V}\}$ such that for all $\pi, \rho \in V(\mathrm{AV})^*$ we have $\mathcal{T}_{\mathcal{G}} \models \varphi(\pi, \rho)$ if, and only if, $\pi \sim \rho$. Notice that as before, a relation $R \subseteq V(\mathrm{AV})^* \times V(\mathrm{AV})^*$ defined by a formula $\varphi(x, y)$ is not guaranteed to be an equivalence relation. The corresponding strategy problem is defined in the same way as for automata recognizable relations, that is,

[10]This argument was suggested to the author by Łukasz Kaiser in personal communication.
[11]Like in the case of ω-words, where the formulas of S1S talk about a structure $(\mathbb{N}, +1, (P_a)_{a \in \Sigma})$.

restricted to those formulas that define in fact an equivalence relation.

Theorem 2.6. *The relation $\sim^* \subseteq V(AV)^* \times V(AV)^*$ is definable in LFP but in general it is not definable in MSO.*

Here, LFP denotes least fixed point logic and MSO denotes monadic second order logic. Definability in LFP is straight forward: We just write down the definition of $\sim^*$ inductively as $\pi \sim^* \rho$ if $\pi, \rho \in V$ and $\pi \sim^V \rho$ or $\pi = \pi' av$, $\rho = \rho' bw$, $v \sim^V w$ and $\pi' \sim^* \rho'$. So with the same argumentation as for automatic relation we can now see that for equivalence relations defined in (the alternation-free fragment of) LFP, the strategy problem is undecidable.

Non-definability in MSO can be seen as follows. Consider the (one-player) game graph $\mathcal{G} = (V, \delta, \sim^V, \sim^A)$ with partial information defined as follows.

- $V = V_0 = \{\varepsilon, 0, 1\}$ and $A = \{a, b\}$
- $\delta(\varepsilon, a) = \delta(0, a) = \delta(1, a) = 0$
- $\delta(\varepsilon, b) = \delta(0, b) = \delta(1, b) = 1$
- $0 \sim^V 1$ and $a \sim^A b$

The structure $\mathcal{T}_\mathcal{G}$ is the full, infinite binary tree with two additional predicates that indicate whether the current node is the 0-child or the 1-child of its unique parent node. Now assume that there is a formula $\varphi(x, y)$ such that for all $\pi, \rho \in V(\mathrm{AV})^* \times V(\mathrm{AV})^*$ we have $\mathcal{T}_\mathcal{G} \models \varphi(\pi, \rho)$ if, and only if, $\pi \sim^* \rho$. Obviously, $\pi \sim^* \rho$ holds if, and only if, $|\pi| = |\rho|$, so φ defines the equal level predicate on $\mathcal{T}_\mathcal{G}$.

Having this formula, it is not hard to construct an S2S-formula $\psi(x, y)$, that is, a formula of the second order logic of two successors, that defines the equal level predicate on the infinite binary tree: Formulas of S2S use the successor functions S_0 and S_1 with $S_j(u) = uj$ and it is easy to define the edge relation E as well as the predicates P_0 and P_1 on the infinite binary tree, using S_0 and S_1. Furthermore, ψ can be used to define the set of Σ-labeled infinite binary trees (for an arbitrary alphabet Σ) such that the labeling is constant on each level. However, this set of trees cannot be recognized by a tree automaton and so, by Rabin's Theorem, it cannot be defined in S2S, cf. Section 3.1. This is a contradiction, so the formulas φ cannot exist.

Asynchronous Observability. As we have mentioned, unless explicitly mentioned otherwise, we consider only games with imperfect information which is defined via *synchronous observability*. In particular, we will not consider strategy problems under imperfect information defined via equivalence relations that are given by automata or logical formulas here any further. We would like to comment on another particular case besides synchronous observability, namely that of *asynchronous observability*.

As we have mentioned, synchronous observability implies the existence of an underlying *shared clock* that can be accessed by all components of the system. If the system under consideration is not adapted to synchronization, that is, cannot be reasonably timed by a global signal, this assumption is not adequate, cf. [170, 173]. Asynchronous systems, where we do not require the existence of such a common clock, are suited for such settings. However, the notion of asynchronous observability is not as canonical as that of synchronous observability:

in a synchronous system, each agent i knows what time it is; ([78], p.135)

More precisely, a player i in an extensive game with imperfect information has a *synchronous view*, if any two histories which are indistinguishable for player i are on the same level of the game tree. Otherwise, player i has an *asynchronous view*. So a player with asynchronous view does not know what time it is but it is also not clear what knowledge he *does* have about the time.

Now, asynchronous observability is a special case of asynchronous imperfect information where the information of the players is defined in terms of observations that they make just as in the case of synchronous observability. However, not all players receive an observation in each step. The decision which processes receive observations in which step is usually made by a *scheduler*, see for example [170, 194]. Such a scheduler can be formalized as an external strategy σ_S that takes a history of a run and yields a set of processes that are scheduled to read and a set of processes that are scheduled to write. Clearly, there are still lots of varieties of this setting. We could consider nondeterministic scheduler that allow several (or maybe even all) possible schedulings, we could require that a process is scheduled to write whenever it is scheduled to read or that each process is scheduled to write in each step and so on. Some work on synthesis of controllers with asynchronous observability can be found in [170, 221, 194].[12]

Here we consider another very special case of asynchronous observability. Consider a *turn-based* game on a finite graph, that means, we have a partition $V = \bigcup_i V_i$ of the positions where $v \in V_i$ means that it is player i's turn. This determines a scheduler for the writing players: the (unique) player that is scheduled to write is the one whose turn it is. Moreover, we define that a process is scheduled to read whenever something happens in the game that he actually can observe. (Intuitively, if some variables of the system are modified to which the player has at least partial access.) Essentially, this is the most basic case of asynchronous observability: It is the same as the synchronous case, except that we remove the shared clock so that a player does receive a simulated tick whenever something happens that he cannot observe at all. This case has been considered in [173] and, similarly, in [47] in the context of timed games.

In other words, this case is obtained from the synchronous case by simply hiding all private moves of the other players from a player i. Where a private move of the players $[n] \setminus \{i\}$ is a move $v \to_a w$ such that $v \notin V_i$ and $v \sim_i^V w$. So, given a game graph $\mathcal{G} = (V, \delta, (\sim_i^V))$, an asynchronous strategy for player i σ takes a history and yields a next action for player i as before, but now has to satisfy the following condition:

$$(\overleftarrow{S}2) \text{ if } \pi \overleftarrow{\sim}_i^* \rho \text{ then } \sigma(\pi) = \sigma(\rho)$$

where $\pi \overleftarrow{\sim}_i^* \rho$ if $\overleftarrow{\pi} \sim_i^* \overleftarrow{\rho}$ and $\overleftarrow{\pi}$ if obtained from π by contracting each maximal sequence $w_0 b_1 w_1 \ldots b_l w_l$ of private moves of the players $[n] \setminus \{i\}$ in π to w_0.

Notice that, originally, we have defined $\sim_i^*$ as an equivalence relation on histories in $\mathcal{G}$ and, clearly, $\overleftarrow{\pi}$ is not a history in $\mathcal{G}$ in general. However, in this regard, we shall treat $\overleftarrow{\pi}$ as though it was a history in $\mathcal{G}$.

As to the question of recognizability and definability of $\overleftarrow{\sim}^*$ we have similar results as for the synchronous case: $\overleftarrow{\sim}^*$ is definable in LFP but not in MSO. As to the latter statement, we just have to modify the game graph so that now, 0 and 1 are both positions of player 1 while ε is still a position of player 0. Moreover, $\overleftarrow{\sim}^*$ is not componentwise recognizable, but now, this depends on $\sim^V$. In a game graph where all positions belong to player 0 and are indistinguishable for player 1, any two histories as equivalent with respect to $\overleftarrow{\sim}^*$ and so the relation is componentwise recognizable. However, in general it is not. But now, it is also not automatic in general. This is also obvious due to the asynchronous nature of $\overleftarrow{\sim}^*$, so we have to invoke asynchronous automata to recognize $\overleftarrow{\sim}^*$ in general: For any $\sim^V$, the relation $\overleftarrow{\sim}^*$ is rational.

We do not consider asynchronous observability in detail but we shall make some remarks. In particular, in Section 3.5.4 we adapt Reif's powerset construction to this case.

2.1.5 Memory Structures

So far, a strategy for a player i has just been a function σ that takes a finite history of a play and yields a next action for player i. The postulation that a strategy for player i uses only information which is actually available to player i has been asserted by a consistency condition with respect to the indistinguishability relation $\sim_i$ of player i or by simply restricting the input

[12]As it turns out, synthesis of controllers with asynchronous observability is even harder as in the corresponding synchronous case: Finkbeiner and Schewe [194] proved that for both, synthesis under full schedulers and scheduler independent synthesis, the strategy problem is undecidable as soon as we consider distributed systems with more than one controller.

of the strategy to the part of the history that player i can observe. In any case, a strategy has been an *infinite object.*

A solution of the controller synthesis problem, however, requires an *implementation* of the strategy by some kind of computing device. In this section we define *finite state strategies*, that is, strategies that can be implemented by finite state automata. In Chapter 5 we consider also strategies that are implemented by *pushdown automata.* Since it resembles physical implementations of strategies more closely, we consider as implementations finite state automata which read only the part of a history which is observable for player i.[13] That is, the automata have as alphabet the set $[V]_{\sim_i^V}[A]_{\sim_i^A}$, in the case of game graphs with partial information and the set $\Sigma_{\text{in}}^{p_i}$ in the case of distributed systems. In the following, let $\mathcal{G} = (V, A, \delta, (\sim_i^V)_i, (\sim_i^A)_i)$ be a game graph with partial information and let $\mathcal{D} = (\mathfrak{A}, (\Sigma_c)_{c\in C})$ be a distributed system.

A *finite memory structure* for player i for $\mathcal{G}$ has the form

$$\mathcal{M} = (M, \delta_{\text{in}}, \delta_{\text{up}})$$

where M is a finite set of states, $\delta_{\text{in}} : V \to M$ is the memory initializing and $\delta_{\text{up}} : M \times [A]_i[V]_i \to M$ is the memory update function. (Notice that we have assumed that each player of the grand coalition knows the initial position.) The state $\delta^*(\pi)$ of $\mathcal{M}$ after some history $\pi = v_0a_1v_1 \dots a_lv_l$ is defined as follows:

$\delta^*(v_0) = \delta_{\text{in}}(v_0)$

$\delta^*(\pi) = \delta_{\text{up}}(\delta^*(v_0a_1v_1 \dots a_{l-1}v_{l-1}), [a_l][v_l])$, if $l > 0$.

A *memory strategy* for player i with memory $\mathcal{M}$ is a function $\sigma_i : M \times [V]_i \to A_i$ such that for all histories $\pi = v_0a_1v_1 \dots a_lv_l$ in $\mathcal{G}$ we have $\sigma_i(\delta^*(\pi, [v_l])) \in \text{act}(v_l)$. A play $\pi = v_0a_1v_1 \dots$ is *compatible* with σ_i, if for all $j \in \mathbb{N}$ we have

$$\text{Pr}_i(a_{j+1}) = \sigma_i(\delta^*(\pi), [v_j]).$$

A memory strategy with memory $\mathcal{M}$ for some finite memory structure $\mathcal{M}$ is also called a *finite state strategy.* A strategy is called *memoryless* or *positional* if it can be implemented by a memory structure with a single memory state.

A finite memory structure for process $p = p_i$ for $\mathcal{D}$ is basically the same as a memory structure for player i for $\mathcal{G}$. However, we do not have an initializing function but a fixed initial state $m_{\text{in}} \in M$ so the structure has the form $\mathcal{M} = (M, m_{\text{in}}, \delta_{\text{up}})$ where the update function δ_{up} is now a function $\delta_{\text{up}} : M \times \Sigma_{\text{in}}^p \to M$. Moreover, $\delta^*(u)$, for $u = a_0a_1 \dots a_l \in (\Sigma_{\text{in}}^p)^*$ is defined by $\delta^*(\varepsilon) = m_{\text{in}}$ and $\delta^*(u) = \delta_{\text{up}}(\delta^*(a_0a_1 \dots a_{l-1}), a_l)$ for $l > 1$. Since we do not have a notion of a *current position* in a game graph, we also re-define the notion of a memory strategy slightly: A *memory strategy* for process i with memory $\mathcal{M}$ is a function $\sigma_p : M \to \Sigma_{\text{out}}^p$ and a local run $\beta_p{}^\frown\alpha_p \in (\Sigma_{\text{in}}^p \times \Sigma_{\text{out}}^p)^\omega$ is consistent with σ_p if $\alpha_p(j) = \sigma_p(\delta^*(\beta_p(<j)))$ for all $j \in \mathbb{N}$.

Notice that with this definition, a memoryless strategy is always trivial in that it yields the same action in each round. To see that this is appropriate, consider a distributed system with a single controller p_1 and without hidden input channels (which constitutes a game with full information) and with a parity specification given by a deterministic parity automaton $\mathcal{A}$ (see Section 2.2). Then in general, process p_1 needs to use at least the current state of $\mathcal{A}$ for his strategy, which is, he has a positional strategy in terms of the *states of* $\mathcal{A}$. Also notice that this definition of a finite state strategy coincides with our claim of computability of a strategy by a finite state machine with output since the strategy σ_p can be seen as the output function of $\mathcal{M}$.

2.2 Winning Conditions

So far, a winning condition has just been any subset of the set of plays in a game with imperfect information. In this section we consider certain special cases of winning conditions, according to

[13] As we have mentioned, the correspondence to strategies that take the whole history but just satisfy consistency constraint (S2) is not as straightforward here as it is when implementations are neglected.

their expressive power and the extent to which they may involve facts, that the players cannot observe. In particular, for an algorithmic treatment of the strategy problem we need not only a finite representation of the game itself, like given by game graphs with partial information and architectures, but we also have to consider winning conditions that admit a finite representation.

As we have already mentioned in the introduction, the definition of a winning condition as a *set of plays* puts our investigation into the realm of *linear time*: A linear time specification determines for each given infinite play whether it is a good play or not, and we require that *all* plays that are compatible with a winning strategy are good. In contrast, a branching time specification does not specify a set of good plays but it defines a set of good trees: A joint winning strategy for the grand coalition is required to be a good tree, that is, it must be a model of the branching time specification. (Or, rather, the corresponding constrained game tree which is, however, essentially the same.) Examples of branching time specifications are computation tree logic CTL, the modal μ-Calculus and, as non-logical formalism, automata on infinite trees. In [6], the even stronger formalism of *alternating* time temporal logic ATL has been introduced. It has been suggested as a natural specification language for open reactive systems, where the paradigm is to incorporate *strategic reasoning* directly into the logic. We do not consider this here any further.

Notice that the linear time setting and the branching time setting are in fact inherently different. Consider, for example, a CTL-formula φ, defining a set of Σ-labeled X-trees for a given finite set X and a given alphabet Σ. Then, in general, there is no set $W \subseteq \Sigma^*$ such that a tree t satisfies φ if, and only if, all infinite paths through t are in W. In particular, in the branching time framework, we can also require the *existence* of certain paths in a given tree which is obviously not possible in the linear time framework.

The advantages and disadvantages of linear and branching time specifications have been subject to many discussions, see for example the fundamental study [131], the subsequent discussions [73, 168] and the more recent paper [214] with many further pointers to the literature. In this work we focus on linear time specifications. We will, however, provide certain remarks on branching time specifications as well, using the tree automata framework from Section 3.1 as specification formalism.

Throughout this section, let

$$\mathcal{G} = (V, \delta, (\sim_i^V)_i, (\sim_i^A)_i) \text{ and } \mathcal{D} = (\mathfrak{A}, (\Sigma_c)_{c \in C})$$

denote a game graph with partial information and $n+1$ players and a distributed system with n controllers and architecture $\mathfrak{A} = (C, w, r)$, respectively.

2.2.1 ω-Regular Winning Conditions

Omega-regular winning conditions can be represented as languages, accepted by ω-automata or, more directly, by means of a coloring $\mathrm{col} : V \to [k]$ of the positions of the game graph (for some $k \in \mathbb{N}$), in case a game graph is given explicitly.

To the latter setting, full effect has been given by the seminal work of McNaughton [142] on infinite games on finite game graphs with *full information*, followed by fundamental studies in [202, 222]. See also [100] for a comprehensive exposition. The most well studied special case are parity games with special attention on algorithmic aspects, see [118, 158, 113].[14] Parity conditions on finite game graphs are also a core issue in our study and we start the exposition of winning conditions with parity games.

Parity Games. A parity condition on $\mathcal{G}$ is given by a coloring $\mathrm{col} : V \to [k]$ for some natural number $k \in \mathbb{N}$. A play $\pi = v_0 a_1 v_1 \ldots \in \Pi$ is won by the grand coalition if $\min(\mathrm{Inf}(\mathrm{col}(\pi)))$ is

[14]The great interest in algorithmic properties of parity games is partly due to the fact that the strategy problem for parity games is polynomial time equivalent to the emptiness problem for tree automata and the model checking problem for the modal μ-calculus, see e.g. [100].

even, where $\mathrm{col}(\pi) = \mathrm{col}(v_0)\mathrm{col}(v_1)\ldots$. That means, the grand coalition wins a play according to the parity condition defined by col, if the least color which is seen infinitely often during the play is even. Notice that a parity conditions depends only on the sequence of *positions* in a play and not on the actions. A *parity game with imperfect information* is a game with imperfect information played on a finite game graph $\mathcal{G}$ where the winning condition W is a parity condition on $\mathcal{G}$, so it is represented as $(\mathcal{G}, \mathrm{col})$. Notice that the winning condition W of a parity game is *position based.*

The parity condition first appeared in [149] in form of the *Rabin Chain Condition* and later, independently, in [75] as the parity condition. Since then, parity games with full information have been extensively studied and many deep and useful results are available, especially on algorithmic aspects. Here we collect some of the fundamental results which are relevant to our work.

It can be shown that sets of plays in a game graph which are given by a parity condition are Borel sets and hence, by Martin's Theorem, parity games with full information are determined. Even more, they enjoy the much stronger property of *uniform memoryless determinacy*, that means, each player $i \in \{0, 1\}$ has a memoryless winning strategy which is winning from any vertex in his winning region. Positional determinacy was first proved by Emerson and Jutla [75] and, independently, by Mostowski [150]. McNaughton [142] gave a constructive proof of uniform memoryless determinacy.

This also yields that the strategy problem for parity games with full information is in NP ∩ co-NP, since a memoryless strategy (for either player!) can be *guessed* and verification of the winning property can easily be performed in polynomial time. However, this still gives an algorithm with running time exponential in the number of positions of the game graph. Up to date it is not known whether parity games with full information can be solved in polynomial time while, as is well known, the question whether they are NP-hard is also open since this would imply NP = co-NP. As we have already suggested, proving that parity games with full information can be solved in polynomial time would be a major contribution to the theory of verification and synthesis.

There are, however, better algorithms for solving the strategy problem for parity games with full information (and constructing uniform memoryless winning strategies for both players on their winning regions) that have a worst case time complexity which is polynomial in the number of vertices of the game graph and only exponential in the number k of colors, see for example [118]. We can summarize these results as follows.

Theorem 2.7. *[75, 150, 142], see also [118]*
Parity games with full information are uniformly determined with memoryless strategies and the winning regions as well as uniform memoryless winning strategies for both players can be constructed in time $O(n^k)$.

We already know that the determinacy result does not carry over to parity games with imperfect information. Moreover, memoryless strategies are not sufficient for winning in games with imperfect information, already for reachability conditions. In Section 3.5.2 we will show that even exponentially large memory is needed in general.

An important subclass of parity games are those with two colors, called *Büchi games.* Usually, we consider Büchi conditions given by a set $R \subseteq V$ where a play π is won by the grand coalition if the set R is visited infinitely often during π.

The simplest winning conditions that we consider are *reachability* and *safety* conditions. A reachability condition on $\mathcal{G}$ is given by a set $R \subseteq V$ of vertices, at least one of which the grant coalition wants to reach. So a play π in $\mathcal{G}$ is won by the grand coalition if some vertex from R occurs in π. The safety condition given by R is the dual of the reachability condition, that is, a play π is won by the grand coalition if each vertex that occurs in π is in R.

Reachability and safety games are not direct special cases of parity games but can be easily seen as special cases using simple modifications of the game graph, both for game graphs with full information as well as those with partial information.[15] In particular, reachability and safety games are uniformly determined with memoryless strategies.

Muller conditions. Another well-studied case of ω-regular winning conditions are *Muller conditions*, introduced by Muller in [151] in the context of ω-automata. We do not consider Muller conditions a priori but use them as a tool in certain constructions and, due to their great significance in the theory of games on graphs as well as ω-automata, we like to comment on them. A Muller condition is given by a set $\mathcal{F} \subseteq 2^V$, and a play π is won by the grand coalition if $\text{Inf}(\pi) \in \mathcal{F}$. Many representations of Muller conditions have been considered and the complexity of the corresponding strategy problem has been investigated, cf. [113], one of the most thriving being Ziełonka's split trees [222]. Notice that Muller games with *explicitly* represented $\mathcal{F}$ can be solved in polynomial time [111].

Muller games on finite graphs with full information are not determined with memoryless strategies in general but determined with finite state strategies [102], see also [142, 202]. The memory structure that is used to implement winning strategies in these games is called latest appearance record (LAR). In [71], a lower bound on the memory which is needed to win in Muller games with full information has been proved which essentially matches the upper bound provided by the LAR-memory. By taking the product of the given game graph and the LAR-memory structure and defining an appropriate coloring, Muller games can be reduced to parity games, see [202]. (Thomas terms this construction as *supplying sufficient memory in the game graphs*, cf. [203],p.55.)

The resulting game graph will, however, be exponentially larger in general since for a given Muller condition over V, the LAR-memory has $|V|!$ states.

ω-Automata. While for game graphs with full information it is more convenient and direct to consider ω-regular winning conditions represented by a coloring of the positions of the game graph, for distributed systems as well as for game graphs with partial information, this is not always appropriate. In particular, a distributed system does not provide a game graph with vertices to color. Moreover, non-observable winning condition (see Section 2.2.3) require automata methods with transformations like determinization involved, even if the original winning condition is in fact given by means of a coloring of the game graph. Hence, we also consider ω-regular winning conditions represented by ω-automata.

We consider ω-automata with Büchi acceptance condition and those with parity acceptance condition. Notice that we obtain these automata also as a natural restriction of both, the tree automata from Section 3.1 as well as the pushdown automata that we shall define below. However, due to their conceptual simplicity and their outstanding importance for the theory of ω-automata, we still like to give an explicit definition of nondeterministic Büchi and parity automata as well as their deterministic versions. Notice that the term Büchi automaton usually means *nondeterministic* Büchi automata, while for parity automata, the deterministic model is the standard concept. This is mainly due to the fact that, while deterministic Büchi automata are strictly weaker in expressive power than nondeterministic ones, deterministic parity automata are equally expressive as the nondeterministic version. For the *alternating* version of parity automata, we refer to tree automata. Notice, however, that ω-automata on words have several properties that are not enjoyed by automata on infinite trees. For example, nondeterministic parity tree automata are strictly more expressive than deterministic parity tree automata.

A nondeterministic *ω-automaton* has the form

$$\mathcal{A} = (\Sigma, Q, q_{\text{in}}, \delta, \text{acc}).$$

- Σ is the finite input alphabet
- Q is the finite set of states and $q_{\text{in}} \in Q$ is the initial state

[15]We just need this fact to make clear that certain upper bounds which we prove for parity games also hold for reachability and safety games.

- $\delta : Q \times \Sigma \to 2^Q$ is the transition function
- $\mathrm{acc} \subseteq Q^\omega$ is the acceptance component

A *run* of $\mathcal{A}$ on $\alpha \in \Sigma^\omega$ is a sequence $\rho = q_0 q_1 \ldots \in Q^\omega$ of states such that, $q_0 = q_{\mathrm{in}}$ and, for all $j \in \mathbb{N}$, $q_{j+1} \in \delta(q_j, \alpha_j)$. The run is *accepting* if $\mathrm{Inf}(\rho) \in \mathrm{acc}$. The automaton *accepts* α if there is an accepting run of $\mathcal{A}$ on α. The set

$$L(\mathcal{A}) = \{\alpha \in \Sigma^\omega \mid \mathcal{A} \text{ accepts } \alpha\}$$

is called the language recognized by $\mathcal{A}$ or simply the language of $\mathcal{A}$.

$\mathcal{A}$ is called Büchi automaton if acc is a Büchi condition, that means, the automaton has the form $\mathcal{A} = (\Sigma, Q, q_{\mathrm{in}}, \delta, F)$ where $F \subseteq Q$ is a set of final states. A run $\rho \in Q^\omega$ is accepting if $\mathrm{Inf}(\rho) \cap F \neq \emptyset$, that is, at least one state from F is seen infinitely often. $\mathcal{A}$ is called parity automaton if acc is a parity conditions, that means, the automaton has the form $\mathcal{A} = (\Sigma, Q, q_{\mathrm{in}}, \delta, \mathrm{col})$ where $\mathrm{col} : Q \to [k]$ is a coloring of the states of $\mathcal{A}$. A run $\rho \in Q^\omega$ is accepting if $\min(\mathrm{Inf}(\mathrm{col}(\rho)))$ is even, that is, the least color which is seen infinitely often during ρ is even. $\mathcal{A}$ is called *deterministic* if for all $(v, a) \in Q \times \Sigma$ we have $|\delta(v, a)| = 1$, that is, δ is a function $Q \times \Sigma \to Q$.

The model of Büchi automata was introduced by Büchi in [43] where he showed that an ω-language is recognizable by such an automaton if, and only if, it is definable in the monadic second order logic of one successor S1S. Since emptiness of Büchi automata is decidable quite easily this yields that satisfiability for S1S is decidable which also implies that the monadic second order theory of $(\mathbb{N}, +1, <, 0)$ is decidable. For in-depth treatments of the intimate connections between logics and automata, we refer the reader to [100] and [83].

Nondeterministic Büchi automata and deterministic parity automata have the same expressive power: they both recognize exactly the ω-regular languages (whichever characterization of these languages one would like to start with as a definition). On the other hand, deterministic Büchi automata don't recognize all regular languages, that is, they are strictly less expressive than nondeterministic Büchi automata. A determinization method for Büchi-automata was first proved by McNaughton in [141] using deterministic automata *Muller automata*. (In fact, McNaughton used a special case of Muller acceptance conditions that is equivalent to the pairs acceptance condition, cf. Section 3.1.1.) The resulting deterministic Muller automata can then be translated into an equivalent deterministic parity automaton using the LAR-memory structure that is also used to reduce Muller games to parity games, see for example [100]. Simulating a parity automaton by a nondeterministic Büchi-automaton is fairly simple, which establishes the expressive equivalence of the automata models. Notice that since deterministic parity automata can be easily complemented (complementing a parity condition can be done by just shifting the colors by one), this also implies the complementation theorem for nondeterministic Büchi automata which has first been proved by Büchi in [43].

During the last decades, a large number of determinization and complementation constructions for Büchi automata have been suggested which use different acceptance conditions and provide better complexity results for the resulting automata, most notably the work of Safra [187] where he provided an essentially optimal translation of nondeterministic Büchi automata with n states to deterministic Rabin automata with $2^{O(n \log(n))}$ states and $O(n)$ pairs. Optimality of this construction was proved in [133, 134]. A deterministic Rabin automata with m states and r pairs can be translated into a deterministic parity automaton with $m \cdot 2^{r \log(r)}$ states and $O(r)$ colors using the so called index appearance record [188, 45, 133] and hence, the original nondeterministic Büchi automaton can be translated into an equivalent deterministic parity automaton with $2^{O(n \log(n))}$ states and $O(n)$ colors as well.

Theorem 2.8. *[187], see also [133]*

(1) A nondeterministic Büchi automaton with n states can be translated into a deterministic parity automaton with $2^{O(n \log(n))}$ states and $O(n)$ colors.

(2) A nondeterministic parity automaton with n states and k colors can be translated into a nondeterministic Büchi automaton with $O(n \cdot k)$ states.

We also like to draw the reader's attention to the work of Muller and Schupp [154] who provided a different construction for determinization of ω-automata, see also the historical remarks in Section 3.1.2. Of the more recent papers on determinization and complementation of Büchi automata we like to mention the work of Piterman [165] and Schewe [192, 193] which provide very good complexity results.

Now an ω-automaton $\mathcal{A}$ representing a winning condition for $\mathcal{G}$ is considered as an automaton over the alphabet $V \cup \mathrm{AV}$ and the winning condition is

$$W = \{\pi \in \Pi \mid \pi \in L(\mathcal{A})\}.$$

So while the automaton may, for example, accept sequences from V^ω or sequences from $V(AV)^\omega$ which are not plays, we are only interested in the *plays* in $L(\mathcal{A})$.

Given a game graph $\mathcal{G}$ and a *deterministic* parity automaton $\mathcal{A}$ representing an ω-regular winning condition for $\mathcal{G}$, we construct a *parity game* as follows:

$$\mathcal{G} \times \mathcal{A} = \Big(V \times Q, (A_i)_i, \Delta^\times, (\sim_i^{V\times Q})_i, (\sim_i^{A})_i, \mathrm{col}^\times\Big)$$

- $\Delta^\times = \{((v,p), a, (w, \delta(p, aw))) \mid (v,a,w) \in \Delta\}$
- $(v,p) \sim_i^{V\times Q} (w,q) :\Longleftrightarrow v \sim_i^V w$
- $\mathrm{col}^\times(v,p) = \mathrm{col}(p)$.

So the game $\mathcal{G} \times \mathcal{A}$ is played on the synchronous product of the game graph $\mathcal{G}$ and the transition graph of $\mathcal{A}$ and we *hide* the states of the automaton from all players of the grand coalition. This is clearly necessary since in the game on $\mathcal{G}$, the players cannot observe the states of $\mathcal{A}$ as well.[16] The parity winning condition on the positions of the new game graph is given in terms of the states of the parity automaton. Notice that while the winning condition W defined by the deterministic parity automaton $\mathcal{A}$ is not necessarily position based, the winning condition of $\mathcal{G} \times \mathcal{A}$ defined by col is position based. The correctness of the construction is proved straightforwardly.

Proposition 2.9. *For any initial position $v_0 \in V$, the grand coalition has a winning strategy for $(\mathcal{G}, \mathcal{A}, v_0)$ if, and only if, the grand coalition has a winning strategy for $(\mathcal{G} \times \mathcal{A}, col^\times, (v_0, \delta(q_{in}, v_0))$.*

Now assume $\mathcal{G}$ is a game graph with *full information*, so $\mathcal{G}$ is a two-player game graph of the form $\mathcal{G} = (V, V_0, \delta)$. Then solving the strategy problem for $\mathcal{G}$ with winning condition given by $\mathcal{A}$ goes like: pick your favorite algorithm for solving parity games and apply it to the parity game graph $(\mathcal{G} \times \mathcal{A}, \mathrm{col}^\times)$.

We have to be careful though, since as with the determinization construction for nondeterministic game graphs in Section 2.1.2, the construction described above yields a game graph with partial information, even if started with a game graph with full information. However, in this case, player 1 can deduce the complete history of states of $\mathcal{A}$ from the history of events in $\mathcal{G}$ which he fully observes. Hence, there is no harm done in making the states of $\mathcal{A}$ visible for player 1 in $\mathcal{G} \times \mathcal{A}$. With this simplification, $\mathcal{G} \times \mathcal{A}$ is a game graph with full information whenever $\mathcal{G}$ is.

Moreover, by the memoryless determinacy of parity games, if player 1 has a winning strategy for $(\mathcal{G} \times \mathcal{A}, \mathrm{col}^\times, (v_0, \delta(q_{\mathrm{in}}, v_0))$ for some initial position $v_0 \in V$ then he has a *memoryless* winning strategy $\sigma_1 : V_1 \times Q \to A$ for $(\mathcal{G} \times \mathcal{A}, \mathrm{col}^\times, (v_0, \delta(q_{\mathrm{in}}, v_0))$ which is provided by the algorithm in

[16] In fact, after some history π has been played, a player i considers a set of states of $\mathcal{A}$ possible, which are exactly those states that are reached when $\mathcal{A}$ reads some history $\rho \sim_i^* \pi$. The same can be inferred by player i in $\mathcal{G} \times \mathcal{A}$. However, no information about the states of $\mathcal{A}$ is given explicitly to the players of the grand coalition.

charge of solving the strategy problem. Notice that $V_1 \times Q$ are the positions in $\mathcal{G} \times \mathcal{A}$ where it it player 1's turn. Now obviously, σ_1 is a *memory* winning strategy σ_1' for player 1 for $(\mathcal{G}, \mathcal{A}, v_0)$ with memory $\mathcal{M} = \mathcal{A}$: We just define $M := Q$, $\delta_{\text{in}}(v) := \delta(q_{\text{in}}, v_0)$ and $\delta_{\text{up}} = \delta$. So implementing a winning strategy for $(\mathcal{G}, \mathcal{A}, v_0)$ is included in strategy synthesis for $(\mathcal{G} \times \mathcal{A}, \text{col}, (v_0, \delta(q_{\text{in}}, v_0))$.

Remark. Notice that the construction does *not* work in general, if $\mathcal{A}$ is a *nondeterministic* automaton. First, in this case, the product game graph $\mathcal{G} \times \mathcal{A}$ would also be nondeterministic. Now, the determinization $\mathcal{G}^d$ as defined in Section 2.1.2 cannot be applied here, since the acceptance condition of $\mathcal{A}$ only requires the *existence* of an accepting run. The solution here is obvious: we give player 1 control over the nondeterministic choices. However, this does not work either since player 1 can make his choices in the game defined by $\mathcal{G} \times \mathcal{A}$ *dependent* on the finite prefix of a run of $\mathcal{A}$ that he has encountered so far which of course increases his capabilities as compared to the game defined by $(\mathcal{G}, \mathcal{A})$.

In [108], a class of *nondeterministic* automata has been provided for which such a product construction can be applied, so called *good for games* automata. Moreover, it has been proved that any Büchi automaton can be transformed into an equivalent good for games parity automaton. Although this transformation also yields an automaton that is, in the worst case, exponentially larger than the original Büchi automaton, it has certain advantages for actual implementations due to its simplicity and the possibility of an incremental approach. Here we shall stick to the product with *deterministic* automata.

Furthermore, we can also easily construct, for a parity game $(\mathcal{G}, \text{col})$, a deterministic parity automaton $\mathcal{A}$, recognizing the parity condition defined by col. So, parity game graphs and game graphs with an ω-regular winning condition defined by a *deterministic parity automaton* are virtually the same.

However, dealing with both these representations does require a careful handling of notions like *memoryless strategies* and parameters such as *complexity of the game graph* and *number of positions*. Moreover, in the context of partial information, some extra care is necessary. On the other hand, if the ω-regular winning condition is not given by a deterministic parity automaton but, say, by a nondeterministic Büchi automaton or an LTL-formula then considerable computational effort has to be put into obtaining such a convenient presentation.

For distributed systems, the situation is essentially the same. Given a distributed system $\mathcal{D}$, an ω-automaton $\mathcal{A}$, defining the specification of the system has now the input alphabet $\Sigma^{\mathcal{D}}$. And, since any sequence $\alpha \in (\Sigma^{\mathcal{D}})^\omega$ constitutes a run of the system, the language $L(\mathcal{A})$ is the specification of the system. Now assume that $\mathcal{A}$ is a deterministic parity automaton. Then we can apply the construction from above and obtain a parity game on a finite game graph with partial information which is, in the case of games with *full information*, basically just the *transition graph* of $\mathcal{A}$ (reminisce about the remark on the original solution of Büchi and Landweber for Church's problem). Moreover, as we have seen above, if player 1 has a winning strategy for this game, then he has a memoryless winning strategy from which we can directly obtain a memory winning strategy with memory $\mathcal{A}$ (that is, a strategy which is positional in the states of the automaton defining the winning condition).

In the case of arbitrary distributed systems, of course, we have to incorporate the partial information of the processes into the game. For this, consider the game graph $\mathcal{G}_{\mathcal{D}}$ with partial information which is obtained by using the translation of a distributed system to a game graph from Section 2.3. Then the product $\mathcal{G}_{\mathcal{D}} \times \mathcal{A}$ as above yields the desired parity game graph with partial information. Notice that the game graph $\mathcal{G}_{\mathcal{D}}$ is a complete graph, so the transition structure of $\mathcal{A}$ will not be restrained by applying this product.

This reduction offers an interesting angle for looking at the controller problem for distributed systems and, especially, Church's problem where we have only a single process with full information. Though, we like to emphasize once more, that for the constructions and proofs that we conduct on the model of distributed systems, the reduction to game graphs will not be of importance. This, in particular, is due to the loss of the explicit communication structure that we encounter when translating distributed systems to game graphs.

2.2.2 Context-free Winning Conditions

The most expressive concrete class of languages that we consider as winning conditions are *context-free languages*. Contextfree languages are those recognizable by parity pushdown automata, that is, parity automata which additionally have access to a stack memory. We use this as a definition and we give only a brief overview over the aspects of context-free languages that are important for us here. For characterizations of this class using, for example, grammars and for an elaborate survey of properties of context-free languages, we refer to [57, 58].

Unlike for finite ω-automata, deterministic parity pushdown automata are strictly less expressive than nondeterministic ones. We call languages accepted by deterministic parity pushdown automata also *deterministic context-free languages* which consequently form a proper subclass of context-free languages. We consider general context-free languages only very briefly since as winning conditions for games, they are too expressive: Games on finite graphs with *full* information are undecidable for context-free languages, which can be proved using undecidability of the *universality problem* for context-free languages [81]. On the other hand, games on finite graphs with full information are decidable for *deterministic* context-free winning conditions [219]. We will study this case thoroughly for games with imperfect information and multiple players.

Pushdown Automata. A *pushdown automaton* has the form

$$\mathcal{P} = (Q, \Sigma, \Gamma, q_{\text{in}}, \delta, \bot, \text{acc})$$

- Q is the finite set of states with initial state q_{in}
- Σ is the input alphabet
- Γ is the stack alphabet with bottom stack symbol $\bot \in \Gamma$
- $\text{acc} \subseteq Q^\omega$ is the acceptance condition

Moreover,

$$\delta : Q \times \Gamma_\bot \times \Sigma_\varepsilon \to 2^{Q \times \Gamma_\bot^{\leq 2}}$$

is the transition function where $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ and, analogously, $\Gamma_\bot = \Gamma \cup \{\bot\}$.

So the transition function takes the current state $q \in Q$, the current top symbol $A \in \Gamma_\bot$ of the stack contents (which may be the bottom symbol $\bot$) and, potentially, the next input symbol $a \in \Sigma$ and yields a set $\delta(q, A, a)$ (or $\delta(q, A, \varepsilon)$) of possible transitions to a next state $q' \in Q$ and a sequence of stack symbols $\gamma \in \Gamma_\bot^{\leq 2}$ of length at most two which is to replace the top stack symbol A. Transitions of the form $\delta(q, A, \varepsilon)$ are called ε-transitions. If performing an ε-transition, $\mathcal{P}$ can change the state and perform a stack operation, but does not read the next letter of the input word. For some stack operation defined by $\gamma \in \Gamma^{\leq 2}$, if $\gamma = \varepsilon$ it deletes the top symbol from the stack and is called pop-operation, if $\gamma = B$, it replaces the top stack symbol A by the symbol B and is called a swap-operation and if $\gamma = AB$, it adds the symbol B on top of the current stack contents and is called a push-operation. Notice that by definition, the stack symbol $\bot$ cannot be written to the stack and we assume that it can neither be deleted from the stack: if $(q', \gamma) \in \delta(q, \bot, a)$ then $\gamma \in \Gamma^{\leq 1}\bot$.

The automaton $\mathcal{P}$ is called *deterministic* if, for all $(q, A, a) \in Q \times \Gamma_\bot \times \Sigma$ we have $|\delta(q, A, a)| + |\delta(q, A, \varepsilon| \leq 1$. The automaton is called a 1-counter automaton if $|\Gamma| = 1$. The automaton is called a realtime automaton if it contains no ε-transitions, that is, the transition function has the form $\delta : Q \times \Gamma_\bot \times \Sigma \to 2^{Q \times \Gamma_\bot^{\leq 2}}$.

A *configuration* of $\mathcal{P}$ is a tuple $C = (q, \gamma) \in Q \times \Gamma^*\bot$. The transition relation $\vdash$ between configurations of $\mathcal{P}$ is defined in terms of the transition function of $\mathcal{P}$: for $a \in \Sigma_\varepsilon$ we have $(q, A\gamma) \overset{a}{\vdash} (q', \gamma'\gamma)$ if $(q', \gamma') \in \delta(q, A, a)$. A *run* of $\mathcal{P}$ on $\alpha \in \Sigma^\omega$ is a sequence $\rho = C_0C_1\ldots$ of configurations of $\mathcal{P}$ such that:

- C_0 is the initial configuration $(q_{\text{in}}, \perp)$
- for all $j \in \mathbb{N}$ we have $C_j \overset{\beta_j}{\longmapsto} C_{j+1}$

where either $\beta = \beta_0\beta_1 \ldots = \alpha$ or β is some finite prefix of α. Notice that the latter case corresponds to a run which ends with an infinite sequence of ε-transitions. The run is called *complete*, if $\beta = \alpha$.

The run ρ is *accepting*, if it is complete and $\mathrm{Pr}_Q(\rho) \in \mathrm{acc}$. Notice that acceptance depends only on the states and not on the stack contents. $\mathcal{P}$ *accepts* a word $\alpha \in \Sigma^\omega$ if there is an accepting run of $\mathcal{P}$ on α and the language recognized by $\mathcal{P}$ is $L(\mathcal{P}) = \{\alpha \in \Sigma^\omega \mid \alpha \text{ accepts } \alpha\}$. We consider here only parity pushdown automata, that is, the automaton comes with a coloring $\mathrm{col} : Q \to [k]$ for some $k \in \mathbb{N}$ and acc is defined via the parity condition given by col: An infinite state sequence $\zeta \in Q^\omega$ is in acc if $\min \mathrm{colInf}(\zeta)$ is even.

A language $L \subseteq \Sigma^\omega$ is called *(deterministic) context-free* (D)CF, if there is a (deterministic) parity pushdown automaton $\mathcal{P}$ with $L(\mathcal{P}) = L$. A language is called *(deterministic) realtime* context-free (D)RCF, if it can be recognized by a (deterministic) realtime pushdown automaton. The classes of *(deterministic) (realtime) 1-counter* (D)(R)1-C languages is defined analogously.

We have already mentioned that the class of context-free languages is strictly larger than the class of deterministic context-free languages. Moreover, while the class of realtime context-free languages is equal to the the class of context-free languages, the class of deterministic realtime context-free languages is strictly included in the class of deterministic context-free languages. In particular, ε-transitions cannot be removed from deterministic parity pushdown automata in general. The aforementioned inclusions as well as some other well-known (strict) inclusions of classes of context-free languages can be summarized as follows:

- DR1-C $\subsetneq$ DRCF $\subsetneq$ DCF $\subsetneq$ RCF $=$ CF
- DR1-C $\subsetneq$ D1-C $\subsetneq$ DCF

Continuity Property. A pushdown automaton $\mathcal{P}$ is said to have the *continuity property* if each run of $\mathcal{P}$ on some input word is complete. Although ε-transitions cannot be removed from deterministic parity pushdown automata, *infinite* ε-runs *can* be removed. That means, for any deterministic parity pushdown automaton $\mathcal{P}$ there exists an equivalent deterministic parity pushdown automaton $\mathcal{P}'$ such that for each input word α, the unique run of $\mathcal{P}'$ on α is complete, that is, $\mathcal{P}'$ has the continuity property [59]. Hence, w.l.o.g. we consider only deterministic parity pushdown automata which have the continuity property.

Games with Contextfree Winning Conditions. Consider a game graph $\mathcal{G}$ with partial information and a context-free winning condition W, that is, W is defined by a parity pushdown automaton $\mathcal{A}$, cf. Section 2.2.1. Now if $\mathcal{A}$ is *deterministic*, the product $\mathcal{G} \times \mathcal{A}$ can be constructed in the same way as described in Section 2.2.1. The result is an *infinite* parity game graph with partial information and in this case, not only the state of the automaton but also the *current contents of the stack* has to be hidden from the players of the grand coalition. As before, for game graphs $\mathcal{G} = (V, \delta)$ with full information, a simplified version of the construction yields a game graph $\mathcal{G} \times \mathcal{A}$ with full information where the players do observe the state and the stack contents. The corresponding parity game with full information is memoryless determined.

However, this construction can *not* be used to infer decidability of the strategy problem and constructibility of winning strategies as we could easily do for the case of ω-regular winning conditions. The reason is twofold. First, the game graph $\mathcal{G} \times \mathcal{A}$ is *infinite*, so the usual algorithms for solving parity games cannot be applied. Moreover, a positional strategy for player 1 for $(\mathcal{G} \times \mathcal{A}, \mathrm{col})$ yields a pushdown strategy which depends on the *complete stack contents.* This is of course undesirable since it is still an infinite representation of the strategy: too much information is needed to implement the strategy in this way.

The first solution for games on finite graphs with deterministic context-free winning conditions has been given in [219] where Walukiewicz proved that the strategy problem for parity games on configuration graphs of (possibly nondeterministic!) pushdown automata (where the parity condition depends only on the state of the automaton) is decidable (in exponential time) and, if player 1 has a winning strategy then he has a pushdown strategy which can be constructed effectively. Where pushdown strategy means now a strategy that is implemented by a deterministic pushdown automaton and which depends only on the state of the automaton and the *top stack symbol*. Notice, however, that this does not imply that the strategy problem for games on finite graphs with arbitrary context-free specifications is decidable (we have already mentioned that, in fact, it is undecidable), since for a nondeterministic pushdown automaton the product construction $\mathcal{G} \times \mathcal{A}$ is not sound. (The reason is exactly the same as for nondeterministic ω-automata, see Section 2.2.1.) An alternative solution has been given in [128] using the alternating two-way tree automata of Vardi [213]. Refinements of this solution for various subclasses of deterministic context-free languages have been studies in [85], see also [178].

2.2.3 Observability

In Section 2.2.1 and Section 2.2.2 we have considered restrictions of the winning conditions with respect to *expressive power*, that is, we have restricted which classes of sets W of plays we consider in principle as winning conditions. Notice that these classes are defined independently of a concrete game graph and, in particular, independently of the *information* of the players. On the contrary, in this section, we consider restrictions of winning conditions with respect to the information of the players, that is, we restrict the extent to which a winning condition may involve facts that some or all of the players do not observe.

A winning condition $W \subseteq V(AV)^\omega$ for $\mathcal{G}$ is called *observable* for player $i \in \{1, \dots, n\}$ if it depends only on the observations that player i makes about the positions in the game graph: If $\pi = v_0 a_1 v_1 \dots$ and $\rho = w_0 b_1 w_1 \dots$ in $\mathcal{G}$ with $v_j \sim_i^V w_j$ for all j then

$$\pi \in W \iff \rho \in W.$$

Such a winning condition can also also be represented as a set $W \subseteq [V]_i([A]_i[V]_i)^\omega$. A winning condition W is called *observably decomposable*, if there are winning conditions W_i, $i = 1, \dots, n$ such that W_i is observable for player i and $W = \bigcap_i W_i$. A winning condition W is called *observable* if it is observable for all players $i = 1, \dots, n$ of the grand coalition.

Remark. In general, a less restrictive notion of observable winning condition would be to require, that the winning condition may depend on the information about the position and the information about the actions which a player observes. This would give a more direct reflection of the actual observability in the game. However, as we have already mentioned in Section 2.3, although in some of our construction we have to deal with wining conditions that may depend on the actions in a play as well, we are particularly interested in winning conditions which do not involve the actions in a play a priori.

Notice that a winning condition that only depends on the observations that a player i makes about the position and actions in the game and which is position based as well, is not necessarily observable for player i in the sense defined above, that means, it is not necessarily completely determined by the observations that player i makes about the positions in the game. Consider for example the game $\mathcal{G} = (V, \delta, \sim^V, \sim^A)$ with

- $V = \{v_0, v_1, v_2, v_3\}$,
- $\delta(v_0, a_i) = v_i$ for $i = 1, 2, 3$
- $a_1 \sim^A a_2$ and $v_1 \sim^V v_2 \sim^V v_3$

Moreover, let $W = \{v_0 a_1 v_1, v_0 a_2 v_2\}$. Then clearly, W is completely determined by the positions that occur in a play and at the same time, W is also completely determined by the observations that player 1 makes about the positions and actions in a play. However, in $v_0 a_2 v_2$ and $v_0 a_3 v_3$, player 1 makes the same observations about the positions but $v_0 v_2 v_2 \in W$ while $v_0 a_3 v_3 \notin W$.

So W is not observable for player 1 in the sense defined above.

In the context of architectures, a winning condition which is observable for a controller p is usually called a *local specification of process* p and is represented as a language $L \subseteq (\Sigma^p)^\omega$ consisting of those local runs which process p wants to ensure. A specification for the whole system $L_p^{\mathcal{D}} \subseteq (\Sigma^{\mathcal{D}})^\omega$ which is local for controller p_i is then given, symbolically, as

$$L_p^{\mathcal{D}} = L_p \times (\Sigma^{\mathcal{D}} \setminus \Sigma^{p_i})^\omega.$$

That means, $L_p^{\mathcal{D}}$ contains exactly those system runs $\alpha \in (\Sigma^{\mathcal{D}})^\omega$ such that $\mathrm{Pr}_{\Sigma^p}(\alpha) \in L_p$. A specification $L \subseteq (\Sigma^{\mathcal{D}})^\omega$ is *locally decomposable*, if there are local specifications $L_p \subseteq (\Sigma^p)^\omega$ for the controllers $p \in P_{\mathrm{con}}$ such that

$$L = \bigcap_p L_p^{\mathcal{D}}.$$

That means, L consists of exactly those system runs $\alpha \in (\Sigma^{\mathcal{D}})^\omega$ such that, for all controllers $p \in P_{\mathrm{con}}$ we have $\mathrm{Pr}_{\Sigma^p}(\alpha) \in L_p$. A system specification L is called local, if it is local for each controller.

Notice that in the definition of local decomposability we can always choose

$$L_p = \mathrm{Pr}_{\Sigma^p}(L)$$

for each $p \in P_{\mathrm{con}}$. Conversely, if we define a collection $(L_{p_1}, \ldots, L_{p_n})$ of local specification for the controllers from an *arbitrary* system specification L by simply setting $L_p := \mathrm{Pr}_{\Sigma^p}(L)$ for $p \in P_{\mathrm{con}}$ then $L(L_{p_1}, \ldots, L_{p_n}) := \bigcap_p L_p^{\mathcal{D}}$ is a locally decomposable by definition. However, in general we have

$$L(L_{p_1}, \ldots, L_{p_n}) \supsetneq L.$$

The same is true, of course, for winning conditions for game graphs as well.

Also notice that a joint strategy $\sigma = (\sigma_{p_1}, \ldots, \sigma_{p_n})$ which is composed of *local* winning strategies, *is* a winning strategy for $(\mathcal{D}, L)$, if L is locally decomposable (contrary to the general case of arbitrary L, see Section 2.1.3). This is the key feature for the easier decidability of distributed systems with locally decomposable specifications that we shall explore in Chapter 5.

The converse, however, is still *not* true: If $\sigma = (\sigma_{p_1}, \ldots, \sigma_{p_n})$ is a joint winning strategy for the grand coalition for a $(\mathcal{D}, L)$, where L is locally decomposable, then an individual strategy σ_p is *not necessarily locally winning* as the input sequences that p may receive from other controllers q in the system runs which are constrained by σ, are constrained by their individual strategies σ_q. This is the reason why the problem does not reduce to solving n individual single controller problems for the local specifications L_{p_i}, $i = 1, \ldots, n$. In fact, the controller problem for locally decomposable regular specifications still becomes undecidable quickly, if the architectures become a little more complex, see Chapter 5.

Remark. A very important observation about observability concerns the difference between observable winning conditions in games on graphs and local specifications in distributed systems. In our model of distributed systems where we do not have broadcast channels, a local system specification L is always trivial, that is, $L = \emptyset$ or $L = (\Sigma^{\mathcal{D}})^\omega$, as soon as we have at least two controllers $p \neq p'$ in the system. This is clear, since if a system specification is completely determined by the local run of p and at the same time it is completely determined by the local run of p' and those two local runs share no common portion, then L is trivial. On the other hand, in games on graphs, the local views of two different players may share a common portion (so in a sense, in games on graphs we do have the feature of broadcasting), hence observable winning conditions are not trivial for multiple players. In fact, the strategy problem for three-player games on finite graphs with observable safety conditions is undecidable, see Chapter 6.

Moreover, for distributed systems with only a single controller and local specifications, the controller problem directly reduces to Church's problem: The only difference are the possible hidden channels from the environment, but since a local system specification does not involve these channels at all, we can simply ignore them. Again, for games on graphs, the situation is quite different. While two-player reachability games on finite graphs with full information can be solved in polynomial time, two-player games on finite graphs with partial information and with *observable* reachability conditions are EXPTIME-hard.

Intuitively, the reason is that the additional structure provided by the *game graph* has to be incorporated in the strategy for player 1, while the course of event through the game graph is *not observable* for player 1 unless the game is one of full information. So the player has to *keep track* of the possible histories of events that he considers possible during a play of the game. Similarly, in a distributed system, a controller has to keep track of the possible histories of transitions through an automaton defining the winning condition that he considers possible – this, however, is only nontrivial in case the winning condition involves facts that the controller cannot observe. In a game graph, this is clearly the case due to the equivalence relations $\sim^V$ and $\sim^A$ which are defined explicitly on the positions and actions of the graph. So in a game on a graph, the specification has always a non-observable part which is the game graph itself.

Also for games with multiple players, the game graph provides the structure that makes the strategy problem for observable winning conditions highly undecidable. It is an inherently *global* structure (a transition in the game graph concerns all the players as their local view is determined by it) and the players have to keep track of their uncertainties about the events in the game graph as well as the uncertainties of the other players. We make this more precise in Section 2.3 where we translate games on graphs into architectures and we shall see that local decomposability of winning conditions is not preserved by this construction. (In fact, there cannot be a reasonable translation of game graphs into architectures which preserves local decomposability of winning conditions since while any distributed system with at most two controllers is decidable for locally decomposable regular winning conditions, see Chapter 5, three player games on graphs are undecidable even for observable safety conditions.)

The idea to *resolve* these uncertainties of the players about the course of events in the game graph and to represent them explicitly leads to the knowledge tracking constructions that we present in Section 3.5 and in Chapter 6.

Observable Parity Conditions. We are particularly interested in parity games with imperfect information and observable winning condition. That means, we consider parity games $(\mathcal{G}, \mathrm{col})$ such that the winning condition W which is defined by col is observable. However, in such cases where the parity condition is given explicitly by a coloring of the game graph, we would like to have a formulation of observability just in terms of the coloring function: we call the coloring col observable for player i if for all $v, w \in V$ with $v \sim_i^V w$ we have $\mathrm{col}_i(v) = \mathrm{col}_i(w)$. The coloring is called observable, if it observable for all players $i = 1, \dots, n$. Notice that the observability of the parity condition W defined by col does not imply the observability of col. Nevertheless, we only consider observable parity conditions given by an observable coloring.

An interesting observation about winning conditions defined by a coloring of a game graph $\mathcal{G}$ with partial information is that for *reachability* as well as *safety* winning conditions we can always assume that they are observable. More precisely, given a game graph $\mathcal{G}$ with partial information and a set $R \subseteq V$ defining a reachability condition on $\mathcal{G}$ we can construct (in linear time) a game graph $\mathcal{G}'$ with $V' = V$ such that R is observable on $\mathcal{G}^o$ and such that, for any $v_0 \in V$, the grand coalition has a winning strategy for $(\mathcal{G}, R, v_0)$ if and only if the grand coalition has a winning strategy for $(\mathcal{G}^o, R, v_0)$.

The construction is really quite trivial, we just redefine $\sim_i^V$ to $\sim_i^{V,o}$ for any player $i = 1, \dots, n$ so that $v \not\sim_i^{V,o} r$ for all $r \in R$ and all $v \in V$ (and $u \sim_i^{V,0} v$ iff $u \sim_i^V v$ for all $u, v \in V \setminus R$). Notice that any player i can define his strategy on a history $\pi = v_0 a_1 v_1 \dots a_l v_l$ such that $v_j \in R$ for some j *arbitrarily*, since the play is already won by the grand coalition. Hence, we can assume just as well that any such history is distinguishable for player i from any other history.

Intuitively, the (nonobservable) structure of the game graph (which is really the core of

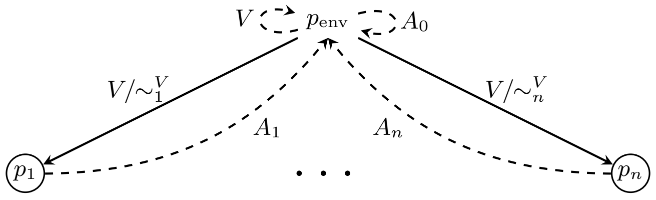


Figure 2.3: Distributed system defined from a game graph with partial information

a reachability condition) comprises the complexity of the nonobservability of the reachability condition. The same construction can be applied to safety conditions on $\mathcal{G}$, given by a set $S \subseteq V$. Then we apply the construction as above with the set $R := V \setminus S$. On the other hand, for Büchi conditions this construction does not work anymore. Moreover, if we have a game graph with partial information and a winning condition that is given by a deterministic reachability (or safety) *automaton*, then this construction cannot be applied directly as well, neither to the game graph nor to the automaton. We have to take the *product* of the game graph and the automaton first, cf. Section 2.2.1.

2.3 Game Graphs vs. Distributed Systems

We now turn to the task of comparing game graphs and distributed systems. We will see that, in principle, it is not hard to translate game graphs with partial information into distributed systems and vice versa. On the other hand, the representation of the elements of interactive situations like *possible moves*, *uncertainties* and *communication* is quite different in the two models. In a sense, different parameters are implemented at different layers of the models and become more visible when we translate one model into the other. This, however, also requires extensive usage of certain resources like graph structure or communication channels. In particular, our translation of a game graph into a distributed system also involves the *specification* of the distributed system, even if we only want to translate the game graph, regardless of a winning condition. The reason is that the *game graph* already specifies certain behaviors of the system, see also the remarks in Section 2.2.3. We will discuss this more concretely after we have seen how the translations work.

Game Graphs to Distributed Systems. Let $\mathcal{G} = (V, \delta, (\sim_i), W)$ be a game graph with partial information and $n+1$ players. To streamline the presentation we consider only the case where the actions are completely invisible for all players, that is, $\sim_i^A = A \times A$ for all $i \in \{1, \ldots, n\}$ and we also restrict to position based winning condition $W \subseteq V^\omega$. It is easy to see how the construction can be adapted to the slightly more general case without these restrictions but it makes the notation less readable.

We define the distributed system $\mathcal{D} = \mathcal{D}(\mathcal{G}) = (\mathfrak{A}, (\Sigma_c)_{c \in C})$ with $n+1$ processes and $\mathfrak{A} = (C, r, w)$ as follows. A schematic presentation of $\mathcal{D}$ is shown in Figure 2.3.

- $C = \{c_0\} \cup \{c_V\} \cup \{c_{0i} \mid i = 1, \ldots, n\} \cup \{c_i \mid i = 1, \ldots, n\}$
- $r(c_0) = w(c_0) = p_0$ and $\Sigma_{c_0} = A_0$
- $r(c_V) = w(c_V) = p_0$ and $\Sigma_{c_V} = V$
- $w(c_{0i}) = p_0$, $r(c_{0i}) = p_i$ and $\Sigma_{c_{0i}} = \{[v]_{\sim_i^V} \mid v \in V\}$ for $i = 1, \ldots, n$
- $w(c_i) = p_i$, $r(c_i) = p_0$ and $\Sigma_{c_i} = A_i$ for $i = 1, \ldots, n$.

Now consider the set $S \subseteq (\Sigma^{\mathcal{D}})^\omega$ consisting of exactly those sequences α which *violate one of* the following conditions for *some* $j \in \mathbb{N}$:

(G1) $\alpha(j) = (a_0, v, [v]_1, \dots, [v]_n, a_1, \dots, a_n)$

(G2) if $\alpha(j) = (a_0, v, [v]_1, \dots, [v]_n, a_1, \dots, a_n)$ then $v = \delta(v', (a_0, a_1, \dots, a_n))$ where $v' = v_0$ if $j = 0$ and $v' = \mathrm{Pr}_V(\alpha(j-1))$ if $j > 0$

Moreover, from a winning condition $W \subseteq V^\omega$ for $\mathcal{G}$ we obtain a specification $L(W) = \{\alpha \in (\Sigma^{\mathcal{D}})^\omega \mid v_0\mathrm{Pr}_V(\alpha) \in W\}$ for $\mathcal{D}(\mathcal{G})$.

The idea behind this construction is the following. In the game played on the graph $\mathcal{G}$, in each step, being in position v', each player i chooses an action a_i and the next position is $v = \delta(v', (a_0, a_1, \dots, a_n))$. So choosing the action a_i is a *local* move of player i but the transition from v' to v is a *global* move which somehow connects all the moves of the players. In a distributed system, in each step, each process p_i chooses an action b_i which yields a joint action $b = (b_0, b_1, \dots, b_n)$, but b does not induce a global transition per se. Global transitions come into play only when we consider a *specification* for the system: consider a regular specification $L \subseteq (\Sigma^{\mathcal{D}})^\omega$ and a deterministic parity automaton $\mathcal{A}$ with $L(\mathcal{A}) = L$. Then a joint action b triggers a transition of $\mathcal{A}$ from the current state q' to the state $q = \delta^{\mathcal{A}}(q', b)$ which is a *global* transition that connects the individual moves of the processes. Hence, we encode the global transition structure provided by the game graph into the specification of the system.

Moreover, the game graph does not only define the possible moves in the game but also the *information* that the players receive during each move of the game. This information, however, is not represented in terms of signals that the players send to each other as in distributed systems but by means of a certain *visible portion* of the current (global) position of the game: the equivalence class $[v]_i$. To simulate the global transition structure provided by the graph and the way in which the information of the players is based on the states of this structure we proceed as follows: The *environment* (which has full information!) chooses not only its own local action a_0 but also serves as a *provider* for the *transition structure* by choosing the global position v of the game and for the *information* of the players by sending to each player i the information $[v]_i$ that this player has about the position v.

The global state is chosen by p_{env} privately by sending it to the channel c_V and the information $[v]_i$ is sent to process p_i through channel c_{01}. Moreover, any controller p_i chooses its actions a_i privately by sending them to the external output channel c_i. To make sure that the system runs of $\mathcal{D}$ in fact correspond to plays in $\mathcal{G}$ we have to require two properties: First, p_{env} must send the correct equivalence class $[v]_i$ to c_{0i} whenever it sends position v to c_V. Second, v must be consistent with the joint action $(a_0, a_1, \dots, a_n)$ and the *previous* position (which, in the first move, is some initial position v_0). This is ensured by the conditions (G1) and (G2): Any play that violates (G1) or (G2) for some $j \in \mathbb{N}$ is won by the controllers.

Notice that in order to be able to satisfy the graph specification S, the environment has to forecast, in each step, the actions of all the controllers. Hence, winning strategies for p_{env} are obviously not necessarily preserved by the construction. On the other hand, since a winning strategy for the grand coalition has to be winning against all possible behaviors of the environment, it has to be winning in particular against those behaviors of the environment where it forecasts these actions correctly. Using these insights it is not hard to see that winning strategies for the grand coalition are preserved:

Proposition 2.10. *For any winning condition $W \subseteq V^\omega$, the grand coalition has a joint winning strategy for $(\mathcal{G}, W, v_0)$ if, and only if, the grand coalition has a joint winning strategy for $(\mathcal{D}(\mathcal{G}), L(W) \cup S)$.*

Notice that $\mathcal{D}(\mathcal{G})$ can be constructed from $\mathcal{G}$ in polynomial time. Moreover, if W is regular then so is $L(W) \cup S$. In fact, S can be recognized by a *deterministic reachability automaton* $\mathcal{A}$ with state set V and the following transitions (being in state u and reading letter $(a_0, v, [w_1]_1, \dots, [w_1]_n, a_1, \dots, a_n)$):

- $u \to v$ if $[w_i]_i = [v]$ for all i and $\delta(u, (a_0, a_1, \dots, a_n)) = v$
- $u \to \top$, else.

The only accepting state of $\mathcal{A}$ is $\top$. Essentially, the transition structure of $\mathcal{A}$ is just the graph structure of $\mathcal{G}$.

Moreover, if W is given by a deterministic ω-automaton $\mathcal{B}$, the construction of a deterministic parity automaton $\mathcal{B}'$ recognizing $L(W) \cup S$ is particularly easy as the language $\mathcal{B}$ can be easily relativized to the language of a deterministic reachability automaton $\mathcal{A}$. For this, we just run the two automata in parallel and we let the new automaton go into a distinct accepting state (colored with 0 and having only selfloops as transitions) as soon as an accepting state of $\mathcal{A}$ is seen. If we don't encounter an accepting state of $\mathcal{A}$, then the run is accepting if the run of $\mathcal{B}$ is accepting.

By construction, the architecture $\mathfrak{A}$ always has information forks, independent of the game graph $\mathcal{G}$. In fact, all the processes $p_1, \dots, p_n$ are pairwise incomparably informed. Hence, for arbitrary regular specifications the architecture $\mathfrak{A}$ is undecidable. On the other hand, this does not imply the undecidability of the following problem: given a winning condition W for $\mathcal{G}$, does the grand coalition have a winning strategy for $(\mathcal{D}(\mathcal{G}), L(W) \cup S)$? Dependent on the game graph $\mathcal{G}$, S may be a set that makes the specification $L(W) \cup S$ very simple. However, automata based methods for solving controller problems for distributed systems cannot take advantage of this possibility since they always rely on a fixed ordering of the processes with respect to informdness [171, 129, 80].

Therefore, so far there is no way known of getting even started with automata based methods, even if $L(W) \cup S$ might be trivial. The method presented in Chapter 6 works on game graphs and, although the algorithm does not terminate on all instances, it takes arbitrary instances as input, independent of the informdness of players, and it can take advantage of simplicity of the game graphs.

Clearly, in general it is unavoidable to have information forks in $\mathcal{D}(\mathcal{G})$ because otherwise we could translate undecidable instances of the strategy problem for game graphs with partial information into decidable instances of the controller problem for distributed systems. Intuitively this is also clear since if, in the game graph $\mathcal{G}$, two players have incomparable information about the positions of the game graph, this information has to be brought somehow to the processes in $\mathcal{D}(\mathcal{G})$ from some process which serves as a provider for the global position v which, in general, can only be the environment.

However, if in the game graph a player i is *better* informed than player j, that is, $\sim_i \subseteq \sim_j$, then the information $[v]_j$ has not to be brought to p_j directly by p_{env} but can be channeled through p_i: the process p_i receives $[v]_i$ and sends $[v]_j$ to p_j. Using this refined construction, the (partial) ordering of the processes with respect to informdness in $\mathcal{G}$ is preserved and, in particular, game graphs with hierarchical information, that means $\sim_1 \subseteq \ldots \subseteq \sim_n$, can be translated into distributed systems with pipeline architectures.

In Section 2.2.3 we have already hinted at the fact that locally decomposable specification can, in general, not be preserved by a translation from game graphs to distributed systems since any distributed system with at most three processes is decidable for locally decomposable regular specifications while games on graphs are undecidable for three players and observable reachability winning conditions. In particular, $L(W) \cup S$ cannot be locally decomposable in general, even if W is observable. Intuitively, the reason is the global nature of the transition structure provided by the game graph which has been revealed very distinctly in the above construction. In fact, S itself is not locally decomposable in general.

Distributed Systems to Game Graphs. Let $\mathcal{D} = (\mathfrak{A}, (\Sigma_c)_{c \in C})$ be a distributed system with $n+1$ processes and architecture $\mathfrak{A} = (C, r, w)$. We define the game graph $\mathcal{G} = \mathcal{G}(\mathcal{D}) = (V, \delta, (\sim_i)_i)$ with partial information and $n+1$ players as follows. Notice that the actions are completely invisible for all players.

- $V = \{v_0\} \uplus \Sigma^{\mathcal{D}}$ and $A_i = \Sigma_{\text{out}}^{p_i}$ for $i \in [n+1]$
- $\delta(v, (a_0, a_1, \dots, a_n)) = (a_0, a_1, \dots, a_n)$ for all $v \in V$

- $v \sim_i w$ if $\mathrm{Pr}_{\Sigma^{p_i}_{\mathrm{in}}}(v) = \mathrm{Pr}_{\Sigma^{p_i}_{\mathrm{in}}}(w)$

Moreover, from a specification $L \subseteq (\Sigma^{\mathcal{D}})^\omega$ we obtain a winning condition $W(L) \subseteq V^\omega$ for $\mathcal{G}$ very directly: $W(L) = \{v_0\alpha \mid \alpha \in L\}$.

Notice that the translation from a distributed system to a game graph with partial information is much simpler than the converse construction which is of course again related to the game graph: a distributed system does not provide any a priori structure on the transitions in system runs. In fact, the actions on the edges of (V, δ) are completely determined by the positions and the corresponding unlabeled graph (V, E) is a *complete* graph.

On the other hand, if L is regular then we can take the product of $\mathcal{G}(\mathcal{D})$ and a deterministic parity automaton $\mathcal{A}$ recognizing L as described in Section 2.2.1 to obtain a game on a potentially more restricted graph. In particular, if we translate a game graph into a distributed system and then apply the converse translation, by taking the product $\mathcal{G}(\mathcal{D}) \times \mathcal{A}$, intuitively, we can (partially) restore the original graph structure.

It is easy to see, that this construction in fact preserves winning strategies for the grand coalition.

Proposition 2.11. *For any specification L, the grand coalition has a winning strategy for $(\mathcal{D}, L)$ if, and only if, the grand coalition has a winning strategy for $(\mathcal{G}(\mathcal{D}), W(L), v_0)$.*

Again, $\mathcal{G}(\mathcal{D})$ can be constructed in polynomial time from $\mathcal{D}$.

Just as before, the informdness of the processes is not necessarily preserved by the construction. In particular, distributed systems with pipeline architectures are *not* translated into hierarchical games. To see the reason for this, consider some pipeline $\mathcal{D}_p$ with at least two controllers p_1 and p_2. Controller p_1 is better informed than p_2, in fact, any information that is sent from the environment to some controller, is sent to p_1. On the other hand, p_1 cannot *observe* the actions chosen by p_2 so in the game graph $\mathcal{G}(\mathcal{D}_p)$, player 1 and player 2 are incomparably informed. (The actions chosen by p_2 are invisible for player 1 in $\mathcal{G}(\mathcal{D}_p)$ and hence, since clearly $\sim_2 \not\subseteq \sim_1$, the game $\mathcal{G}(\mathcal{D}_p)$ is not hierarchical.) Nevertheless, as we have demonstrated in Section 2.1.3, once p_1 and p_2 have committed themselves to a joint strategy, p_1 can *deduce* the actions of p_2 since they are uniquely determined by its own actions and the strategy that p_2 uses.

In order to preserve the informdness of the processes we have to modify the construction so that, whenever process p is better informed than process p', we make the actions of p' *observable* for the player corresponding to p in $\mathcal{G}(\mathcal{D})$ *explicitly.* Also notice that, in the representation of external output channels that we have chosen here, a player does not even observe his *own* actions, so we also have to make, for each process p, the actions of p observable for the corresponding player in $\mathcal{G}(\mathcal{D})$.

Remark. As we have mentioned, the translation of game graphs with partial information into distributed systems can be easily adapted to the slightly more general case of game graphs which have arbitrary observabilities for the players of the grand coalition on both, the positions and the actions and where also not necessarily position based winning conditions are allowed. On the other hand, for the converse direction, the general case offers a different possibility to translate distributed systems into game graphs with partial information: We could then let the game graph $\mathcal{G}(\mathcal{D})$ have only a single position $\{v\} = V$ and the same set of actions as before but with appropriately adjusted observabilities. Of course, in this case the winning condition has to depend highly on the actions.

Chapter 3

Solving Interaction under Partial Information

In this chapter, we review fundamental methods for solving interaction under partial information or, more precisely, strategy problems for games on finite graphs with partial information and distributed systems.

In Section 3.1 we introduce and discuss the essential concept of alternating automata on infinite trees which is at the heart of many powerful and elegant solutions for synthesis under partial information. Section 3.1.1 is devoted to the definitions and basic results that are important for wielding tree automata safely, like closure properties and decidability of the emptiness problem. In Section 3.1.2 we demonstrate how alternation can be removed from parity tree automata which is among the core issues for the solution methods that we present in Section 3.2 as well as the proofs of our results in Chapter 5. We also introduce and discuss pushdown tree automata (Section 3.1.3) that we shall use in Chapter 5 as well. While alternation cannot be removed from such automata in general, the most central aspect here is that the emptiness problem is still decidable for nondeterministic automata.

The presentation in Section 3.2 ranges from the groundbreaking works of Rabin [177] and Pnueli and Rosner [169, 171] on synthesis for single controller systems and pipelines to the full characterization of all decidable architectures in [80]. Section 3.2.1 focuses on the significance of the concept of alternation for dealing with partial information in synthesis. Moreover, in Section 3.2.2, we accentuate the connection between partial information and the concept of alternation in tree automata by showing that games with imperfect information are the semantic games for alternating tree automata in a very natural way. We also present some results on the computational complexity of strategy problems in Section 3.3.

In Section 3.4 we prove a very tight undecidability result for strategy problems under partial information with deterministic context-free winning condition. This sharpens and extends a result by Peterson and Reif [163] and serves as a preparation for the exposition in Chapter 5. While the result from Section 3.4 shows that there are no non-trivial decidable subcases of the strategy problem for distributed systems with arbitrary deterministic context-free specifications, in Chapter 5 we shall see that this is not true for locally decomposable specifications.

Finally, we present the powerset construction for two-player games on finite graphs with partial information. This method has originally been used by Reif in [181] for reachability games. We present the basic method and we show how it can be extended to more general classes of (two-player) games. We also discuss its connection to the concept of knowledge tracking. In Section 3.5.2 we demonstrate how finite state implementations of strategies can be obtained from the powerset construction and we also prove a matching lower bound on the number of states.

In Section 3.5.3 we discuss an optimized version of the powerset construction for observable parity conditions that has been developed in [66, 52, 22] and which shows a substantially improved performance in benchmarks. However, its worst case running time is exponential and it seems hard to find large classes of game graphs on which the method runs in polynomial time. The powerset construction also forms the basis for a solution method presented in Chapter 4 which, in contrast, yields a positive polynomial time upper bound for game graphs with bounded complexity (with respect to appropriate measures) but under the *additional* restriction of bounded partial information. Moreover, the powerset construction is a preparation for the construction in Chapter 6.

We conclude the exposition of the powerset construction and related concepts with an extension to the case of imperfect information defined via asynchronous observability as introduced in Section 2.1.4.

3.1 Trees and Tree Automata

Infinite games with with imperfect, as well as with perfect, information, are intimately linked to *infinite trees.* The notion of a tree itself is among the most fundamental concepts in computer science. Depending on the application at hand, the definition of a tree may vary considerably: trees may be finite or infinite, have bounded or unbounded branching degree, may be labeled with elements from finite alphabets or from infinite data domains and so on.

We use trees to represent sets of possible infinite runs of nonterminating systems where, in each step, some event is performed which is composed of moves of the individual components of the system, each of which is chosen from a predefined, finite set of actions. The tree of all such runs is then an infinite tree with a *maximum finite* branching degree which represents all possible behaviors of the system and the game played on the tree captures the interaction of the environment and the controllers of the system. A *strategy* for some player in this game is a *labeling of the game tree* with actions from the *finite* set of actions of this player which, of course, has to fulfill certain conditions, see Section 2.1.1.

Now obviously, there is nothing substantial about the representation of a strategy σ for player i as a labeled tree $t : T \to A_i$: it is just another name for the same (infinite!) object. But it is the possibility to use *finite automata* on infinite trees to recognize *sets* of trees that represent strategies which makes this view so appealing. In this section we introduce the relevant notions like trees and tree automata. We also present some constructions and fundamental results which support the usage of tree automata for solving games with imperfect information. Elaborate surveys can be found, for example, in [200, 203, 100]. As an expedient compilation of surveys and recent research papers we recommend [83].

As mentioned before, to represent strategies in our setting, trees will have a maximum finite branching degree and labels from finite alphabets: For a set X, an *X-tree* is a prefix-closed set $T \subseteq X^*$. T is called *full* if $T = X^*$. The set X will be finite most of the time. For a finite alphabet Σ, a Σ-labeled X-tree is a function $t : T \to \Sigma$. The tree t is called full, if T is full. The set of all (full) Σ-labeled X-trees is denoted X_Σ. Unless explicitly stated otherwise we consider only full trees (and ignore certain branches of the tree that we are not interested in). Notice that regular tree languages (cf. Section 3.1.1) can easily be relativized to full trees since a tree automaton can check in a deterministic fashion that a given tree is full. However, we will use tree that are not full and where the set X is infinite, for example, when we consider *subtrees* and to define the notion of a *run* of an alternating automaton on a given tree.

A *path* through T is an ω-word $\alpha \in X^\omega$ such that, for all $j \in \mathbb{N}$ we have $\alpha_{<j} \in T$ (notice that for a full tree this is trivial). By $t(\alpha) \in \Sigma^\omega$ we denote the labeling of α, that is, $t(\alpha)_j = t(\alpha_{<j})$. Since we usually do not refer to the tree T explicitly, we allow ourselves to be a bit sloppy about this notation. We sometimes call a path α through T a path through t, but we also refer to the ω-word $t(\alpha) \in \Sigma^\omega$ as a path through t. It will be clear from the context whether we refer to ω-word from X^ω or from Σ^ω.

Now, for games with imperfect information on finite graphs, a strategy for player i can be viewed as an A_i-labeled VA-tree $t : (\mathrm{VA})^* \to A_i$ which fulfills the conditions (S1) and (S2). We can also view a strategy for player i as an A_i-labeled $[V]_{\sim_i^V}[A]_{\sim_i^A}$-tree for which the consistency condition (S2) is redundant, cf. Section 2.1.2. This view is particularly appealing as the set of all A_i-labeled VA-trees which fulfill conditions (S1) and (S2) as stated in Section 2.1.2, is *not regular*: While it is easy for a tree automaton to check that a given tree fulfills condition (S1) for all nodes which correspond to histories in $\mathcal{G}$, the consistency condition (S2) cannot be checked by a tree automaton. In fact, if it could be, then the strategy problem would be decidable for arbitrary games with imperfect information on finite graphs with omega-regular winning conditions. We will return to this issue later.

We will use tree automata mainly to solve controller problems for distributed systems. For a distributed system $\mathcal{D} = (\mathfrak{A}, (\Sigma_c)_{c \in C})$, a strategy for a process p is a Σ^p_{out}-labeled Σ^p_{in}-tree. However, a joint strategy σ for the grand coalition should now not be noted as just a tuple $\sigma = (\sigma_1, \dots, \sigma_n)$ since a tuple of trees is not a tree per se. Instead we use the representation as a function $\sigma : (\Sigma^{\mathcal{D}})^* \to \prod_{i=1}^n \Sigma^{p_i}_{\mathrm{out}}$ such that $\mathrm{Pr}_{\Sigma^{p_i}_{\mathrm{out}}}(\sigma)$ fulfills the consistency condition (S2) for each i. Such a strategy can then be represented as a $\prod_{i=1}^n A_i$-labeled $\Sigma^{\mathcal{D}}$-tree or, as we have demonstrated in Section 2.1.3, as a $\prod_{i=1}^n \Sigma^{p_i}_{\mathrm{out}}$-labeled $\Sigma^{p_{\mathrm{env}}}_{\mathrm{out}}$-tree since once the controllers have commited themselves to a joint strategy, the outputs of all controllers are completely determined by the inputs that they receive from the environment. Remind that in this case, the consistency conditions require not only that certain labels are uniform over certain branches of the tree but also that certain labels are uniform over certain sequences of labels of the branches. We shall keep this observation in mind since it is useful for automata based methods to solve controller problems for distributed systems.

Finite automata on infinite trees were introduced by Rabin in [176] where he used Muller acceptance on the states of the automaton as well as acceptance conditions given by deterministic ω-automata. Notice that as for winning conditions for games, these two representations of acceptance conditions are essentially equivalent: Consider an alternating parity tree automaton $\mathcal{A}$ and a deterministic ω-automaton $\mathcal{B}$ over $\Sigma = Q$ with, say, parity acceptance condition which defines the acceptance component acc of $\mathcal{A}$. Then we obtain an alternating parity tree automaton $\mathcal{A}'$ with $L(\mathcal{A}') = L(\mathcal{A})$ via the synchronized product $\mathcal{A}' = \mathcal{A} \times \mathcal{B}$ and a corresponding coloring $\mathrm{col}^{\times}$ just as for game graphs. Moreover, if $\mathcal{A}$ is nondeterministic (deterministic) then $\mathcal{A}'$ is nondeterministic (deterministic).

Rabin proved that a language of infinite trees is recognizable by such an automaton if, and only if, it is definable in the monadic second order logic of two successors S2S. He also showed decidability of the emptiness problem which yields that satisfiability for S2S is decidable. (This implies that the monadic theory of the infinite binary is decidable.) Notice that this result is analog to Büchi's result for S1S, see Section 2.2.1, and the references [100] and [83].

Rabin used *nondeterministic* tree automata for which the key step in the construction is complementation (the same also holds for Büchi's construction for Büchi automata). In [127], Kupferman and Vardi suggested that *alternation* is a suitable and useful mechanism for dealing with partial information and this concept has been applied successfully to several cases of synthesis under partial information, quite beyond the scope of [127]. While in [127] the main contribution has been the use of alternation to deal with branching time specification in the context of partial information, in [137] the power of alternation has been used in a different way to deal with local linear specifications for possible more complex architectures. Moreover, alternation also yields elegant and coherent solutions synthesis under partial information for global linear specifications.[1]

Alternating computing devices have first been introduced in [49], see also [48] where alternating Turing machines, pushdown automata and finite automata on *finite* words have been considered. Alternating automata on infinite objects first occurred in [147] where it has

[1]This apparent connection between alternating tree automata and strategy problems under partial information has been accentuated in [172, 164] where it has been observed that two-player games with imperfect information correspond directly to *universal* tree automata and can also be used to deal with alternation, cf. Section 3.2.

been shown that alternating Büchi automata (i.e., on infinite *words*) can be translated into nondeterministic Büchi automata with an exponential blow up of the state space.

More precisely, if the original automaton has n states, then the resulting automaton has at most 3^n states. (Very recently it has been shown that this bound it essentially optimal [38].) The construction in [147] can also be extended to Büchi automata on infinite trees which demonstrates that an alternating Büchi tree automata with n states can be transformed into a nondeterministic Büchi tree automaton with at mos 3^n states.

To use tree automata to deal with arbitrary ω-regular winning condition, we need the more general model of alternating *parity* tree automata. Notice that alternating parity tree automata are strictly more expressive than alternating Büchi tree automata. On the other hand, in contrast to the case of ω-automata, the classes of trees that are recognizable by deterministic parity tree automata and nondeterministic Büchi automata, respectively, are incomparable, see our collections of results that we provide in the following sections.

Alternating automata on infinite trees have been introduced by Muller and Schupp [152, 153] where they considered several types of acceptance conditions which encompass all regular types of acceptance condition. While, by now, there are several definitions of alternating tree automata around, we will use the one given by Muller and Schupp which has been used successfully for the solution of strategy problems under partial information, cf. [127, 137, 80].

3.1.1 Tree Automata

An *alternating tree automaton* over Σ-labeled X-trees has the form

$$\mathcal{A} = (\Sigma, Q, q_{\text{in}}, \delta, \text{acc})$$

where Σ is the alphabet of labels, Q is the finite set of states, $q_{\text{in}} \in Q$ is the initial state and $\delta : Q \times \Sigma \to \mathcal{B}^+(X \times Q)$ is the transition function where $\mathcal{B}^+(X \times Q)$ denotes the set of all positive Boolean formulas over propositional variables from $X \times Q$, and $\text{acc} \subseteq Q^\omega$ is the acceptance condition. As usual, for Boolean formulas we assume that $\wedge$ takes precedence over $\vee$ and that the empty conjunction is a tautology and the empty disjunction is unsatisfiable. We also usually assume that the formulas from $\mathcal{B}^+(X \times Q)$ are given in disjunctive normalform and we denote such formulas φ in DNF also as sets $\varphi = \{\psi_0, \dots, \psi_{l-1}\}$ of conjuncts and we denote the conjuncts ψ also as sets $\psi \subseteq X \times Q$ of propositional variables. This is convenient as it allows to use the notation $\psi \in \varphi$ and $(x, q) \in \psi$ unambiguously. Moreover, we use the notation $\text{Dir} = \{\downarrow_x \mid x \in X\}$ to emphasize the character of the elements $x \in X$ as *directions* to which the automaton proceeds.

An alternating *parity* tree automaton comes with a coloring $\text{col} : Q \to^[k]$ for some $k \in \mathbb{N}$ instead of acc and $\text{acc} \subseteq Q^\omega$ is then defined via the parity condition given by col: A sequence $\pi = q_0q_1q_2 \ldots \in Q^\omega$ (with $q_0 = q_{\text{in}}$) is in acc if $\min \text{Infcol}(\pi)$ is even. We also consider alternating *Muller* tree automata which are the same as alternating parity tree automata but the coloring function col is replaced by a collection $\mathcal{F} \subseteq 2^Q$ of sets of states which define the acceptance condition acc via the Muller condition: A sequence $\pi = q_0q_1q_2 \ldots$ is in acc if $\text{Inf}(\pi) \in \mathcal{F}$. Analogously, we define *Büchi*, *reachability* and *safety* automata that come with a designated state set $F \subseteq Q$ defining acc as described in Section 2.2.

The behavior of the automaton on some Σ-labeled X-tree $t : X^* \to \Sigma$ is given by the notion of a run. A *run* of $\mathcal{A}$ on t is a *not necessarily* full Σ_r-labeled $\mathbb{N}$-tree $\rho : T \to \Sigma_r$, where $\Sigma_r = X^* \times Q$ such that the following conditions hold.

(1) $\varepsilon \in T$ and $\rho(\varepsilon) = (\varepsilon, q_{\text{in}})$

(2) if $v \in T$ with $\rho(v) = (u, q)$ and $\varphi = \delta(q, t(u))$ then there is some conjunct $\psi = \{(\downarrow_{x_0}, q_0), \dots, (\downarrow_{x_{l-1}}, q_{l-1})\} \subseteq \text{Dir} \times Q$ in (the DNF of) φ such that for $j \in [l]$, v has a successor $v \cdot j$ with $\rho(v \cdot j) = (u \cdot x_j, q_j)$.

So a typical run of $\mathcal{A}$ goes like: Start at the root ε of t in the initial state q_{in} and look at $\varphi = \delta(q_{\text{in}}, t(\varepsilon)$. *Choose* (nondeterminism) a conjunct $\psi = \{(\downarrow_{x_0}, q_0), \dots, (\downarrow_{x_{l-1}}, q_{l-1})\}$ from φ and then, proceed with *all* (universality) possibilities given by ψ. That means, the automaton sends a copy with state q_0 to direction x_0, a copy with state q_1 to direction x_1 and so on. Then each copy goes on like this individually, that is, the copy corresponding to $(\downarrow_{x_j}, q_j)$ looks at $\varphi = \delta(q_j, t(x_j))$ and makes the next step as before.

The crux of universality is that several of the directions x_j may be *identical*, so that the automaton may send several different copies into the same direction. Hence, different nodes $v \neq v'$ in the run ρ may correspond to the same node $\text{Pr}_{X^*}(v) = \text{Pr}_{X^*}(v')$ in t which is the reason why a run is a tree over a possibly larger set of directions than X. Notice though, that using the infinite set $\mathbb{N}$ as directions of a run is an overapproximation: Clearly, $X \times Q$ is sufficient if we disallow useless conjunctions like $(q, j) \wedge (q, j)$ in the transition function. Nevertheless, we shall usually stick to the definition using $\mathbb{N}$ as directions. For a node $v \in T$ in the run ρ with $\rho(v) = (u, q)$, we call u the X^*-component of $\rho(v)$ and q the Q-component of $\rho(v)$. Moreover, we sometimes refer to the last symbol $u(|u| - 1)$ of u as the *X-component* of $\rho(v)$.

The run is called *accepting* if each infinite path π through ρ is accepting. More precisely, for all infinite paths α through T the sequence of Q-components in $\rho(\pi)$ is accepting, that is, $\text{Pr}_Q(\rho(\pi)) \in \text{acc}$. The automaton $\mathcal{A}$ *accepts* a tree t if there *exists* an accepting run of $\mathcal{A}$ on t. The language recognized by $\mathcal{A}$ is $L(\mathcal{A}) = \{t \in X_\Sigma \mid \mathcal{A} \text{ accepts } t\}$. We call a language $\mathcal{T} \subseteq X_\Sigma$ *regular* if there is a *nondeterministic parity* tree automaton $\mathcal{A}$ with $L(\mathcal{A}) = \mathcal{T}$.

Notice that in the definition of a run, we have not excluded the possibility that the automaton sends copies into certain directions x (with some state q) that are not required by the conjunct ψ. Since the notion of acceptance asks for the *existence* of an accepting run, there is obviously no harm done by allowing the automaton to send more copies than required by the transition function. However, sometimes it is more convenient not to have such useless branches in a given run, so we usually assume that a run is *minimal* in the sense that it is build strictly according to the transition function. That means, we consider runs which satisfy the following new condition (2) instead of the old condition (2):

(2) if $v \in T$ with $\rho(v) = (u, q)$ and $\varphi = \delta(q, t(u))$ then there is some conjunct $\psi = \{(\downarrow_{x_0}, q_0), \dots, (\downarrow_{x_{l-1}}, q_{l-1})\} \subseteq \text{Dir} \times Q$ in φ such that the set of successors of v in T is $\{v \cdot j \mid j = 0, \dots, l - 1\}$ and $\rho(v \cdot j) = (u \cdot x_j, q_j)$.

Now assume w.l.o.g. that $X = [d]$ for some $d \in \mathbb{N}$. The automaton $\mathcal{A}$ is called *universal*, if for all $q \in Q$ and all $a \in \Sigma$ we have $|\delta(q, a)| = 1$, that means, the formula $\varphi = \delta(q, a)$ consists of a single conjunct $\varphi = (\downarrow_{x_0}, q_0) \wedge \dots \wedge (\downarrow_{x_{l-1}}, q_{l-1})$. The automaton $\mathcal{A}$ is called *nondeterministic* if, for all $q \in Q$ and all $a \in \Sigma$, $\delta(q, a)$ has the form

$$\bigvee_{j=0}^{m-1} (\downarrow_0, q_0^j) \wedge \dots \wedge (\downarrow_{d-1}, q_{d-1}^j)$$

for some $m \in \mathbb{N}$. The automaton is called *deterministic* if it is universal *and* nondeterministic, that is, for all $q \in Q$ and all $a \in \Sigma$, $\delta(q, a)$ has the form as for nondeterministic automata and, additionally, we have $m = 1$. We call the attribute of being deterministic, nondeterministic, universal or (strictly) alternating the *alternation mode* of the automaton.

We represent the transition function of a nondeterministic tree automaton also as a relation $\Delta \subseteq Q \times \Sigma \times Q^d$ and for a deterministic tree automaton as a function $\delta\colon Q \times \Sigma \to Q^d$. Moreover, if $\mathcal{A}$ is nondeterministic, a run of $\mathcal{A}$ on a Σ-labeled X-tree $t\colon X^* \to \Sigma$ can be viewed as a Q-labeled Xtree $\rho\colon X^* \to Q$ with $(\rho(u), t(u), \rho(u \cdot 0), \dots, \rho(u \cdot (d - 1))) \in \Delta$ for all $u \in X^*$.

We like to note that sometimes, the definition of nondeterministic and deterministic automata is somewhat relaxed by allowing the automaton to send copies only to some directions instead of all directions. This, however, is only a technical difference: since a run is accepting if *all infinite* paths through the run are accepting according to acc, we can always make the transition structure of the automaton complete with respect to the set Dir of directions, by introducing a default accepting state which is sent to all directions to which the automaton doesn't send a copy. In fact, when constructing tree automata, we allow ourselves to use the relaxed version of deterministic and nondeterministic automata.

Notice that both, deterministic and universal tree automaton, have a uniquely determined run on each given input tree t. However, while a deterministic automaton sends, in each step of this uniquely determined run, exactly one copy into each direction of the tree, a universal automaton may send several copies into the same direction. A nondeterministic automaton, on the other hand, can choose, in each step of a run, from at most m different conjuncts but once a conjunct is chosen, the automaton sends exactly one copy into each direction.

Remark. As it turns out, on *trees*, the possibility to have several different runs is a stronger gain than being able to send several copies into the same direction: while universal parity tree automata are equally expressive as the deterministic model, nondeterministic parity tree automata are strictly more powerful, cf. Section 3.1.2. On the other hand, on words, the situation is quite different: Alternating and deterministic parity ω-automata have the same expressive power. The same holds for Muller automata and, while deterministic Büchi automata are strictly less expressive than nondeterministic Büchi automata, the nondeterministic and universal model have the same expressive power.

The difference here is of course that on trees, a nondeterministic automaton is still somehow alternating, that is, it has $\vee$ and $\wedge$ in the transition formulas $\delta(q, a)$. Clearly, this asymmetry between nondeterministic and universal automata is implied by the objects on which the automata operate – to exploit the branching character of a tree, an automaton should be able, during any step of a run, to proceed to each direction of the tree simultaneously.

Before we go into the more sophisticated results about tree automata we state some well-known properties that give a bit of a conception of their expressive power. We write down a short proof just to get started with tree automata. For an ω-language $L \subseteq \Sigma^\omega$ and some finite set X, by $\mathcal{T}(L, X)$, we denote the set of all Σ-labeled X-trees t such that all paths through t are in L. Moreover, consider the following tree languages:

- $\mathcal{T}_{\exists b} = \{t \in \mathbb{B}_{\{a,b\}} \,|\, t(u) = b \text{ for some } u \in \mathbb{B}^*\}$
- $\mathcal{T}_{t^0=t^1} = \{t \in \mathbb{B}_{\{a,b\}} \,|\, t(0u) = t(1u) \text{ for all } u \in \mathbb{B}^*\}$
- $\mathcal{T}_{|b|<\omega} = \{t \in \mathbb{B}_{\{a,b\}} \mid \text{ each path of } t \text{ has only finitely many letters } b\}$

Proposition 3.1. *(1) If $L \subseteq \Sigma^\omega$ is a regular ω-language then $\mathcal{T}(L, X)$ can be recognized by a deterministic parity tree automaton.*

(2) The language $\mathcal{T}_{\exists b}$ is recognizable by a nondeterministic reachability automaton but not recognizable by any deterministic tree automaton.

(3) The language $\mathcal{T}_{t^0=t^1}$ is not recognizable by any tree automaton.

(4) The language $\mathcal{T}_{|b|<\omega}$ is recognizable by a deterministic parity automaton but not recognizable by a nondeterministic Büchi automaton.

Proof. Proposition (1) is obvious: We just construct a deterministic tree automaton for $\mathcal{T}(L, X)$ where the acceptance condition acc is given by a deterministic ω-automaton recognizing L and, as we have already mentioned, the synchronized product gives a deterministic parity tree automaton recognizing $\mathcal{T}(L, X)$. For proposition (2), the first part is again obvious: A nondeterministic reachability automaton just guesses a path on which it expects to find a letter b and sends F-states into all other branches of the tree. On the path that is checks it stays in a state $Q \setminus F$ as long as it does not see a letter b but as soon as it does, from this point on it only sends F-states to all directions.

For the second part of (2), assume that $\mathcal{A} = (\{a, b\}, Q, q_{\text{in}}, \delta, \text{acc})$ is a deterministic tree automaton such that $L(\mathcal{A}) = \mathcal{T}_{\exists b}$ and consider the tree $t_a : \mathbb{B}^* \to \{a, b\}$ with $t_a(u) = a$ for all

$u \in \mathbb{B}^*$. Since $t \notin \mathcal{T}_{\exists b}$, the unique run ρ_a of $\mathcal{A}$ on t_a is not accepting, that means, there is at least one path π in ρ_a such that $\rho_a(\pi) \notin \text{acc}$. Now let the first transition in ρ_a be $(q_{\text{in}}, a, q_0, q_1)$ for some $q_0, q_1 \in Q$. W.l.o.g., we assume that the path π goes through the subtree of ρ rooted in 0, that is, $\pi(0) = \varepsilon$ and $\pi(1) = 0$. We define the tree $t : \mathbb{B}^* \to \Sigma$ by $t(1) = b$ and $t(u) = a$ for all $u \in \mathbb{B}^* \setminus \{1\}$. Then the unique run ρ of $\mathcal{A}$ on t starts with the transition $(q_{\text{in}}, a, q_0, q_1)$ as well and the subtree t^0 of t rooted in 0 coincides with the subtree t_a^0 of t_a rooted in 0. (Formally, t^0 is the tree $t^0 : 0X^* \to \Sigma$ with $t^0(0u) = t(0u)$ for all $u \in X^*$.) Therefore, the subtree of the run ρ rooted in 0 coincides with the subtree of the run ρ_a rooted in 0. Hence, the path π goes through ρ as well and so, ρ is not accepting. But since $t \in \mathcal{T}_{\exists b}$ this is a contradiction to $L(\mathcal{A}) = \mathcal{T}_{\exists b}$.

To prove (3), assume there is some tree automaton $\mathcal{A} = (\{a, b\}, Q, q_{\text{in}}, \delta, \text{acc})$ such that $L(\mathcal{A}) = \mathcal{T}_{t^0 = t^1}$ and consider the language $\mathcal{T} = \{t_l \mid l \in \mathbb{N}\}$, where for $l \in \mathbb{N}$, t_l is defined by $t_l(u) = a$ for all $u \in X^*$ with $|x| \leq l$ and $t_l(u) = b$ for all $u \in X^*$ with $|u| > l$. Clearly, $\mathcal{T} \subseteq \mathcal{T}_{t^0 = t^1}$ so for each $t \in \mathcal{T}$ there exists an accepting run ρ_t of $\mathcal{A}$ on t. Consider, for each $t \in \mathcal{T}$, a conjunct $\psi_t \in B^+(X \times Q)$ such that the first transition in ρ_t is specified by ψ_t. More precisely, if the root $(q_{\text{in}}, \varepsilon)$ of ρ_t has successors labeled with $(j_0, q_0), \ldots, (j_l, q_l) \in \{0, 1\} \times Q$, then $\psi_t = (\downarrow_{j_0}, q_0) \wedge \ldots \wedge (\downarrow_{j_l}, q_l)$. Let $\Delta_{\mathcal{T}} = \{\psi_t \mid t \in \mathcal{T}\}$. Since $\mathcal{T}$ is infinite while $\Delta_{\mathcal{T}}$ is finite, there are trees $t_0, t_1 \in \mathcal{T}$ with $t_0 \neq t_1$ and $\psi := \psi_{t_0} = \psi_{t_1}$. Now let the tree $t : \mathbb{B}^* \to \{a, b\}$ be defined by $t(\varepsilon) = a$, $t(0u) = t_0(0u)$ and $t(1u) = t_1(1u)$ and consider the run ρ of $\mathcal{A}$ on t which is defined as follows. The first transition of ρ is ψ and then, for each literal $(\downarrow_j, q)$ in ψ, the subtree of ρ rooted in (j, q) is the subtree of ρ_{t_0} rooted in (j, q) if $j = 0$ and it is the subtree of ρ_{t_1} rooted in (j, q), if $j = 1$. Since $\psi = \psi_{t_0} = \psi_{t_1}$, ρ is indeed a run of $\mathcal{A}$ on t and since both, ρ_{t_0} and ρ_{t_1}, are accepting, ρ is accepting as well. But $t \notin \mathcal{T}_{t^0 = t^1}$ which is a contradiction to out assumption that $L(\mathcal{A}) = \mathcal{T}_{t^0 = t^1}$.

Finally, the first part of proposition (4) is again obvious: A deterministic parity automaton $\mathcal{A}$ recognizing $\mathcal{T}_{|b| < \omega}$ uses colors 1 and 2 and it takes color 2 whenever it encounters a letter a and color 1 whenever it encounters a letter b. For a given tree $t \in \mathbb{B}_{\{a,b\}}$, the least color seen infinitely often is 2 on each path of the uniquely determined run of $\mathcal{A}$ if, and only if, each path of t contains only finitely many letters b. For a proof of the second part of proposition (4) see, for example, [100]. (The result has originally been proved by Rabin, cf. [177, 100].) ☐

Closure Properties. An important aspect of any automaton model is *closure* of the languages that can be recognized by these automata under certain operations like union, intersection and projection. This is especially important for applications to logic: Intersection, complementation and projection correspond to conjunction, negation and existential quantification, respectively. Closure under union and intersection are both easy to prove for nondeterministic as well as for alternating tree automata. On the other hand, while closure under projection is easy for nondeterministic tree automata but hard for alternating tree automata, for closure under complement it is the other way round. *Deterministic* (and, hence, universal) tree automata are closed under intersection, but in general not closed under any other operation. These properties are well-known. We give short proofs which fortify the understanding of tree automata and help to comprehend the different branching modes. In the following, let X and Y be finite sets of directions and let Σ and Γ be finite alphabets.

Proposition 3.2. *Nondeterministic and alternating parity tree automata are closed under union and intersection. Deterministic parity tree automata are closed under intersection but not union.*

Proof. Let $\mathcal{A}_j = (\Sigma, Q^j, q^j_{\text{in}}, \delta^j, \text{col}^j)$, $j = 1, 2$ be alternating parity tree automata. We describe the construction of the parity tree automaton $\mathcal{A} = (\Sigma, Q, q_{\text{in}}, \delta, \text{acc})$ recognizing the union (intersection) of $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$ informally. First, in both cases, Q is the disjoint union of Q^1 and Q^2 together with a fresh initial state q_{in} and we define $\text{col}(q) = \text{col}^j(q)$, where $q \in Q^j$ and $\text{col}(q_{\text{in}}) = 0$. (In fact, the color of q_{in} is insignificant as q_{in} will occur in each run of $\mathcal{A}$ exactly once.) Moreover, $\delta(q, a) = \delta^j(q, a)$ for all $q \neq q_{\text{in}}$, where $q \in Q^j$.

Now, for the automaton $\mathcal{A}$ tackling the *union* of $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$ we define $\delta(q_{\text{in}}, a) = \delta^1(q^1_{\text{in}}, a) \vee \delta^2(q^2_{\text{in}}, a)$ for $a \in \Sigma$. So at the beginning of a run, the automaton $\mathcal{A}$ decides whether to perform a run of $\mathcal{A}_1$ or a run of $\mathcal{A}_2$: Since we take the *disjoint* union of Q^1 and Q^2, once the automaton has chosen $\delta^j(q^j_{\text{in}}, a)$ for some $j \in \{1, 2\}$, it performs a run of $\mathcal{A}_j$. So $\mathcal{A}$ accepts a given input tree t if, and only if, there is an accepting run of $\mathcal{A}_1$ on t or there is an accepting run of $\mathcal{A}_2$ on t. Hence, $L(\mathcal{A}) = L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$. Notice that if $\mathcal{A}_1$ and $\mathcal{A}_2$ are both nondeterministic, then so is $\mathcal{A}$. On the other hand, $\mathcal{A}$ is not necessarily deterministic, even if $\mathcal{A}_1$ and $\mathcal{A}_2$ both are.

For the automaton that is intended to recognize the *intersection* of $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$ we define $\delta(q_{\text{in}}, a) = \delta^1(q^1_{\text{in}}, a) \wedge \delta^2(q^2_{\text{in}}, a)$ for $a \in \Sigma$. So the automaton $\mathcal{A}$ simulates a run of $\mathcal{A}_1$ and a run of $\mathcal{A}_2$ in parallel: Since we take the *disjoint* union the two runs are completely independent of each other and each path through either one of these runs has to be accepting. So $\mathcal{A}$ accepts a given input tree t if, and only if, there is an accepting run of $\mathcal{A}_1$ on t and there is an accepting run of $\mathcal{A}_2$ on t. Hence, $L(\mathcal{A}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$. Now, however, $\mathcal{A}$ is *not* necessarily nondeterministic, even if $\mathcal{A}_1$ and $\mathcal{A}_2$ both are.

To see that deterministic parity tree automata are not closed under union, consider the $\{a, b\}$-labeled $\mathbb{B}$-trees t_a with $t_a(u) = a$ for all $u \in \mathbb{B}^*$ and t_b with $t_b(\varepsilon) = a$ and $t_b(u) = b$ for all $u \in \mathbb{B}^* \setminus \{\varepsilon\}$. The languages $\{t_a\}$ and $\{t_b\}$ can be recognized by deterministic parity tree automata but the language $\{t_a, t_b\}$ cannot be recognized by any deterministic tree automaton: Consider the uniquely determined runs ρ_a and ρ_b of a given deterministic tree automaton $\mathcal{A}$ on t_a and on t_b, respectively and consider the tree t which is obtained by replacing the subtree of t_a rooted in 1 by the subtree of t_b rooted in 1. Then the run ρ of $\mathcal{A}$ on t is also obtained by replacing the subtree of ρ_a rooted in 1 by the subtree of ρ_b rooted in 1. Hence, since both ρ_a and ρ_b are accepting, the run ρ is accepting. Since $t \notin \{t_a, t_b\}$ we have $L(\mathcal{A}) \neq \{t_a, t_b\}$. $\square$

In Section 3.1.2 we will see that any alternating parity tree automaton can be translated into an equivalent nondeterministic parity tree automaton. Hence, the construction above shows, nevertheless, that nondeterministic parity tree automata are closed under intersection. Though, the construction for removing alternation from parity tree automata is complicated and there is a much simpler way of constructing an automaton $\mathcal{A}$ recognizing the intersection of $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$ if $\mathcal{A}_1$ and $\mathcal{A}_2$ are nondeterministic, which is also standard for other automaton models: the *synchronous product* $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$. The state set of $\mathcal{A}$ is $Q = Q_1 \times Q_2$ and the acceptance condition acc consists of those sequences $\alpha \in Q^\omega$ such that $\text{Pr}_{Q_1}(\alpha) \in \text{acc}_1$ and $\text{Pr}_{Q_2}(\alpha) \in \text{acc}_2$. During a run, the automaton $\mathcal{A}$ simulates $\mathcal{A}_1$ and $\mathcal{A}_2$ in parallel and accepts, if both automata accept. Clearly, if there is an accepting run of $\mathcal{A}_1$ on a given input tree t and an accepting run of $\mathcal{A}_2$ on t then this run can be easily combined to an accepting run of $\mathcal{A}$. Conversely, since $\mathcal{A}_1$ and $\mathcal{A}_2$ are both *nondeterministic* automata, an accepting run of $\mathcal{A}$ on t can also directly be decomposed into individual accepting runs of $\mathcal{A}_1$ and $\mathcal{A}_2$ on t. Hence, $L(\mathcal{A}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$. This is no longer true, if $\mathcal{A}_1$ or $\mathcal{A}_2$ is not nondeterministic. We discuss the reason for this at the end of this section. Moreover, if $\mathcal{A}_1$ and $\mathcal{A}_2$ are both deterministic, then $\mathcal{A}$ is deterministic as well. Notice, however, that acc is not a parity condition per se. What we do is, we first consider the Muller acceptance condition given by $\mathcal{F} = \{F \subseteq Q_1 \times Q_2 \mid \min \text{col}_j(\text{Pr}_{Q_j}(F))) \text{ is even for } j \in \{1, 2\}\}$. It is not hard to see that this acceptance condition is precisely acc. Then we can turn the resulting nondeterministic Muller tree automaton into an equivalent nondeterministic parity tree automaton.

Now we consider closure under projection for nondeterministic automata. For alternating tree automata, closure under projection then follows from the results described in Section 3.1.2. Let $\iota : \Sigma \to \Gamma$ be some function. The operator $\text{Pr}_\iota : X_\Sigma \to X_\Gamma$ yields, for a Σ-labeled X-tree t, the Γ-labeled X-tree $t' = \text{Pr}_\iota(t)$ with $t'(x) = \iota(t(x))$. A particularly interesting special case of projection is given by alphabets Σ of the form $\Sigma = \Sigma_1 \times \Sigma_2$ and functions $\iota : \Sigma \to \Sigma_1$. We denote the corresponding projection operator Pr_ι also by Pr_{Σ_1}. For a Σ_1-labeled X-tree t_1 we call a Σ_2-labeled tree t_2 also a Σ_2-annotation to t_1 and we call the product tree $t_1 \times t_2$ with

$t_1 \times t_2(u) = (t_1(u), t_2(u))$ a Σ_1-labeled Σ_2-annotated X-tree. Hence, for $\mathcal{A} \subseteq X_\Sigma$ we have

$$\mathrm{Pr}_{\Sigma_1}(\mathcal{T}) = \{t_1 \in X_{\Sigma_1} \mid \text{ there is a } \Sigma_2 - \text{annotation } t_2 \text{ of } t_1 \text{ such that } t_1 \times t_2 \in \mathcal{T}\}.$$

Proposition 3.3. *Nondeterministic tree automata are closed under projection but deterministic tree automata are not.*

Proof. Let $\mathcal{A} = (\Sigma, Q, q_{\text{in}}, \Delta, \text{acc})$ be a nondeterministic tree automaton over Σ-labeled $[d]$-trees and let $\iota : \Sigma \to \Gamma$ be a function. We define the automaton $\mathcal{A}' = (\Gamma, Q, q_{\text{in}}, \Delta', \text{acc})$ by $\Delta' = \{(q, \iota(a), q_0, \dots, q_{d-1}) \mid (q, a, q_0, \dots, q_{d-1}) \in \Delta\}$. (Or, using the notation of the transitions as formulas, $\delta'(q, a) = \bigvee_{b \in \Sigma, \iota(b)=a} \delta(q, b)$). Notice that in the case of a projection operator Pr_{Σ_1}, Δ' is obtained from Δ by deleting the Σ_2-component from the label of each transition.

Clearly, if $s \in \mathrm{Pr}_\iota(L(\mathcal{A})$, that is, s is obtained from a Σ-labeled $[d]$-tree $t \in L(\mathcal{A})$ by applying the function ι to the labels of t, then we obtain an accepting run of $\mathcal{A}'$ on s from an accepting run of $\mathcal{A}$ on t by applying the function ι to the labels of the transition that are used in the run. Hence, $s \in L(\mathcal{A}')$. Now let conversely $s \in L(\mathcal{A}')$ and consider a run $\rho : [d]^* \to Q$ of $\mathcal{A}'$ on s. Then, for each $u \in [d]^*$, there is a transition $\zeta(u) = (q, b, q_0, \dots, q_{d-1}) \in \Delta'$ with $q = \rho(u)$, $b = s(u)$ and $q_l = \rho(u \cdot l)$ for $l \in [d]$. By construction of $\mathcal{A}'$, $b = \iota(a_u)$ for some $a_u \in \Sigma$ such that $(q, a_u, q_0, \dots, q_{d-1}) \in \Delta$. So if we define the Σ-labeled $[d]$-tree t by $t(u) = a_u$, then ρ is an accepting run of $\mathcal{A}$ on t which yields $t \in L(\mathcal{A})$. Since by definition, $\iota(a_u) = s(u)$ for all $u \in [d]^*$ we have $\mathrm{Pr}_\iota(t) = s$ and hence $s \in \mathrm{Pr}_\iota(L(\mathcal{A})$.

To see that deterministic tree automata are not closed under projection, instead of providing a concrete example language, we refer to the construction for removal of alternation from Section 3.1.2 which translates an alternating parity tree automaton into a nondeterministic parity tree automaton. A careful examination of this construction reveals that the only source of nondeterminism is a projection step. So if deterministic parity tree automata were closed under projection, then any alternating parity tree automaton could be translated into an equivalent deterministic parity tree automaton which, according to Proposition 3.1, is not possible. □

Here, the constructions that are performed on the given automaton to obtain an automaton recognizing the projection of the original language are trivial. In particular, the resulting automaton has the same size as the original automaton and also uses the same acceptance condition acc. Moreover, if $\mathcal{A}$ is nondeterministic, then $\mathcal{A}'$ is nondeterministic and, if $\mathcal{A}$ is deterministic and ι is injective, then so is $\mathcal{A}'$. However, if ι is not injective, the latter statement does not hold anymore.

We also need the more special operation of *widening*. The operator $\text{wide}_Y : X_\Sigma \to (X \times Y)_\Sigma$ yields, for a Σ-labeled X-tree t, the Σ-labeled $X \times Y$-tree $t' = \text{wide}_Y(t)$ with $t'(x, y) := t(x)$.

However, we do not prove *closure* under widening. In fact, tree automata are *not* closed under widening which can be proved with a similar reasoning as in the proof that $\mathcal{T}_{t^0=t^1}$ cannot be recognized by any tree automaton. It is also interesting to notice that if tree automata were closed under widening, then the strategy uniformity condition (S2) could also be checked by tree automata. We discuss this in further detail in Section 3.2. The property that does hold for tree automata and the widening operator has been the key step in the solution of the strategy problem for branching time specifications in [127] and is formulated in the following proposition. We provide a proof of this result in Section 3.2.

Proposition 3.4. *[127] For any alternating parity tree automaton $\mathcal{A}$ over Σ-labeled $X \times Y$-trees there is an alternating parity tree automaton $narrow_Y(\mathcal{A})$ over Σ-labeled X-trees such that $t \in L(narrow_Y(\mathcal{A}))$ if, and only if, $wide_Y(t) \in L(\mathcal{A})$. Moreover, $narrow_Y(\mathcal{A})$ has the same size as $\mathcal{A}$ and uses the same acceptance condition.*

The same result can also be proved for the x-ray operator. We will use the x-ray operator only rather implicit, but like to state it as a general principle to keep in mind. The operator $\text{xray} : X_\Sigma \to X_{\Sigma \times X}$ yields, for Σ-labeled X-tree t, the $\Sigma \times X$-labeled X-tree $t' = \text{xray}(t)$ with

$t'(x) = (t(x), x(|x|-1))$. (For the root $x = \varepsilon$, $x(|x|-1)$ is some designated element from X, called the root direction.) In this case, the proof is even much easier than in the case of the widening. The new automaton $\mathcal{B}$, while sending a copy to some direction, keeps the direction in the state to which it switches. In this way, it can simulate the given automaton $\mathcal{A}$ directly, by feeding it's states into the transition function of $\mathcal{A}$. The acceptance condition of $\mathcal{B}$ is essentially just the acceptance condition of $\mathcal{A}$. However, the size of $\mathcal{B}$ now depends on the size of the set X.

Emptiness of Tree Automata. The most important property of finite automata on infinite trees is the *decidability* of the emptiness problem, that is, it is decidable whether $L(\mathcal{A}) = \emptyset$ for a given alternating parity tree automaton $\mathcal{A}$. The emptiness problem for nondeterministic tree automata with Muller or pairs acceptance condition has been solved by Rabin [176, 177] where he used the decidability of the problem to prove decidability of the monadic theory of the infinite binary tree and to provide an alternative solution of Church's problem.

The pairs condition is given by a collection $((E_1, F_1), ..., (E_l, F_l))$ of pairs $E_j, F_j \subseteq Q$ which define the acceptance condition acc as follows: a sequence $\pi \in Q^\omega$ of states is in acc if for some j we have $\mathrm{Inf}(\pi) \cap E_j = \emptyset$ and $\mathrm{Inf}(\pi) \cap F_j \neq \emptyset$. This acceptance condition (or winning condition, in the context of games) was introduced by Rabin in [177] and is now called Rabin condition. In [149], Mostowski considered the parity condition in form of the Rabin chain condition, that is, Rabin conditions with $E_1 \subseteq F_1 \subseteq \ldots \subseteq E_l \subseteq F_l$. It is not hard to see that the parity conditions correspond in fact precisely to the Rabin chain conditions.

Subsequently, several improved algorithms for the emptiness problem have been developed of which we like to mention the work of Emerson and Jutla [74] as well as Pnueli and Rosner [169] who have provided algorithms for solving the nonemptiness problem for Rabin tree automata with a time complexity exponential in the number of Rabin pairs but only polynomial in the number of states. In [74] it was also shown that the problem is NP-complete in general, while Pnueli and Rosner [169] used their algorithm to obtain a solution of Church's problem for LTL specifications which has doubly exponential time complexity in the length of the specification. So in particular, the emptiness problem for parity tree automata can be solved in time exponential in the number of colors but polynomial in the number of states.

Theorem 3.5. *[176, 74, 169], see also [100]*
The emptiness problem for nondeterministic Rabin tree automata is decidable and can be solved in time exponential in the number of Rabin pairs and polynomial in the number of states of the given automaton.

For nondeterministic parity tree automata, however, more can be said, using a simple reduction of the nonemptiness problem to the strategy problem for parity games with full information. The construction is well known, see for example [203, 222]. As it is quite helpful for the understanding of tree automata, we provide a brief presentation of the construction. This also serves as a preparation of the more general construction for universal and alternating tree automata that we discuss in Section 3.2.

Let $\mathcal{A} = (\Sigma, Q, q_{\mathrm{in}}, \Delta, \mathrm{col})$ be a nondeterministic parity tree automaton where $\Delta \subseteq Q \times \Sigma \times Q^d$ for some $d \in \mathbb{N}$ and $\mathrm{col} : Q \to [k]$ for some $k \in \mathbb{N}$. We define the parity game graph $\mathcal{G}(\mathcal{A}) = (V, V_0, E, \mathrm{col})$ (with $E \subseteq V \times V$) as follows:

- $V = \Delta \cup Q$ and $V_0 = \Delta$
- E consists of the following moves:
  - $q \to (q, a, q_0, \ldots, q_{d-1}))$ for $(q, a, q_0, \ldots, q_{d-1}) \in \Delta$
  - $(q, a, q_0, \ldots, q_{d-1}) \to q_j$ for $(q, a, q_0, \ldots, q_{d-1}) \in \Delta$ and $j \in [k]$
- $\mathrm{col}(v) = \mathrm{col}(\mathrm{Pr}_Q(v))$ for $v \in V$.

So player 1 resolves the nondeterministic choices and, by doing so, implicitly also constructs a tree $t \in X_\Sigma$: the labels of t are the labels of the transitions chosen by player 1. Player 0, on the other hand, resolves the directions in the tree and a play π is won by player 1 if the corresponding infinite state sequence $\mathrm{Pr}_Q(\pi) \in Q^\omega$ is accepting. Hence, a winning strategy for player 1 from the initial position q_{in} guarantees the existence of a tree and a run on that tree such that all possible paths $\pi \in Q^\omega$ through that run are accepting and vice versa.

Proposition 3.6. *$L(\mathcal{A}) \neq \emptyset$ if and only if player 1 has a winning strategy for $\mathcal{G}(\mathcal{A})$ from q_{in}.*

Obviously, the size of the game graph $\mathcal{G}(\mathcal{A})$ is linear in the size of $\mathcal{A}$ and since the strategy problem for parity games with full information is in NP ∩ co-NP, the same holds for the emptiness problem for nondeterministic parity tree automata. Moreover, the problem can be solved in time $O((|Q| + |\Delta|)^k)$.

Now, if player 1 has a winning strategy, then he also has a positional winning strategy, since $\mathcal{G}(\mathcal{A})$ is a parity game graph. From such a positional winning strategy for player 1 for $\mathcal{G}(\mathcal{A})$ from q_{in} we can construct a finite automaton with output, which generates some tree that is accepted by $\mathcal{A}$. More precisely, there is a finite automaton $\mathcal{B} = (X, Q^\mathcal{B}, \delta^\mathcal{B}, q^\mathcal{B}_{\mathrm{in}}, \tau^\mathcal{B})$ with output function $\tau^\mathcal{B} : Q^\mathcal{B} \to \Sigma$ and state set $Q^\mathcal{B} = Q$ such that the Σ-labeled X-tree $t^\mathcal{B} : X^* \to \Sigma$ which is obtained by unfolding the transition graph of $\mathcal{B}$ (i.e., $t^\mathcal{B}(u) = \tau^\mathcal{B}((\delta^\mathcal{B})^*(u))$ for $u \in X^*$) is in $L(\mathcal{A})$. Trees which are generated by finite automata in this way are called regular. This result has first been proved by Rabin in [177] for the case of Rabin tree automata.

Theorem 3.7. *[177] If $L(\mathcal{A}) \neq \emptyset$ then $L(\mathcal{A})$ contains a regular tree.*

Proof. Let $\sigma : Q \to \Delta$ be a positional winning strategy for player 1 for $\mathcal{G}(\mathcal{A})$ from q_{in} and consider some $q \in Q^\mathcal{B} = Q$ with $\sigma(q) = (q, a, q_0, \ldots, q_{d-1})$. We define $\tau(q) = a$ and $\delta^\mathcal{B}(q, l) = q_l$. By construction of $t^\mathcal{B}$, the tree is regular and by definition of the game $\mathcal{G}(\mathcal{A})$ we have $t^\mathcal{B} \in L(\mathcal{A})$. □

Notice that, since we can construct σ from $\mathcal{A}$ in time $O((|Q| + |\Delta|)^k)$, the regular tree $t^\mathfrak{B}$ can be constructed in time $O((|Q| + |\Delta|)^k)$ as well.

Also notice that the finite automaton $\mathcal{B}$ closely resembles our definition of a finite memory structure which is not coincidental. In fact, Rabin's approach to Church's problem [177] can be paraphrased as follows. A strategy $\sigma : \mathbb{B}^* \to \mathbb{B}$ for the controller in Church's setting is a $\mathbb{B}$-labeled $\mathbb{B}$-tree t_σ. Now for a given linear specification φ written in monadic second order logic, construct a deterministic Rabin ω-automaton $\mathcal{S}$ such that $L(\mathcal{S}) = L(\varphi) \subseteq \mathbb{B}^\omega$. From this automaton, one can easily construct a finite Rabin tree automaton $\mathcal{A}$ (even a deterministic one) such that $\mathcal{A}$ accepts a tree $t_\sigma \in \mathbb{B}_\mathbb{B}$ if, and only if, all paths through t_σ which are compatible with the labeling (all plays which are compatible with σ) are in $L(\mathcal{S})$ (won by the controller).

This very simple and direct mutual reducibility reveals that nondeterministic tree automata and two-player games with full information are essentially quite the same. Nondeterministic tree automata have been used successfully for the solution of synthesis problems under full information [169, 1, 211, 221]. In [221], also certain cases of synthesis under partial information have been solved using nondeterministic tree automata for *linear* specifications. There, however, certain constructions have to be carried out *a priori* that essentially reduce the strategy problem under partial information to one under full information, cf. Sections 3.2 and 3.5.

On the other hand, as we have mentioned, in [127] Kupferman and Vardi have suggested that *alternation* is a proper concept to deal with partial information, in particular in the branching time setting. There, the difficulty of reducing the strategy problem under partial information to one under full information is shifted to the nonemptiness problem for alternating tree automata. However, for this problem, a straightforward and efficient game approach like for nondeterministic tree automata is not available. In fact, the emptiness problem is Exptime-hard for alternating automata with very simple acceptance condition like Büchi and even reachability acceptance condition. (Reachability acceptance conditions for tree automata are defined analogously to reachability winning conditions for infinite games on finite graphs.) The Exptime-hardness follows from the fact that already the universality problem for nondeterministic automata on finite trees is Exptime-hard [197]. (We will see later that, by a very direct and natural reduction from two-player reachability games with imperfect information on finite graphs, even the emptiness problem for *universal* tree automata with reachability acceptance condition can be shown to beEXPTIME-hard.[2])

The reason why the straightforward game approach for nondeterministic tree automata fails in the case of arbitrary alternating automata is that in the emptiness game as we have described it, played on an *alternating* tree automaton, player 1 could make the labeling of the tree dependent on the branches of the *run* which now do not correspond to the branches of an input tree. We will discuss this in Section 3.2. Hence, deciding emptiness for alternating tree automata is usually done by *translating* an alternating tree automaton into an equivalent nondeterministic tree automaton. Before we review the basic ideas of alternation removal for tree automata, we like to make some comments.

First, at this point we should anticipate that in Section 3.2 we will argue that, in fact, there is a straightforward game approach to the nonemptiness of alternating tree automata which, however, leads to games with *imperfect* information. While this accentuates the statement from [127] and offers a coherent perspective on the connection between alternating tree automata and synthesis under partial information, it does not help for the algorithmic solution of the emptiness problem as games with imperfect information are hard to solve as well.

Moreover, we like to mention the work [130] where an alternative approach has been developed to check emptiness of alternating tree automata. The key feature is to avoid removal of alternation and instead, reduce an alternating parity tree automaton to a nondeterministic Büchi tree automaton. The worst case complexity of this construction is essentially the same as for the method using removal of alternation. However, the potential advantage of the construction in [127] is that it is more transparent and simpler to implement than the usual constructions for removing alternation from parity tree automata and that it can be decomposed into several smaller steps which are all amenable to optimization.

Though, we are interested in alternation removal not only as a method for testing emptiness of a single given alternating tree automata. We also need to test emptiness of the intersection of a nondeterministic parity tree automaton $\mathcal{A}_1$ and a nondeterministic parity *pushdown* tree automaton $\mathcal{A}_2$, see Section 3.1.3. For this, we cannot take the *disjoint union* as described in Proposition 3.2 since this would yield an alternating parity pushdown tree automaton: alternating parity pushdown tree automata are strictly more expressive than nondeterministic ones and the emptiness problem for those automata is even undecidable. Our solution here is to apply a slightly advanced version of the usual product construction to $\mathcal{A}_1$ and $\mathcal{A}_2$.

However, as we have already mentioned in the proof of Proposition 3.2, for this product construction to be sound, $\mathcal{A}_1$ and $\mathcal{A}_2$ have to be *both* nondeterministic. This fact is completely independent of $\mathcal{A}_2$ being a pushdown tree automaton: To see why the construction fails if we apply it directly to alternating tree automata $\mathcal{A}_1$ and $\mathcal{A}_2$, consider a run of the resulting product automaton $\mathcal{A}$. Some branches in this run correspond to universal choices of $\mathcal{A}_1$ and some branches correspond to universal choices of $\mathcal{A}_2$. In this joint run, however, $\mathcal{A}_1$ can also make it's *nondeterministic* choices dependent on the branches which are induced by universal choices of $\mathcal{A}_2$ and vice versa!

At this point, we should anticipate some detail from Section 3.1.2 where we will show that, in fact, for any given input tree t which is in the language of some alternating parity tree automaton, there is also a run of this automaton on t in which the nondeterministic choices do *not* depend on the particular universal branch of the automaton but only on the current *node* in the tree t and the current *state* of the automaton. However, for an alternating automaton, there are still several possible states in which the automaton can be at a given node of the tree t. And, in the case of the product automaton $\mathcal{A}$, this state is composed of the individual states of the automata $\mathcal{A}_1$ and $\mathcal{A}_2$. Hence, the automaton $\mathcal{A}_1$ can still make it's nondeterministic choices

[2]Notice that due to the asymmetry of the nondeterministic and the universal alternation mode, this does not immediately follow from the EXPTIME-hardness of the universality problem for nondeterministic reachability tree automata.

depend on information that is immanent to the automaton $\mathcal{A}_2$ and vice versa.

The same problem arises for projection. The nondeterministic automaton that we have constructed in the proof of Proposition 3.3 implicitly *guesses* a Σ-labeling for the Γ-labeled $[d]$-tree on which it runs by nondeterministically choosing transitions with Γ-labels for which there are corresponding Σ-labels. Now if we apply the construction directly to an alternating automaton then the nondeterministic choices of the automaton and, hence, the labels that it guesses can depend on the state of the automaton which is not uniquely determined for a given node of the input tree.

The solution is to remove alternation from the automata that are involved *a priori*, that is, to construct equivalent nondeterministic tree automata, and to apply the operation (intersection, projection) afterwards. The problem about the construction in [130] is that the nondeterministic Büchi tree automaton $\mathcal{B}$ that is constructed from a given alternating parity tree automaton $\mathcal{A}$ is *not* necessarily equivalent to $\mathcal{A}$, that is, in general we have $L(\mathcal{B}) \neq L(\mathcal{A})$. We have $L(\mathcal{B}) \neq \emptyset$ if, and only if, $L(\mathcal{A}) \neq \emptyset$ which is clearly sufficient for checking emptiness, but not for constructing an automaton that recognizes the intersection of two languages.

These observations are particularly interesting in the context of interaction under partial information. Dependence and independence of nondeterministic choices in runs of tree automata on universal branches of the run which are not apparent in the given input tree are intimately linked to dependence and independence of strategies on certain information in interactive situations as we consider them here. We discuss this in Section 3.2 and in Chapter 5.

3.1.2 Removal of Alternation

The main aspect of this section is removal of alternation from parity tree automata: we show how to translate an arbitrary alternating parity tree automaton into an equivalent nondeterministic parity tree automaton. We present only the main ideas of the construction.

The two main ingredients for removing alternation from parity tree automata are determinization of ω-automata and memoryless determinacy of parity games. Having these tools at hand, we can proceed as follows: Given an alternating parity tree automaton $\mathcal{A} = (\Sigma, Q, q_{\text{in}}, \delta, \text{col})$ with $\text{col} : Q \to [k]$ over Σ-labeled X-trees, consider the membership game (or semantic game) $\mathcal{G}(\mathcal{A}, t)$ of $\mathcal{A}$ on any given tree $t \in X_\Sigma$. This game is defined similar to the emptiness game but now, the tree t is given in advance and a winning strategy for player 1 is a witness that $t \in L(\mathcal{A})$. Obviously, with the tree t fixed, the game can be played on alternating automata as well: Player 1 resolves the nondeterministic choices and player 0 resolves the universal choices. More precisely, in a given node u of the tree t and state q of the automaton, player 1 chooses a conjunct ψ from $\varphi = \delta(q, t(u))$ and player 0 chooses a literal (q', j) from ψ. Then the game proceeds to node uj and state q'. In this way, the players form a path through the run of $\mathcal{A}$ on t and player 1 wins if this run is accepting. It is not hard to see that, in fact, player 1 has a winning strategy for $\mathcal{G}(\mathcal{A}, t)$ if, and only if, $t \in L(\mathcal{A})$.

So the game graph of $\mathcal{G}(\mathcal{A}, t)$ consists of positions (u, q), where $u \in X^*$ and $q \in Q$ and the initial position is always $(\varepsilon, q_{\text{in}})$. (Notice that for a complete description of all moves in the game graph, we would have to have also positions (u, ψ), where ψ is a conjunct from $\delta(t(u), q)$. Moreover, if $\delta(t(u), q)$ is not given in DNF, several alternating moves of the players might be necessary until a literal (q', j) is reached. Here, w.l.o.g., we shall stick to our more compact representation.) Since t is infinite, this game graph is infinite and, in general, it is not possible to play this game on a finite graph, since t may not admit a finite presentation. On the other hand, the game graph is not a tree: A position (u, q) may be reached via several different sequences of states (which correspond to different branches of the run ρ of $\mathcal{A}$ on t which player 1 is constructing). Notice, though, that the graph is acyclic.

Now in order to remove alternation from $\mathcal{A}$, we have to construct a *nondeterministic* parity tree automaton $\mathcal{B}$ over Σ-labeled X-trees, which takes a tree t and checks, whether player 1 has a winning strategy for $\mathcal{G}(\mathcal{A}, t)$. There are to main problems that we encounter when trying to accomplish this task. First, when running over branches $u = x_0x_1x_2\dots$ of t, there may be several paths through the game graph of $\mathcal{G}(\mathcal{A}, t)$ which correspond to u and which have to be traced simultaneously. The idea to solve this problem is very similar to the solution presented in Section 2.2.1 for (not necessarily observable) ω-regular winning conditions: We construct a universal parity ω-automaton $\mathcal{C}$ that runs over a branch of t and universally branches over all possible plays π in $\mathcal{G}(\mathcal{A}, t)$ that correspond to this branch. At the same time, $\mathcal{C}$ checks that those are all won by player 1. This is where we use our first tool, determinization of ω-automata: For a tree automaton to be able to simulate $\mathcal{C}$ on all branches, $\mathcal{C}$ must be deterministic.

Of course, only those plays should be checked by $\mathcal{C}$, that are compatible with some strategy of player 1. However, of this strategy, the tree automaton $\mathcal{B}$ is in charge, that is, $\mathcal{B}$ should guess such a strategy. To deal with this situation, we first extend the labeling of the input trees by certain information, specifying a strategy for player 1, thereby providing the automaton $\mathcal{C}$ access to the strategy. Now, there we face the second main difficulty: A strategy for player 1 is in general a complicated object – it may depend on the complete sequence of states of $\mathcal{A}$ by which a position (u, q) in the game graph is reached (that means, it depends on the branch of the run). So we cannot incorporate enough information into the labels of an input tree to encode *arbitrary* strategies for player 1.

But since $\mathcal{A}$ is a parity tree automaton, $\mathcal{G}(\mathcal{A}, t)$ is a parity game (with full information) and hence we can put our second tool into effect which yields that *memoryless* strategies suffice to win in parity games with full information. Such a memoryless strategy for player 1 for the game $\mathcal{G}(\mathcal{A}, t)$ depends only on the current position (u, q) in the game graph (that is, it is now independent of the particular branch of the run!) and yields, for any such position, one of the conjuncts of $\delta(t(u), q)$. Now, this information can be easily annotated to the labels of t: For each node $u \in X^*$, we extend the label $t(u)$ of u by a function $\sigma_u : Q \to B^+(X \times Q)$ such that $\sigma_u(q)$ is a conjunct of $\delta(t(u), q)$ for all $q \in Q$.

Now, having all these conceptions at hand, we can accomplish the construction as suggested above. First, we construct a universal parity ω-automaton $\mathcal{C}$ that runs over words $\alpha \in (X \times \Sigma \times \Gamma)^\omega$, where Γ denotes the set of all functions $\sigma : Q \to B^+(X \times Q)$. This automaton checks that all sequences of states that correspond to $\mathrm{Pr}_{X\times\Sigma}(\alpha)$ *and* are consistent with the memoryless strategy for player 1 specified by $\mathrm{Pr}_\Gamma(\alpha)$ are in acc.

This automaton can then be turned into a deterministic parity ω-automaton using determinization of Büchi ω-automata as follows: complement $\mathcal{C}$ by taking the dual automaton $\tilde{\mathcal{C}}$ of $\mathcal{C}$ (as defined below) which is a nondeterministic parity ω-automaton recognizing the complement of $L(\mathcal{C})$. Then, simulate $\tilde{\mathcal{C}}$ by a nondeterministic Büchi automaton and translate this automaton into an equivalent deterministic parity automaton. Finally, complement this automaton by complementing the parity acceptance condition which yields a deterministic parity automaton $\mathcal{D}$ such that $L(\mathcal{D}) = L(\mathcal{C})$. Notice that complementation of deterministic as well as alternating parity automata does not increase the size of the automaton. Hence, using Theorem 2.8, $\mathcal{D}$ can be seen to have at most $2^{O(|Q|\cdot k)\cdot\log(|Q|\cdot k)}$ states and $O(|Q| \cdot k)$ colors.

Now we construct a nondeterministic tree automaton $\mathcal{B}'$ that runs over $\Sigma\times\Gamma$-labeled X-trees, and simulates $\mathcal{D}$ on all branches. So $\mathcal{B}'$ accepts a tree $s \in X_{\Sigma\times\Gamma}$, if the Γ-annotations to the labels define a positional winning strategy for player 1 for $\mathcal{G}(\mathcal{A}, t)$, where t is obtained from s by deleting these Γ-annotations. Since $\mathcal{D}$ is a parity ω-automaton, $\mathcal{B}'$ is a parity tree automaton.

Finally, we use the fact that nondeterministic tree-automata are closed under projection: By deleting the Γ-components from the labels in the transitions of $\mathcal{B}'$, we obtain a nondeterministic parity tree automaton $\mathcal{B}$ that accepts a Σ-labeled X-tree t if, and only if, there *exists* a Γ-annotation of t such that the resulting tree $s \in X_{\Sigma\times\Gamma}$ is in $L(\mathcal{B}')$, cf. Proposition 3.3. So $\mathcal{B}$ accepts a tree $t \in X_\Sigma$ if, and only if, there exists a winning strategy for player 1 for $\mathcal{G}(\mathcal{A}, t)$ and hence, $L(\mathcal{B}) = L(\mathcal{A})$.

Historical Remarks. The membership game has first been described by Gurevich and Harrington in their seminal work [102] for nondeterministic Muller tree automata (where they used the famous names of automaton and pathfinder for player 0 and player 1, respectively) and has later been adapted to alternating tree automata by Muller and Schupp [152, 153]. Gurevich and Harrington gave an alternative prove of Rabin's result that nondeterministic tree automata can be effectively complemented, using a similar kind of reasoning as we have used above. However, since they used Muller tree automata, they could not proceed via memoryless determinacy but they proved that Muller games are determined with finite memory (LAR) strategies.[3] (In fact, although we didn't use this in our argumentation, determinacy was important in the proof of Gurevich and Harrington since for *complementation*, one has to construct a nondeterministic automaton that accepts a given tree t if, and only if, player 1 does *not* have a winning strategy for $\mathcal{G}(\mathcal{A}, t)$ for which they used that this is equivalent to player 0 having a winning strategy for $\mathcal{G}(\mathcal{A}, t)$.) Clearly, finite memory determinacy is enough to be able to annotate strategies to the labels of the input trees.

In their famous paper [75], Emerson and Jutla also gave a proof of Rabin's theorem. They showed that alternating tree automata and the modal μ-calculus (which is closed under negation by definition) are equally expressive and they proved that alternation can be removed from tree automata, using the argumentation presented above. (In fact, their proof of memoryless determinacy of parity games comes from this work.) Emerson and Jutla argued that, implicitly, the concept of alternation had also been used in earlier proofs of Rabin's theorem, in particular in [102]. To be able to comprehend this claim, consider the definition of a dual automaton as given below. Complementing a nondeterministic automaton $\mathcal{A}$ can be done by dualization which leads to an alternating tree automaton $\mathcal{B}$. Knowing this fact and the construction for removing alternation as described above, the constructions in the various proofs of Rabin's theorem can be seen as constructions for removing alternation from $\mathcal{B}$. However, in contrast to Gurevich and Harrington, Emerson and Jutla used alternating tree automata (introduced by Muller and Schupp in [152, 153]) explicitly and formulated the removal of alternation as a separate theorem.

In [154], Muller and Schupp gave an alternative construction for removal of alternation which neither relies on previous constructions for determinization of ω-automata nor on known results on finite memory determinacy of games. In fact, their construction includes new proofs for such results. In particular, as Vardi and Wilke state, it has an alternative construction for determinization of ω-automata built-in ([83],p.661-662). Muller and Schupp were also the first to consider the special case of universal automata explicitly. The interesting new techniques and the coherent exposition together with proper complexity results and treatment of special cases makes this paper a highly cited milestone in the theory of alternating automata.

Removal of Universality. Consider the special case where $\mathcal{A}$ is *universal*, that is, each formula $\delta(q, a)$ is a conjunction of literals. In this case, player 1 does not make any real moves in the game $\mathcal{G}(\mathcal{A}, t)$ for a given tree $t \in X_\Sigma$, so a positional (or, in fact, any) strategy for player 1 is always trivial. Hence, the Γ-annotations of the Σ-labeled X-trees are futile and can be ignored. Moreover notice that the parity tree automaton $\mathcal{B}'$ which simulates the deterministic parity ω-automaton $\mathcal{D}$ on all branches of a given tree X_Σ can do this deterministically and since we don't have Γ-annotations, the projection step does not apply. Therefore, the nondeterministic parity tree automaton $\mathcal{B}$ with $L(\mathcal{B}) = L(\mathcal{A})$ is actually deterministic. Hence, the construction above yields, as a special case, that universal parity tree automata and deterministic parity tree automata are equally expressive. This result has first been formulated explicitly by Muller and Schupp [154]. So removal of universality from parity tree automata is basically just determinization of ω-automata. Hence, solving the emptiness problem for such automata essentially reduces to determinizing an ω-automaton and solving a parity game with *full information* on a finite graph.

Complementing Alternating Automata. Rabin [176] proved that *nondeterministic* Muller tree automata can be effectively complemented which is a deep result that also relies on

[3]It should be noted that Gurevich and Harrington themselves refer to a manuscript sent to them by Büchi which apparently contains the first proof of this fact, cf. [102].

determinization of ω-automata. On the other hand, an *alternating* Muller tree automaton $\mathcal{A} = (\Sigma, Q, q_{\text{in}}, \delta, \mathcal{F})$ can be complemented quite easily by taking the *dual* automaton $\tilde{\mathcal{A}}$ of $\mathcal{A}$ which is obtained from $\mathcal{A}$ by dualizing the transition function δ to $\tilde{\delta}$ and complementing the acceptance condition acc to $Q^\omega \setminus \text{acc}$. Dualizing the transition function means that we exchange $\wedge$ with $\vee$ in the formulas $\delta(q, a) \in B^+(Q \times X)$. Moreover, the complement of a Muller condition is again a Muller condition (with $\tilde{\mathcal{F}} = 2^Q \setminus \mathcal{F}$). Likewise, the complement of a parity condition is also again a parity condition (with a shift of the coloring by one) and so the reasoning here applies to parity conditions just as well.

Now it is easy to see that the semantic game $\mathcal{G}(\tilde{\mathcal{A}}, t)$ for a given Σ-labeled X-tree t is obtained from the game $\mathcal{G}(\mathcal{A}, t)$ also by dualizing the game to $\tilde{\mathcal{G}}(\mathcal{A}, t)$, that is, by swapping ownership of the vertices in the game and by complementing the winning condition. Notice that technically, this construction is only sound if we do not assume that the transition formulas of an alternating tree automaton are in DNF: If the formulas $\delta(q, a)$ are in DNF then the formulas $\tilde{\delta}(q, a)$ are in CNF and $\mathcal{G}(\tilde{\mathcal{A}}, t) = \tilde{\mathcal{G}}(\mathcal{A}, t)$ holds in general only if we consider the game $\mathcal{G}(\tilde{\mathcal{A}})$ as being played directly on the formulas $\tilde{\delta}(q, a)$ without transforming them into DNF. So for now, we drop the assumption. Notice that then, for a given position $(u, q) \in X^* \times Q$ of the game $\mathcal{G}(\mathcal{A}, t)$, player 0 and player 1 may have to take several turns until a literal of $\delta(t(u), q)$ is reached.

Now it is easy to see that in the game $\tilde{\mathcal{G}}(\mathcal{A}, t) = \mathcal{G}(\tilde{\mathcal{A}}, t)$ player 0 has a winning strategy if, and only if, player 1 has a winning strategy in $\mathcal{G}(\mathcal{A}, t)$ and so, since Muller games as well as parity games are *determined*, player 0 has a winning strategy for $\mathcal{G}(\tilde{\mathcal{A}}, t)$ if, and only if, player 1 does *not* have a winning strategy for $\mathcal{G}(\mathcal{A}, t)$. Since player 1 has a winning strategy for $\mathcal{G}(\tilde{\mathcal{A}}, t)$ if, and only if, $t \in L(\mathcal{A})$, it follows that $L(\tilde{\mathcal{A}}) = X_\Sigma \setminus L(\mathcal{A})$.

This complementation construction for alternating tree automata has first been described by Muller and Schupp [152, 153]. In a sense, this shows that alternating tree automata are the natural logical completion of the existential fragment of the automaton model to the full automaton model where both existential and universal branches are possible and allowed to alternate in an arbitrary fashion. As we have argued, this enables easy negation by dualization and allows a smoother explanation of the proofs of Rabin's theorem by making the implicit use of alternation explicit, cf. [75]. On the other hand, on trees, the concept is not as coherent as it is on words. While alternating Turing machines and alternating automata on words have the property that the dual of a nondeterministic automaton is always a universal automaton (and vice versa) this is no longer true for automata on trees. As we have mentioned, nondeterministic tree automata are also alternating in that they can send copies to *all* directions of the tree. Consequently, the dual of a nondeterministic automaton is a (seriously) alternating automaton in general. On the other hand, the dual of a universal automaton *is* a nondeterministic automaton (a special one that sends, in each step of a run, a copy to at most one direction).

After this brief exposition of some fundamental aspects of finite automata on infinite trees we assemble some main statements which help to apprehend the expressive power of tree automata and which can be used as tools for solving synthesis problems under partial information.

Theorem 3.8. *[75, 154], see also [100]*
Let $\mathcal{A} = (\Sigma, Q, q_{in}, \delta, acc)$ be an alternating tree automaton.

(1) If $\mathcal{A}$ is a parity automaton with $col\colon Q \to [k]$, one can construct a nondeterministic parity tree automaton $\mathcal{B}$ with at most $2^{O(|Q| \cdot k \cdot \log(|Q| \cdot k)}$ states and at most $O(|Q| \cdot k)$ colors such that $L(\mathcal{A}) = L(\mathcal{B})$. If $\mathcal{A}$ is universal, then $\mathcal{B}$ is deterministic.

(2) If $\mathcal{A}$ is a Muller automaton with $\mathcal{F} \subseteq 2^Q$, one can construct an alternating parity automaton with at most $|Q| \cdot |Q|!$ states and at most $2 \cdot |Q|$ colors. The same holds for deterministic, universal and nondeterministic automata.

(3) Alternating parity tree automata are closed under complement. This does neither hold for deterministic parity automata nor for alternating Büchi automata.

(4) Alternating parity tree automata are strictly more expressive than alternating Büchi tree automata. On the other hand, deterministic parity tree automata and nondeterministic Büchi tree automata are incomparable.

Proof. We have proven (1) in the exposition above. For (2), we use the LAR-construction just as for games and ω-automata. (Or, to use the known construction more directly, take a Muller ω-automaton recognizing acc, turn it into an equivalent parity ω-automaton $\mathcal{C}$ and then take the synchronous product of $\mathcal{A}$ and $\mathcal{C}$.) Notice that the construction does not affect the type of the automaton. As to (3), complementation of alternating parity tree automata has been described above. To see that the class of deterministically parity (Büchi) recognizable tree languages is not closed under complementation, consider the language $\mathcal{T}_{\exists b}$ (see Section 3.1.1. The complement of $\mathcal{T}_{\exists b}$ consists only of the constant tree t_a with $t_a(u) = a$ for all $u \in \mathbb{B}^*$ so it is clearly recognizable by a deterministic parity (Büchi and, in fact, even reachability) automaton, while by Proposition 3.1, the complement of L is not recognizable by any deterministic tree automaton.

To prove proposition (4), consider the tree language $\mathcal{T}_{|b|<\omega}$. By Proposition 3.1 this language can be recognized by a deterministic parity tree automaton but not by a nondeterministic Büchi automaton. As we have mentioned, alternating Büchi tree automata can be translated into nondeterministic Büchi tree automata, so $\mathcal{T}_{|b|<\omega}$ is not recognizable by an alternating Büchi tree automaton as well. Hence, there is a language that is recognizable by a deterministic parity tree automaton but not recognizable by an alternating Büchi automaton. This proves one direction of the second part of the statement, and, since alternating parity tree automata are a generalization of alternating Büchi tree automata, it also proves the first part of the statement. On the other hand, again by Proposition 3.1, $\mathcal{T}_{\exists b}$ is a tree language that is recognizable by a nondeterministic reachability automaton, and hence also by a nondeterministic Büchi automaton, but not recognizable by any deterministic tree automaton which proves the other direction of the second part of the statement. $\square$

3.1.3 Pushdown Tree Automata

An *alternating pushdown tree automaton* over Σ-labeled X-trees has the form

$$\mathcal{A} = (\Sigma, \Gamma, Q, q_{\text{in}}, \delta, \perp, \text{acc})$$

where Σ, $Q \ni q_{\text{in}}$ and acc are as before, cf. Section 3.1.1.

Moreover, Γ is the finite stack alphabet, $\perp \notin \Gamma$ is the initial stack symbol and δ is the transition function which now also depends on the top stack symbol and performs a stack operation during each transition. As for pushdown ω-automata we assume that a stack operation increases the height of the stack by at most one. Furthermore, since we will use pushdown tree automata to deal with context-free winning conditions in games, the automata should be able to simulate (deterministic) pushdown ω-automata along all paths of a given input tree, hence, they need also ε-transitions. Notice that an alternating pushdown automaton could be defined in such a way that it has the possibility to send certain copies to successors and certain other copies via ε-transitions to the current node. However, for simplicity we restrict the use of ε-transitions in that an alternating pushdown automaton has to decide whether it takes an ε-transition or a transition that sends copies to successors and, if it takes an ε-transition, then it is required to send exactly one state. Clearly this restricted use of ε-transitions is sufficient to simulate deterministic pushdown ω-automata. So the transition function δ is the union $\delta = \delta_{\downarrow} \cup \delta_{\circlearrowleft}$ with

- $\delta_{\downarrow} : Q \times \Gamma_{\perp} \times \Sigma \to B^+(X \times Q \times \Gamma_{\perp}^{\leq 2})$
- $\delta_{\circlearrowleft} : Q \times \Gamma_{\perp} \to 2^{Q \times \Gamma_{\perp}^{\leq 2}}$.

Similar as for pushdown ω-automata, we write δ also as $\delta : Q \times \Gamma_{\perp} \times \Sigma_{\varepsilon} \to B^+(X \times Q \times \Gamma_{\perp}^{\leq 2}) \cup 2^{Q \times \Gamma_{\perp}^{\leq 2}}$. All the basic definitions and observations concerning the set $B^+(X \times Q \times \Gamma_{\perp}^{\leq 2}$

of positive Boolean formulas over variables from $X \times Q \times \Gamma_\perp^{\leq 2}$ carry over from Section 3.1.1. As for pushdown ω-automata we assume that the initial stack symbol $\perp$ can neither be deleted from nor written to the pushdown stack. That is, if $\varphi \in \delta(q, \perp, a)$ and $(\downarrow_x, q', \gamma)$ is a literal in φ, then $\gamma \in \Gamma^{\leq 1}\perp$, and analogously for ε-transitions $\delta(q, \perp, \varepsilon)$.

A *run* of $\mathcal{A}$ on a Σ-labeled X-tree $t : X^* \to \Sigma$ is a *not necessarily full* Σ_r-labeled $\mathbb{N}$-tree $\rho : T \to \Sigma_r$ with $\Sigma_r = X^* \times Q \times \Gamma^*\perp$ such that the following conditions hold.

(1) $\varepsilon \in T$ and $\rho(\varepsilon) = (\varepsilon, q_{\text{in}}, \perp)$

(2) if $v \in T$ with $\rho(v) = (u, q, A\gamma)$ then one of the following conditions holds:

  (2.1) there is a unique successor $v \cdot 0$ of v in T such that $\rho(v \cdot 0) = (u, q', \gamma'\gamma)$ for some $(q', \gamma') \in \delta(q, A, \varepsilon)$.

  (2.2) the set of successors of v in T is $\{v \cdot j \mid j \in [l]\}$ for some $l \in \mathbb{N}$ such that for all $j \in [l]$ we have $\rho(v \cdot j) = (u \cdot x_j, q_j, \gamma_j\gamma)$ where $\psi = \{(x_0, q_0, \gamma_0), \ldots, (x_{l-1}, q_{l-1}, \gamma_{l-1})\} \subseteq X \times Q \times \Gamma_\perp$ is a conjunct in $\varphi = \delta(q, A, t(u))$.

The run is called *complete* if there is no infinite path $v_0, v_1 \ldots$ of nodes in T with $v_j \sqsubseteq v_{j+1}$ for all j such that there is some $u \in X^*$ with $\text{Pr}_{X^*}(v_j) = u$ for some $u \in X^*$ for all j. As for pushdown automata on words, this excludes the possibility of infinite sequences of ε-transitions. A pushdown tree automaton $\mathcal{A}$ is said to have the continuity property, if each run of $\mathcal{A}$ on some input tree is complete. Remind that, w.l.o.g., we consider only deterministic parity pushdown ω-automata which have the continuity property, hence, for our concerns pushdown tree automata which have the continuity property suffice as well. So from now on we assume that a pushdown tree automaton *has* the continuity property.

The run is called *accepting* if, for each infinite path π through ρ, we have $\text{Pr}_Q(\pi) \in \text{acc}$. (Notice that the run is complete by the assumption that $\mathcal{A}$ has the continuity property.) So the acceptance of a run depends only on the sequence of states and not on the stack contents. As before we consider parity pushdown tree automata and Muller pushdown tree automata where acc is defined via a parity condition given by $\text{col} : Q \to [k]$ and a Muller condition given by $\mathcal{F} \subseteq 2^Q$, respectively. As before, $\mathcal{A}$ accepts a tree $t \in X_\Sigma$ if there is an accepting run of $\mathcal{A}$ on t and $L(\mathcal{A}) = \{t \in X_\Sigma \mid \mathcal{A} \text{ accepts } t\}$.

Now assume w.l.o.g. that $X = [d]$ for some $d \in \mathbb{N}$. The automaton $\mathcal{A}$ is called *nondeterministic* if, for all (q, A, a), $\delta(q, A, a)$ has the form

$$\bigvee_{j=0}^{m-1} (\downarrow_0, q_0^j, \gamma_0^j) \wedge \ldots \wedge (\downarrow_{d-1}, q_{d-1}^j, \gamma_{d-1}^j)$$

for some $m \in \mathbb{N}$. Notice that nondeterminism is a restriction to the function $\delta_\downarrow$ only while $\delta_\circlearrowleft$ has the same format as for alternating pushdown automata. This is, of course, due to the fact that we have already started with alternating pushdown automata that can use ε-transitions only in a restricted way. The automaton $\mathcal{A}$ is called *universal*, if for all $(q, A, a) \in Q \times \Gamma_\perp \times \Sigma$ we have $|\delta(q, A, a)| + |\delta(q, A, \varepsilon)| \leq 1$. The automaton is called deterministic if it is both, nondeterministic and universal.

We represent the transition function δ of a nondeterministic tree automaton also as a relation $\Delta = \Delta_\downarrow \cup \Delta_\circlearrowleft$ with $\Delta_\downarrow \subseteq Q \times \Gamma_\perp \times \Sigma \times (2^Q \times \Gamma_\perp^{\leq 2})^d$ and $\Delta \subseteq Q \times \Gamma_\perp \times (2^Q \times \Gamma_\perp^{\leq 2})$. However, due to the ε-transitions, a run of a nondeterministic automaton can now *not* be viewed as a $Q \times \Gamma^*\perp$-labeled X-tree. Moreover, while for a run ρ of an alternating tree automaton on some Σ-labeled X-tree t, all nodes of ρ that correspond to the same node $u \in X^*$ of t are *on the same level* of ρ, this does not hold anymore for pushdown tree automata, even for deterministic ones. On the other hand, for a run ρ of a *nondeterministic* pushdown tree automaton on some Σ-labeled X-tree t, it holds that if two nodes v_1 and v_2 in ρ correspond to the same node $u \in X^*$ of t, then v_1 and v_2 are *not* on the same level of ρ.

Notice that for nondeterministic pushdown tree automata, the assumption that transitions to successor and ε-transitions cannot be mixed is indispensable. If a nondeterministic pushdown tree automaton could perform a transition to successors and, at the same time, also perform an ε-transition, then it could easily simulate an alternating tree automaton: It could send a copy to each successor and at the same time perform an ε-transition. Then it could use the copy that was sent by the ε-transition, to send another copy to each successor and so on. Also notice that for a nondeterministic pushdown tree automaton $\mathcal{A}$, the assumption that a run $\rho : T \to \Sigma_r$ of $\mathcal{A}$ on some Σ-labeled X-tree t is complete is equivalent to the assumption that for all $u \in X^*$ there exists some $v \in T$ such that $\mathrm{Pr}_{X^*}(\rho(v)) = u$.

In the same way as for pushdown ω-automata we define 1-counter and realtime pushdown tree automata: $\mathcal{A}$ is called 1-counter automaton if $|\Gamma| = 1$ and it is called realtime automaton if it does not contain any epsilon transitions, that is, $\delta = \delta_{\downarrow}$. As for ω-languages we define (deterministic) context-free, 1-counter and realtime tree languages.

Properties of Pushdown Tree Automata. We will not discuss the properties of pushdown tree automata in detail, but we make some general remarks and collect some facts which are important to us. A very simple observation is that, just as deterministic parity tree automata, deterministic parity pushdown tree automata can simulate deterministic parity pushdown ω-automata along the paths of a given tree. However, a nondeterministic parity pushdown tree automaton can *not* simulate a nondeterministic parity pushdown ω-automaton along the paths of a given tree in general. Remind that nondeterministic parity pushdown ω-automata are strictly more expressive than deterministic ones. In fact, it can be seen that there are context-free languages $L \subseteq \Sigma^\omega$ such that the set of all Σ-labeled Σ-trees t with $\pi \in L$ for any path π in t is not a context-free tree language. This can be shown, for example, by a reduction from the universality problem for nondeterministic context-free ω-languages, which is undecidable [81].

The most important property of pushdown tree automata is that the emptiness problem for nondeterministic parity pushdown tree automata is decidable in exponential time. It has been shown in [106] that the nonemptiness problem for nondeterministic Büchi pushdown tree automata can be solved in triple exponential time and in [125] this has been improved to parity acceptance and single exponential time:

Theorem 3.9. *[125] The emptiness problem for nondeterministic parity pushdown tree automata can be solved in exponential time.*

On the other hand, in general, the emptiness problem for alternating parity pushdown tree automata is undecidable, even if we consider only *universal* automata. This follows directly from the aforementioned fact that the universality problem for nondeterministic ω-*languages* is undecidable: To recognize arbitrary context-free ω-languages we don't need ε-transitions (cf. Section 2.2.2) and the dual of a nondeterministic realtime parity pushdown ω-*automaton* is a universal realtime parity pushdown ω-automaton.

For arbitrary alternating automata undecidability of the emptiness problem can also be seen from the fact that the emptiness problem for the intersection of context-free languages of finite words is already undecidable. Intuitively, solving the emptiness problem for the intersection of two context-free languages makes it necessary to let two finite state devices, each of which has access to a *separate* stack memory, run in parallel over the same object, which somehow establishes a connection between the devices. This is quite reminiscent of pushdown machines with two stack memories which can, in turn, simulate Turing machines and hence have an undecidable emptiness problem. Now an alternating tree automaton can easily simulate two given tree automata in parallel, recognizing the intersection of the given languages, cf. Proposition 3.2.

This also demonstrates that there is no effective translation of universal parity pushdown tree automata to nondeterministic parity pushdown tree automata. In fact, the expressive power of universal and nondeterministic parity pushdown tree automata is incomparable and hence, alternating parity pushdown tree automata are strictly more expressive than the nondeterministic model. Moreover, while nondeterministic parity pushdown tree automata are closed under union, they are neither closed under complement nor under intersection. We have described the problem with intersection already above. Moreover, if the automata were closed under complement, then we could, for any universal automaton, take the dual of that automaton, which is a nondeterministic parity pushdown tree automaton and then we could complement this automaton and obtain a nondeterministic parity pushdown tree automaton which is equivalent to the given universal automaton. This however, is not possible in general.

A property that *does* hold is that the intersection of a *regular* tree language and a *context-free* tree language is again a context-free tree language and hence, the emptiness problem for the intersection of nondeterministic parity tree automata and nondeterministic parity pushdown tree automata is decidable. The rest of this section contains the corresponding product construction.

Let $\mathcal{A} = (\Sigma, Q^{\mathcal{A}}, q^{\mathcal{A}}_{\text{in}}, \delta^{\mathcal{A}}, \text{col}^{\mathcal{A}})$ be a nondeterministic parity tree automaton and let $\mathcal{B} = (\Sigma, \Gamma, \perp, Q^{\mathcal{B}}, q^{\mathcal{B}}_{\text{in}}, \delta^{\mathcal{B}}, \text{col}^{\mathcal{B}})$ be a nondeterministic parity pushdown tree automaton, which both run on Σ-labeled $[d]$-trees for some $d \in \mathbb{N}$. Notice that due to the ε-transitions, a direct synchronous product as in Proposition 3.2 cannot be applied. Instead, while $\mathcal{B}$ performs ε-transitions, $\mathcal{A}$ has to be kept waiting. This product has a weak form of asynchronism with one component idling from time to time while the other component performs certain steps but still, the two components are always in the same location of the tree.

Also notice that with the parity acceptance condition we run into a similar problem as for the product of two nondeterministic parity tree automata: The accepting sequences of states of the product automaton cannot be directly defined by a parity condition using the colorings of the given automata. Hence, we again take a detour through Muller pushdown tree automata. It is clear that the LAR-construction works here as well, so we can transform the automaton into a nondeterministic parity pushdown tree automaton.

$$\mathcal{A} \times \mathcal{B} = (\Sigma, \Gamma, \perp, Q, q_{\text{in}}, \delta, \mathcal{F})\}$$

- $Q = Q^{\mathcal{A}} \times Q^{\mathcal{B}}$ and $q_{\text{in}} = (q^{\mathcal{A}}_{\text{in}}, q^{\mathcal{B}}_{\text{in}})$
- $\mathcal{F}$ consists of those subsets F of Q such that $\min\{\text{col}^{\mathcal{A}}(\text{Pr}_{Q^{\mathcal{A}}}(\overline{q})) \mid \overline{q} \in F\}$ is even and $\min\{\text{col}^{\mathcal{B}}(\text{Pr}_{Q^{\mathcal{B}}}(\overline{q})) \mid \overline{q} \in F\}$ is even
- for all $(p, q) \in Q$, all $a \in \Sigma$ and all $A \in \Gamma_{\perp}$ we define

$$\delta((p,q), a, A) = \bigvee_{[\psi^{\mathcal{A}} \in \delta^{\mathcal{A}}(p,a)]} \bigvee_{\psi^{\mathcal{B}} \in \delta^{\mathcal{B}}(q,a,A)]} \bigwedge_{j \in [d]} (\downarrow_j, (p_j, q_j), \gamma_j)$$

  where $(\downarrow_j, p_j) \in \psi^{\mathcal{A}}$ and $(\downarrow_j, q_j, \gamma_j) \in \psi^{\mathcal{B}}$
- for all $(p, q) \in Q$ and all $A \in \Gamma_{\perp}$ we define

$$\delta((p,q), A, \varepsilon) = \{((p, q'), \gamma) \mid (q', \gamma) \in \delta^{\mathcal{B}}(q, A)\}$$

Proposition 3.10. $L(\mathcal{A} \times \mathcal{B}) = L(\mathcal{A}) \cap L(\mathcal{B})$.

Proof. First, since $\mathcal{B}$ has the continuity property (and $\mathcal{A}$ has no ε-transitions), it is easy to see that $\mathcal{A} \times \mathcal{B}$ has the continuity property as well.

Let t be an Σ-labeled $[d]$-tree. First, assume that $t \in L(\mathcal{A}) \cap L(\mathcal{B})$, that means, there is an accepting run $\rho^{\mathcal{A}} : X^* \to Q^{\mathcal{A}}$ of $\mathcal{A}$ on t and an accepting run $\rho^{\mathcal{B}} : T \to X^* \times Q^{\mathcal{B}} \times \Gamma^*\perp$ of $\mathcal{B}$ on t, where $T \subseteq \mathbb{N}^*$. We define $\rho : T \to X^* \times (Q^{\mathcal{A}} \times Q^{\mathcal{B}}) \times \Gamma^*\perp$ by $\rho(v) = (u, (p, q), \gamma)$ where $(u, q, \gamma) = \rho^{\mathcal{B}}(v)$ and $p = \rho^{\mathcal{A}}(u)$. It is easy to see that, by definition of $\mathcal{A} \times \mathcal{B}$, ρ is an accepting run of $\mathcal{A} \times \mathcal{B}$ on t which yields $t \in L(\mathcal{A} \times \mathcal{B})$.

Now let conversely $t \in L(\mathcal{A} \times \mathcal{B})$, that means, there is an accepting run $\rho : T \to X^* \times Q \times \Gamma^*\perp$ of $\mathcal{A} \times \mathcal{B}$ on t, where $T \subseteq \mathbb{N}^*$. First observe that if $v, v' \in T$ (with $|v| \leq |v'|$) such that $\rho(v) = (u, (p, q), \gamma)$ and $\rho(v') = (u, (p', q'), \gamma')$ then, by definition of $\mathcal{A} \times \mathcal{B}$, $v \sqsubseteq v'$ and v' is reached in ρ from v by a sequence of ε-transitions of $\mathcal{A} \times \mathcal{B}$. Hence, $v' = v0^*$ and $p' = p$.

We define $\rho^{\mathcal{A}} : X^* \to Q^{\mathcal{A}}$ by $\rho^{\mathcal{A}}(u) = \text{Pr}_{Q^{\mathcal{A}}}(\rho(v))$ for some $v \in T$ such that $\text{Pr}_{X^*}(\rho(v)) = u$. As we have just shown, p is independent of the particular choice of v. To see that $\rho^{\mathcal{A}}$ is in fact a run of $\mathcal{A}$ on t, consider some $u \in X^*$ with $\rho^{\mathcal{A}}(u) = p$ and consider some $v \in T$ with $\mathrm{Pr}_{X^*}(\rho(v)) = u$. Since $\mathcal{B}$ has the continuity property, ρ is complete, so there is a maximal $m \in \mathbb{N}$ such that $\mathrm{Pr}_{X^*}(\rho(v0^m)) = u$. Let $w := v0^m$ and $\rho(w) = (u, (p, q), \gamma)$. The set of successors of $w := v0^m$ in T is $\{w \cdot j \mid j \in [d]\}$ such that for all $j \in [d]$ we have $\rho(w \cdot j) = (u \cdot j, (p_j, q_j), \gamma_j\gamma)$ where $\psi = \{(\downarrow_j, (p_j, q_j), \gamma_j) \mid j \in [d]\}$ is a conjunct in $\varphi = \delta((p, q), A, t(u))$. By definition of δ, $\{(\downarrow_j, p_j) \mid j \in [d]\}$ is a conjunct in $\varphi^{\mathcal{A}} = \delta^{\mathcal{A}}(p, t(u))$. Therefore, $\rho^{\mathcal{A}}$ is a run of $\mathcal{A}$ on t. Now consider any path π through X^*. Then there is a unique path π' in T such that π is obtained from π' by projecting $\rho(\pi')$ to the X^*-component and contracting all multiple occurrences of nodes $u \in X^*$ to a single occurrence. Then we have $\mathrm{Inf}(\rho^{\mathcal{A}}(\pi)) = \mathrm{Inf}(\mathrm{Pr}_{Q^{\mathcal{A}}}(\rho(\pi')))$ and since ρ is accepting this yields that $\mathrm{Inf}(\rho^{\mathcal{A}}(\pi)) \in \mathcal{F}$ which, by definition of $\mathcal{F}$ means that $\min \mathrm{col}(\mathrm{Inf}(\rho^{\mathcal{A}}(\pi)))$ is even. So $\rho^{\mathcal{A}}$ is accepting and hence, $t \in L(\mathcal{A})$.

Moreover, for $v \in T$ we define $\rho^{\mathcal{B}}(v) = (u, q, \gamma)$ where $\rho(v) = (u, (p, q), \gamma)$ for some p. Since $\rho : T \to X^* \times (Q^{\mathcal{A}} \times Q^{\mathcal{B}}) \times \Gamma^*\bot$ is an accepting run of $\mathcal{A} \times \mathcal{B}$, the definition of $\mathcal{A} \times \mathcal{B}$ easily yields that $\rho^{\mathcal{B}} : T \to X^* \times Q^{\mathcal{B}} \times \Gamma^*\bot$ is an accepting run of $\mathcal{B}$ on t. (Notice that, in particular, ρ and $\rho^{\mathcal{B}}$ are defined over the same tree $T \subseteq \mathbb{N}^*$.) Hence, $t \in L(\mathcal{B})$. $\square$

3.2 Tree Automata and Partial Information

We have already hinted at the intimate connection between alternating tree automata and interaction under partial information in Section 3.1. This was explicitly suggested by Kupferman and Vardi [127] and is underpinned by various successful applications of this concept to synthesis under partial information [127, 129, 137, 80]. Some intuitive evidence on the connection between tree automata and partial information has also been given in Section 3.1.1 where we have argued why constructions like product of automata and projection cannot be applied directly to alternating (or, in fact, even universal) automata. In this Section, we study the relation between alternating tree automata and interaction under partial information more closely. We present basic solutions of strategy problems under partial information which use universal or general alternating tree automata in an elegant way and we discuss how the emptiness problem for such automata can in turn be reduced to the strategy problem for games with imperfect information.

To start with, let us review how tree automata can be used to solve games with full information and *linear* specifications. For this, consider Church's setting, where we have the environment p_0 and a single controller p_1, which communicate via a channel c_{01} from p_0 to p_1 and, additionally, p_1 has an external output channel. The alphabets on the channels are both the boolean alphabet $\mathbb{B}$. Formally, this defines the distributed system

$$\mathcal{D}_1 = (\mathfrak{A}, (\Sigma_c)_{c \in C})$$

with $\mathfrak{A} = (C, r, w)$ where

- $C = \{01, 1\}$
- $w(01) = p_0$ and $r(01) = p_1$
- $w(1) = p_1$ and $r(1) = p_0$

Moreover, $\Sigma_c = \mathbb{B}$ for all channels $c \in C$.

Now consider some regular specification $L \subseteq (\Sigma^{\mathcal{D}_1})$. A strategy for p_1 is a Σ_1-labeled Σ_{01}-tree $t : \Sigma_{01}^* \to \Sigma_1$, cf. Section 3.1. The strategy is winning if it is satisfies the following condition:

(W) each $\alpha = \alpha_{01}{}^\frown\alpha_1 \in (\Sigma^{\mathcal{D}_1})^\omega$ which is consistent with t is in L.

Notice that we could represent a strategy for p_1 also in the same way as we represent joint strategies for the grand coalition (which now consists only of p_1), that is, as Σ_1-labeled $(\Sigma^{\mathcal{D}_1})^*$-trees $t : (\Sigma^{\mathcal{D}_1})^* \to \Sigma_1$. Since $\Sigma^{\mathcal{D}_1} = \Sigma_{01} \times \Sigma_1$ and p_1 can always deduce his own actions from his strategy and the sequence of signals it has received via c_{01}, there is no harm done in providing p_1 with this additional information in the tree representation of a strategy. On the other hand, since the branches of such a strategy t which correspond to histories that are *consistent* with the strategy are uniquely determined by the labeling of the tree we can skip all other branches and we prefer to use the more evident representation of a strategy as a Σ_{01}-tree.

The key observation for solving the controller problem for $\mathcal{D}_1$ with tree automata is that

$$\mathcal{T}_{\text{win}} = \{t \in \mathbb{B}_{\mathbb{B}} \,|\, t \text{ is a winning strategy for } p_1\}$$

is a regular tree language. $\mathcal{T}_{\text{win}}$ can even be recognized by a *deterministic* parity tree automaton: Construct a deterministic parity automaton $\mathcal{B} = (\mathbb{B} \times \mathbb{B}, Q, q_{\text{in}}, \delta, \text{col})$ with $L(\mathcal{B}) = L$. Then construct a deterministic parity tree automaton $\mathcal{A}$ which simulates $\mathcal{B}$ on all $\alpha \in (\Sigma^{\mathcal{D}_1})^\omega$ that are consistent with the given input tree t. More precisely, $\mathcal{A} = (\mathbb{B}, Q, q_{\text{in}}, \delta^{\mathcal{A}}, \text{col})$ has transitions $\delta(q, b) = (\downarrow_0, \delta(q,(0,b))) \wedge (\downarrow_1, \delta(q,(1,b)))$ for all $q \in Q$ and all $b \in \mathbb{B}$. So $\mathcal{A}$ always feeds both *directions* 0 and 1 into the automaton $\mathcal{B}$ which are the possible actions of p_0 in this situation, but as corresponding signal from Σ_1 it takes the *label* of the current node which is the action chosen by p_1 in this situation, according to the strategy t. So $L(\mathcal{A}) = \mathcal{T}_{\text{win}}$ and therefore, p_1 has a winning strategy for $(\mathcal{D}_1, L)$ if, and only if, $L(\mathcal{A}) \neq \emptyset$. We already know that the latter problem is decidable and if $L(\mathcal{A}) \neq \emptyset$ then we can construct a regular tree $t \in L(\mathcal{A})$ which yields a finite state implementation of a winning strategy for p_1. As we have already mentioned in Section 3.1, this approach to Church's problem was originally presented by Rabin [177].

Notice that while the alphabet of the automaton $\mathcal{B}$ recognizing L is $\mathbb{B} \times \mathbb{B}$, the labeling alphabet of the tree automaton $\mathcal{A}$ is $\mathbb{B}$. Clearly, the automaton $\mathcal{B}$ needs to access both components of a system run of $\mathcal{D}_1$ to be able to recognize the specification of $\mathcal{D}_1$ which it can only do via the input alphabet. On the other hand, the tree automaton $\mathcal{A}$ has implicit access to the $\Sigma_{\text{in}}^{p_1}$-component of the system runs via the *directions* of the tree which we have used to simulate $\mathcal{B}$ on the system runs along the paths of a given input tree t. In a sense, we have made implicit use of the x-ray operator, cf. Section 3.1.1. We will do this frequently in our tree automata constructions.

Now consider a simple extension of $\mathcal{D}_1$ to an interactive scenario where p_1 has *partial information*: Let the distributed system $\mathcal{D}_{1,h}$ be obtained from $\mathcal{D}_1$ by adding a *hidden channel* c_0 of the environment to C, that is, $C = \{0, 01, 1\}$ and $w(0) = r(0) = p_0$ and we define $\Sigma_0 = \mathbb{B}$. All the other components are as before. Now consider again some (regular) specification $L \subseteq (\Sigma^{\mathcal{D}_{1,h}})^\omega$. A *strategy* for p_1 is, just as before, Σ_1-labeled Σ_{01}-tree $t : \Sigma_{01}^* \to \Sigma_1$ and the strategy is winning, if it satisfies the following condition:

(W) each $\alpha = \alpha_0 {}^\frown \alpha_{01} {}^\frown \alpha_1 \in (\Sigma^{\mathcal{D}_{1,h}})^\omega$ which is consistent with t is in L.

Now, if we try to solve the controller problem for $\mathcal{D}_{1,h}$ just as for the system $\mathcal{D}_1$ we run into a problem: A deterministic automaton which runs over a Σ_1-labeled Σ_{01}-tree t cannot check the condition (W) since it has no access to the Σ_0-component of the system runs α. The automaton that we have constructed above had access to the Σ_{01}-component of the system runs via the *directions* of the input tree. In the representation of a strategy of p_1 for $\mathcal{D}_{1,h}$ as a Σ_1-labeled Σ_{01}-tree, though, the Σ_0-directions are not present. However, we can also represent a strategy for p_1 for $\mathcal{D}_{1,h}$ as a Σ_1-labeled $\Sigma_0 \times \Sigma_{01}$-tree $t : (\Sigma_0 \times \Sigma_{01})^* \to \Sigma_1$. Now this tree contains the Σ_0-components of a system run as directions and a deterministic tree automaton running over t could check condition (W). On the other hand, a Σ_1-labeled $\Sigma_0 \times \Sigma_{01}$-tree has to fulfill the consistency condition (S2) in order to actually represent a strategy for p_1, cf. Section 2.1. As we have mentioned before, (S2) *cannot* be checked by a tree automaton, that is, the set of all Σ_1-labeled $\Sigma_0 \times \Sigma_{01}$-trees that satisfy condition (S2) is not regular, cf. Section 3.1.

Intuitively, this is clear once we know that the set $\mathcal{T}_{t^0=t^1}$ as defined in Section 3.1 is not regular. However, since this observation is one of the most essential aspects of synthesis under partial information, we give a short explicit proof sketch. To simplify the reasoning let us redefine Σ_{01} from $\mathbb{B}$ to the single element alphabet $\{|\}$. A strategy for p_1 can then be described as a function $\sigma : \Sigma_0^* \to \mathbb{B}$ which satisfies condition (S2). That is, σ takes the sequence of hidden signals of the environment p_0 as input but at the same time it has to be independent of the particular sequence of signals it reads. Hence, the value of σ only depends on the length of the input. In other words, the set of all Σ_1-labeled Σ_0-trees which satisfy condition (S2) (that is, which are strategies of p_1), is just the set of all $\mathbb{B}$-labeled $\mathbb{B}$-trees t such that the labeling of t is constant on each level. Now, it is not hard to see that the language $\mathcal{T}_{t(u)=t(|u|)} = \{t \in \mathbb{B}_{\mathbb{B}} \mid t(u) = t(v) \text{ for all } u, v \in \mathbb{B}^* \text{ with } |u| = |v|\}$ is not recognizable by any tree automaton. (The proof is essentially the same as for the set $\mathcal{T}_{t^0=t^1}$: A nondeterministic tree automaton $\mathcal{A}$ that accepts all trees in $\mathcal{T}_{t(u)=t(|u|)}$ accepts in particular the tree t_0 with $t_0(u) = 0$ for all $u \in \mathbb{B}^*$ and the tree t_1 with $t_1(\varepsilon) = 0$ and $t_1(u) = 1$ for all $u \in \mathbb{B}^* \setminus \{\varepsilon\}$. Now if ρ_j is an accepting run of $\mathcal{A}$ on t_j for $j = 0, 1$, by combining the subtree of ρ_0 which is rooted in 0 and the subtree of ρ_1 which is rooted in 1 we obtain an accepting run of $\mathcal{A}$ on the tree t_{01} with $t_{01}(u) = 1$ for $u \in 1\mathbb{B}^*$ and $t_{01}(u) = 0$ for $u \in \mathbb{B}^* \setminus 1\mathbb{B}^*$. Since $t_{01} \notin \mathcal{T}_{t(u)=t(|u|)}$, $L(\mathcal{A}) \neq \mathcal{T}_{t(u)=t(|u|)}$.)

Clearly, regularity of (S2) depends on the system at hand. If, for example, the output alphabet of p_1 is $\{|\}$ then (S2) becomes trivial. Moreover, even if (S2) is not regular, the question whether (W) $\wedge$ (S2) is regular still depends on the condition (W) which is parametrized by L. If L is empty, then clearly the set of all trees which satisfy (W) and (S2) is empty as well and hence, (W) $\wedge$ (S2) is regular. On the other hand, if $L = (\Sigma_0 \times \Sigma_{01} \times \Sigma_1)^\omega$ then (W) $\wedge$ (S2) $\equiv$ (S2).

So the approach to solve the controller problem for $\mathcal{D}_{1,h}$ by constructing a tree automaton that runs over Σ_1-labeled $\Sigma_0 \times \Sigma_{01}$-trees and checks that the input tree is a winning strategy for p_1 does not work either. Since for a Σ_1-labeled Σ_{01}-tree we know at least that it always represents a *strategy* for p_1, let us reconsider this representation. Remind that the problem there is that a tree automaton running over such a tree cannot check condition (W) since it does not have access to the Σ_0-components. The key observation needed to overcome this problem is that the following condition is equivalent to (W):

($W_{\forall\alpha_0}$) if $\alpha_{01}{}^\frown\alpha_1 \in (\Sigma_{01} \times \Sigma_1)^\omega$ with $\alpha_1(j) = t(\alpha_{01}(<j))$ for all $j \in \mathbb{N}$ then $\alpha_0{}^\frown\alpha_{01}{}^\frown\alpha_1 \in L$ *for all* $\alpha_0 \in \Sigma_0^\omega$.

The reason for the equivalence of (W) and ($W_{\forall\alpha_0}$) is that the sequences $\alpha_0 \in \Sigma_0$ are not involved in the requirement that a system run $\alpha = \alpha_0{}^\frown\alpha_{01}{}^\frown\alpha_1$ is *consistent* with t: Neither does the strategy depend on α_0 nor is α_0 constrained by the strategy. Now ($W_{\forall\alpha_0}$) can easily be rephrased as condition (W), by shifting the universal quantification over the sequences α_0 to the definition of the language $L_{\forall\alpha_0} = \{\alpha_{01}{}^\frown\alpha_1 \mid \alpha_0{}^\frown\alpha_{01}{}^\frown\alpha_1 \in L \text{ for all } \alpha_0\}$.

(W) if $\alpha_{01}{}^\frown\alpha_1 \in (\Sigma_{01} \times \Sigma_1)^\omega$ with $\alpha_1(j) = \sigma(\alpha_{01}(<j))$ for all $j \in \mathbb{N}$ then $\alpha_0{}^\frown\alpha_{01}{}^\frown\alpha_1 \in L_{\forall\alpha_0}$

Now (W) is the condition in terms of the distributed system $\mathcal{D}_1$ but with the modified specification $L_{\forall\alpha_0}$. Assume that $\mathcal{B}$ is a deterministic parity automaton recognizing L. Then we can construct a deterministic parity automaton $\mathcal{B}_{\forall\alpha_0}$ recognizing $L_{\forall\alpha_0}$ as follows: First, we construct a nondeterministic parity automaton $\mathcal{C}$ which, while running over a word $\alpha_{01}{}^\frown\alpha_1 \in (\Sigma_{01} \times \Sigma_1)^\omega$, $\mathcal{B}_{\forall\alpha_0}$ *guesses* a sequence $\alpha_0 \in \Sigma_0^\omega$ and checks that $\mathcal{B}$ does *not* accept the corresponding word $\alpha_0{}^\frown\alpha_{01}{}^\frown\alpha_1$. (By simulating $\mathcal{B}$ on this word but using the complementary parity acceptance condition.) Then we can determinize and complement $\mathcal{C}$ to obtain $\mathcal{B}_{\forall\alpha_0}$. Having constructed $\mathcal{B}_{\forall\alpha_0}$, we can solve the controller problem for $(\mathcal{D}_{1,h}, L)$ by solving the controller problem for $(\mathcal{D}_1, L_{\forall\alpha_0})$ just as before.

This solution is due to Wong-Toi and Dill [221]. Notice that it uses only deterministic tree automata and nondeterministic ω-automata.

Theorem 3.11. *[221] The controller problem for $\mathcal{D}_{1,h}$ is decidable for regular specifications.*

However, the universal quantification over the sequences α_0 in ($W_{\forall\alpha_0}$) can also be directly simulated by a *universal* tree automaton: Consider some deterministic parity automaton $\mathcal{B}$ recognizing L. Then we can construct a universal parity tree automaton $\mathcal{A}$ that (universally) branches over the possible input signals from Σ_0 while running over the paths of the given input tree $t : \Sigma_{01}^* \to \Sigma_1$ and simulates $\mathcal{B}$ on all resulting paths. That is, $\mathcal{A}$ spans the Σ_1-labeled

$\Sigma_0 \times \Sigma_{01}$-tree $\text{wide}_{\Sigma_0}(t) = t' : (\Sigma_0 \times \Sigma_{01})^* \to \Sigma_1$ with $t'(u) = t(\text{Pr}_{\Sigma_{01}}(u))$ and checks that t' is a winning strategy for p_1 just like the deterministic automaton for solving the controller problem for $\mathcal{D}_1$ as above. Hence, $t \in L(\mathcal{A})$ if, and only if, t is a winning strategy for p_1 for $(\mathcal{D}_{1,h}, L)$.

Then we can translate $\mathcal{A}$ into an equivalent deterministic parity tree automaton $\mathcal{A}'$ as described in Section 3.1. Once we have constructed $\mathcal{A}'$, the controller problem for $(\mathcal{D}_{1,h}, L)$ can be solved by checking nonemptiness of $\mathcal{A}'$ and, if $L(\mathcal{A}') \neq \emptyset$, then we can construct a regular tree $t \in L(\mathcal{A}')$ which yields a finite memory strategy for p_1 for $\mathcal{D}_{1,h}$ just as in the case of $\mathcal{D}_1$.

Notice that, on the technical level, this approach is really just the same as the approach of Wong-Toi and Dill. As we have seen, removing universality from the tree automaton $\mathcal{A}$ amounts exactly to determinization of a nondeterministic parity automaton which is the dual of a universal parity automaton that runs over the universal branches of $\mathcal{A}$ in parallel – this nondeterministic automaton is essentially the automaton $\mathcal{C}$ as defined above. In a sense, while the universal automaton $\mathcal{A}$ directly checks that a given strategy t is winning with respect to $\text{Tr}(L)$ by using universal branching to simulate $\mathcal{B}$ on the paths of the tree $\text{wide}_{\Sigma_0}(t)$, the approach of Wong-Toi and Dill is more modular and constructs an automaton recognizing $L_{\forall\alpha_0}$ explicitly. This shifts the determinization of $\mathcal{C}$ to a more visible layer of the construction.

These observations also show that reducing the strategy problem for regular two-player games with imperfect information to parity games with full information, essentially just amounts to determinization of ω-automata.

3.2.1 Alternating Tree Automata Solutions

Wong-Toi and Dill also extended their solution to several other, more general cases which, however, are all restricted to a single controller and linear time specifications. On the other hand, the more direct tree automata approach using universality as an integral part is at the heart of many powerful and elegant solutions for more sophisticated synthesis problems involving multiple controllers, branching time specifications and locally decomposable specifications, see for example [127, 129, 137, 80]. In this section, we discuss some of the main ideas of these advanced constructions.

Branching Time Specifications. In [127], Kupferman and Vardi considered the controller problem for distributed systems with a single controller and branching time specifications written in CTL [56] and its extension CTL* [73]. We do not introduce these logics here but instead, we use tree automata as specification formalism which encompasses all specifications expressible in CTL* [77, 72, 20] (and even the modal μ-Calculus [75]). So consider the distributed system $\mathcal{D}_{1,h}$ as described above and let $\mathcal{S} = (\Sigma_1, Q, q_{\text{in}}, \delta, \text{col})$ be an alternating parity tree automaton defining the branching time specification for $\mathcal{D}_{1,h}$. We consider the following variant of the controller problem for $\mathcal{D}_{1,h}$: is there a strategy $t : (\Sigma_0 \times \Sigma_{01})^* \to \Sigma_1$ for p_1 such that $t \in L(\mathcal{S})$?

Notice that here, we again use the representation of strategies for p_1 as $\Sigma_0 \times \Sigma_{01}$-trees. The reason is that a branching time specification should also be able to address the Σ_0-component of the system runs which it cannot do if it talks about Σ_1-labeled Σ_{01}-trees. Now, since the branching time specification defines a set of good trees that a winning *strategy* is required to be in, strategies for p_1 have to represented as $\Sigma_0 \times \Sigma_{01}$-trees as well.

While for the system $\mathcal{D}_1$ with no hidden channels from the environment, the controller problem for branching time specifications could be solved directly by checking emptiness of $\mathcal{S}$, this is not the case for $\mathcal{D}_{1,h}$: A tree $t \in L(\mathcal{S})$ is not necessarily a strategy for p_1 since it might not satisfy the consistency condition (S2). As we already know, in general there is no tree automaton recognizing $\mathcal{S} \cap \text{(S2)}$. Notice that since the condition (S2) is not regular, the set $T_{\text{wide}_{\Sigma_0}} = \{\text{wide}_{\Sigma_0}(t) \,|\, t \in (\Sigma_{01})_{\Sigma_1}\}$ is not regular as well since $T_{\text{wide}_{\Sigma_0}} = T_{(S2)} = \{t' \in (\Sigma_0 \times \Sigma_{01})_{\Sigma_1} \,|\, t' \text{ satisfies (S2) }\}$. This substantiates the remark from Section 3.1: The class of regular tree languages is not closed under widening.

To solve the controller problem for $\mathcal{D}_{1,h}$ and $L(\mathcal{S})$ we present a solution given by Kupferman and Vardi in [127].[4] It can be viewed as an extension of the solution of the controller problem for the system $\mathcal{D}_{1,h}$ for linear time specifications to branching time specifications.

To see how this can be done, remind that we have noted that the universal tree automaton $\mathcal{A}$ as constructed above, while running over an input tree $t : \Sigma_{01} \to \Sigma_1$, uses its *universal* power to *span* the tree $t' = \mathrm{wide}_{\Sigma_0}(t)$ and simulate the deterministic parity automaton $\mathcal{B}$ on all paths of t'. Now the important point is that an alternating parity tree automaton can also simulate the *tree automaton* $\mathcal{S}$ on the tree t' while running over the input tree t instead of the automaton $\mathcal{B}$. That means, while $T_{\mathrm{wide}_{\Sigma_0}}$ is not regular, the set

$$\mathrm{narrow}(L(\mathcal{S})) := \{t \in (\Sigma_{01})_{\Sigma_1} \mid \mathrm{wide}_{\Sigma_0}(t) \in L(\mathcal{S})\}$$

is regular. This set can be seen as the analog to the set $L_{\forall\alpha_0}$ in the case where L is a linear time specification.

Notice that the regularity of $\mathrm{narrow}(L(\mathcal{S}))$ exactly amounts to the claim of Proposition 3.4 that for a given alternating parity tree automaton $\mathcal{A}_{X\times Y}$ over Σ-labeled $X \times Y$-trees we can construct an alternating parity tree automaton $\mathrm{narrow}_Y(\mathcal{A}_{X\times Y})$ over Σ-labeled X trees such that $\mathrm{narrow}_Y(\mathcal{A}_{X\times Y})$ accepts a tree t if, and only if, $\mathcal{A}_{X\times Y}$ accepts $\mathrm{wide}_Y(t)$.

To see how this automaton can be obtained, let the automaton to be constructed again be called $\mathcal{A}$. The transition $\delta^{\mathcal{A}}(q, a_1)$ of $\mathcal{A}$ for a given state $q \in Q^{\mathcal{A}}$ and a label $a_1 \in \Sigma_1$ is obtained from the transition $\delta^{\mathcal{S}}(q, a_1)$ of $\mathcal{S}$ by replacing each literal $(\downarrow_{(a_0,a_{01})}, q) \in (\Sigma_0 \times \Sigma_{01}) \times Q$ by the literal $(\downarrow_{a_{01}}, q)$. So, when $\delta^{\mathcal{S}}(q, a_1)$ contains a conjunct

$$\bigwedge_{a=(a_0,a_{01})\in\Sigma_0\times\Sigma_{01}} \bigwedge_{q\in Q_a} (\downarrow_a, q)$$

then $\delta^{\mathcal{A}}(q, a_1)$ contains the conjunct

$$\bigwedge_{a=(a_0,a_{01})\in\Sigma_0\times\Sigma_{01}} \bigwedge_{q\in Q_a} (\downarrow_{a_{01}}, q).$$

So, for a given direction $a_{01} \in \Sigma_{01}$, the automaton $\mathcal{A}$ sends all the states $q \in \bigcup_{a_0\in\Sigma_0} Q_{(a_0,a_{01})}$ that $\mathcal{S}$ sends to *some* direction (a_0, a_{01}) to the direction a_{01}. In this way, $\mathcal{A}$ simulates $\mathcal{S}$ on $\mathrm{wide}_{\Sigma_0}(t)$ for a given input tree $t \in (\Sigma_{01})_{\Sigma_1}$. Clearly, this construction introduces additional universality.

To see that $L(\mathcal{A}) = \mathrm{narrow}(L(\mathcal{S}))$ it is sufficient to note that a run $\rho^{\mathcal{A}}$ of $\mathcal{A}$ on a given input tree $t \in (\Sigma_{01})_{\Sigma_1}$ corresponds to a run $\rho^{\mathcal{S}}$ of $\mathcal{S}$ on $\mathrm{wide}_{\Sigma_0}(t)$ and, moreover, $\rho^{\mathcal{A}}$ is accepting if, and only if, $\rho^{\mathcal{S}}$ is accepting. To make this concept more evident, it is convenient to incorporate the directions $a_0 \in \Sigma_0$, over which the automaton $\mathcal{A}$ universally branches, explicitly into the state space of $\mathcal{A}$. In this way, the correspondence between the runs $\rho^{\mathcal{A}}$ and $\rho^{\mathcal{S}}$ can be established directly via the labels of $\rho^{\mathcal{A}}$ which makes the proof more transparent. So the state space $Q^{\mathcal{A}}$ is $Q^{\mathcal{S}} \times \Sigma_0$ and $\delta^{\mathcal{A}}((q, b_0), a_1)$ for a given state $(q, b_0) \in Q^{\mathcal{A}}$ and a label $a_1 \in \Sigma_1$ is obtained from $\delta^{\mathcal{S}}(q, a_1)$ by replacing each literal $(\downarrow_{(a_0,a_{01})}, q) \in (\Sigma_0 \times \Sigma_{01}) \times Q$ by $(\downarrow_{a_{01}}, (q, a_0))$.

We conclude that in fact $L(\mathcal{A}) = \mathrm{narrow}(L(\mathcal{S}))$, that means, the automaton $\mathcal{A}$ accepts a Σ_1-labeled Σ_{01}-tree which represents a strategy for p_1 in $\mathcal{D}_{1,h}$ if, and only if, $\mathcal{S}$ accepts the Σ_1-labeled $\Sigma_0 \times \Sigma_{01}$-tree $\mathrm{wide}_{\Sigma_0}(t)$ which represents the same strategy of p_1. Hence, $L(\mathcal{A}) \neq \emptyset$ if, and only if, p_1 has a strategy $\mathcal{D}_{1,h}$ whose representation as a $\Sigma_0 \times \Sigma_{01}$-tree is in $L(\mathcal{S})$. If $L(\mathcal{A}) \neq \emptyset$, as usual, a regular tree in $L(\mathcal{A})$ yields a finite state implementation of a strategy for p_1. This solves the controller problem for $\mathcal{D}_{1,h}$ with branching time specifications.

Theorem 3.12. *[127] The branching time controller problem for $\mathcal{D}_{1,h}$ is decidable for regular tree languages.*

Multiple Controllers. Distributed systems as we consider them here have first been studied by Pnueli and Rosner in[171]. They considered, among others, the system

$$\mathcal{D}_2 = (\mathfrak{A}, (\Sigma_c)_{c\in C})$$

with $\mathfrak{A} = (\{01, 02, 1, 2\}, r, w)$ where

[4] An earlier version of this paper appeared as [126].

- $w(0i) = p_0$ and $r(0i) = p_i$
- $w(i) = p_i$ and $r(i) =_0$

Moreover, $\Sigma_c = \mathbb{B}$ for all $c \in C$. They showed that $\mathcal{D}_2$ is undecidable for LTL-specifications.[5] Originally, Pnueli and Rosner used more general alphabets but by encoding each element of the larger alphabets as a bit sequence, the proof can be adapted to binary alphabets. The proof can also be slightly refined so that the specification can be recognized by a safety automaton.[6]

Theorem 3.13. *[163, 171] The controller problem for $\mathcal{D}_2$ is undecidable for safety specifications.*

Now let $L \subseteq (\Sigma^{\mathcal{D}_2})^\omega$ be a specification for $\mathcal{D}_2$. To analyze why the approach for the system $\mathcal{D}_{1,h}$ cannot be extended to $\mathcal{D}_2$, first notice that due to the different information that p_1 and p_2 have about the system $\mathcal{D}_2$, the definition of $L_{\forall\alpha_0}$ as given for $\mathcal{D}_{1,h}$ does not apply here directly. Now consider the following naive approach to generalize the definition of $L_{\forall\alpha_0}$ to $\mathcal{D}_2$: We construct specifications $\overline{L}_1$ and $\overline{L}_2$, where $\overline{L}_i$ is defined with respect to the information of p_i, that is, it consists of those $\alpha_{01}{}^\frown\alpha_1 \in (\Sigma_{01} \times \Sigma_1)^\omega$ such that for all $\alpha_{02}{}^\frown\alpha_2 \in (\Sigma_{02} \times \Sigma_1)^\omega$ we have $\alpha_{01}{}^\frown\alpha_{02}{}^\frown\alpha_1{}^\frown\alpha_2 \in L$ and $\overline{L}_2$ analogously. Then we solve, for $i = 1, 2$, the controller problem for p_i with respect to $\overline{L}_i$.

In fact, this approach is partially correct: If a strategy $\sigma = (\sigma_1, \sigma_2)$ can be synthesized in this way then σ is actually a winning strategy for the controllers for $(\mathcal{D}_2, L)$. The converse, however, is clearly not true: The requirement that for each sequence $\alpha_{01}{}^\frown\alpha_1$ which is consistent with σ_1 we have $\alpha_{01}{}^\frown\alpha_{02}{}^\frown\alpha_1{}^\frown\alpha_2 \in L$ *for all* $\alpha_{02}{}^\frown\alpha_2 \in (\Sigma_{02} \times \Sigma_1)^\omega$ is too strong since for a run $\alpha = \alpha_{01}{}^\frown\alpha_{02}{}^\frown\alpha_1{}^\frown\alpha_2$ we require merely that $\alpha \in L$, if α is consistent with the joint strategy σ. So the sequences $\alpha_{02}{}^\frown\alpha_2$ that occur in such runs are constrained by σ_2 and the sequences $\alpha_{01}{}^\frown\alpha_1$ are constrained by σ_1.

Due to this mutual dependency it is quite intricate to reduce controller problems for distributed systems with multiple controllers to controller problems with full information in a direct way like for $\mathcal{D}_{1,h}$. In Chapter 6 we present a construction that takes into account the different information of p_1 and p_2 and also incorporates strategic dependencies. Clearly, since the controller problem for $\mathcal{D}_2$ is undecidable, there cannot be an effective reduction to a controller problem with full information which works in general. In fact, our construction does not always yield a finite result, cf. Section 3.5 and Chapter 6. Here we review how the controller problem can be solved for systems with pipeline architectures and regular specifications.

Consider a pipeline with $n + 1$ processes

$$\mathcal{D}_{n,p} = (\mathfrak{A}, (\Sigma_c)_{c\in C}).$$

W.l.o.g. we assume that from a process p_i with $i < n$, there is exactly one channel to p_{i+1}. Additionally, the environment has a hidden channel and each process p_i has an external output channel. There are no other channels in the architecture $\mathfrak{A}$. We denote $\mathfrak{A} = (C, r, w)$ as follows.

- $C = \{c_0, c_{0,h}\} \cup \{c_i, c_{i,e} \mid i = 1, \dots, n-1\} \cup \{c_{n,e}\}$
- $w(c_i) = p_i$, $r(c_i) = p_{i+1}$ and $w(c_{0,h}) = r(c_{0,h}) = 0$
- $w(c_{i,e}) = p_i$ and $r(c_{i,e}) = p_0$

Moreover, assume $\Sigma_c = \mathbb{B}$ for all $c \in C$ and for a controller p_i let us denote

- $\Sigma_i := \prod_{c\in O_{p_i}\cap I_{p_{i+1}}} \Sigma_c$
- $\Sigma_{\text{out}}^{\geq i} := \Sigma^{\geq p_i} (= \prod_{j=i}^{n} \Sigma_{\text{out}}^{p_j})$ (Notice that $\Sigma_{\text{out}} = \Sigma_{\text{out}}^{\geq 1}$)

[5]Peterson and Reif showed already in [163] that the strategy problem for three-player games on finite graphs with partial information is undecidable for reachability conditions. However, they used a model with a certain kind of asynchronous observability.

[6]We provide a detailed proof in Chapter 5.

As we have demonstrated in Section 2.1.3, a joint strategy $\sigma = (\sigma_1, \dots, \sigma_n)$ for the controllers can be represented as a Σ_{out}-labeled Σ_0-tree $t : \Sigma_0^* \to \Sigma_{\text{out}}$ since given the fixed strategy σ, the sequence of signals that p_0 sends to the controllers uniquely determines the output of all controllers. In the case of the pipeline, the underlying information flow is linear which makes it particularly simple, to trace how the outputs are determined: In the first step, the action of each process p_i, $i = 1, \dots, n$ is uniquely determined by σ_i. In the second step, the action of process p_1 is $\sigma_1(a)$, where a is the signal chosen by p_0 in the first step, the action of process p_2 is $\sigma_2(\sigma_1(a))$ and so on. In the third step, the action of process p_1 is $\sigma_1(ab)$, where b is the signal chosen by p_0 in the second step, the action of process p_2 is $\sigma_2(\sigma_1(a)\sigma_1(ab))$ and so on. In this way, the actions of all processes $p_1, \dots, p_n$ are determined in each step.

Now let $L \subseteq (\Sigma^{\mathcal{D}_{n,p}})^\omega$ be a regular specification for $\mathcal{D}_{n,p}$ and let $\mathcal{B}$ be a deterministic parity automaton with $L(\mathcal{B}) = L$. We already know how to construct a universal parity tree automaton $\mathcal{A}_1$ that runs over a tree $t : \Sigma_0^* \to \Sigma_{\text{out}}$ and checks that all path through $\text{wide}_{\Sigma_{0,h}}(t)$ are in L by simulating $\mathcal{B}$ along these paths. This takes care of the hidden channel $c_{0,h}$ from p_0.

Moreover, we know how to transform $\mathcal{A}_1$ into an equivalent deterministic parity tree automaton $\mathcal{A}_1'$. But we also know that no tree automaton can check that t is in fact a strategy, because for this to be true, for each $i = 2, \dots, n$ the tree t has to fulfill the consistency condition $(\text{S2})_{p_i}$. For the case of the pipeline $\mathcal{D}_{n,p}$, these conditions have the following form:

$$(\text{S2})_i \quad \text{Pr}_{\Sigma_{i-1}}(t^*(u^{-1})) = \text{Pr}_{\Sigma_{i-1}}(t^*(v^{-1})) \implies \text{Pr}_{\Sigma_{\text{out}}^{p_i}}(t(u)) = \text{Pr}_{\Sigma_{\text{out}}^{p_i}}(t(v)).$$

Now given a tree $t : \Sigma_0^* \to \Sigma_{\text{out}}$ which fulfills condition $(\text{S2})_i$ we can define a strategy $t_i : \Sigma_{i-1}^* \to \Sigma_{\text{out}}^{p_i} = \Sigma_i \times \Sigma_{i,e}$ for p_i as follows: First, $t_i(\varepsilon) = \text{Pr}_{\Sigma_{\text{out}}^{p_i}}(t(\varepsilon))$. Moreover, for $v \in \Sigma_{i-1}^+$ let $t_i(v) = \text{Pr}_{\Sigma_{\text{out}}^{p_i}}(t(ua))$ for some $ua \in \Sigma_0^*$ such that $\text{Pr}_{\Sigma_{i-1}}(t^*(u)) = v$. Since t fulfills condition $(\text{S2})_i$, this definition is independent of the particular u and a.

Since the condition (S2) is not regular, the conditions $(\text{S2})_i$ are clearly not regular in general as well. Notice however, that we have not required a condition $(\text{S2})_1$. In fact, $\text{Pr}_{\Sigma_{\text{out}}^{p_1}}(t)$ is a strategy for p_1 for any tree $t : \Sigma_0^* \to \Sigma_{\text{out}}$. The reason for this is that p_1 is the uniquely determined best informed process in the system $\mathcal{D}_{n,p}$, that is, p_1 reads all the signals from Σ_0 that the environment sends to the controllers directly.

This is already the most important step towards a solution of the controller problem for $\mathcal{D}_{n,p}$: An extended strategy for p_1 is here a joint strategy $\sigma_{\geq 1} : \Sigma_0^* \to \Sigma_{\text{out}}$ for the grand coalition. Likewise, any controller p_i can take the role of all processes $p_i, \dots, p_n$ by playing an extended strategy $t_{\geq i} : \Sigma_{i-1}^* \to \Sigma_{\text{out}}^{\geq i}$. As we have mentioned, by playing an extended strategy $t_{\geq i}$, p_i can abuse the information on which it bases its decisions, that means, $t_{\geq i}$ does not necessarily fulfill the consistency conditions $(\text{S2})_j$ for $j > i$. The solution to the controller problem for $(\mathcal{D}_{n,p}, L)$ is an iterative construction of alternating parity tree automata $\mathcal{A}_1, \dots, \mathcal{A}_n$ (with $\mathcal{A}_1$ already given) that successively eliminates this abuse of information. That means, for $i \geq 2$, $\mathcal{A}_i$ is an automaton over $\Sigma_{\text{out}}^{\geq i}$-labeled Σ_{i-1}-trees such that $t \in L(\mathcal{A}_i)$ if, and only if:

There is a $\Sigma_{\text{out}}^{\geq i-1}$-labeled Σ_{i-2}-tree s such that

- s is in $L(\mathcal{A}_{i-1})$ and fulfills $(\text{S2})_i$
- $s_i = t$, where s_i is the extended strategy $s_i : \Sigma_{i-i}^* \to \Sigma_{\text{out}}^{\geq i}$ for process p_i, determined by s

By induction, $L(\mathcal{A}_n) \neq \emptyset$ if, and only if, there is a tree $t \in L(\mathcal{A}_1)$ such that t fulfills $(\text{S2})_i$ for all $i = 2, \dots, n$. Clearly, there is a joint winning for the controllers for $(\mathcal{D}_{n,p}, L)$ if, and only if, there is a tree with these properties, that means, if, and only if, $L(\mathcal{A}_n) \neq \emptyset$.

We do not give all details of the construction here but we merely describe the automaton $\mathcal{A}_2$ informally, which is sufficient to understand the general principle behind the construction. To see how $\mathcal{A}_2$ works, let us start at the root of some input tree $t : \Sigma_1^* \to \Sigma_{\text{out}}^{\geq 2}$. There, the automaton *nondeterministically* guesses an element $(a, b) \in \Sigma_1 \times \Sigma_{1,e}$ which is the action chosen by p_1 in the first step according to some strategy that $\mathcal{A}_2$ is guessing. Then the automaton *universally* branches over all elements $c_1, \dots, c_r \in \Sigma_0$ ($r = |\Sigma_0|$), which are the possible signals send by p_0 to p_1 in the first step, and proceeds to copies $(\downarrow_a, c_1), \dots, (\downarrow_a, c_r)$. Analogously, being at any node $(u, w) \in \Sigma_1^* \times \Sigma_0^*$ of the run, $\mathcal{A}_2$ again guesses an element $(a, b) \in \Sigma_{\text{out}}^{p_i}$, branches universally over all c_j, and proceeds to copies $(\downarrow_a, c_1), \dots, (\downarrow_a, c_r)$. In this way, $\mathcal{A}_2$ guesses a $\Sigma_{\text{out}}^{p_i}$-labeled Σ_0-tree $t' : \Sigma_0^* \to \Sigma_{\text{out}}^{p_i}$ which is a strategy for p_1. Moreover, via the *labels* of t we immediately obtain a $\Sigma_{\text{out}}^{\geq 1}$-labeled Σ_0-tree s from t': The $\Sigma_{\text{out}}^{\geq 2}$-label of ε is $t(\varepsilon)$, the $\Sigma_{\text{out}}^{\geq 2}$-label of any node c_j is $t(a^0)$ and so on. By construction, s satisfies $(\text{S2})_2$ and the strategy s_2 for p_2 which is determined by s_2 coincides with t.

In order to guarantee that also $s \in L(\mathcal{A}_1)$ holds, we simulate the equivalent deterministic parity tree automaton $\mathcal{A}_1'$ (over $\Sigma_{\text{out}} = \Sigma_{\text{out}}^{\geq 1}$-labeled Σ_0-trees) on s which $\mathcal{A}_2$ guesses. To see how this is done, let us again start from the top of t. Remind that $\mathcal{A}_2$ guesses a $(a, b) \in \Sigma_1 \times \Sigma_{1,e}$, branches over all $c_1, \dots, c_r \in \Sigma_0$ and proceeds to copies $(\downarrow_a, c_1), \dots, (\downarrow_a, c_r)$. In order to simulate $\mathcal{A}_1'$, we keep the state q of $\mathcal{A}_1'$ in the state of $\mathcal{A}_2$ and we look at the transition $\delta^{\mathcal{A}_1'}(q, s(\varepsilon)) = \delta^{\mathcal{A}_1'}(q, ((a, b), t(\varepsilon)))$ which tells us, for each direction c_j in s, the state of $\mathcal{A}_1'$. Analogously, in any node (u, w) of the run, $\mathcal{A}_2$ again guesses a $(a, b) \in \Sigma_{\text{out}}^{p_1}$, branches over all c_j, and proceeds to copies $(\downarrow_a, c_1), \dots, (\downarrow_a, c_r)$. Hence, take again the state q of $\mathcal{A}_1'$ and look at $\delta^{\mathcal{A}_1'}(q, s(u)) = \delta^{\mathcal{A}_1'}(q, ((a, b), t(u)))$.

Notice that the way in which we use the concept of alternation here differs considerably from the way in which we have used the concept of universality for spanning the Σ-labeled $X \times Y$-tree $\text{wide}_Y(t)$ over a Σ-labeled X-tree t: There we have send all the possible Y-elements into *all* X-directions of the tree and we have kept the labels from the tree t completely. In particular, simulating a given tree automaton $\mathcal{A}$ on $\text{wide}_Y(t)$ while running over t introduces only additional universality but no additional nondeterminism. Using this observation it's easy to see that this can be done safely, even if $\mathcal{A}$ is alternating. Here the situation is different: $\mathcal{A}_2$ branches universally over the Σ_0-elements but sends them all to the *same* Σ_1-direction which the automaton guesses *nondeterministically*. In this way, the tree s is not really spanned over t but somehow superimposed.[7]

Now consider what would happen if we simulated the universal parity tree automaton $\mathcal{A}_1$ directly on the tree s. This automaton universally branches over the possible *hidden* inputs from $\Sigma_{0,h}$ in each step. So when we simulate $\mathcal{A}_1$ on s, a branch in the run does not correspond to a node $u \in \Sigma_0^*$ of s anymore but to a node $(u, v) \in (\Sigma_0 \times \Sigma_{0,h})^*$ of $\text{wide}_{\Sigma_{0,h}}(s)$ and the automaton $\mathcal{A}_2$ can make its nondeterministic choices *dependent* on v. But the nondeterministic choices of $\mathcal{A}_2$ guess a strategy of p_1 which has to be *independent* of v. Notice that, as we already know, the nondeterministic choices of $\mathcal{A}_2$ can be made independent of the particular branch of the run by using a positional strategy for player 1 for the semantic game $\mathcal{G}(\mathcal{A}_2, t)$. But nevertheless, the choices still depend on the particular node of t *and* the state of the automaton. But the state of $\mathcal{A}_2$ also contains the state of $\mathcal{A}_1$ and since $\mathcal{A}_1$ simulates the specification $\mathcal{B}$ along the paths of $\text{wide}_{\Sigma_{0,h}}(s)$, the state of $\mathcal{A}_1$ contains the state of $\mathcal{B}$ from which one can still infer information about the sequence v.

However, since the automaton $\mathcal{A}_1'$ is deterministic, it can be seen that we have in fact $t \in L(\mathcal{A}_2)$ if, and only if, there is a tree $s \in L(\mathcal{A}_1)$ such that s satisfies condition $(\text{S2})_2$ and the strategy s_2 coincides with t. Clearly, since $\mathcal{A}_2$ is not just universal but alternating, there is no equivalent deterministic parity tree automaton in general. However, from our explanation above it is clear that it suffices to have an equivalent *nondeterministic* automaton $\mathcal{A}_2'$ to be able to construct the automaton $\mathcal{A}_3$ in the very same way as we have constructed $\mathcal{A}_2$.

We have already noted that once the iteration is finished and we have accomplished the construction of $\mathcal{A}_n$, it holds that $L(\mathcal{A}_n) \neq \emptyset$ if, and only if, there is a $t \in L(\mathcal{A}_1)$ that fulfills $(\text{S2})_i$ for $i = 2, \dots, n$. This settles the decidability of the controller problem for $(\mathcal{D}_{n,p}, L)$: To decide whether there is a winning strategy for the controllers for $(\mathcal{D}_{n,p}, L)$ we transform $\mathcal{A}_n$ into an equivalent nondeterministic parity tree automaton and check whether $L(\mathcal{A}_n) \neq \emptyset$.

[7]Clearly, terms like *spanning* and *superimposing* have no precise definition but they seem meaningful for distinguishing the different ways in which we have used the concept of alternation.

Implementation of Strategies. Once we know that $L(\mathcal{A}_n) \neq \emptyset$ we would like to synthesize a winning strategy for the controllers. This, however, is not as simple as for $\mathcal{D}_{1,h}$: A regular tree t_n in $L(\mathcal{A}_n)$ does not directly provide a finite state implementation of a joint winning strategy for the controllers but merely a finite state implementation of a strategy for p_n, but, with the additional property that *there exists* a regular tree $t_{n-1} \in L(\mathcal{A}_{n-1})$ such that t_{n-1} satisfies condition $(\mathrm{S2})_n$ and $(t_{n-1})_n = t_n$. So in order to actually synthesize a joint winning strategy for the controllers, we have to proceed iteratively again by going backwards through the inductive construction of the automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$.

Clearly, in the second step of this iteration it is not sufficient to check just the emptiness of the nondeterministic automaton $\mathcal{A}'_{n-1}$ and to construct a regular tree in $L(\mathcal{A}_{n-1})$ but we have to make sure that we get hold of a regular tree $t_{n-1} \in L(\mathcal{A}_{n-1})$ such that $(t_{n-1})_n = t_n$. To guarantee this, we don't just play the usual emptiness game $\mathcal{G}(\mathcal{A}'_{n-1})$ but we play the game $\mathcal{G}(\mathcal{A}'_{n-1}, t_n)$ which is relativized to the tree t_n and which is obtained by taking the product of the game graph $\mathcal{G}(\mathcal{A}'_{n-1})$ and a finite automaton $\mathcal{B}_n$ that generates t_n.

More precisely, the game graph $\mathcal{G}(\mathcal{A}'_{n-1}, t_n)$ has positions (q, p) for player 1 and positions $(q, ((a, b), c), q_1, \ldots, q_s, p)$ for player 0 where $s = |\Sigma_{n-2}|$ and $(q, ((a, b), c), q_1, \ldots, q_s)$ is a transition of $\mathcal{A}'_{n-1}$, p is a state of $\mathcal{B}_n$ and $c = \tau^{\mathcal{B}_n}(p)$. (Notice that $\mathcal{A}'_{n-1}$ runs over $\Sigma_{n-1} \times \Sigma_{n-1,e} \times \Sigma_n$-labeled Σ_{n-2}-trees.) The initial position is $(q_{\mathrm{in}}, p_{\mathrm{in}})$ from which player 1 can move to any position $(q_{\mathrm{in}}, ((a, b), \tau^{\mathcal{B}_n}(p_{\mathrm{in}})), q_1, \ldots, q_s, p_{\mathrm{in}})$. From this position, player 0 can move to any position $(q_j, \delta^{\mathcal{B}_n}(p_{\mathrm{in}}, a))$ and so on. So we also feed the Σ_{n-1}-label chosen by player 1 into the transition function of $\mathcal{B}_n$. In this way, player 1 constructs a $\Sigma_{n-1} \times \Sigma_{n-1,e} \times \Sigma_n$)-labeled Σ_{n-2}-tree $t_{n-1} \in L(\mathcal{A}'_{n-1})$ where the Σ_n-component of the labels is determined by the corresponding label of t_n at the node which is given by the sequence of Σ_{n-1}-labels of t_{n-1} – that means $(t_{n-1})_n = t_n$.

The first solution for the controller problem for $(\mathcal{D}_{n,p}, L)$ has been given by Pnueli and Rosner in [171]. They used only nondeterministic tree automata and branching time logics, exploiting their well-known equivalence to nondeterministic tree automata. In [129] Kupferman and Vardi extended the solution of Pnueli and Rosner to certain more general classes of architectures and to branching time specifications using alternating tree automata. Clearly, the solution presented above works for branching time specifications as well.

In [80], Finkbeiner and Schewe used similar constructions for their decision procedures but extended them to a somewhat more general setting where *broadcast* channels are allowed, cf. Section 2.1.3. Finkbeiner and Schewe also considered so called *white box* processes, that is, the architectures may have certain processes which already *have* an implementation by means of a finite state device.[8] Finkbeiner and Schewe showed that *any* architecture without an information fork (cf. Section 2.1.3) is decidable for regular and for branching time specifications written in the modal μ-calculus. The proof proceeds via certain architecture manipulations which allow to transform any given information fork free architecture into one which has the shape of a pipeline. So essentially, the decision procedure that we have presented above is sufficient for deciding the controller problem for any distributed system with a decidable architecture.

Finkbeiner and Schewe also proved that the information fork criterion is necessary for decidability of a given architecture for both, regular and branching time specification. To prove that any architecture with an information fork is undecidable for regular specifications, the original proof of Pnueli and Rosner for the architecture $\mathcal{D}_2$. has to be refined in two directions. First, one has to deal with arbitrary information forks instead of just the simple on in the architecture $\mathcal{D}_2$. Second, one has to deal with the fact that the two processes in the given information fork do not necessarily have external output channels but it may be that all their

[8]In fact, they showed that white boxes can be eliminated. Essentially each white box can be simulated by an appropriate black box and the behaviors which are prescribed by the implementation of the white box can be added to the specification. However, for *modeling* an actual system, white boxes may be comfortable as they allow to incorporate partial system designs into the architecture, rather than the specification, cf. [80].

outputs are read by the other process. The proof in [80] can again be strengthened to safety conditions, so we obtain the following result.

Theorem 3.14. *[171, 129, 80]*

(1) If $\mathfrak{A}$ has an information fork, $\mathfrak{A}$ is undecidable for safety and CTL specifications.

(2) If $\mathfrak{A}$ does not have an information fork, it is decidable for regular and CTL* specifications.

3.2.2 Emptiness Games for Alternating Tree Automata

In Section 3.1 we have described the emptiness game for *nondeterministic* tree automata. We have also mentioned that the game cannot be directly generalized to *alternating* tree automata. We could easily incorporate the alternating transition structure of the automaton into the game just as we have done this for the semantic game but the problem is that then, player 1 can make his choices of transitions dependent on the particular branch of the run. This is ok, as long as the input tree is fixed (the nondeterministic choices of the automaton may depend on these particular branches) but it does not work if the input tree is to be constructed by means of these choices (the labels of the input tree must not depend on these particular branches).

Consider alternating parity tree automaton on Σ-labeled $[d]$-trees

$$\mathcal{A} = (\Sigma, Q, q_{\text{in}}, \delta, \text{col}).$$

Universal Tree Automata. If $\mathcal{A}$ is universal, there is an easy way to fix the problem described above by introducing partial information to the emptiness game. For this, we slightly modify the positions of player 1 so that they have the form (q, x) where $q \in Q$ and $x \in [d]$. From such a position, player 1 can move (with action a) to any position (q, x, a) of player 0 with $a \in \Sigma$. Then, for $\delta(q, a) = (\downarrow_{x_1}, q_1) \wedge \ldots \wedge (\downarrow_{x_s}, q_s)$, player 0 can move (with action j) from (q, x, a) to any position (q_j, x_j). Player 1 has partial information about the positions and actions in the game: Of a position (q, x) or (q, x, a) he observes only x, that is $v \sim^V w$ if $\Pr_{[d]}(v) = \Pr_{[d]}(w)$. Moreover, for any two actions j and j' of player 0 we have $j \sim^A j'$. So a strategy for player 1 in the emptiness game $\mathcal{G}(\mathcal{A})$ is essentially a function $\sigma : [d]^* \to \Sigma$, that is, it is a Σ-labeled $[d]$-tree and obviously, σ is a winning strategy for player 1 if, and only if, $\sigma \in L(\mathcal{A})$.

This yields a linear time reduction of the emptiness problem for universal parity tree automata to the strategy problem for two-player parity games with imperfect information on finite graphs. Since we already have a linear time reduction in the other direction, this demonstrates that the problems are linear time equivalent (by means of simple and natural reductions). However, so far we have no way of making use of this fact in either one of the directions: For both problems, the known solutions involve determinization of ω-automata and solving a parity game with full information on a finite graph, see also Section 3.5.

Alternating Tree Automata. For an arbitrary alternating tree automata $\mathcal{A}$, however, this solution does still not work: The *nondeterministic* choices depend on the particular branch of the run and not just on the branch $u \in [d]^*$ of the input tree. (In fact, as we know, they depend only on the state of the automaton, but this information is not available for player 1 either.) Hence, in the game $\mathcal{G}(\mathcal{A})$, player 1 does not have sufficient information for choosing the transitions of the automaton so we have to extend the game $\mathcal{G}(\mathcal{A})$ further.

In [172] it has been suggested to *split* player 1 into two players, one choosing the labels of the input tree and having partial information just like player 1 before and the other choosing the transitions of the automaton. The necessary modifications of the game as described above are straightforward. This is still a linear time reduction but now, we obtain a game with three players where one player has partial information.

This can be seen as a special case of a pipelines with two processes where we have no hidden channels from the environment and the regular specification is given by a deterministic parity automaton. If we inspect the solution for the system $\mathcal{D}_{n,p}$, we observe that in this special case, the construction starts with a deterministic parity tree automaton $\mathcal{A}_1$ and requires only a single application of the construction of an alternating parity tree automaton $\mathcal{A}_2$. This tree automaton can be seen as running over strategies for p_2 (the process with partial information) while guessing a strategy for p_1 (the process with full information). Since we don't need any removal of alternating during the construction it is not hard to see that $\mathcal{A}_2$ can be constructed in linear time. So again, the nonemptiness problem for alternating parity tree automata and the strategy problem for three-player parity games with imperfect information on finite graphs are linear time equivalent in a very natural way.

For the reduction of the emptiness problem of alternating parity tree automata to games with imperfect information, another solution has been suggested in [164] which involves only two-player games but needs exponential time. The idea is that player 1 chooses the labels of the input tree as before and at the same time, being in node $u \in [d]^*$ of the input tree, chooses a *positional strategy* $\sigma_u : Q \to B^+([d] \times Q)$ such that $\sigma_u(q)$ is a conjunct in $\delta(t(u), q)$, where $t(u)$ is the label of t at u that player 1 has chosen. (So player 1 chooses the labels of the Γ-annotated Σ-labeled $[d]$-tree as described in Section 3.1.2.)

More precisely, player 1 has the same information as in the game above and can move (with action (a, σ) from a position (q, x) to any position of the form (q, x, a, σ) with $a \in \Sigma$ and $\sigma : Q \to B^+([d] \times Q)$. Then, if $\sigma(\delta(q, a)) = (\downarrow_{x_1}, q_1) \wedge \ldots \wedge (\downarrow_{x_s}, q_s)$, player 0 can move (with action j) from (q, x, a, σ) to any position (q_j, x_j). Since the number of functions $Q \to B^+([d] \times Q)$ is exponential in the size of $\mathcal{A}$, this is an exponential time reduction which, essentially, reduces the problem of removing alternation from an arbitrary alternating parity tree automaton to removing universality from a universal parity tree automaton. Notice that the correctness of this construction heavily relies on positional determinacy of parity games with full information just in the way as we have used them for removal of alternation from tree automata, cf. Section 3.1.2.

In a nutshell, these reductions fortify and conclude our exposition of the intimate connection between alternating tree automata and interaction under partial information. We have seen several solutions of strategy problems under partial information using alternating tree automata in a powerful and elegant way. In Chapter 5, we shall explore the potential of these methods further.

3.3 Complexity

In Section 3.2, we were only concerned with *decidability* of strategy problems but we did neither consider the time or space bounds of the decision procedures that we have presented nor the inherent computational complexity of the problems. Now we give a brief overview over some fundamental complexity results on strategy problems under partial information.

One of the most fundamental results on the complexity of strategy problems has been given in Theorem 2.7: The strategy problem for parity games with *full information* is in NP ∩ co-NP and can be solved in time exponential in the number of colors but polynomial in the number of positions of the game graph. However, the problem is not known to be NP-complete and the question whether the problem can be solved by an algorithms whose overall time complexity is polynomial is one of the major open questions in the theory of verification and synthesis. We have discussed this issue briefly in Section 2.2 and we will return to it in Chapter 4.

The complexity of parity games is especially important since any ω-regular language can be recognized by a deterministic parity automaton. Clearly, the size of this automaton depends on the original representation of the language. If it is given by a nondeterministic Büchi automaton, then the number of states is exponential but the number of colors is only polynomial. The same holds for deterministic Muller automaton. On the other hand, for a given LTL-specification, the size of the automaton is doubly exponential and the number of colors is exponential in the length of the formula. Even worse, if the language is given by an S1S-formula, the size of the parity automaton is not elementary in general. We provide basic complexity results for some of the most fundamental such cases. We try and proceed in historical order.

Remark. Notice that we can always derive lower bounds for strategy problems from the *satisfiability* problem (emptiness problem, respectively) of the given specification formalism: For branching time specifications we have already noted that the controller problem with respect to the system $\mathcal{D}_1$ as defined in Section 3.2 *is* the satisfiability problem for the given specification. Moreover, for linear specifications, the satisfiability problem is a very special case of the controller problem: if we consider the system $\mathcal{D}_{1,-}$ which is obtained from the system $\mathcal{D}_1$ by deleting the channel c_{01} from the environment to the controller, then a winning strategy for p_1 is just a linear model for the given specification.

$\mathcal{D}_1$ and S1S. The first fundamental solution to synthesis of reactive systems has been given by Büchi and Landweber [44]. They started from an S1S-formula, translated it into an equivalent nondeterministic Büchi automaton and used McNaughton's result [141] (which they called the fundamental lemma on ω-automata) to translate this automaton into an equivalent deterministic Muller automaton $\mathcal{B}$. Then they solved the controller problem for the system $\mathcal{D}_1$ with the specification given by $\mathcal{B}$, which is essentially just the Muller game played on the transition graph of $\mathcal{B}$. The translation of S1S-formulas into Büchi automata is due to Büchi [43]. The complexity of this translation, however, is not elementary, which is accompanied by a matching lower bound according to Meyer [145].

Theorem 3.15. *[44, 145] The controller problem for $\mathcal{D}_1$ is nonelementary recursive with respect to specifications written in* S1S.

$\mathcal{D}_1$ and LTL. In [169], Pnueli and Rosner proved that for linear temporal logic LTL [167], the controller problem for $\mathcal{D}_1$ can be solved in doubly exponential time. They used a translation of LTL-formulas into nondeterministic Büchi automaton of size exponential in the length of the formula [77, 216], see also [212]. Using Theorem 2.8 and Theorem 2.7 it can then be seen that the controller problem for $\mathcal{D}_1$ for LTL-specifications can be solved in time doubly exponential in the size of the formula. From the satisfiability problem for LTL we only get a PSPACE lower bound, but in [170], Pnueli and Rosner proved a matching 2-EXPTIME lower bound for the controller problem for $\mathcal{D}_1$ with LTL specifications.[9]

Theorem 3.16. *[169, 170] The controller problem for $\mathcal{D}_1$ is complete for 2-*EXPTIME *with respect to specifications written in linear temporal logic.*

$\mathcal{D}_{1,h}$ and CTL*. The previous two results deal with controller problems under *full information* with *linear* specifications. In [127], Kupferman and Vardi extended these solutions to both, partial information and branching time specifications. As we have mentioned in the introduction, strategy problems under partial information are usually harder to solve than those under full information. However, for all the specification languages considered in [127], this is not the case. Even more, for CTL and CTL*, the controller problem for $\mathcal{D}_{1,h}$ has the same complexity as the satisfiability problem for these logics. Intuitively, the reason that the controller problem for $\mathcal{D}_1$ is not harder than satisfiability in the branching time setting is that branching time specifications talk about trees.[10] So, in a sense, they already comprise the interaction between the environment and the controller implicitly. In fact, modulo some technical details of the representation of strategies as trees, the branching time controller problem for the system $\mathcal{D}_1$ *is* the satisfiability problem for branching time specifications.

[9]In this paper, Pnueli and Rosner consider in fact mostly asynchronous systems. The 2 EXPTIME-hardness of the controller problem, however, is proved out for synchronous systems.

[10]In general, branching time specifications talk about transition systems, but due to the tree model property, we can restrict our attention to trees.

Theorem 3.17. *[127] The controller problem for $\mathcal{D}_{1,h}$ is* Exptime*-complete for CTL specifications and 2-*Exptime*-complete for CTL* and LTL specifications.*

The lower bounds for CTL and CTL* follow immediately from those for the satisfiability problems, the lower bound for LTL follows from that for $\mathcal{D}_1$. To see why we obtain the same upper bound as for $\mathcal{D}_1$, let us first consider CTL*.

In [20] it is shown that by using *alternating* tree automata one can save an exponential in the translation of CTL* to tree automata, cf. [127]. Now, in the case of the system $\mathcal{D}_{1,h}$, to deal with partial information, an application of the widening operator is enough. But the construction of the automaton $\mathrm{narrow}_Y(\mathcal{A})$ can be applied directly to an alternating tree automaton $\mathcal{A}$ and does not increase the size of the automaton. (While applied to a given nondeterministic automaton it yields an alternating automaton, entailing an exponential blow up of the size!) So the alternation capability of the automaton comprises both, the complexity caused by the branching time specification and the the complexity caused by partial information, at the same time. Hence, we have to apply the translation into an equivalent nondeterministic tree automaton only once and end up with a total complexity that is doubly exponential in the length of the CTL* formula.

Now, for a given LTL formula φ, the controller problem for $(\mathcal{D}, \varphi)$ is equivalent to the controller problem $(\mathcal{D}, \forall\varphi)$ (notice that $\forall\varphi \in$ CTL*), so the same complexity holds for LTL. On the other hand, for the case of CTL, to obtain a total complexity that is only exponential in the length of the formula, the construction has to be modified seriously, cf. [127]. Details can be found in [126].

$\mathcal{D}_{1,h}$ and ω-Automata. Now we consider the case where the specification is already given by a deterministic parity automaton $\mathcal{B}$. In the case of $\mathcal{D}_1$, solving the controller problem amounts to solving a parity game with full information on a finite graph: The controller problem for $(\mathcal{D}_1, L(\mathcal{B}))$ can be reduced to the emptiness problem for a *deterministic* parity tree automaton $\mathcal{A}$ which simulates $\mathcal{B}$ along the paths of a tree. In particular, $\mathcal{A}$ has the same set of states and the same coloring as $\mathcal{B}$. Now solving the emptiness problem for $\mathcal{A}$ means to solve a parity game with full information on a finite graph.[11] So the controller problem for $(\mathcal{D}_1, L(\mathcal{B}))$ is in $\mathrm{NP} \cap \text{co-NP}$ and can be solved in time polynomial in the number of states of $\mathcal{B}$ and exponential in the number of colors of $\mathcal{B}$.

For the system $\mathcal{D}_{1,h}$ solving the controller problem means solving the emptiness problem for a *universal* parity tree automaton $\mathcal{A}$ that simulates $\mathcal{B}$ along the paths of a tree which it spans over the input tree. Again, $\mathcal{A}$ has the same set of states and coloring as $\mathcal{B}$, but now, checking emptiness of $\mathcal{A}$ requires time that is exponential in both, the number of states and the number of colors of $\mathcal{B}$.

The construction of $\mathcal{A}$ can also be easily adapted to the case of game graphs with partial information. (For this, the automaton only has to make sure, that it checks only paths which actually correspond to plays on the game graph.) Moreover, Reif has shown that the strategy problem for games on finite graphs with partial information is Exptime-hard, even for reachability and safety conditions [181, 182]. As we have mentioned, for reachability conditions we can always assume that they are observable. Hence, we obtain the following result.

Theorem 3.18. *[182, 221] The strategy problem for games with imperfect information on finite graphs and two players can be solved in exponential time for parity conditions and is* Exptime*-hard for observable reachability conditions.*

Notice that the precise running times of the algorithms for the system $\mathcal{D}_{1,h}$ (and $(\mathcal{D}_1$ as well) clearly depend on the size of the alphabet $\Sigma^{\mathcal{D}_{1,h}}$. In the case of logical specifications like LTL-formulas, this can be bound by $2^{O(|\varphi|)}$ since we can restrict our attention to those signals that are actually addressed in the specification. (Which are, however, exponentially many in

[11] Another way of viewing the controller problem for $(\mathcal{D}_1, L(\mathcal{B}))$ as a parity game with full information on a finite graph is to use the reduction to games on graphs from Section 2.3.

the length of the formula, since LTL-formulas address atomic propositions from a set Prop and in the reactive system, the processes choose actions which are sets $a \in 2^{\mathrm{Prop}}$ of atomic propositions.) In the case of a deterministic parity automaton recognizing the specification, the size of $\Sigma^{\mathcal{D}_{1,h}}$ is bounded by the number of transitions of $\mathcal{B}$. So the construction of the tree automaton $\mathcal{A}$ can be done in time linear in the size of $\mathcal{B}$ and hence, the overall complexity is in fact exponential in the size of $\mathcal{B}$.

Remark. We have already suggested that the emptiness problem for universal parity tree automata can be shown to be EXPTIME-hard by a natural reduction of the strategy problem for two-player parity games with imperfect information. This can now be seen in fact using the previous theorem even for reachability conditions: We have seen how to reduce the strategy problem for two-player reachability games with imperfect information on finite graphs to the emptiness problem of universal reachability tree automata in linear time. Since the former problem is EXPTIME-hard the same holds for the latter.

So far, the only complexity beyond doubly exponential time was that of the controller problem for $\mathcal{D}_1$ where the specification is written in S1S which is due to the fact that S1S is nonelementary more succinct than automata. Monadic second order logic, however, is rarely used to specify systems in practice. The good news is that for the practically appealing formalism of CTL as well as for parity games with imperfect information, the complexity is only exponential, which is the best we can hope for strategy problems under partial information in general, since they always imply a certain form of nondeterminism.

$\mathcal{D}_{n,p}$ and LTL**.** For multiple controllers, the situation is much worse. Apart from the result that most architectures are in fact undecidable, pipelines have a nonelementary complexity, already for simple specifications like LTL: Given a pipeline $\mathcal{D}_{n,p}$ with n controllers and an LTL specification φ, solving the controller problem for $(\mathcal{D}_{n,p}, \varphi)$, asymptotically, takes time at least $\exp_n(|\varphi|)$, where

$$\exp_1(k) = 2^k \quad \text{and} \quad \exp_{n+1}(k) = 2^{\exp_n(k)}.$$

This nonelementary lower bound has been proven in [163] and in [171], Pnueli and Rosner have restated the result in terms of distributed systems.[12]

Theorem 3.19. *[163, 171] The controller problem for $\mathcal{D}_{n,p}$ is n-*EXPTIME*-hard for LTL specifications.*

So while for games with only two players, an important task is to find special cases of the strategy problem which can be solved more efficiently for games with multiple players, the focus is usually on characterizing decidable subcases. We tackle the first challenge in Chapter 4. Contributions to the second issue are presented in Chapter 5 and Chapter 6.

3.4 Context-free Specifications

In Chapter 5 we prove certain decidability results for distributed systems where the specifications may be deterministic context-free but are required to be locally decomposable. In fact, we show that in the decomposition of the specification, at most one of the individual specifications may be context-free. Here we show that for arbitrary (that means, not necessarily locally decomposable) deterministic context-free specifications most controller problems are undecidable. For the case of the system $\mathcal{D}_{1,h}$ this has already been shown in [163].[13] We prove that even for deterministic

[12]It should be mentioned that the lower bound of $\exp_n(|\varphi|)$ is a prudent estimation: It holds for the stark, straight pipeline regardless of details like hidden channels from the environment that can be easily overlooked.

[13]Their proof of this result does not use the asynchronous observability of their model.

realtime 1-counter specifications and any nontrivial extension of the system $\mathcal{D}_1$, the controller problem is not even recursively enumerable.

For the proof we use the fact that the non-halting problem for 2-register machines is not recursively enumerable. Since we use this problem also for our undecidability proofs for locally decomposable specifications in Chapter 5, let us fix some notation for 2-register machines.

A 2-register machine $\mathcal{R}$ consists of a sequence $I_0, \ldots, I_{k-1}, I_k$ of instructions where $I_k = \texttt{stop}$ and for any $j \in [k]$, I_j is one of the following instructions:

- $\texttt{inc}(R_\iota)$ or $\texttt{dec}(R_\iota)$ where $\iota \in \{1,2\}$
- $\texttt{if}\ R_\iota = 0\ \texttt{goto}\ l$ where $\iota \in \{1,2\}$ and $l \in [k+1]$.

So $\mathcal{R}$ has two registers for storing natural numbers and may, in each step increase or decrease either one of the registers. The machine starts with instruction I_0 and after performing an increase or a decrease operation in instruction I_j, the machine just jumps to the next instruction I_{j+1}. Moreover, the machine has conditional jumps, depending on whether or not register 1 (register 2, respectively) is empty.

A configuration of $\mathcal{R}$ has the form $C = (l, x_1, x_2)$ where $x_1, x_2 \in \mathbb{N}$, so it contains the current instruction number and the current contents of the two registers. If $C' = (l', x_1', x_2')$ is the successor configuration of a configuration C of $\mathcal{R}$ then we write $C \vdash C'$. Moreover, if $C'' = (l'', x_1'', x_2'')$ then we write $C \overset{\iota}{\vdash} C''$ if $l'' = l'$ and $x_\iota'' = x_\iota'$, that means, C'' is the successor configuration of C with respect to the instruction number and the contents of register ι (but not necessarily with respect to the other register).

It is well known that the question whether the unique computation of a given 2-register machine when started with empty registers is finite (halting problem) is not decidable. However, the problem is recursively enumerable and so the question whether this unique computation of a given 2-register is *infinite* (non-halting problem) is not recursively enumerable.

Theorem 3.20. *The controller problem for a given architecture $\mathfrak{A}$ with $|P_{con}| \geq 1$ is decidable for DR1-C specifications if, and only if, one of the following two conditions holds.*

(1) $O_{p_0} = \emptyset$

(2) $H_{p_0} = \emptyset$ *and* $|P_{con}| = 1$.

Moreover, if the problem is decidable then it is decidable for arbitrary DCF specifications. If it is undecidable then it is not recursively enumerable.

Proof. First we show the if-direction. If condition (1) holds, then the controller problem for $\mathfrak{A}$ is just the emptiness problem for the given specification which is decidable for DR1-C specifications. If, on the other hand, condition (2) holds, then the controller problem for $\mathfrak{A}$ for an arbitrary deterministic context-free specification can be reduced to solving a parity game with *full information* on a pushdown graph which is decidable [219] (cf. Section 2.2.2).

Now we turn to the only-if-direction, so assume that conditions (1) and (2) do not hold, that is, $O_{p_0} \neq \emptyset$ and first let $H_{p_0} \neq \emptyset$. Moreover, let $p_1 \in P_{\text{con}}$ be some controller, c_0 some hidden channel of p_{env} and let c_1 be an external output channel of p_1. Now let $\mathcal{R}$ be a 2-register machine as described above.

The idea for the reduction is as follows. Process p_1 has the task of writing the unique computation of $\mathcal{R}$ when started on the empty registers to the channel c_1 configuration by configuration. That is, in each macro step, it writes the number of some instruction, two sequences of symbols representing the contents of the registers and then proceeds again with the number of some instruction. To make sure that this sequence of configurations is indeed the computation of $\mathcal{R}$ when started on the empty registers, we code the initial configuration into the specification and we also give p_0 the possibility to check one of the registers at some unique point in time secretly as follows. Process p_1 has three symbols $0, 1, 2$ at its disposal for its hidden channel c_0. If p_0 writes 0 then this means that it does not want to check any of the registers and if it sends $\iota \in \{1,2\}$, then this means that it wants to check register ι. So when p_0

writes ι to c_0, then this triggers the deterministic pushdown automaton (that recognizes the specification of the system) to store the contents of register ι that p_1 writes for the *current* configuration (in the current macro step) and then check that the contents of this register that p_1 writes for the next configuration is built according to the instruction of $\mathcal{R}$ that p_1 has chosen at the beginning of the previous macro step (which can be stored in the state space of the automaton). We exclude the possibility for p_1 to write the instruction number k, so that when the computation of $\mathcal{R}$ is finite, at some point, p_1 will make a mistake.

More precisely, we define the system $\mathcal{D} = (\mathfrak{A}, (\Sigma_c)_{c \in C})$ by $\Sigma_{c_0} = \{0, 1, 2\}$ and $\Sigma_{c_1} = [k] \cup \{A_1, A_2\}$ where A_1 and A_2 are distinct symbols which are used to represent the contents of the registers. We silence all other channels $c \in C$ in the architecture by defining $\Sigma_c = \{|\}$. (Notice that we do this in particular for all possible channels from p_0 to p_1.) The specification $L \subseteq (\Sigma_{c_0} \times \Sigma_{c_1})^\omega$ is defined as follows: $\alpha_0 {}^\frown \alpha_1 \in (\Sigma_{c_0} \times \Sigma_{c_1})^\omega$ is in L, if the following conditions hold.

(1) α_1 has the form $C_0 C_1 \ldots$ where $C_j \in [k] \cdot \{A_1\}^* \cdot \{A_2\}^*$ for each j

(2) $C_0 = 0\varepsilon\varepsilon$

(3) if there exists a $r \in \mathbb{N}$ with $\alpha_0(r) \in \{1, 2\}$ then consider the smallest such r and the smallest $s \geq r$ with $\alpha_1(s) \in [k]$, and let $\alpha_1 = \alpha_1(<s) C C' \gamma$.
Then $C \overset{\alpha_0(r)}{\vdash} C'$.

So condition (1) ensures that p_1 really writes a configuration of $\mathcal{R}$ in each macro step while condition (2) restricts the first configuration to the initial configuration of $\mathcal{R}$ when started on empty registers. Clearly, conditions (1) and (2) are regular. Moreover, condition (3) implements the possibility of p_0 to check the correctness of the construction that p_1 provides: Upon the first occurrence of a symbol $\iota \in \{1, 2\}$ in α_0, we consider the *next two* configurations C and C' that p_1 constructs. (Notice that if $\alpha_0(0) \in \{1, 2\}$, that is, p_0 wants to check register $\alpha_0(0)$ right at the beginning of the construction, then we consider the first and the second configuration that p_1 constructs.) Since we check the correctness of the construction for C and C' only with respect to *one* of the registers, it is not hard to see that condition (3) is in fact DR1-C recognizable.

If $\mathcal{R}$ does not halt when started on empty registers then writing the unique run of $\mathcal{R}$ with initially empty registers is clearly a winning strategy for p_1 for $(\mathcal{D}, L)$. Now let conversely be σ_1 a winning strategy for p_1 for $(\mathcal{D}, L)$. Notice that except the channel c_0, all channels from p_0 are silenced and so σ_1 is a function $\{|\}^* \to \Sigma_{c_1}$. That means, there is an $\alpha_{\sigma_1} \in (\Sigma_{c_1})^\omega$ such that *for all* $\alpha_0 \in \Sigma_{c_0}^\omega$, the system run $\alpha_0 {}^\frown \alpha_1$ is consistent with σ_1. Moreover, since σ_1 is a winning strategy, each such system runs satisfies conditions (1) – (3). In particular, α_{σ_1} satisfies conditions (1) and (2), that is, $\alpha_{\sigma_1} = C_0 C_1 \ldots$ where $C_j \in [k] \cdot \{A_1\}^* \{A_2\}^*$ for each j and $C_0 = 0\varepsilon\varepsilon$. If $C_j \vdash C_{j+1}$ for all $j \in \mathbb{N}$ then clearly $\mathcal{R}$ does not halt when started on empty registers.

Now assume that $C_j \not\vdash C_{j+1}$ for some $j \in \mathbb{N}$ and consider the smallest such j. Then we have $C_j \overset{1}{\not\vdash} C_{j+1}$ or $C_j \overset{2}{\not\vdash} C_{j+1}$. In the first case set $\iota = 1$ and in the second case let $\iota = 2$. Now define $\alpha_{\sigma_0} = 0^{|C_0 \ldots C_{j-1}|}\, \iota\, 0^\omega$. As we have observed, $\alpha_{\sigma_0} {}^\frown \alpha_{\sigma_1}$ is consistent with σ_1. However, the smallest $r \in \mathbb{N}$ with $\alpha_0(r) \in \{1, 2\}$ is $r = |C_0 \ldots C_{j-1}|$ and the smallest $s \geq r$ with $\alpha_1(s) \in [k]$ is $s = r$. Moreover, $\alpha(<s) C C' \gamma = C_0 \ldots C_{j-1} C_j C_{j+1} \gamma$ and since $C_j \overset{\iota}{\not\vdash} C_{j+1}$, $\alpha_{\sigma_0} {}^\frown \alpha_{\sigma_1}$ does not satisfy condition (3) which contradicts the fact that σ_1 is a winning strategy.

Now consider the case $H_{p_0} = \emptyset$ but $O_{p_0} \neq \emptyset$ and $|P_{\text{con}}| \geq 2$. Let $p_1, p_2 \in P_{\text{con}}$ such that p_0 sends information to p_2. We simulate the proof given above as follows. Process p_1 takes the same role as before and the function of p_2 is only to receive the information that p_0 has sent to channel c_0 in the proof given above. Process p_1 cannot see this information and so the proof works just as before. □

Remark. Notice that the specification L constructed in the proof can even be recognized by a deterministic realtime 1-counter automaton that merely uses a reachability acceptance condition.

Consider the system $\mathcal{D}_{1,h}$. We have reduced the controller problem for $(\mathcal{D}_{1,h}, L)$ with *regular* L to the emptiness problem for universal parity tree automata. In the very same way we can reduce the controller problem for $(\mathcal{D}_{1,h}, L)$ with *deterministic context-free* L to the nonemptiness problem for universal parity pushdown tree automata. This gives an alternative proof of the fact that the nonemptiness problem for universal parity pushdown tree automata is undecidable and it also demonstrates that this holds even for universal realtime parity 1-counter tree automata.

3.5 Powerset Construction

In [182], Reif presented a powerset construction for solving the strategy problem for two-player games on finite graphs $\mathcal{G}$ with partial information by turning them into two-player games on finite graphs $2^{\mathcal{G}}$ with full information.[14] This game graph can be viewed as a compressed representation of the possible states of knowledge of player 1 during plays of the original game and the dynamics of this knowledge. In this sense, it is a succinct version of a *knowledge tracking* construction, as mentioned in the introduction.

Historically, this method precedes the automata based methods for solving controller problems that we have presented in the previous section but was presented in a quite different context of Turing machines. Reif aimed at generalizing the theory of alternating computing devices initiated by Chandra, Kozen and Stockmeyer [49, 48], which correspond to two-player games with *full information.* The private alternating machines of Reif correspond to two-player games with *imperfect information.*[15] Therefore, the method was originally restricted to games with winning conditions that correspond to the acceptance condition of Turing machines, that is, reachability conditions.[16]

Due to this quite different research context and the small class of winning condition to which the method was applied in [182], it was not very popular in the theory of synthesis under partial information. Although certain ideas from [182] and [163] have been adopted in the work of Pnueli and Rosner [171] as well as Kupferman and Vardi [129] on controller problems for distributed systems, strong results for games with expressive classes of winning conditions and multiple players have been obtained using tree automata, cf. Section 3.2.

On the other hand, the method of Reif can be easily generalized to two-player games with arbitrary *observable* winning conditions and for games on graphs, observable winning conditions form a non-trivial and relevant special case which still has a high complexity: The strategy problem for two-player games on finite game graphs with observable reachability conditions is EXPTIME-hard [181], see also Sections 2.2.3 and 3.3. More recently, there has been ample research in how Reif's method can be used to obtain more efficient solutions for this special case.

A popular approach is the antichains method that has first been proposed in [66] for safety games and has been extended to games with observable parity conditions in [52]. This method yields an algorithm that has still an exponential running time in the worst case, but outperforms the algorithm that results from explicitly carrying out the powerset construction in practical benchmarks.[17] We will discuss the antichains approach briefly in Section 3.5.3.

In contrast, in Chapter 4, we utilize the powerset construction to provide an algorithm that runs in polynomial time on certain classes of graphs, even in the worst case. More precisely, we

[14]In [181], Reif already presented an algorithm which uses the same idea. It was only in [182] though that Reif made this view as powerset construction explicit.

[15]In [163], Peterson and Reif generalized this concept further to games with *multiple players.*

[16]Notice that for Turing machines that have only finite computations, the winning condition for the existential player can also be represented as safety conditions.

[17]There are also classes of games where the result of the powerset construction has exponential size but the antichains algorithm needs only polynomial time to solve the strategy problem for these games. However, no general criteria are known which would guarantee that the algorithm runs in polynomial time.

prove that on two-player game graphs with bounded partial information *and* bounded DAG-width, the strategy problem can be solved in polynomial time for observable parity conditions. This gives a general criterion that guarantees that the strategy problem is in PTIME and the corresponding classes of games still form a relevant special case. On the other hand, the efficiency of this method relies on these structural restrictions and it may be much worse on other classes of game graphs, in particular compared to the antichains approach.

In this section, we describe the powerset construction of Reif and we demonstrate how it can be generalized not only to observable winning conditions but to arbitrary winning conditions. We also adapt the construction to a certain case of asynchronous observability and we show that in both cases, ω-regular winning conditions are effectively preserved by this construction. This section is partially based on work presented in [172, 173].[18]

3.5.1 Generalized Construction

We start with the original powerset construction for two-player reachability games with imperfect information on finite graphs.[19] So, let $\mathcal{G} = (V, \delta, (\sim^V), (\sim^A))$ be a game graph with partial information and two players and let $R \subseteq V$ be a set defining a reachability condition on $\mathcal{G}$. We make the usual assumptions on $\mathcal{G}$, in particular, we assume that $\mathcal{G}$ is turn based and player 1 always knows when it is his turn (for all $u \in V$, $[u] \subseteq V_i$ for some $i \in \{0, 1\}$). Moreover, player 1 always knows which actions are available to him ($u \sim^V v \implies \mathrm{act}_1(u) = \mathrm{act}_1(v)$) and he can distinguish any two of his own actions ($a, b \in A_1 \implies a \not\sim^A b$).[20]

First, reconsider the solution of the controller problem for the system $\mathcal{D}_{1,h}$ from Section 3.2 for a regular specification $L \subseteq (\Sigma^{\mathcal{D}_{1,h}})^\omega$ represented by a deterministic parity automaton $\mathcal{B}$. There we have defined the specification $L_{\forall\alpha_0} \subseteq \Sigma^{\mathcal{D}_1}$ that reduced the controller problem for $\mathcal{D}_{1,h}$ to one for the system $\mathcal{D}_1$. Remind that the specification $L_{\forall\alpha_0}$ requires, for a given run $\alpha_{01}{}^\frown\alpha_1 \in \Sigma^{\mathcal{D}_1}$ that for *all* hidden sequences $\alpha_0 \in \Sigma_0^\omega$ we have $\alpha_0{}^\frown\alpha_{01}{}^\frown\alpha_1 \in L$.

This construction can be seen as introducing *nondeterminism* to the automaton $\mathcal{B}$ that corresponds to the *partial information* of p_1: A history $u_{01}{}^\frown u_1$ induces a set of runs of $\mathcal{B}$ each of which corresponds to a particular hidden history u_0 that p_1 cannot observe. We have then constructed a deterministic parity automaton $\mathcal{B}_{\forall\alpha_0}$ recognizing $L_{\forall\alpha_0}$ which involves determinization of ω-automata. However, if $\mathcal{B}$ is a *reachability* automaton, then for this determinization, the usual powerset construction is sufficient.[21]

This yields a reduction of the controller problem for $\mathcal{D}_{1,h}$ for reachability specification to the controller problem for $\mathcal{D}_1$ that only involves a simple powerset construction. Now the reduction of game graphs with partial information to distributed systems from Section 2.3 preserves reachability conditions, so we can solve the strategy problem for $(\mathcal{G}, R)$ using this translation.

Reif's construction, on the other hand, can be seen as a direct implementation of this construction on the given game graph, where partial information is defined by the equivalence relations $\sim^V$ and $\sim^A$. Having the foregoing considerations in mind, it is now comprehensible how the game graph $2^{\mathcal{G}}$ should be defined. We denote $2^{\mathcal{G}} = (2^V, \overline{\Delta})$ and the positions of player 0 are 2^{V_0}. Moreover,

$$(\overline{u}, a, \overline{v}) \in \overline{\Delta} \;:\Longleftrightarrow\; \overline{v} = \mathrm{Post}_{[a]}(\overline{v}) \cap [v] \text{ for some } v \in \overline{v}.$$

[18] Certain technical proofs that are omitted here can be found in [172] and in the full version of [173], currently available at www.logic.rwth-aachen.de.

[19] Reif's model is in fact a somewhat restricted special case of our game graphs with partial information. However, the adaption to our more general model is straightforward.

[20] As we have mentioned, these properties are just technical simplifications. We can assume them w.l.o.g., but the powerset construction could also easily be adapted to cover cases in which they do not hold.

[21] Notice that we have to assume a certain normalform where the accepting states have as outgoing transition only a selfloops with all possible input letters on it. This corresponds to the assumption that a reachability condition is observable. Since we use these observations only as an intuition for the powerset construction, we don't go into details.

The definition of the Post-operator is as usual: For $U \subseteq V$ and $B \subseteq A$,

$$\mathrm{Post}_B(U) = \{v \in V \mid \exists\, u \in U\ \exists\, b \in B : \delta(u, b) = v\}.$$

We also denote $2^{\mathcal{G}}$ as $\overline{\mathcal{G}} = (\overline{V}, \overline{\Delta})$. Notice that $2^{\mathcal{G}}$ is a nondeterministic game graph with full information. In Section 2.1.2 we have seen how we can easily construct an equivalent deterministic game graph with full information.

Now, to appreciate the term of knowledge tracking that we have mentioned at the beginning of this section, for a history $\pi \in v_0(\mathrm{AV})^*$ in $\mathcal{G}$, let us define

$$\mathrm{Tr}(\pi) = \{\rho \sim^* \pi \mid \rho \in v_0(\mathrm{AV})^* \text{ is a history in } \mathcal{G}\}.$$

So $\mathrm{Tr}(\pi)$ is an explicit representation of the knowledge of player 1 after the history π has been played. [22] However, $\{\mathrm{Tr}(\pi) \mid \pi \in v_0(\mathrm{AV})^* \text{ is a history in } \mathcal{G}\}$ is obviously infinite. We approximate this representation by identifying sets $\mathrm{Tr}(\pi)$ and $\mathrm{Tr}(\rho)$ if π and ρ induce the same *set of positions* that player 1 considers possible after playing π and ρ, respectively. We denote this set as follows:

$$\mathrm{core}(\mathrm{Tr}(\pi)) = \{\mathrm{last}(\rho) \mid \rho \in \mathrm{Tr}(\pi)\}.$$

For readability we also abbreviate $\mathrm{core}(\mathrm{Tr}(\pi))$ by $V(\pi)$. It is easy to see that $V(v_0) = \{v_0\}$ and, for any history $\pi a v \in v_0(\mathrm{AV})^+$ in $\mathcal{G}$, the following holds.

Proposition 3.21. $V(\pi a v) = \mathrm{Post}_{[a]}(V(\pi)) \cap [v]$.

So, by induction, any position in $2^{\mathcal{G}}$ that is reachable from $\{v_0\}$ has the form $\mathrm{core}(\mathrm{Tr}(\pi))$ for some history $\pi \in v_0(\mathrm{AV})^*$ in $\mathcal{G}$ and the edges in $\overline{\Delta}$ reflect how moves in plays of $\mathcal{G}$ cause transitions between these states.

Remark. Notice that $\mathrm{core}(\mathrm{Tr}(\pi)) = \mathrm{core}(\mathrm{Tr}(\pi))$ does not imply $\pi = \rho$. (This is trivial because only histories of equal length can be equivalent. But in general, also for histories of equal length, this implication does not hold.) However, we will see that player 1 has a winning strategy for $(\mathcal{G}, R, v_0)$ if, and only if, he has a winning strategy for $(2^{\mathcal{G}}, 2^R, \{v_0\})$. Since $2^{\mathcal{G}}$ is a game graph with *full information*, in this sense, the representation of $\mathrm{Tr}(\pi)$ by $\mathrm{core}(\mathrm{Tr}(\pi))$ as defined above is in fact adequate for two-player games. In particular, since positional strategies suffice to win in reachability games with full information, whenever player 1 has a winning strategy for $(\mathcal{G}, R, v_0)$ he also has one that depends only on $\mathrm{core}(\mathrm{Tr}(\pi))$ instead of $\mathrm{Tr}(\pi)$. It is crucial to mention, however, that for games with multiple players, this is not the case! We will discuss this in Chapter 6.

Now a history $\overline{\pi} = \overline{w}_0 b_1 \overline{w}_1 \ldots b_l \overline{w}_l$ in $\overline{\mathcal{G}}$ induces a *set* of histories in $\mathcal{G}$ which we denote $\Pi(\overline{\pi})$ and which can be described as follows:

$$\text{A history } \pi = v_0 a_1 v_1 \ldots a_l v_l \text{ is in } \Pi(\overline{\pi}) \text{ if } v_j \in \overline{w}_j \ \text{ and } \ a_j \sim^A b_j.$$

A fundamental property of the powerset construction is that for each history $\overline{\pi}$ in $\overline{\mathcal{G}}$, the set $\Pi(\overline{\pi})$ is nonempty. The proof is straightforward by induction, using Proposition 3.21.

Proposition 3.22. *Let $v_l \in \overline{w}_l$.*

(1) There is a history $\pi \in \Pi(\overline{\pi})$ such that $\mathrm{last}(\pi) = v_l$

(2) For each history $\pi \in (\overline{\pi})$ we have $V(\pi(\leq j)) = \overline{w}_j$.

[22]Notice that we have relativized the definition to histories that start in v_0 which reflects the assumption that player 1 *knows* the initial position, cf. Section 2.1.2.

Winning Conditions. So far, we have only defined the powerset game graph $2^{\mathcal{G}}$ but we did not define the winning condition for this new game. We have started with an arbitrary reachability condition but, as we have noted in Section 2.2.3 we can assume, w.l.o.g., that R is observable for player 1, that is, for each $\overline{u} \subseteq [v]$ for some $v \in V$ we have either $\overline{u} \in 2^R$ or $\overline{u} \in 2^{V \setminus R}$. Now once we have this assumption, we can derive the definition of the winning condition from the powerset construction for reachability automata. Notice, however, that for the construction of a deterministic reachability automaton recognizing $L_{\forall\alpha_0}$ we either have to start with a *universal* automaton or to finish the construction with a *complementation* step. So while for the determinization of a nondeterministic reachability automaton we define the accepting states to be those that *contain* an accepting state of the original automaton, here the winning states for player 1 are those that consist entirely of winning states from R, that is, $\overline{R} = 2^R$.

Now, it is not hard to see that this definition can be easily extended to observable parity conditions. If $\mathrm{col} : V \to [k]$ is an observable coloring of $\mathcal{G}$ then we can define the coloring $\overline{\mathrm{col}} : 2^V \to [k]$ as follows. For any $\overline{u} \in 2^V$ we define $\overline{\mathrm{col}}(\overline{u}) := \mathrm{col}(u)$ for some $u \in \overline{u}$. So if $\overline{u} \subseteq [v]$ for some $v \in V$ then this is independent of the chosen u. If, on the other hand, $\overline{u}$ is not a subset of some equivalence class, then the color of $\overline{u}$ does not matter since $\overline{u}$ is not reachable in $2^{\mathcal{G}}$ from the initial position $\{v_0\}$.

For non-observable parity conditions, however, it is not clear how to define a coloring on $2^{\mathcal{G}}$ in terms of the coloring on $\mathcal{G}$ in such a direct way. Instead, for non-observable winning conditions, we define the winning condition for $2^{\mathcal{G}}$ in a similar way as the winning condition $L_{\forall\alpha_0}$. However, due to the underlying game graph and the way in which partial information is defined on the graph, the definition is a little more involved than in the case of $L_{\forall\alpha_0}$: In the same way as for histories, a play $\overline{\pi} = \overline{w}_0 b_1 \overline{w}_1 b_2 \overline{w}_2 \dots$ in $2^{\mathcal{G}}$ induces a *set of plays* in $\mathcal{G}$ which we also denote as $\Pi(\overline{\pi})$ and which is defined accordingly:

A play $\pi = v_0 a_1 v_1 a_2 v_2 \dots$ is in $\Pi(\overline{\pi})$ if $v_j \in \overline{w}_j$ and $a_j \sim^A b_j$ for all j.

Now, analogously to the definition of $L_{\forall\alpha_0}$, a play $\overline{\pi}$ can be defined as being winning for player 1 if *for all* plays $\pi \in \Pi(\overline{\pi})$ are won by player 1. Remind that in the system $\mathcal{D}_{1,h}$, the hidden sequences α_0 are generated by p_0. Similarly, the uncertainties of player 1 in the game played on $\mathcal{G}$ are generated by the moves of player 0. So, given a winning condition $W \subseteq \Pi$ for player 1 for $\mathcal{G}$, we define the winning condition $\overline{W} \subseteq \overline{\Pi}$ by

$$\overline{\pi} \in \overline{W} \;:\Longleftrightarrow\; \Pi(\overline{\pi}) \subseteq W.$$

Notice that $\overline{W}$ is not position based. This is clearly unavoidable if W is not position based. But even if W is position based, in general we can not define a position based winning condition on $2^{\mathcal{G}}$ such that winning strategies of player 1 are preserved. Intuitively this is quite clear:[23] The actions of player 0 may carry information that is important for player 1 to win the game but which he cannot infer from the observations that he makes about the positions. Now, if we re-define $\Pi(\overline{\pi})$ by omitting the condition on the actions (of player 0), this would be essentially the same as hiding the information about the actions from player 1 since we would require that player 1 wins certain plays uniformly (by choosing the same actions) that he actually can distinguish. On the other hand, the dependence of the winning condition on the actions of player 1 guarantees that for a play $\overline{\pi}$ that is consistent with some strategy for player 1 for $2^{\mathcal{G}}$, all plays in $\Pi(\overline{\pi})$ are also consistent with some corresponding strategy of player 1 for $\mathcal{G}$.[24] So by omitting the condition on the actions of player 1 in definition of $\Pi(\overline{\pi})$ would require player 1 to win certain plays that are not consistent with his winning strategy for $\mathcal{G}$.

Figure 3.1 shows a game graph $\mathcal{G}$ with partial information and with a position based winning condition where the winning condition $\overline{W}$ for $2^{\mathcal{G}}$ depends on the actions of player 0 and of

[23]For the technical details, see the proof of Proposition 3.26.

[24]Notice that we have assumed that player 1 can distinguish all his actions, so $a_j \sim^A b_j$ implies $a_j = b_j$. Without this assumption, we would have to require in the definition of $\Pi(\overline{\pi})$ explicitly $a_j = b_j$ if $v_{j-1} \in V_1$.

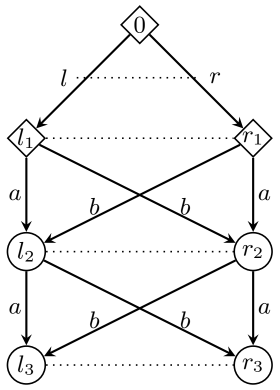


Figure 3.1: $W = \{0l_1l_2l_3, 0r_1r_2r_3, 0l_1r_2l_3, 0r_1l_2r_3\}$ is position based but $\overline{W}$ depends on actions

player 1. Player 1 clearly has a winning strategy for $\mathcal{G}$ from 0: He can simply choose action a when it is his turn. On the other hand, in the definition of $\overline{W}$ as given above, dependence on actions is crucial since otherwise, $\overline{W}$ would be empty, leaving player 1 out in the cold.

Notice that the example from Figure 2.1 does not show this: There the winning condition $\overline{W}$ is position based and so, while player 1 does not have a position based winning strategy for $(\mathcal{G}, W)$, he does have one for $(2^{\mathcal{G}}, \overline{W})$. The reason is that the information about the actions of player 0 is included in the sets that we obtain in the powerset construction: While $\text{Post}_{[a_1]}(0) = \{1, 2\}$, we have $\text{Post}_{[a_3]}(v_0) = \{2, 3\}$. This, however, is sufficient only because the W is *observable.*

In fact, this observation generalizes to arbitrary observable winning conditions: If W is observable, then $\overline{W}$ is position based. To show this we first prove that for a play $\overline{\pi}$ in $2^{\mathcal{G}}$ we have $\Pi(\overline{\pi}) \neq \emptyset$. Since $\mathcal{G}$ is finite, this follows from Proposition 3.22 using König's Lemma.

Proposition 3.23. *For each play $\overline{\pi}$ in $2^{\mathcal{G}}$, the set $\Pi(\overline{\pi})$ is not empty.*

Proof. Let $\overline{\pi} = \overline{w}_0 b_1 \overline{w}_1 b_2 \overline{w}_2 \ldots$ with $\overline{w}_0 = \{v_0\}$. We define a tree $\zeta(\overline{\pi})$ as follows. The l-th level of $\zeta(\overline{\pi})$ consists of all finite histories $\pi \in \Pi(\overline{\pi}(< l + 1))$. In particular, the root of $\zeta(\overline{\pi})$ is v_0 and, for a history $\rho = \pi a v$ for some $a \in A$ and some $v \in V$, the unique predecessor of ρ in $\zeta(\overline{\pi})$ is π. By Proposition 3.22, each level of $\zeta(\overline{\pi})$ is nonempty and since $\mathcal{G}$ is finite, $\zeta(\overline{\pi})$ is finitely branching. Therefore, by Königs Lemma, $\zeta(\overline{\pi})$ contains an infinite path $\pi = v_0 a_1 v_1 a_2 v_2 \ldots$ which is a play in $\mathcal{G}$ and, by construction of $\zeta(\overline{\pi})$, is contained in $\Pi(\overline{\pi})$. □

Now, for a play $\pi = v_0 a_1 v_1 a_2 v_2 \ldots$ we define the sequence $\text{obs}(\pi) = [v_0][v_1][v_2] \ldots$ of equivalence classes that player 1 observes during the play π and for a play $\overline{\pi} = \overline{w}_0 b_1 \overline{w}_1 \ldots$ in $2^{\mathcal{G}}$ we define $\text{obs}(\overline{\pi}) = \text{obs}(\pi)$ for some $\pi \in \Pi(\overline{\pi})$. Moreover, let $\text{obs}(W) := \{\text{obs}(\pi) \mid \pi \in W\}$.

Proposition 3.24. *If W is observable, then $\overline{W} = \{\overline{\pi} \in \overline{\Pi} \mid \text{obs}(\overline{\pi}) \in \text{obs}(W)\}$.*

In particular, if W is observable, then $\overline{W}$ is position based. Moreover, this observation demonstrates, that observable parity conditions are included in the general definition of $\overline{W}$ as a special case: If W is a parity condition defined by an observable coloring $\text{col} : V \to [k]$ then $\overline{W}$ coincides with the winning condition defined by $\overline{\text{col}}$.

Now, the powerset construction yields for a game graph $\mathcal{G}$ and an arbitrary winning condition $W \subseteq \Pi$ a new game graph $\overline{\mathcal{G}}$ and a corresponding winning condition $\overline{W} \subseteq \overline{\Pi}$ such that winning strategies for player 1 are preserved. However, we still have to show that if the winning condition W is in fact regular, then $\overline{W}$ is regular as well. For this, assume that W is given by a deterministic parity automaton $\mathcal{B} = (\Sigma, Q, q_{\text{in}}, \delta^{\mathcal{B}}, \text{col})$ with $\Sigma = V \cup \text{AV}$, that is, $W = \{\pi \in \Pi \mid \pi \in L(\mathcal{B})\}$. Notice that this includes the case of parity games with non-observable coloring.

The idea for the automaton $\overline{\mathcal{B}} = (\overline{\Sigma}, \overline{Q}, \overline{q}_{\text{in}}, \delta^{\overline{\mathcal{B}}}, \overline{\text{col}})$ with $\overline{\Sigma} = \overline{V} \cup \text{A}\overline{\text{V}}$ defining $\overline{W}$ is the same for the automaton $\mathcal{B}_{\forall\alpha_0}$ recognizing $L_{\forall\alpha_0}$: It runs on a play $\overline{\pi} = \overline{w}_0 b_1 \overline{w}_1 b_2 \overline{w}_2 \ldots \in \overline{\Pi}$, universally branches over all plays $\pi = v_0 a_1 v_1 a_2 v_2 \ldots \in \Pi(\overline{\pi})$ and at the same time simulates $\mathcal{B}$ on these plays π. The requirement that only sequences which are actually *plays* in $\mathcal{G}$ should be checked, as well as the conditions defining $\Pi(\overline{\pi})$, can obviously be implemented into an automaton.

Formally, we define $\overline{\mathcal{B}}$ as follows. Notice that we consider only plays $\overline{\pi}$ in $\overline{\mathcal{G}}$ that start in the initial position $\overline{v}_0 = \{v_0\}$.

- $\overline{Q} = \{\overline{q}_{\text{in}}\} \cup Q \times V$, $\overline{\text{col}}(\overline{q}_{\text{in}}) = \text{col}(q_{\text{in}})$ and $\overline{\text{col}}(q, v) = \text{col}(q)$
- $\delta^{\overline{\mathcal{B}}}(\overline{q}_{\text{in}}, \overline{v}_0) = (\delta^{\mathcal{B}}(q_{\text{in}}, v_0), v_0)$

Moreover, for $(q, v) \in \overline{Q}$ and $a\overline{w} \in \text{A}\overline{\text{V}}$,

$$\delta^{\overline{\mathcal{B}}}((q, v), b\overline{w}) = \bigwedge_{[a \sim^A b]} \bigwedge_{[w \in \text{Post}_a(v) \cap \overline{w}]} (\delta^{\mathcal{B}}(q, aw), w).$$

So $\overline{\mathcal{B}}$ always stores the position v that has been chosen for the current branch which, in the first step, is only v_0. Then, reading $b\overline{w}$, the automaton $\overline{\mathcal{B}}$ branches over all actions $a \sim^A b$ and over all a-successors $w \in \overline{w}$ of v. Notice that, for each action $a \sim^A b$ there is *at most* one such a-successor w of v that is contained in $\overline{w}$. At the same time, $\overline{\mathcal{B}}$ simulates $\mathcal{B}$ on the plays that it generates in this way and so it is not hard to see that a play $\overline{\pi} \in \overline{\Pi}$ is in $L(\overline{\mathcal{B}})$ if, and only if, $\Pi(\overline{\pi}) \subseteq L(\mathcal{B})$, that means, $\overline{\pi} \in \overline{W}$.

Proposition 3.25. $\overline{W} = \{\overline{\pi} \in \overline{\Pi} \mid \overline{\pi} \in L(\overline{\mathcal{B}})\}$.

So, regular winning conditions are effectively preserved by the powerset construction, using the general definition of $\overline{W}$ as presented above. Using this, we can solve a parity game with imperfect information on a finite graph by performing the powerset construction, constructing $\overline{\mathcal{B}}$ and taking the product parity game graph $(\overline{\mathcal{G}} \times \overline{\mathcal{B}}, \overline{\text{col}}^{\times}, (\{v_0\}, \delta^{\overline{\mathcal{B}}}(q_{\text{in}}, \{v_0\}), \overline{q}_{\text{in}}))$ with full information. Then, some algorithm for parity games with full information can be applied.

Finally, we consider context-free specifications. As we know, arbitrary context-free specifications are too strong for games, so we restrict to deterministic ones. However, as we have seen in Section 3.4, they are still too strong for two-player games with imperfect information. The construction that we have presented for regular specifications fails for deterministic context-free specifications, because the definition of $\overline{W}$ requires a quantification over certain plays in $\mathcal{G}$ and so it inevitably introduces nondeterminism. Therefore, $\overline{W}$ is not necessarily deterministic context-free, even if W is. Hence, the generalized powerset construction reduces a game with imperfect information and deterministic context-free specification to a game with full information and nondeterministic context-free specification, which is not decidable in general.

On the other hand, if W is defined by a deterministic pushdown automaton $\mathcal{P}$ over the alphabet $[V]_{\sim^V}$[25], that is, $W = \{\pi \in \Pi \mid \text{obs}(\pi) \in L(\mathcal{P})\}$, then the strategy problem for $(\mathcal{G}, W)$ is decidable: By Proposition 3.24, $\overline{W} = \{\overline{\pi} \in \Pi \mid \text{obs}(\overline{\pi}) \in L(\mathcal{P})$, so $\overline{W}$ is deterministic context-free.

So far, we have only claimed that the powerset construction in fact preserves winning strategies for player 1. We conclude this section with a proof of this fact and some remarks on infinite game graphs.

Proposition 3.26. *Player* 1 *has a winning strategy for* $(\mathcal{G}, W, v_0)$ *if, and only if, he has a winning strategy for* $(\overline{\mathcal{G}}, \overline{W}, \{v_0\})$.

Proof. Let first σ be a winning strategy for player 1 for $(\mathcal{G}, W, v_0)$. We construct the strategy $\overline{\sigma}$ for player 1 for $(\overline{\mathcal{G}}, \overline{W}, \{v_0\})$ as follows. For a history $\overline{\pi}$ in $\overline{\mathcal{G}}$ from $\overline{w}_0 = \{v_0\}$ with $\text{last}(\overline{\pi}) \in \overline{V}_1$, let $\pi \in \Pi(\overline{\pi})$ and define $\overline{\sigma}(\overline{\pi}) := \sigma(\pi)$. Notice that the definition of $\overline{\sigma}(\overline{\pi})$ is independent of the particular history π.

[25]Analogously to parity conditions defined by an observable coloring

Now consider a play $\overline{\pi} = \overline{w}_0 b_1 \overline{w}_1 b_2 \overline{w}_2 \dots$ in $\overline{\mathcal{G}}$ from $\overline{w}_0 = \{v_0\}$ that is consistent with $\overline{\sigma}$. Let $\pi = v_0 a_1 v_1 a_2 v_2 \dots \in \Pi(\overline{\pi})$ and let $j > 0$ such that $v_{j-1} \in V_1$. Then $\pi(<j) \in \Pi(\overline{\pi}(<j))$, so by definition of $\overline{\sigma}$ we have $\sigma(\pi(<j)) = \overline{\sigma}(\overline{\pi}(<j)) = b_j$. Moreover, the definition of $\Pi(\overline{\pi})$ yields $b_j \sim^A a_j$ and, as player 1 can distinguish any two of his actions, $b_j = a_j$. Therefore, π is consistent with σ. (Notice that for this argumentation it is crucial that $\Pi(\overline{\pi})$ depends on the actions in $\overline{\pi}$!) Since σ is a winning strategy, $\Pi(\overline{\pi}) \subseteq W$, so $\overline{\pi} \in \overline{W}$.

Now let, conversely, $\overline{\sigma}$ be a winning strategy for player 1 for $(\overline{\mathcal{G}}, \overline{W}, \{v_0\})$. To construct the strategy σ for player 1 for $(\mathcal{G}, W, v_0)$, we first fix a subgraph $\overline{\mathcal{H}}$ of $\overline{\mathcal{G}}$ that fulfills the following condition: For each edge $(\overline{u}, a, \overline{v})$ in $\overline{\mathcal{G}}$, there is exactly one $b \in [a]$ such that $(\overline{u}, b, \overline{v})$ is an edge in $\overline{\mathcal{H}}$. (That is, we delete from each edge in $\overline{\mathcal{G}}$ all but one equivalent actions.) Now, for a history $\pi = v_0 a_1 v_1 \dots a_l v_l$ in $\mathcal{G}$ with $v_l \in V_1$, let $\overline{w}_j := V(\pi(\leq j))$ for $j \in \{0, \dots, l\}$. (Notice that $\overline{w}_0 = \{v_0\}$.) Then there are uniquely determined actions $b_j \sim^A a_j$ such that $\overline{\pi} = \overline{w}_0 b_1 \overline{w}_1 \dots b_l \overline{w}_l$ is a history in $\overline{\mathcal{H}}$. Define $\sigma(\pi) = \overline{\sigma}(\overline{\pi})$. Notice that by Proposition 3.22 (2), if $\pi \sim^* \rho$, then $\overline{\pi} = \overline{\rho}$ and so $\sigma(\pi) = \sigma(\rho)$.

Now consider a play $\pi = v_0 a_1 v_1 a_2 v_2 \dots$ in $\mathcal{G}$ that is consistent with σ. As before, if $\overline{w}_j = V(\pi(\leq j))$ for $j \in \mathbb{N}$ then there are uniquely determined actions $b_j \sim^A a_j$ such that $\overline{\pi} = \overline{w}_0 b_1 \overline{w}_1 b_2 \overline{w}_2 \dots$ is a play in $\overline{\mathcal{H}}$. Now let $j > 0$ such that $\overline{w}_{j-1} \in \overline{V}_1$. By definition of σ we have $\overline{\sigma}(\overline{\pi}(<j)) = \sigma(\pi(<j)) = a_j$. Moreover, $b_j \sim^A a_j$ and, since player 1 can distinguish any two of his actions, $b_j = a_j$. Therefore, $\overline{\pi}$ is consistent with $\overline{\sigma}$ and since $\overline{\sigma}$ is a winning strategy we obtain $\Pi(\overline{\pi}) \subseteq W$. Moreover, by construction of $\overline{\pi}$, we have $\pi \in \Pi(\overline{\pi})$, so $\pi \in W$. □

Notice that the construction of the subgraph $\overline{\mathcal{H}}$ in the proof makes sure that σ does not distinguish between actions of player 0 that are actually indistinguishable for player 1: In $\overline{\mathcal{G}}$, player 1 observes any action of player 0, so $\overline{\sigma}$ can depend on these actions, even if they indistinguishable for player 1 in $\mathcal{G}$. On the other hand, if player 0 chooses some action a to go from position $\overline{u}$ to $\overline{v}$, the definition of $\overline{W}$ guarantees that he can choose any action $b \in [a]$ instead just as well. So deleting equivalent actions from the edges in $\overline{\mathcal{G}}$ does not limit the powers of player 0 but makes sure that we construct σ with respect to fixed reference actions of player 0. This also demonstrates that if player 1 has a winning strategy for $(\overline{\mathcal{G}}, \overline{W}, \{v_0\})$, then he has one which does distinguish between equivalent actions.

Infinite Game Graphs. Proposition 3.26 holds also for infinite game graphs, so the generalized definition of $\overline{W}$ extends Reif's powerset construction to both, non-observable winning conditions and arbitrary game graphs.

On the other hand, the proof of Proposition 3.24 requires a game graph that is at least finitely branching. In fact, it is easy to see that there are infinite game graphs with observable winning condition such that the canonical definition of a winning condition for $2^{\mathcal{G}}$ by means of observations does not work (for an example, see [172]). Moreover, in the case of infinitely branching game graphs, observable winning conditions do not even guarantee that $\overline{W}$ is position based.

A particularly interesting case of infinite game graphs is that of graphs generated by pushdown automata, cf. Section 2.2.2. However, since the strategy problem is undecidable for games with imperfect information on finite graphs and specifications defined by deterministic pushdown automata with reachability acceptance, the strategy problem for two-player reachability games with imperfect information on graphs generated by pushdown automata is undecidable as well. So while the generalized powerset construction demonstrates that such games can be turned into games with full information in principle, there is no possibility to obtain an effective presentation of the powerset graph in this case.

Now, Reif's powerset construction offers a method to solve games with imperfect information on finite graphs. There are, however, two drawbacks. First, if the winning condition is not observable, then already for simple Büchi conditions, determinization of ω-automata is needed. So instead of performing the powerset construction and determinization separately, we could also translate the game graph into a distributed systems which merges the winning condition and the structure of the game graph into the specification. Then, the powerset construction is included in the determinization step. Essentially, the complexity will be the same. It is, however, somehow appealing to have a direct method that does not go via this translation as well.

Second, even if the winning condition is observable, in general the powerset construction requires to construct a graph that is exponentially larger than the original graph. However, no automata theoretic methods are needed in this case, so when studying solution methods for the strategy problem for parity games with imperfect information and *observable* coloring, we can focus on the powerset construction. The challenge here is to find optimized versions of the powerset construction as well as subclasses of parity games with imperfect information for which the strategy problem can be solved more efficiently. Remind that for two-player games with imperfect information on finite graphs the strategy problem is EXPTIME-hard, even for observable reachability condition.

In Section 3.5.3, we discuss a construction that avoids the explicit construction of the whole powerset graph in advance. Instead, it proceeds by evaluating formulas of the modal μ-Calculus on the powerset graph and uses and optimized representation of the sets of positions. Moreover, in Chapter 4 we analyze the complexity of the strategy problem for parity games with imperfect information on graphs of bounded complexity and, in particular, the performance of methods that are derived from the powerset construction.

3.5.2 Implementation of Strategies

We have seen how the powerset construction can be used to solved the strategy problem for parity games with imperfect information on finite graphs. As usual, once we know that a winning strategy for player 1 exists, we would also like to be able to construct an implementation of such a strategy (from the information that the algorithm generates).

First, we consider the case of games with observable parity winning conditions, so let $\mathcal{G} = (V, V_0, \delta, \sim^V, \sim^A)$ be a game graph with partial information and let $\mathrm{col} : V \to [k]$ be an observable coloring. Moreover, let $v_0 \in V$ be some initial position such that player 1 has a winning strategy for $(\mathcal{G}, \mathrm{col}, v_0)$. Then player 1 has a winning strategy for $(\overline{\mathcal{G}}, \overline{\mathrm{col}}, \{v_0\})$ and since $\overline{\mathcal{G}}$ is a game graph with full information, player 1 has a *positional* winning strategy $\overline{\sigma} : 2^{V_1} \to A_1$ for $(\mathcal{G}, \mathrm{col}, v_0)$. Now this strategy can also be viewed as a memory strategy σ for $\mathcal{G}$, where the memory structure is, essentially, the game graph $\mathcal{G}$.

More precisely, we define the memory structure $\mathcal{M} = (M, \delta_{\mathrm{in}}, \delta_{\mathrm{up}})$ as follows. The set of states is $M = 2^V$ and the initializing function yields, for some position $v \in V$ the initial state $\delta_{\mathrm{in}}(v) = \{v\}$. The update function is defined according to the transitions in $\overline{\mathcal{G}}$:

$$\delta_{\mathrm{up}}(\overline{u}, [a][v]) := \mathrm{Post}_{[a]}(\overline{u}) \cap [v].$$

Moreover, we define the memory strategy $\sigma : M \to A_1$ by $\sigma(\overline{u}) := \overline{\sigma}(\overline{u})$, if $\overline{u} \subseteq V_1$, and if $\overline{u} \subseteq V_0$ then $\sigma(\overline{u})$ is an arbitrary action from A_1.

According to Proposition 3.21, we have $\delta^*([\pi]) = V(\pi)$ for all histories $\pi \in v_0(\mathrm{AV})^*$ in $\mathcal{G}$. Using this, similarly as in the proof of Proposition 3.26, one can show that for any play in $\mathcal{G}$ that is consistent with σ, the corresponding play in $\overline{\mathcal{G}}$ as constructed in the proof, is consistent with $\overline{\sigma}$ and so, σ is a winning strategy for player 1 for $(\mathcal{G}, \mathrm{col}, v_0)$. (Notice that the proof is even simpler since $\overline{\sigma}$ is a positional strategy and so in particular, it does not depend on the actions in a history.)

In the case of a parity condition that is not necessarily observable, we first construct a deterministic parity automaton $\mathcal{B} = (\Sigma, Q, q_{\mathrm{in}}, \delta, \mathrm{col})$ over the alphabet [V]∪A[V], defining the winning condition $\overline{W}$ and then we take the product parity game graph $(\overline{\mathcal{G}} \times \overline{\mathcal{B}}, \overline{\mathrm{col}}^{\times}, (\{v_0\}, \delta^{\overline{\mathcal{B}}}(\overline{q}_{\mathrm{in}}, \{v_0\})))$. For the game played on this graph, player 1 has again a positional winning strategy $\overline{\sigma}$, but now, this strategy depends not only on the positions $\overline{u} \in 2^V$ of $\overline{\mathcal{G}}$, but also on the state of the automaton $\overline{\mathcal{B}}$. Hence, we have to adapt the definition of $\mathcal{M}$ accordingly: $M = 2^V \times Q$, $\delta_{\text{in}}(v) = (\{v\}, \delta^{\overline{\mathcal{B}}}(q_{\text{in}}, \{v\}))$ and

$$\delta_{\text{up}}((\overline{u}, q), [a][v]) := (\overline{v}, \delta^{\overline{\mathcal{B}}}(q, b\overline{v}))$$

for some $b \in [a]$, where $\overline{v} = \text{Post}_{[a]}(\overline{u}) \cap [v]$. As before, the memory strategy $\sigma : M \to A_1$ is defined by $\sigma(\overline{u}, q) := \overline{\sigma}(\overline{u}, q)$, if $\overline{u} \subseteq V_1$, and if $\overline{u} \subseteq V_0$ then $\sigma(\overline{u}, q)$ is an arbitrary action from A_1.

So for parity game graphs with observable coloring, if player 1 has a winning strategy from some position v_0, then he also has a finite memory winning strategy that uses at most $2^{|V|}$ many memory states.[26] In the remainder of this section we demonstrate, that this bound is essentially optimal, even for safety games.

For $m \in \mathbb{N}$, consider the game graph $\mathcal{G}_m = (V_m, \delta_m, \sim_m^V, \sim_m^A)$, where:

$V_m^0 = \{v_0\} \cup \{v_j^0, v_j^1 \mid j = 1, \dots, m\}$

$V_m^1 = \{w_j^0, w_j^1 \mid j = 1, \dots, m\} \cup \{\text{waste}, \text{haven}\}$

$A = \{\downarrow_1, \dots, \downarrow_m\} \cup \{\neg_1, \dots, \neg_m\} \cup \{?, !, \circlearrowleft\}$.

The transitions of $\mathcal{G}_m$ are defined as follows:

$v_0 \xrightarrow{\downarrow_j} v_j^0$

$v_j^0 \xrightarrow{\neg_j} v_j^1, \quad v_j^1 \xrightarrow{\neg_j} v_j^0 \quad \text{and} \quad v_j^\iota \xrightarrow{\neg_l} v_j^\iota \quad \text{for} \quad l \neq j$

$v_j^\iota \xrightarrow{?} w_j^\iota$

$w_j^0 \xrightarrow{\neg_j} w_j^1, \quad w_j^1 \xrightarrow{\neg_j} \text{waste} \quad \text{and} \quad w_j^\iota \xrightarrow{\neg_l} w_j^\iota \quad \text{for} \quad l \neq j$

$w_j^0 \xrightarrow{!} \text{waste} \quad \text{and} \quad w_j^1 \xrightarrow{!} \text{haven}$

$\text{waste} \xrightarrow{\circlearrowleft} \text{waste} \quad \text{and} \quad \text{haven} \xrightarrow{\circlearrowleft} \text{haven}$

Finally, the indistinguishabilities of player 1 are:

$\downarrow_1 \sim^A \downarrow_2 \sim^A \ \dots \ \sim^A \downarrow_m$

$v_j^\iota \sim^V v_l^\kappa$ for all $j, l \in \{1, \dots, m\}$ and all $\iota, \kappa \in \{0, 1\}$

$w_j^\iota \sim^V w_l^\kappa$ for all $j, l \in \{1, \dots, m\}$ and all $\iota, \kappa \in \{0, 1\}$

Moreover, we define the safety condition $S_m = V_m \setminus \{\text{waste}\}$ on $\mathcal{G}_m$. The game graph $\mathcal{G}_2$ is depicted in Figure 3.2. Diamond positions belong to player 0 while round positions belong to player 1. Dotted lines indicate the indistinguishabilities of player 1. The selfloops at waste and haven are omitted.

So after the first move of player 0 from v_0, player 1 will consider exactly the set $\{v_1^0, \dots, v_m^0\}$ of positions possible which can be seen as the digital number $0 \dots 0$. Then, by applying $\neg_j$, player 0 can change the position v_j^0 in this set to v_j^1 and vice versa, that is, he can flip the j-th bit in the digital number. Therefore, the sets that player 0 can generate correspond to all digital numbers with m bits. Then, having generated some digital number $\{v_1^{\iota_1}, \dots, v_m^{\iota_m}\}$, player 0 can apply the action ? which gives control to player 1. Now player 1 can flip the j-th bit from 0 to 1 by using also the action $\neg_j$, but by trying to flip a bit that already has the value 1, he will lose. Additionally, player 1 will also lose, if he plays the action ! when being in some position w_j^0 that corresponds to a bit with value 0. Hence, to apply the action ! safely and reach the position haven, player 1 must first generate the digital number $1 \dots 1$. But in order to be sure that he does not try to flip a bit that already has the value 1, he must exactly know the digital number that player 0 has created. Since there are 2^m such digital number, any finite memory strategy for player 1 will have at least 2^m memory states.

[26]For parity game graphs with not necessarily observable coloring, the construction above shows that exponential memory is sufficient as well, but $2^{|V|}$ states will not be enough in general.

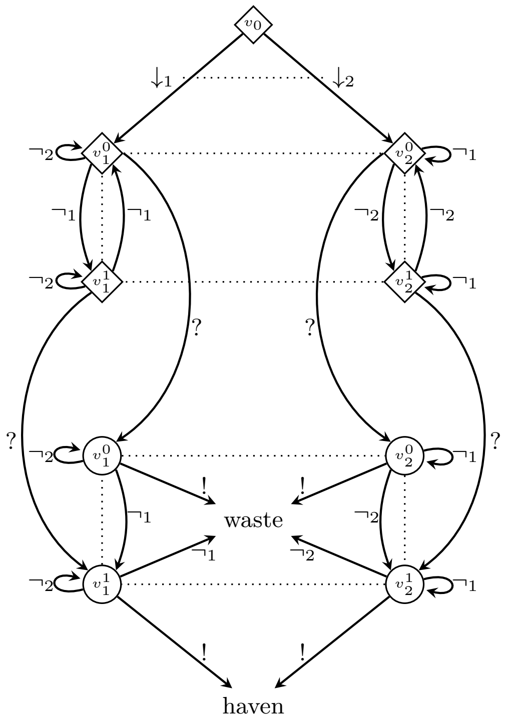


Figure 3.2: The game graph $\mathcal{G}_2$. Player 1 needs at least 4 memory states.

Proposition 3.27. *Player* 1 *has a memory winning strategy for* $(\mathcal{G}_m, S_m, v_0)$ *that uses at most* 2^m *memory states but he does not have one that uses less than* 2^m *memory states.*

Proof. It is easy to see that player 1 has a memory winning strategy for $(\mathcal{G}_m, S_m, v_0)$ that uses at most 2^m memory states: If player 0 never plays the action ? then the game will stay in the safe positions v_j^ι forever, so player 1 wins the game. If player 0 plays action ? after a finite number of steps, then he has generated a digital number $\iota_1 \dots \iota_m$ $(V(\pi) = \{w_1^{\iota_1}, \dots, w_m^{\iota_m}\})$ and player 1 chooses actions $\neg_j$ for exactly those bits j which are 0 $(w_j^0 \in V(\pi))$. Thereafter he knows that the game is in position w_j^1 for some $j \in \{1, \dots, m\}$, so he can safely choose action ! and reach position haven.

Now let $\mathcal{M} = (M, \delta_{\text{in}}, \delta_{\text{up}})$ be a memory structure for player 1 with $|M| < 2^m$ and assume there is a winning strategy $\sigma : M \times [V_1] \to A_1$ for player 1 with memory $\mathcal{M}$. It is easy to see that for all tuples $(\iota_1, \dots, \iota_m) \in \{0,1\}^m$ there is some σ-history π such that $V(\pi) = \{w_1^{\iota_1}, \dots, w_m^{\iota_m}\}$. (Notice that any history π in which ? does not occur is consistent with σ as all positions in π belong to player 0.) Since there are 2^m such tuples but less than 2^m memory states in M, there are σ-histories π, ρ in $\mathcal{G}_n$ with $V(\pi) = \{w_1^{\iota_1}, \dots, w_m^{\iota_m}\}$ and $V(\rho) = \{w_1^{\kappa_1}, \dots, w_m^{\kappa_m}\}$ for some tuples $\bar{\iota} = (\iota_1, \dots, \iota_m) \neq (\kappa_1, \dots, \kappa_m) = \bar{\kappa} \in \{0,1\}^m$ such that $\delta^*(\pi) = \delta^*(\rho)$.

Consider the (unique) extensions $\pi a_1 v_1 a_2 v_2 \dots$ and $\rho b_1 w_1 b_2 w_2 \dots$ of π and ρ to σ-plays. For $l \in \mathbb{N}$, define the history $\pi^l = \pi a_1 v_1 \dots a_l v_l$ and for any l with $l = 0$ or $v_l \notin \{\text{waste}, \text{haven}\}$, let $Z(\pi^l) = \{j \in \{1, \dots, m\} \mid w_j^0 \in V(\pi^l)\}$ be the bits in the digital number interpretation of $V(\pi^l)$ which are 0. The histories ρ^l and sets $Z(\rho^l)$ are defined analogously. Notice that $\bar{\iota} \neq \bar{\kappa}$ implies $Z(\pi) \neq Z(\rho)$. Since all positions w_j^ι are indistinguishable for player 1 it is easy to see that $a_1 = b_1$ and, for all $l \geq 1$ with $v_l, w_l \notin \{\text{waste}, \text{haven}\}$, we have $\delta^*(\pi^l) = \delta^*(\rho^l)$ and $a_{l+1} = b_{l+1}$.

First we show that there is some $l \in \mathbb{N}$ with $a_{l+1} = !$. Otherwise, $a_1 = \neg_j$ for some j and for all $l \geq 1$ with $v_l \notin \{\text{waste}, \text{haven}\}$ we also have $a_{l+1} = \neg_j$ for some j. Moreover, if $j \in Z(\pi^l)$, then $Z(\pi^{l+1}) = Z(\pi^l) \setminus \{j\}$ and $v_{l+1} \notin \{\text{waste}, \text{haven}\}$, so there is some $l \in \mathbb{N}$ with

$v_l \notin \{\text{waste}, \text{haven}\}$ such that $j \notin Z(\pi^l)$. This means $w_j^0 \notin V(\pi^l)$ and so $w_j^1 \in V(\pi^l)$. But then σ cannot be winning: $w_j^1 \in V(\pi^l)$ means that there is some history $\vartheta \sim^* \pi^l$ with $\text{last}(\vartheta) = w_j^1$. Since π^l is consistent with σ and $\vartheta \sim^* \pi^l$, ϑ is consistent with σ as well. (Notice that player 1 can distinguish any two of his actions.) After ϑ has been played, player 1 chooses the action $\sigma(\delta^*(\vartheta), [\text{last}(\vartheta)]) = \sigma(\delta^*(\pi^l), [\text{last}(\pi^l)]) = a_{l+1} = \neg_j$, so the unique extension of ϑ to a σ-play is *lost* by player 1 since $\delta(w_j^1, \neg_j) = \text{waste}$. This contradicts our assumption that σ is a winning strategy, so there is some $l \in \mathbb{N}$ with $a_{l+1} = !$. Analogously, there is some $l' \in \mathbb{N}$ with $b_{l'+1} = !$.

Now consider the (unique) $l, l' \in \mathbb{N}$ such that $a_{l+1} = !$ and $b_{l'+1} = !$. W.l.o.g., assume $l \leq l'$. Then $v_l, w_l \notin \{\text{waste}, \text{haven}\}$ as the action ! is not available in either one of these positions. As we have noticed, this yields $a_k = b_k$ for $k = 1, \ldots, l+1$. Therefore, $a_{l+1} = b_{l+1} = !$. Moreover, since $Z(\pi) \neq Z(\rho)$ and $a_k = b_k \in \{\neg_1, \ldots, \neg_m\}$ for $k = 1, \ldots, l$, we also have $Z(\pi^l) \neq Z(\rho^l)$, so at least one of these sets is not empty. If $Z(\pi^l) \neq \emptyset$, then σ cannot be winning: $Z(\pi^l) \neq \emptyset$ yields a history $\vartheta \sim^* \pi^l$ such that $\text{last}(\vartheta) = w_j^0$ for some j. As before, ϑ is consistent with σ. After history ϑ, player 1 chooses action $\sigma(\delta^*(\vartheta), [\text{last}(\vartheta)]) = \sigma(\delta^*(\pi^l), [\text{last}(\pi^l)]) = a_{l+1} = !$, so the unique extension of ϑ to a σ-play is *lost* by player 1 because $\delta(w_j^0, !) = \text{waste}$. If $Z(\rho^l) \neq \emptyset$, then completely analogously, σ cannot be winning as well. This contradicts our assumption that σ is a winning strategy, so there cannot be a memory winning strategy for player 1 for $(\mathcal{G}_m, S_m, v_0)$ that uses less than 2^m memory states. □

Notice that the number of positions and actions in the game graph $\mathcal{G}_m$ are linear in m. Moreover, $\mathcal{G}_m$ also contains linearly many edges, but the $3m$ selfloops in the graph are each labeled with exactly $m-1$ actions. Therefore, the lower bound is subexponential in the total size $|\mathcal{G}|$ of the game graph: the number of memory states needed to implement a winning strategy for player 1 is at least $2^{\sqrt{|\mathcal{G}|/c}}$ for some c. However, it is a clear and tight lower bound of $2^{|V|/5}$, for game graphs with linearly many actions and (unlabeled) edges. In this sense, the upper bound of $2^{|V|}$ is essentially optimal for safety games.

On the other hand, the construction cannot be directly applied to reachability conditions: The reachability condition $R_m = \{\text{haven}\}$ is *not* equivalent to the safety condition S_m because player 0 can stay in the positions v_j^ι forever. To use reachability conditions we would have to change the game graphs so that player 0 can make at most m moves before giving control to player 1 (which is sufficient to generate any digital number with m bits from $0 \ldots 0$). But this causes a blow up of the set of positions, yielding a $2^{\sqrt{|V|/c}}$ lower bound.

In [172] a construction has been presented that also yields a $2^{\sqrt{|V|/c}}$ lower bound for reachability games but uses exponentially many actions. This can be avoided by using the fact that any permutation is a product of certain transpositions which, however, causes a blow up of the set of positions, yielding a $2^{\sqrt[3]{|V|}}$ lower bound. A tight inspection of the construction may give a $2^{\sqrt{|V|/c}}$ lower bound for the case of linearly many actions, and a $2^{|V|/c}$ lower bound for the case of exponentially many actions. Whether the tight lower bound for safety games can be established for reachability games is not quite clear.

It is also interesting to note that the game graphs in [172] are different in that player 1 does not consider exactly m positions possible in each step (until a special position is reached) and has to turn this set into one which allows for a safe execution of some winning action. Instead, player 1 removes in each step (at most) one element from the set of positions that he considers possible and he can only safely execute a winning action in a situation where this set is a singleton. Moreover, the interaction between player 0 and player 1 is also different: While in the games on $\mathcal{G}_m$, player 0 makes an arbitrary number of moves (m are sufficient) but then gives control to player 1 for the rest of the game, in the games used in [172], the players alternate at least m times before player 1 gets a chance to win.

Another lower bound for reachability games has been obtained in [22]. The exact bound has not been calculated, but it is clearly superpolynomial.[27] A distinct feature of the game

[27]The number of positions of the game graphs $\mathcal{H}_m$ from [22] is the sum of the first m prime numbers, while

graphs presented in [22] is that player 0 only makes a single move right at the beginning and that all the actions of player 0 are indistinguishable for player 1. We could also modify the game graphs $\mathcal{G}_m$ so that all actions of player 0 are indistinguishable for player 1 by encoding the actions of player 0 into the positions of the graph, cf. Section 2.1.2. This, however, would induce quadratically many positions, so the lower bound would be $2^{\sqrt{|V|}/c}$ as well. For game graphs that satisfy this special condition, it is also not quite clear whether a $2^{|V|/c}$ lower bound holds, even for safety conditions.

3.5.3 Antichains

Throughout this section, we consider only *observable* parity conditions. We give merely some intuition and main ideas of the antichains approach. For details we refer to [66, 52, 22].

In [66], the following approach to solve reachability and safety games with imperfect information on finite graphs $\mathcal{G} = (V, \delta, (\sim^V), (\sim^A)$ has been suggested:[28] Instead of constructing the whole powerset graph $2^{\mathcal{G}}$ in advance, one performs the usual fixed point computation on $2^{\mathcal{G}}$, that iterates the controllable predecessor operator $\mathrm{CPre} : 2^{(2^V)} \to 2^{(2^V)}$ with

$$\mathrm{CPre}(\overline{X}) = \{\overline{v} \in 2^{V_0} \mid \mathrm{Post}(\overline{v}) \subseteq \overline{X}\} \cup \{\overline{v} \in 2^{V_1} \mid \mathrm{Post}(\overline{v}) \cap \overline{X} \neq \emptyset\}.$$

So $\mathrm{CPre}(\overline{X})$ contains exactly those positions in $2^{\mathcal{G}}$ from which player 1 can force the game into $\overline{X}$ in a single step. Clearly, CPre is a monotone operator. So, for a reachability winning condition given by a set $R \subseteq V$ of winning positions, to compute the winning region of player 1 in $2^{\mathcal{G}}$ we can start with $2^R \subseteq 2^V$ and then apply the controllable predecessor operator until we reach a fixed point.

This computation obviously does not require access to the whole game graph in advance: We start with 2^R and we compute only as much of the graph structure of $2^{\mathcal{G}}$ as is needed in each iteration of CPre. Although this avoids the computation of parts of the game graph that are irrelevant for determining the winning region of player 1, the representation of those parts that are relevant is still extensive. The important observation is now, that each set $\overline{X}$ of positions that occurs during the fixed point computation on $2^{\mathcal{G}}$ is *downward closed* with respect to inclusion, that is, if $\overline{v} \in \overline{X}$ then $\overline{u} \in \overline{X}$ for all $\overline{u} \subseteq \overline{v}$. (For $\overline{X} = 2^R$ this is obvious and if $\overline{X}$ is downward closed, then $\mathrm{CPre}(\overline{X})$ can be easily shown to be downward closed as well.)

Therefore, the set $\overline{X}$ is uniquely determined by the set of maximal elements with respect to $\subseteq$, that is, we can represent $\overline{X}$ by the set

$$\lceil \overline{X} \rceil = \{\overline{v} \in \overline{X} \mid \text{ there is no } \overline{u} \in \overline{X} \text{ such that } \overline{v} \subsetneq \overline{u}\}.$$

Now such a set $\lceil \overline{X} \rceil$ forms an *antichain* in the lattice $(2^{(2^V)}, \subseteq)$ of subsets. The basic idea for the antichains method is now to compute the fixed point of CPre on the lattice of antichains of $(2^{(2^V)}, \subseteq)$ instead of $(2^{(2^V)}, \subseteq)$ directly. In this way, the sets that occur during the computation are potentially smaller than the sets during the computation using the extensive representation of sets of positions. It can be shown that there are classes of safety games with imperfect information such that performing the powerset construction explicitly requires exponential time while the antichains method runs in polynomial time [66].

These classes, though, are rather particular and there seem to be no comprehensive characteristics known that would guarantee that the antichains methods needs only polynomial time to solve the strategy problem for the corresponding class of games with imperfect information. However, in actual benchmarks, the antichains method usually shows a quite advanced performance, so for problems whose solution forms a part of certain toolboxes like standard automata construction, the antichains method may be a good alternative, if applicable.

the number of states needed to implement a winning strategy for player 1 is the product of these numbers.

[28]In fact, in [66], the approach has been described only for safety conditions but the adaption the reachability conditions is straightforward.

In fact, in [65], this approach has been used to develop an improved method for solving the universality problem for nondeterministic automata $\mathcal{A}$ on finite words. This problem can be easily reduced to the strategy problem for two-player *reachability* games with imperfect information: Player 1 tries to find a word $u \in \Sigma^*$ that is *not* in the language of $\mathcal{A}$, while player 0 tries to find an accepting run of $\mathcal{A}$ on the word that player 1 constructs.

In each step, player 1 chooses a letter a from Σ and player 0 chooses an a-transition from the current state. Player 1 also has a special action $\perp$ that he can choose to indicate the end of the word. This action leads from accepting states of $\mathcal{A}$ to a state v_{fail} which is losing for player 1 and it leads from rejecting states of $\mathcal{A}$ to a state v_{goal} which is winning for player 0. So $R = \{v_{\text{goal}}\}$. The states of the automaton are invisible for player 1, that means, all states form a single $\sim^V$-equivalence class. In this way, player 1 observes only the number of steps (and his own actions, which is irrelevant as we know), so strategies for player 1 are functions $\sigma : \mathbb{N} \to \Sigma \cup \{\perp\}$ which correspond to finite words u. It is not hard to see that a strategy is in fact winning if, and only if, $u \notin L(\mathcal{A})$. Experimental results on this solution of the universality problem show that it often outperforms the standard powerset construction.

In [52], the solution has been extended to parity winning conditions. For this, one uses the fact that the winning region of player 1 in a parity game with *full* information can be defined in the modal μ-calculus [75]. More precisely, given a parity game graph $(\mathcal{G}, \text{col})$ with *full information*, we can construct a formula $\varphi \in L_\mu(V_0, V_1, P_1, \dots, P_k)$ where $\text{col}(V) \subseteq [k]$ such that for all $v \in V$ we have $\mathcal{G}, v \models \varphi$ if, and only if, player 1 has a winning strategy for $(\mathcal{G}, \text{col}, v)$.

Essentially, evaluating the formula φ on $\mathcal{G}$ is also done by iteratively computing fixed points of CPre, but now, we have to compute several nested greatest and smallest fixed points. To see that this computation can be performed on the lattice of antichains as well, one has to demonstrate that intersection, union and taking fixed points of monotone operators preserve downward closedness of the sets $\overline{X} \subseteq 2^V$ too. (Notice that complement does not preserve downward closedness, but for the formula φ, negation is not needed.)

These results provide the equipment to apply the antichains method to the strategy problem for parity games. On the other hand, since the antichains algorithm does not construct the powerset graph $2^{\mathcal{G}}$ explicitly, it is not directly clear how to obtain an implementation of a strategy for player 1 from a positive answer of the algorithm. Of course it could be possible that whenever the antichains method generates only sets of polynomial size, then we can also find a succinct representation of a winning strategy that needs only polynomially many states as well. However, in [22] it has been shown that there is a class of game graphs $\mathcal{G}_n$ with partial information and reachability winning condition such that each game graph has size polynomial in n and, while the fixed point computed by the antichains algorithm also has size polynomial in n, any winning strategy for player 1 for $\mathcal{G}_n$ has size at least exponential in n, cf. Section 3.5.2.

Intuitively, this demonstrates that the information generated by the antichains algorithm during a computation is not sufficient in general, to extract winning strategies. Hence, there can be no simple patch of the algorithm that upgrades it to a strategy constructing version but still retains its performance as it is. Nevertheless, in [22], an antichains method has been presented that is capable of constructing finite state implementations of winning strategies, which is a serious extension of the antichains approach as described before. In [23], a prototype implementation of this algorithm has been presented.

Summing up, the antichains method in its general form, is an optimized implementation of the powerset construction that avoids an explicit construction of the whole game graph in advance and represents the sets of positions more succinctly. Experimental results show a quite improved performance compared to more direct implementations. A shortcoming of the method is that so far there are no sufficient conditions known that would guarantee that the method runs in polynomial time on certain (preferably large) classes of games. One of the core issues about the method for both, adequate experimental performance and theoretical analysis, is the implementation of the controllable predecessor operator on the lattice of antichains. This operator forms an integral part of the method and influences its performance crucially [22].

Some remarks on this can also be found in [172].

3.5.4 Asynchronous Observability

We conclude the presentation of the powerset construction and related concepts with a little side trip to games with imperfect information on finite game graphs, where the information of the players is given in terms of *asynchronous observability.* As we have mentioned, there are several results on certain cases of asynchronous observability, cf. Section 2.1.4.

Here we demonstrate how the powerset construction can be adapted to the asynchronous case defined by the equivalence relation $\overleftarrow{\sim}^*$ as introduced in Section 2.1.4. Remind that $\pi \overleftarrow{\sim}^* \rho$ if $\overleftarrow{\pi} \sim^* \overleftarrow{\rho}$ where $\overleftarrow{\pi}$ is obtained from π by contracting each maximal sequence $v_j a_{j+1} v_{j+1} \dots a_k v_k$ of private moves of player 0 in π to v_j. That is, π and ρ are indistinguishable for player 1 in the asynchronous case just as in the synchronous case, if we additionally *hide* all private moves of player 0 completely from player 1. Some basic mechanisms that we use for this adaption of the powerset construction are similar to those presented in [221, 194, 47].

As we have mentioned, the method of knowledge tracking, that is, constructing an explicit representation of the possible states of knowledge of player 1 during plays of the original game and the dynamics of this knowledge, is of independent interest, beyond the mere task of strategy synthesis, see also Chapter 6. In the following, let $\mathcal{G} = (V, \delta, \sim^V, \sim^A)$ be a game graph with partial information.

To see how the powerset construction should be defined for the case of games with imperfect information defined by asynchronous observability, reconsider the set

$$\mathrm{Tr}(\pi) = \{\rho \sim^* \pi \mid \rho \in v_0(\mathrm{AV})^* \text{ is a history in } \mathcal{G}\}.$$

Analogously, we can define the set

$$\overleftarrow{\mathrm{Tr}}(\pi) = \{\rho \overleftarrow{\sim}^* \pi \mid \rho \in v_0(\mathrm{AV})^* \text{ is a history in } \mathcal{G}\}.$$

The core-operator is then defined just as before, that is, $\mathrm{core}(\overleftarrow{\mathrm{Tr}}(\pi)) = \{\mathrm{last}(\rho) \mid \rho \in \overleftarrow{\mathrm{Tr}}(\pi)\}$. We abbreviate $\mathrm{core}(\overleftarrow{\mathrm{Tr}}(\pi))$ by $\overleftarrow{V}(\pi)$.

Notice that for any history π in $\mathcal{G}$, we have $\overleftarrow{\mathrm{Tr}}(\pi) = \overleftarrow{\mathrm{Tr}}(\overleftarrow{\pi})$. Clearly, $\overleftarrow{\pi}$ is not a history in $\mathcal{G}$ in general, but in this context we treat $\overleftarrow{\pi}$ as though it was a history in $\mathcal{G}$. So $\overleftarrow{\mathrm{Tr}}(\overleftarrow{\pi}) = \{\rho \overleftarrow{\sim}^* \overleftarrow{\pi} \mid \dots\} = \{\rho \overleftarrow{\sim}^* \pi \mid \dots\} = \overleftarrow{\mathrm{Tr}}(\pi)$. Therefore, $\overleftarrow{V}(\pi) = \overleftarrow{V}(\overleftarrow{\pi})$ for all π.

To compute the sets $\overleftarrow{V}(\pi)$ iteratively as we have done for the sets $V(\pi)$ we have to take into account the private moves of player 0. For this we define, for a set $U \subseteq V$ of positions, the set $\mathrm{Reach}_p(U) \subseteq U$ of all positions that can be reached from some position in U by a (possibly empty) sequence of private moves of player 0. Notice that if $U \subseteq V_1$ then $\mathrm{Reach}_p(U) = U$. It is easy to see that $\overleftarrow{V}(v_0) = \mathrm{Reach}_p(\{v_0\})$.

Moreover, for a set $U \subseteq V$, an action a and a position $w \in \mathrm{Post}_a(U)$, we define $U[aw]$ to be U, if $U \subseteq V_0$ and $w \sim^V v$ for some $v \in U$ and we define $U[aw]$ to be $\mathrm{Reach}_p(\mathrm{Post}_{[a]}(U) \cap [w])$, otherwise. These definitions reflect the fact that player 1 does not notice the private moves of player 0 but as soon as a non-private move is performed, player 1 has to take all subsequently possible private moves of player 0 into account at once. Having this in mind, it is easy to prove that for any history $\pi a v \in v_0(\mathrm{AV})^+$ in $\mathcal{G}$ the following holds.

Proposition 3.28. $\overleftarrow{V}(\pi a v) = \overleftarrow{V}(\pi)[av]$.

Now, the powerset graph for $\mathcal{G}$, which we denote $\tilde{G} = (\tilde{V}, \tilde{\Delta})$ in the asynchronous case, is defined accordingly: Clearly, $\tilde{V} = 2^V$ and $\tilde{V}_0 = 2^{V_0}$. Moreover,

$$(\tilde{v}, a, \tilde{w}) \in \tilde{\Delta} \;:\Longleftrightarrow\; \tilde{w} = \tilde{v}[aw] \text{ for some } w \in \tilde{w}.$$

Again, an important property for the correctness of this construction is that each history $\tilde{\pi} = \tilde{w}_0 b_1 \tilde{w}_1 \dots b_l \tilde{w}_l$ in $\tilde{\mathcal{G}}$ induces at least one history in $\mathcal{G}$. In the asynchronous case, the set $\overleftarrow{\Pi}(\tilde{\pi})$ of histories induced by $\tilde{\pi}$, can be described as:

$$\text{A history } \pi = v_0 a_1 v_1 \dots a_r v_r \text{ is in } \overleftarrow{\Pi}(\tilde{\pi}) \text{ if } \overleftarrow{\pi} \in \Pi(\tilde{\pi}).$$

In the asynchronous case, however, $\overleftarrow{\Pi}(\tilde{\pi}) \neq \emptyset$ holds in general only if there are no selfloops of player 0 in $\tilde{\pi}$: Selfloops of player 0 in $\tilde{\pi}$ correspond to private moves of player 0 in $\mathcal{G}$, but in $\overleftarrow{\pi}$, all such moves are erased. In fact, sequences of selfloops that player 0 plays during a play $\tilde{\pi}$ may make it impossible to construct a corresponding play in $\mathcal{G}$ directly, since there may be no appropriate sequences of private moves of player 0 that can be plugged in at these points of the play.[29] However, the definition of the winning condition will guarantee that we can restrict to plays (and therefore histories) in which player 0 does not play any such sequences of selfloops.

Proposition 3.29. *Assume that $\tilde{w}_j \neq \tilde{w}_{j+1}$ if $\tilde{w}_j \in \tilde{V}_0$ and let $v_r \in \tilde{w}_l$.*

(1) There is a history $\pi \in \overleftarrow{\Pi}(\tilde{\pi})$ such that $\text{last}(\pi) = v_r$.

(2) For each history $\pi \in \overleftarrow{\Pi}(\tilde{\pi})$ we have $\overleftarrow{V}(\overleftarrow{\pi}(\leq j)) = \tilde{w}_j$.

To see how to define an appropriate winning condition $\tilde{W}$ for $\tilde{\mathcal{G}}$ from a winning condition $W \subseteq \Pi$ for $\mathcal{G}$, reconsider the definition of the winning condition $\overline{W}$ for $\overline{\mathcal{G}}$. There, for a play $\overline{\pi}$ to be won by player 1 we have required that any play $\pi \in \Pi(\overline{\pi})$ is won by player 1. Here we use exactly the same definition but of course we have to adapt the definition of $\Pi(\overline{\pi})$, taking into account the private moves of player 0 just as for finite histories.

For this, let $\overleftarrow{\pi}$ be obtained from a play π in $\mathcal{G}$ by contracting each *maximal finite* sequence $v_j a_{j+1} v_{j+1} \dots a_k v_k$ of private moves of player 0 in π to v_j. That means, all finite sequences of private moves are erased but if π ends with some infinite sequence of private moves, then $\overleftarrow{\pi}$ ends with the same infinite sequence of private moves. Given a play $\tilde{\pi} = \tilde{w}_0 b_1 \tilde{w}_1 b_2 \tilde{w}_2 \dots$ in $\tilde{G}$ the set $\overleftarrow{\Pi}(\tilde{\pi})$ is defined accordingly and, using this set, the winning condition $\tilde{W}$ is defined as before:

$$\text{A play } \pi \text{ in } \mathcal{G} \text{ is in } \overleftarrow{\Pi}(\tilde{\pi}) \text{ if } \overleftarrow{\pi} \in \Pi(\tilde{\pi})$$
$$\tilde{\pi} \in \tilde{W} \ :\Longleftrightarrow\ \overleftarrow{\Pi}(\tilde{\pi}) \subseteq W$$

Notice that the $\Leftarrow$-direction of the definition of $\tilde{W}$ guarantees that any play in which player 0 plays a selfloop that is not part of an infinite sequence of selfloops, is *lost* by player 0. On the other hand, player 0 does not need to play any such selfloops because the $\Rightarrow$-direction of the definition of $\tilde{W}$ ensures that for any position $\tilde{w}_j \in \tilde{V}_0$ in $\tilde{\pi}$, *all* possible finite sequences of private moves of player 0 in $\mathcal{G}$ that travel through $\tilde{w}_j$ are taken into account. Hence, player 0 needs to play selfloops only if he wants to play an *infinite* sequence of selfloops that corresponds to an infinite sequence of private moves in $\mathcal{G}$. So we can restrict to plays in which player 0 does not play any useless selfloops and apply Proposition 3.29 to any history that is relevant.

The proof that with this definition of $\tilde{W}$, winning strategies of player 1 are actually preserved, is pretty much the same as in the synchronous case. The finite sequences of selfloops of player 0 have already been taken care of, so essentially the only difference is that we have to pay some attention to the infinite sequences of selfloops, which correspond to infinite sequences of private moves.

Proposition 3.30. *Player 1 has a winning $\leftarrowtail^*$-strategy for $(\mathcal{G}, W, v_0)$ if, and only if, he has a winning strategy for $(\tilde{\mathcal{G}}, \tilde{W}, \overleftarrow{V}(v_0))$.*

[29] Notice that this is true also for positions $\tilde{w} \in \tilde{V}_0$ that actually do induce loops in $\mathcal{G}$, if player 0 proceeds to some position $\tilde{w}' \neq \tilde{w}$ later on in the play. These selfloops in $\tilde{\mathcal{G}}$, however, cannot be simply removed because player 0 needs them to be able to simulate infinite sequences of private moves in $\mathcal{G}$.

Omega-Regular Winning Conditions. Now we want to show that just as in the synchronous case, ω-regular winning conditions are effectively preserved by the construction. So assume that W is ω-regular. Here we do not, however, construct a universal parity automaton defining $\tilde{W}$ from a deterministic parity automaton defining W but we go through the complement $\Pi \setminus W$ and use nondeterministic Büchi automata which makes it somehow easier to deal with the sequences of private moves of player 0. So assume that $\Pi \setminus W$ is defined by a nondeterministic Büchi automaton $\mathcal{C} = (\Sigma, Q, q_{\text{in}}, \Delta^{\mathcal{C}}, F)$. (Notice that from a deterministic parity automaton defining W we can easily obtain such an automaton $\mathcal{C}$ of size polynomial in the size of the original automaton by complementation and simulation.)

The main idea for the automaton $\tilde{\mathcal{C}} = (\tilde{\Sigma}, \tilde{Q}, \tilde{q}_{\text{in}}, \Delta^{\tilde{\mathcal{C}}}, \tilde{F})$ defining $\tilde{\Pi} \setminus \tilde{W}$ is now the dual as in the synchronous case: While reading a play $\tilde{\pi}$, $\tilde{\mathcal{C}}$ guesses a play $\pi \in \overleftarrow{\Pi}(\tilde{\pi})$ and at the same time simulates $\mathcal{C}$ on this play. From $\tilde{\mathcal{C}}$ we can then construct a deterministic parity automaton defining $\tilde{W}$ in the usual way. We restrict to plays that do not end with an infinite sequence of selfloops of player 0. We comment on the general case after we have seen how the automaton works but we do not want to incorporate this into the definition.

Formally, we define $\tilde{\mathcal{C}}$ as follows. Clearly, $\tilde{\Sigma} = \tilde{V} \cup A\tilde{V}$ and, moreover, $\tilde{Q} = \{\tilde{q}_{\text{in}}\} \cup Q \times V \times \{0,1\}$ and $\tilde{F} = \{(q, v, \iota) \mid q \in F\} \cup \{(q, v, 1) \mid q \in Q\}$. Given $(p, v, \iota) \in Q \times V \times \{0,1\}$, the possible transitions from this state are:

$((p, v, \iota), b\tilde{w}, (q, w, \kappa)) \in \Delta^{\tilde{\mathcal{C}}} \quad :\Longleftrightarrow$
there is a history $v_1 a_2 v_2 \dots a_l v_l$ in $\mathcal{G}$ such that:

1. $l = 1$, if $\tilde{w} \in \tilde{V}_1$ and $v \not\sim^V v_1$ if $\tilde{w} \in \tilde{V}_0$
2. there is some $a \sim^A b$ with $\delta(v, a) = v_1$
3. $v_j \in \tilde{w}$ for all $1 \le j \le l$ and $v_l = w$
4. there are $q_1, \dots, q_{l-1} \in Q$ with
   - 4.1 $(p, av_1, q_1), (q_1, a_2 v_2, q_2), \dots, (q_{l-1}, a_l v_l, q) \in \Delta^{\mathcal{C}}$
   - 4.2 if $\kappa = 1$ then $q_j \in F$ for some $1 \le j \le l-1$.

The possible transitions $(\tilde{q}_{\text{in}}, \tilde{w}_0, (q, w, \kappa))$ from the initial state $\tilde{q}_{\text{in}}$ are defined accordingly but due to the longish formulation we do not write them down.

So $\tilde{\mathcal{C}}$ always stores the position v that is has chosen from the previous set $\tilde{w}$ which, in the first step, is some $v \in \text{Reach}_p(\{v_0\})$. Then, reading $b\tilde{w}$, the automaton guesses a finite sequence of private moves of player 0 starting from some $[b]$-successor of v (2), that travels through $\tilde{w}$ and ends in some position w (3). This sequence must be, of course, empty if $\tilde{w} \in \tilde{V}_1$ and it must be maximal within the play that $\tilde{\mathcal{C}}$ guesses if $\tilde{w} \in \tilde{V}_0$ (1).

At the same time, $\mathcal{C}$ is simulated on the play that is generated in this way by the requirement that for this finite sequence of private moves of player 0, there is a corresponding finite run of $\mathcal{C}$ that ties in with the previous run prefix ending in p and ends in q (4.1). Moreover, since we do not visit the states from this finite run explicitly, we use a flag $\kappa \in \{0,1\}$ to indicate, whether during this finite run some accepting state has been visited (4.2). If all these requirements are fulfilled, then we move on to state (q, w, κ).

Now it is not hard to see how to handle the general case: The automaton *guesses* whether the play will end in an infinite sequence of selfloops of player 0 and it guesses a position $s \in \mathbb{N}$ where this sequence starts. For all positions $\ge s$, the sequence $v_1 a_2 b_2 \dots a_l v_l$ in the definition of $\Delta^{\tilde{\mathcal{C}}}$ has to fulfill $l = 1$ in *each* step, that is, also if $\tilde{w} \in \tilde{V}_0$. The other conditions remain unchanged. Since $\tilde{\mathcal{C}}$ always memorizes a position v from the position $\tilde{w}$ that is has read in the last step, it can check for consecutive identical position by testing $v \in \tilde{w}$ for the position $\tilde{w}$ that it reads in the current step. If $\tilde{\mathcal{C}}$ encounters two different positions after position s, then it immediately rejects.

Finally, let $\tilde{\mathcal{B}}$ be a deterministic parity automaton such that $L(\tilde{\mathcal{B}})$ recognizes the complement of $L(\tilde{\mathcal{C}})$ (given the automaton $\tilde{\mathcal{C}}$ for the general case).

Proposition 3.31. $\tilde{W} = \{\tilde{\pi} \in \tilde{\Pi} \mid \tilde{\pi} \in L(\tilde{\mathcal{B}})\}$.

To see that this construction is in fact effective we have to show that 1. - 4. are decidable.

Proposition 3.32. *Given $t = ((p, v, \iota), b\tilde{w}, (q, w, \kappa))$, we can decide whether $t \in \Delta^{\tilde{\mathcal{C}}}$?*

Proof. We construct a nondeterministic finite automaton $\mathcal{B}_{\text{fin}}$ over $\mathrm{V} \cup \mathrm{AV}$ accepting precisely those words which are finite histories in $\mathcal{G}$ such that conditions 1. - 3. hold. Now let $\mathcal{C}_{\text{fin}}$ be the product automaton of $\mathcal{B}_{\text{fin}}$ and the nondeterministic finite automaton $\mathcal{A}_{\text{fin}}$, obtained from $\mathcal{B}$, by defining all states to be accepting and augmenting $\mathcal{B}$ by a state to remember whether some state in F has been seen if $\kappa = 1$. Then $\mathcal{C}_{\text{fin}}$ accepts precisely those words which are finite histories in $\mathcal{G}$ such that conditions 1. - 4. hold. So $t \in \Delta^{\tilde{\mathcal{C}}}$ if, and only if, $L(\mathcal{C}_{\text{fin}}) \neq \emptyset$. $\square$

Observable Winning Conditions. Finally we consider observable winning conditions. Of course, in the asynchronous case, the observations that player 1 makes during a play π have to be defined using $\overleftarrow{\pi}$, that is, we set $\overleftarrow{\mathrm{obs}}(\pi) := \mathrm{obs}(\overleftarrow{\pi})$. The winning condition W is called observable in the asynchronous case if for each two plays π and ρ in $\mathcal{G}$ with $\overleftarrow{\mathrm{obs}}(\pi) = \overleftarrow{\mathrm{obs}}(\rho)$ we have $\pi \in W$ if, and only if, $\rho \in W$. In other words, $W = \{\pi \in \Pi \mid \mathrm{obs}(\pi) \in \mathrm{obs}(W)\}$.

Notice that in a play π, player 1 will never make two consecutive identical observations $[v][v]$ with $v \in V_0$. So, to define whether a play $\tilde{\pi}$ is won by player 1 just by means of the observations of player 1, we have to hide selfloops of player 0 in $\tilde{\pi}$ as well: Let $\overleftarrow{\tilde{\pi}}$ be obtained from $\tilde{\pi}$ by contracting each maximal finite sequence $\tilde{v}a_1\tilde{v}\dots a_l\tilde{v}$ of selfloops of player 0 in $\tilde{\pi}$ to $\tilde{v}$. Again, if $\tilde{\pi}$ ends with an infinite sequence of selfloops then $\overleftarrow{\tilde{\pi}}$ ends with the same sequence of selfloops. Now let accordingly $\overleftarrow{\mathrm{obs}}(\tilde{\pi}) := \mathrm{obs}(\overleftarrow{\tilde{\pi}})$ and consider $\{\tilde{\pi} \in \tilde{\Pi} \mid \overleftarrow{\mathrm{obs}}(\tilde{\pi}) \in \overleftarrow{\mathrm{obs}}(W)\}$.

In general, this condition will not be equal to $\tilde{W}$ and, in fact, not adequate for $\tilde{\mathcal{G}}$. The reason is that player 0 can still play an infinite sequence of selfloops on a position $\tilde{w}$ that does not induce a cycle in $\mathcal{G}$. Such a play will be won by player 0, because there is no play in $\mathcal{G}$ that induces the same sequence of observations. Notice, however, that the only way how player 0 can take advantage of these selfloops is by abusing them in this way: A *finite* sequence of selfloops $\tilde{w}a_1\tilde{w}\dots a_l\tilde{w}$ does not influence the winner of the play. So we can deal with this problem by simply deleting all selfloops from any position $\tilde{w} \in \tilde{V}_0$ that does not induce a cycle in $\mathcal{G}$. (Which is a very simple transformation of the graph $\tilde{\mathcal{G}}$.) Call the resulting graph $\hat{\mathcal{G}}$ and consider $\{\hat{\pi} \in \hat{\Pi} \mid \overleftarrow{\mathrm{obs}}(\hat{\pi}) \in \overleftarrow{\mathrm{obs}}(W)\}$.

Again, in general there are plays $\hat{\pi}$ with $\hat{\pi} \in \tilde{W}$ but $\overleftarrow{\mathrm{obs}}(\hat{\pi}) \notin \overleftarrow{\mathrm{obs}}(W)$. The reason is that $\tilde{W}$ stipulates that any play where player 0 plays a selfloop is won by player 1 which is clearly not the case for the winning condition described above since the observations of player 1 in $\hat{\pi}$ are the same as in $\overleftarrow{\hat{\pi}}$. To see that the condition is adequate for $\tilde{\mathcal{G}}$ nevertheless, let us define $\hat{W} := \{\hat{\pi} \in \hat{\Pi} \mid \overleftarrow{\hat{\pi}} \in \tilde{W}\}$. (Notice that $\hat{W} = \{\hat{\pi} \in \hat{\Pi} \mid \overleftarrow{\Pi}(\overleftarrow{\hat{\pi}}) \subseteq W\}$.) It is easy to see that player 1 has a winning strategy for $(\tilde{\mathcal{G}}, \tilde{W}, \overleftarrow{V}(v_0))$ if, and only if, he has a winning strategy for $(\hat{\mathcal{G}}, \hat{W}, \overleftarrow{V}(v_0))$ and, moreover, if W is ω-regular then so is $\hat{W}$. But now,

$$\{\hat{\pi} \in \hat{\Pi} \mid \overleftarrow{\mathrm{obs}}(\hat{\pi}) \in \overleftarrow{\mathrm{obs}}(W)\} = \hat{W}.$$

For plays that do not end in an infinite sequence of selfloops, this can be shown just as in the synchronous case, using König's Lemma. Plays that do, have to be considered separately, as usual. That is, for the modified powerset construction yielding $(\hat{\mathcal{G}}, \hat{W}, \overleftarrow{V}(v_0))$, observable winning conditions can be easily treated as a special case. Finally, consider an observable coloring $\mathrm{col} : V \to [k]$ of $\mathcal{G}$ and a set $X \subseteq [k]^\omega$ defining $W = \{\pi \in \Pi \mid \mathrm{col}(\pi) \in X\}$. Then for W to be actually observable in the asynchronous case, in general, the set X has to be *stutter closed*[30], that is, invariant under extending and contracting finite sequences of identical colors.

[30]Clearly, W can be observable if X is not stutter closed because the possible private moves of player 0 are constrained by the game graph. Stutter closedness is, however, a convenient sufficient condition for observability.

In this case $\hat{W}$ coincides with the winning condition defined by $\hat{\mathrm{col}} : \hat{V} \to [k]$ with $\hat{\mathrm{col}}(\hat{v}) = \mathrm{col}(v)$ for some $v \in \hat{v}$. Notice that if X is defined by a parity condition, then X is stutter closed.

We have seen that, essentially, all techniques that we have presented for the synchronous case can also be applied to the asynchronous case, albeit with some greater technical effort. There are two small exceptions, tough, that we like to mention. First, the example from Section 3.5.2 showing the $2^{|V|/5}$ lower bound on the number of states does obviously not work in this way for the asynchronous case: All moves of player 0 in the gadget that allows him to create the digital number are private moves, so this whole gadget collapses to a single position when we apply the powerset construction. It is easy to adapt the example to the asynchronous case similar as it can be adapted to reachability conditions by taking m distinct copies of the gadget and letting player 0 move from copy to copy. But in this way, we obtain a $2^{\sqrt{|V|}/c}$ lower bound for the asynchronous case which can also be obtained from the examples in [22] and [172].

Second, in the synchronous case, from a given S1S formula defining a winning condition W for $\mathcal{G}$, we can easily construct an S1S-formula defining the winning condition $\overline{W}$ for $\overline{\mathcal{G}}$, see for example [172]. On the other hand, the sequences of private moves of player 0 make it much more intricate to construct such a formula in the asynchronous case. Clearly, the formula can be obtained from the automaton $\tilde{\mathcal{B}}$ but this gives a formula of length *nonelementary* in the length of the original formula. It is not clear, whether this blow up is avoidable.

Direct Reduction. We conclude the discussion of the asynchronous case with a short remark on a more direct solution for the strategy problem by reducing it to the synchronous case. This can be done by adding edges to the game graph $\mathcal{G}$ which allow player 0 to *skip* the sequences of private moves that he makes and instead go directly to any position that he could reach by such a sequence of private moves and one subsequent non-private move. In turn, we can let player 1 observe all moves of player 0, so we end up with a game with imperfect information and synchronous observability.

Clearly, a single move that player 0 makes in this game corresponds now in general to a set of sequences of moves in the original game, so we have to adapt the winning condition accordingly. Roughly speaking, a play π' in the new game is in W' if for all plays π in $\mathcal{G}$ with $\overleftarrow{\pi} = \pi'$ we have $\pi \in W$. Similarly as for $\tilde{W}$ it can be shown that W' is ω-regular if W is. Moreover, if W is observable then, essentially, we have $W' = \{\pi' \mid \overleftarrow{\pi'} = \overleftarrow{\pi} \text{ for some } \pi \in W\}$. In fact, if W is a parity condition with an observable coloring col, then W' is also a parity condition given by col.

This provides a reduction of the asynchronous strategy problem to the synchronous strategy problem which preserves ω-regular winning condition and observable parity conditions. In particular, for the case of observable parity conditions, the reduction can be performed in polynomial time. On the other hand, it should be emphasized that, while the construction preserves winning strategies for player 1, it does not preserve the dynamics of the knowledge of player 1 in the game.

Chapter 4

Parity Games on Simple Graphs

The question whether the strategy problem for parity games on finite game graphs with full information can be solved in polynomial time is one of the major open questions in the theory of verification and synthesis of finite state systems.

Parity games are the model checking games for the modal μ-calculus [76, 199], a fixed point extension of propositional modal logic [122], which subsumes many linear and branching time specification formalisms, in particular CTL* [61, 33], which in turn encompasses LTL, as well as the popular dynamic logic PDL, introduced in [82]. As we have seen in Section 3.1, parity games with full information can also be used to decide the emptiness problem for nondeterministic parity tree automata and constructing a regular tree for each given nonempty regular tree language. Moreover, parity conditions are capable of expressing *all* ω-regular winning conditions in such a way that the resulting games are always positionally determined, cf. Section 2.2. [1]

During the last 15 years, much research on finding a polynomial time algorithm for parity games with full information has been done, see for example [118, 157, 113], and [119, 189, 190] for recent (optimized) variants of algorithms presented in [142, 222, 118]. However, so far no polynomial time algorithm for solving the problem is known. In fact, very recently it has been shown that one of the most prominent candidates for polynomial time algorithm – the strategy improvement method of Vöge and Jurdziński – requires an exponential number of iterations in general [87]. This result also holds for several variants of the algorithm like that of Schewe [190].[2]

The essential algorithmic facts which are known about parity games with full information in general are formulated in Theorem 2.7: The strategy problem for parity games with full information is in NP$\cap$co-NP and can be solved in time $O(|V|^k)$ where k is the number of colors. Moreover, the algorithm presented in [119] yields a subexponential lower bound of $|V|^{O(\sqrt{|V|})}$.

On the other hand, it has been shown that for graphs of bounded complexity, parity games can often be solved in polynomial time. More precisely, on classes of graphs with bounded tree-width [156], DAG-width [24, 25] and entanglement [26], parity games with full information can be solved in polynomial time.[3] There are also several other classes of graphs for which this holds, like for example graphs of bounded Kelly-width [115] or clique width [159] and also certain classes of graphs that are not defined in terms of bounded graph complexity. However, we consider only classes of graphs defined by boundedness of certain graph complexity measures and we focus on tree-width, DAG-width and entanglement.

[1] In fact, parity conditions are the only possibility for this.

[2] It is interesting to note that this work of Friedmann [87] has been the starting point for a number of other lower bound constructions including subexponential lower bounds for several pivoting rules for simplex algorithms that are used to solve linear programs [88, 89].

[3] Nevertheless, although parity games *can* be solved in polynomial time on these classes of graphs, a close analysis of the counterexamples of Friedmann reveals that they have bounded complexity as well [46]. Hence, the strategy improvement algorithm does *not* work in polynomial time on graphs of bounded complexity.

The natural question arises, whether these result for parity games with full information can be transferred (at least partially) to games with imperfect information. In this chapter we conduct a detailed analysis of the influence of graph complexity on the computational complexity of the strategy problem for *parity* games with imperfect information on finite graphs with *two* players and *observable* coloring. We will see that in general, bounding the graph complexity does not decrease the complexity of this problem. However, if we additionally bound the *size of the equivalence classes of positions* in the game graphs, certain PTIME results can be obtained. For this, we build on the powerset construction from Section 3.5. The basic idea is to transform a parity game graph with partial information into one with full information using the powerset construction and prove that the transformation preserves boundedness of the graph complexity. Then the PTIME results on parity games with full information can be applied.

Notice that the powerset construction in the presented form works only for game graphs with two players. The more general construction from Chapter 6 may give a starting point to deal with the case of multiple players, but we do not pursue this here. Furthermore, as we have seen, for dealing with non-observable parity conditions, determinization of ω-automata is needed which goes beyond the usual powerset construction, and we have to use the product construction from Section 2.2.1 which is also delicate in the context of graph complexity. It may be possible to transfer some of our positive results to the non-observable case as well, but this requires a substantial investigation of the constructions. We do not consider this here either.

In Section 4.2 we start with the general case, where the size of the equivalence classes of positions may be unbounded. We first demonstrate that the approach using the powerset construction as described above is bound to fail in this case. More precisely (and even worse), we show that the powerset construction may yield graphs of *exponential* complexity (and hence exponential size) when applied to graphs of bounded complexity. Furthermore, the first main result of this chapter states that, indeed, any approach to exploit the simplicity of the graphs is bound to fail: The strategy problem for parity games with imperfect information is EXPTIME-hard on graphs of entanglement at most two and DAG-width at most three. Moreover, on graphs of entanglement zero and DAG-width one the problem is still PSPACE-hard.[4]

In Section 4.3, we consider game graphs with bounded partial information, that means, for a given class of game graphs, we have a fixed r such that each equivalence class of positions has size at most r. In this case, the powerset construction yields graphs which are only polynomially larger. Hence, parity games on graphs with bounded partial information and a fixed number of priorities can be solved in polynomial time for *arbitrarily complex* graphs.

However, for parity games with arbitrarily many priorities this is not clear (and equivalent to the problem for games with full information). What remains to prove in order to show that parity games on graphs with bounded partial information can be solved in polynomial time if the graph complexity is bounded is that applying the powerset construction to such game graphs preserves boundedness of the graph complexity. We will see that, while for tree-width and entanglement this is not the case, the powerset construction does preserve boundedness of directed path-width. Hence, the strategy problem for parity games on graphs with bounded partial information and bounded directed path-width can be solved in polynomial time.

On the other hand, boundedness of directed path-width is a very restrictive assumption: Intuitively this amounts to the assumption that the game graphs are only boundedly different from directed paths, which are very simple graphs. However, a close analysis of the reasons why the powerset construction does preserve boundedness of directed path-width reveals a possibility to transfer this result to DAG-width: The graph searching games that characterize directed path-width are also games of partial information in that the robber is invisible for the cops, see Section 4.1. As it turns out, this allows for a particularly easy translation of cops' strategies from the original to the powerset graph. In a sense, the partial information in the graph searching game captures the uncertainties of player 1 which are explicitly represented in the powerset

[4]In fact, these results hold even for reachability conditions.

graph. Now, to be able to translate strategies for the cop player from the original graph to the powerset graph, in the case of bounded partial information, it is not necessary to have a completely invisible robber. In fact, if r is the maximal size of the subsets in the powerset graph, a robber which may be on at most r possible vertices at each point, is sufficient.

To exploit these observations, we use a concept of graph searching in Section 4.4, where the cops do not try to capture a single directed robber on the graph but several robbers simultaneously. The main technical result is that to capture r directed robbers simultaneously on the graph, $k \cdot r$ cops are sufficient, if k cops are sufficient to capture a single directed robber.[5] This result allows us then to translate cops' strategies for the DAG-width game on the original game graph to the powerset game graph, using only boundedly many cops, with the multiple robbers game as a whistle-stop. Finally, this implies the second main result of this chapter: The strategy problem for parity games on graphs with bounded partial information and bounded DAG-width can be solved in polynomial time.

This chapter is based on [174, 175].

4.1 Graph Complexity

Graph complexity measures were originally defined for *undirected graphs.* The most prominent measure for undirected trees is that of *tree-width* which has first been considered in [103]. Later on, Robertson and Seymour have introduced the concept independently [184] in the course of their famous graph minor project, eventually resulting in the graph minor theorem (also known as Robertson-Seymour Theorem), which is one of the deepest results of graph theory: The minor relation is a well-quasi ordering on the class of all undirected finite graphs. Or, equivalently, each infinite class of undirected finite graphs contains two graphs of which one is a minor of the other. The entire graph minor project appeared as a series of papers tagged with *Graph minors* and consecutive numbers, starting with I. The final proof of the graph minor theorem appears in *Graph Minors. XX. Wagner's Conjecture* [185], a nice introductory course can be found in [67].

In [184], tree-width has been defined in the form as it is probably best known, using the notion of a *tree decompositions* of a graph. This concept is also of great importance for algorithmic graph theory, since decomposability can be used to develop dynamic programs for decision problems on graphs, see for example [7]. Tree-width has been particularly successful in this respect and is by now a central notion in algorithmic graph theory.

A tree decomposition anatomizes the given graph into (possibly quite complex) parts that are only sparsely related in a certain sense and which, altogether, form a tree. The general idea is now, roughly speaking, to solve the problem on the individual parts and combine the partial solution in a bottom up manner along the tree. The point here is, that on classes of graphs where all the individual parts have bounded size, that is to say, the graphs have *bounded tree-width*, then the solution of the problem on the individual parts also takes only a fixed amount of time. So if there is a way to combine the partial solutions efficiently, then the whole problem can be solved efficiently as well.

This approach has been shown to be applicable to a large number of decision problems on graphs which are NP-hard, including all MSO-definable graph properties [60]. For an overview we refer to [35]. On the other hand, in general, game graphs do not have a symmetric edge relation and obviously, the direction of the edges is of vital importance for the notion of (winning) strategies. Still, we can measure the complexity of a directed graph using tree-width as well, by simply taking the tree-width of the symmetric closure of the graph. It has been shown in [156], that the strategy problem for parity games with full information can be solved in polynomial time on classes of graphs with bounded tree-width.

[5] We like to anticipate that the central issue here is that of *monotonicity*, cf. Section 4.1.

However, this assumption is very strong: There are acyclic graphs of arbitrarily large tree-width, so it is expedient to look for complexity measures that take into account the whole structure of a directed graph − in particular, the direction of the edges. One of the earliest and most popular approaches is *directed tree-width*, defined in [116].[6] Although certain fixed parameter tractability results (in the sense described above) have been proved for classes of graphs with bounded directed tree-width [116], the measure has also certain undesirable properties [2] and is somehow difficult to handle, see also [157]. In particular, the game characterization of directed tree-width is somehow inconvenient, especially for our concerns. We will make some remarks on this later on. Whether the strategy problem for parity games with full information can be solved in polynomial time on graphs of bounded directed tree-width is still open.

During the last decade, much research has been done on the development of complexity measures for directed graphs that are more appropriate for applications in games and logic, especially parity games and the modal μ-calculus [21, 157]. The properties and special merits of the different measures are still subject to intensive research, see for example [114, 123, 91] and also [25, 27] for recent articles reviewing and sharpening much previous research. Although we focus here on complexity measures for directed graphs, especially DAG-width, due to its central significance in the structure theory of finite graphs we start our presentation of the different measures with tree-width. This will also serve as a preparation for the definition of DAG-width which is a particularly natural generalization of tree-width in terms of the game characterization that we shall mostly work with. We will also briefly consider tree-width later on where we demonstrate the high potential of the powerset construction to create graph structure if the direction of the edges is neglected. The corresponding result for tree-width will also prelude the main result from Section 4.4 on DAG-width. First, we review some very basic notions about graphs that we need to make quite precise in the context of graph complexity.

Graphs. We consider graphs $G = (V, E)$ where $E \subseteq V \times V$, so graphs are directed, possibly with selfloops but without multiedges. G is called *undirected* if E is symmetric. The symmetric closure of G is the graph $G^{\text{sym}} = (V, E^{\text{sym}})$ with $(u, v) \in E^{\text{sym}}$ if $(u, v) \in E$ or $(v, u) \in E$.

For $u, v \in V$, a path in G from u to v is a sequence $u = u_0, u_1, \dots, u_l = v$ such that $(u_j, u_{j+1}) \in E$ for $j \in [l]$. For a set $X \subseteq V$, the subgraph $(X, E \cap X \times X)$ of G is called the subgraph induced by the vertices X and is denoted $G \cap X$. The subgraph $G \cap (V \setminus X)$ induced by the vertices $V \setminus X$ is denoted $G - X$. Moreover, for a set $Y \subseteq V$, $\text{Reach}_G(Y)$ denotes the set of vertices $u \in V$ that are reachable in G from Y, that means, there is some $v \in Y$ such that there is a (possibly empty) path from v to u in G. (Hence, $Y \subseteq \text{Reach}_G(Y)$.)

G is called strongly connected if, for all $u, v \in V$, there is a path from u to v in G. G is called connected if G^{sym} is strongly connected. A set $U \subseteq V$ is called (strongly) connected if $G \cap U$ is. A strongly connected component of G is a maximal strongly connected set $U \subseteq V$ and for a vertex $v \in V$ and a set $U \subseteq V$, by $C_U(v)$ we denote the (unique) strongly connected component C of $G - U$ such that $v \in C$. Let SCC denote the set of all strongly connected components of G. The graph $B(G) = (\text{SCC}, E^{\text{scc}})$ where $(X, Y) \in E^{\text{scc}}$ if there is some $u \in X$ and some $v \in Y$ with $(u, v) \in E$ is a directed acyclic graph (DAG), called the block graph of G.

We sometimes identify a graph G with its set of vertices, allowing notation like $u \in G$ and $G - H$, where H is the subgraph of G induced by U. (Notice that $G - H$ is not well-defined if H is not an induced subgraph, so there is no danger of confusing the notation.)

Selfloops and Terminal Positions. As we have mentioned above, we allow directed graphs $G = (V, E)$ to contain selfloops $(u, u) \in E$. Usually, in the context of graph complexity, selfloops are discarded because they bring along certain undesirable technical properties. For example, in the presence of selfloops, graphs of DAG-width one are not necessarily acyclic in a strict sense but may have arbitrary selfloops. This simplification can usually be justified very easily,

[6]Notice that there exists an addendum to this paper, currently available at http://people.math.gatech.edu/~thomas/PAP/.

because for many graph properties and graph theoretic decision problems, selfloops can be neglected or replaced by small gadgets in a generic way. For example, Hamilton cycle is invariant under deleting and adding selfloops. Moreover, if we consider the strategy problem for parity games with full information, we can simply replace any selfloop (u, u) in the game graph by an undirected edge $(u, \tilde{u})$ to a new vertex $\tilde{u}$ that belongs to the same player and is colored with the same color as u. Moreover, $\tilde{u}$ does not have any other edges. Notice that a player needs to play a selfloop in a parity game with full information only if he intends to stay on the vertex forever.

On the other hand, for game graphs with partial information, this is not true: The execution of a selfloop during a play may reveal and conceal information about the possible positions to and from player 1, respectively. See for example the game graphs constructed in Section 3.5.2 and also the corresponding class in [172]. Therefore, selfloops are more weighty in games with imperfect information and can have a material impact on the result of the powerset construction.

Nevertheless, it is possible, though technically more costly, to replace selfloops in game graphs with partial information by small gadgets: We have to make the positions u and v indistinguishable for player 1 and mount all edges from u on v as well. More precisely, consider a game graph $\mathcal{G} = (V, \delta, \sim^V, \sim^A)$ with partial information, let $u \in V$ such that there is a selfloop on u and let $A_{u,\circlearrowleft} = \{a \mid \delta(u, a) = u\}$ be the set of all action-labels on this selfloop. We add a new vertex $\tilde{u}$ to V and we define $u \sim^V \tilde{u}$, that is, we put $\tilde{u}$ in the equivalence class of u, and we re-define $\delta(u, a) = \tilde{u}$ for all $a \in A_{u,\circlearrowleft}$. Moreover, for any vertex v and action a with $\delta(u, a) = v$ we set $\delta(\tilde{u}, a) = v$ as well. In particular, we have an undirected edge between u and $\tilde{u}$.

Unfortunately, the impact of this construction on the structure of the game graph is less transparent than in the case of game graphs with full information where we just add an undirected edge to a a vertex that is, apart from that edge, isolated. With respect to directed edges, $\tilde{u}$ is still only sparsely integrated into the game graph. However, if the direction of the edges is neglected then $\tilde{u}$ is intertwined more strongly with the rest of the graph. Indeed, avoiding selfloops makes formally defining and reasoning about such classes of game graphs technically often much more involved. Hence, we prefer to keep selfloops.

Nevertheless, essentially, all our results can be proved also for the case where we discard selfloops. (Using the construction described above or slight improvements, adapted to the particular game graphs.) We briefly go over the differences: The precise bound from Section 3.5.2 is slightly different: We obtain a lower bound of roughly $2^{|V|/8}$ on the size of memory needed to implement winning strategies in safety games with imperfect information on finite game graphs with partial information and without selfloops. Moreover, the game graphs used in Proposition 4.8 have tree-width at most three instead of two and the same holds for the game graphs used in Proposition 4.7. All other results remain unchanged. In particular, we like to emphasize that the main results are not affected by this issue. Theorems 4.11, 4.12 and 4.13 hold as stated, if we disallow selfloops. (Notice that then, Theorem 4.12 could be formulated in terms of DAG-width one instead of acyclic graphs.) Moreover, the result in Theorem 4.20 is also clearly invariant under disallowing selfloops.

Another issue that arises in the context of structural complexity of graphs is that of terminal positions. For example, a DAG in the strict sense, does not have any cycles so it inevitably contains terminal positions. Terminal positions, however, incur finite plays and the kind of winning conditions that we have defined so far does not determine the winner of a finite play. Usually we assume that game graphs are non-terminating so that finite plays are not possible. But throughout this chapter we do allow terminal positions in game graphs and we assume that the set of these positions is divided into those where player 0 has won and those where player 1 has won. Clearly, this can also be modeled by game graphs which are non-terminating using a parity (or even reachability) condition: We can put selfloops on the terminal positions and color positions where player i has won with color $1-i$. (In fact, in the context of reachability conditions we consider the terminal positions where player 1 has won as part of the reachability set R.) However, to be able to refer to trees and acyclic graphs in the usual (strict) sense, we allow terminal positions explicitly.

4.1.1 Tree-Width

A *tree-decomposition* of an undirected graph $G = (V, E)$ is a pair $(T, \mathcal{X})$, where T is a tree and $\mathcal{X} : T \to 2^V$ is a function such that the following conditions hold.

(T1) $V = \bigcup\{\mathcal{X}(t) \mid t \in T\}$.

(T2) For each edge $(u, v) \in E$ there is some $t \in T$ such that $\{u, v\} \subseteq \mathcal{X}(t)$.

(T3) For all $v \in V$, the set $\{t \in T \mid v \in \mathcal{X}(t)\}$ induces a connected subtree of T.

So each edge of G has to be contained in some of the bags $\mathcal{X}(t)$. Furthermore, if for $t_1, t_2 \in T$, a node $v \in V$ occurs in the bag of t_1 and t_2, then it occurs in each bag along the unique path from t_1 to t_2 in T. The *width* of $(T, \mathcal{X})$ is the maximal size of a bag occurring in the decomposition, minus one: $\text{width}(T, \mathcal{X}) = \max_{t \in T} |\mathcal{X}(t)| - 1$. The *tree-width* of G, denoted $\text{tw}(G)$, is the minimum width of all possible tree-decompositions of G. So, intuitively, the tree-width measures how close an undirected graph is to a (undirected) tree. (The -1 is to make trees have tree-width one.)

Interfaces. Now we can also make more precise what we mean by saying that the individual parts (bags) are only sparsely related within the given graph: Consider some designated node t_0 of T as the fixed root of the tree. We define the partial order $\preceq$ on T by $t \preceq s$ if the unique path from s to t_0 contains t. Moreover, for a node t, the subtree of T rooted in t is $T_{t\preceq} := \{s \in T \mid t \preceq s\}$. Now we set $\mathcal{X}(t \preceq) := \bigcup\{\mathcal{X}(s) \mid t \preceq s\}$ and $\mathcal{X}(t \prec) := \mathcal{X}(t \preceq) \setminus \mathcal{X}(t)$. Now assume that $T, \mathcal{X})$ is a tree-decomposition and t a node of T.

Then the only vertices in $\mathcal{X}(t \preceq)$ that are adjacent to vertices in $V \setminus \mathcal{X}(t \preceq)$ are those belonging to $\mathcal{X}(t)$. To see this, consider any edge $(u, v) \in E$ with $u \in \mathcal{X}(t \preceq)$ and $v \in V \setminus \mathcal{X}(t \preceq)$. Then $u \in \mathcal{X}(s)$ for some $t \preceq s$ and according to condition (T2), $\{u, v\}$ is contained in a bag $\mathcal{X}(s')$ for some node $s' \in T$. Since $v \notin \mathcal{X}(t \preceq)$ we have $t \not\preceq s'$ and so, by condition (T3), $u \in \mathcal{X}(t)$. So the subgraphs $G \cap \mathcal{X}(t \prec)$ and $G - \mathcal{X}(t \preceq)$ are only related via the interface $\mathcal{X}(t)$![7]

Cops and Robber Games. The tree-width of an undirected graph $G = (V, E)$ can be characterized in terms of a game: There are two players, a cop player and a robber player. The robber player moves a robber token along the edges of G at unlimited speed. The cop player has a number cops at his disposal and he may place these cops on the vertices of G arbitrarily. When the cop player chooses a new set U of positions to be occupied by the cops, the robber player may move his token along any path on which there is no position that has been previously occupied by a cop and is also occupied by a cop according to U. The goal of the cop player is to land a cop on the robber, the robber player's goal is to elude capture. We also speak of the cops and the robber as autonomous entities, allowing terms like cops' moves or robber's position.

Formally, cops' positions are of the form (U, v) where the cops occupy the set U and the robber is on vertex v. The cops can move to any position (U, U', v), indicating the set U' as the positions that will be occupied next. We say that the cops $U \setminus U'$ are taken from the graph, the cops $U' \setminus U$ are placed on the graph and the cops $U \cap U'$ stay put. In turn, robber's positions are of the form (U, U', v) and the robber can move to any position (U', w) with $w \in \text{Reach}_{G - U \cap U'}(v)$. Positions (U', w) with $w \in U'$ are the *terminal positions* of the game graph where the robber is captured and the cops have won. All infinite plays are won by the robber.

The particular property of the robber's move is that he can run at the moment, where the cops $U \setminus U'$ are already removed from the graph but the cops $U' \setminus U$ have not yet been placed on the graph (we say that they are *about to land*). So he can use any path that avoids $U \cap U'$. Additionally, we have a unique initial position $\perp$, where it is the robber's turn and from which he can go to any position $(\emptyset, v)$. So a play in the game has the form $\pi = (U_0, v_0)((U_0, U_1), v_0)(U_1, v_1) \ldots$ where $U_0 = \emptyset$ and the initial position $\perp$ is usually omitted.

[7]We refer to this kind of property also as separator property and it is precisely this kind of property that allows to use such decompositions for dynamic programs.

Notice that the condition that w is reachable from v in $G{-}U \cap U'$ is equivalent to v and w being in the same connected component of $G{-}U \cap U'$ (since G is undirected). Moreover, as the robber can run at unlimited speed, the particular position of the robber within the connected component of $G{-}U$ is not important, so we could describe positions of the cops and robber game also as (U, U', C) and (U, C), where C is a connected component of $G{-}U$. However, for our purposes it is often more convenient to refer to the exact position of the robber.

The game is played by two players on a finite graph of exponential size $2^{O(|V|)}$. The k cops and robber game is played on the same graph, restricted to positions (U, U', v) with $|U|, |U'| \leq k$. Hence, this game graph has polynomial size. Remind that a play in the game is won by the robber if it is infinite and won by the cops if it is finite. So, essentially, cops and robber games are reachability games (with full information), which implies that they are positionally determined and can be solved in time linear in the size of the game graph.

Monotonicity. Now the games that we are actually interested in are *monotone* cops and robber games, which are played on the same game graph but the winning condition is modified: A play is monotone if the robber can never return to a position that has previously been unavailable for him. Formally, a play is monotone if for all positions (U, U', v) we have $\text{Reach}_{G-U\cap U'}(v) \subseteq \text{Reach}_{G-U}(v)$. A finite play in the monotone cops and robber game is won by the cops only if it is monotone. This can be formulated as a winning condition that is a conjunction of a reachability and a safety condition for the cop player, so these games are also positionally determined. Moreover, a strategy σ for the cops is called monotone, if any play which is consistent with σ is monotone. So monotone winning strategies for the cops are precisely the winning strategies for the cops for the monotone cops and robber game. If the cop player wins the (monotone) k cops and robber game on G then we also say that k cops (monotonously) capture the robber on G.

The definition of monotonicity that we have given here is also called robber-monotonicity. On the other hand, cop-monotonicity is defined by the requirement that no vertex is visited twice by the cops. (Notice that we do not distinguish between the individual cops.) Formally, a play $\pi = (U_0, v_0)((U_0, U_1), v_0)(U_1, v_1)\ldots$ is cop-monotone, if for all j and all $v \in U_j \setminus U_{j+1}$ we have $v \notin U_l$ for all $l > j$. However, it is easy to see that whenever the cops have a winning strategy that is robber-monotone or cop-monotone, they also have a winning strategy that is both. So in this sense, the notion of monotonicity is canonical.

In fact, a strategy for the cops that is cop-monotone is also robber-monotone. To see this, notice that a play is monotone if it never reaches a position (U, U', v) such that there is some $w \in U \setminus U'$ and some $u \in \text{Reach}_U(v)$ such that $(u, w) \in E$. (In other words, if it never reaches a position such that $(U \setminus U') \cap \text{Reach}_{G-U\cap U'}(v) \neq \emptyset$.) That means, a play is monotone if the robber can never reach a position from which a cop is just leaving. (Clearly this is necessary for monotonicity but also, if the robber can reach a position that was unavailable for him previously then he must travel through such a position u.) Using this, robber-monotonicity follows directly from cop-monotonicity. In our proofs, we use this formulation of robber-monotonicity.

To see that a robber-monotone winning strategy can be made cop-monotone as well, consider a robber-monotone winning strategy σ for k cops and assume that at some point during some σ-play a cop is placed on a vertex u that is not reachable for the robber, that is, $u \in U' \setminus U$ but $u \notin \text{Reach}_{G-U\cap U'}(v)$. It is easy to see that if we just omit placing a cop on u, then the resulting strategy will be winning and, moreover, this transformation also preserves robber-monotonicity of σ. Hence, there is a robber-monotone winning strategy for k cops that never places a cop at a vertex that is unavailable for the robber. But this strategy is also cop-monotone: Since the strategy is robber-monotone, a cop can be taken from a vertex only if that vertex is already unavailable for the robber – but then it will never be re-occupied. Notice that the condition that cops are only placed in the area which is still reachable for the robber is equivalent to the requirement that cops are only placed inside the connected component of the robber! We shall keep this observation in mind, because the *failure* of this latter property for *directed* graphs will be important for us later.

Characterizations of Tree-Width. In [198], the following characterization has been shown.

Theorem 4.1. *[198] The following statements are equivalent for any undirected graph G:*

(1) G has tree-width at most $k-1$.

(2) k cops monotonously capture a robber on G.

(3) k cops capture a robber on G.

So by using the implications (3) to (1) and then (1) to (2) we see that, in fact, monotone winning strategies suffice to win k cops and robber games, that is, if k cops capture a robber on G, then k cops capture a robber on G monotonously. We also say that tree-width has monotonicity cost 0 or simply *is monotone*.

Clearly, the implication from (2) to (3) is trivial. To obtain a monotone strategy from a tree-decomposition is also not too hard: Given a tree-decomposition T of width $k-1$ of a graph G, then k cops play along T as follows: As before, we fix some root t_0 of T. The cops occupy $b_0 = \mathcal{X}(t_0)$, so the robber will be in a bag $\mathcal{X}(t)$ for some $t_0 \neq t$. Now consider the unique successor t_1 of t_0 with $t_1 \preceq t$. Then the set $\mathcal{X}(t_1 \preceq)$ is a trap for the robber: He could go to vertices outside $\mathcal{X}(t_1 \preceq)$ only via $\mathcal{X}(t_0)$ (using conditions (T2) and (T3) as above), but these vertices are occupied by cops. Now consider the cops' move from $\mathcal{X}(t_0)$ to $\mathcal{X}(t_1)$ (in particular, the cops on $\mathcal{X}(t_0) \cap \mathcal{X}(t_1)$ stay put) and consider some edge $(u, v) \in E$ with $u \in \mathcal{X}(t_1 \preceq)$ and $v \in \mathcal{X}(t_0)$. Then, by (T2), $\{u, v\}$ is contained in some bag $\mathcal{X}(s)$. Moreover, by (T3), if $t_1 \preceq s$ then $v \in \mathcal{X}(t_1)$ and if $t_1 \not\preceq s$ then $u \in \mathcal{X}(t_0)$ which demonstrates that the move from $\mathcal{X}(t_0)$ to $\mathcal{X}(t_1)$ is robber-monotone. So in this way the robber is chased down the tree T in a robber-monotone way and, eventually, captured in a leaf of T.

The more difficult part of the theorem is the implication from (3) to (1). Assume that σ is a (positional) winning strategy for k cops on G (which depends only on the connected component of the robber). Let the vertices of T be of the form (U, C) where $U \subseteq V$ and C is a connected component of $G-U$ and there is an edge between (U, C) and (U', C') if $(U, C) \to (U', C')$ is a move in the k cops and robber on G that is compatible with σ. That means, given the position (U, C), the cops move to U' according to σ and subsequently, the robber has the possibility to run to the connected component C'.

Since σ is a winning strategy for the cop player, there are no cycles in T. Now if we define the bag of a node (U, C) by $\mathcal{X}(U, C) = U$ then conditions (T1) and (T2) hold. (Notice that, since all edges are undirected, the robber may bum around at any given set $\{u, v\}$ with $(u, v) \in E$, as long as not both, u and v, are occupied by cops at the same time. So, for each edge $(u, v) \in E$ there is some (U, C) such that $\{u, v\} \subseteq U$.) However, condition (T3) does not hold in general.[8]

For this implication, Seymour and Thomas used the concept of a *screen*:[9] For an undirected graph $G = (V, E)$, two sets $U_1, U_2 \subseteq V$ are called *touching*, if $U_1 \cap U_2 \neq \emptyset$ or if there are $u_1 \in U_1$ and $u_2 \in U_2$ such that $(u_1, u_2) \in E$. A set $U \subseteq V$ *covers* a set $\mathcal{U} \subseteq 2^V$, if for each $U' \in \mathcal{U}$ we have $U' \cap U \neq \emptyset$. A *screen* in G is a set of connected, mutually touching subsets of V. The minimum size of all sets covering a screen is called its *thickness*. The main theorem of [198], which also yields the implication from (3) to (1), is called *tree-width duality theorem*. A simpler proof has been given in [14].

Theorem 4.2. *[198] (Tree-width Duality Theorem) An undirected graph has a screen of thickness at least k if and only if it has tree-width at least $k-1$.*

Notice that the theorem gives in fact a dual characterization of tree-width: A tree-decomposition of width *at most* $k-1$ does *not* exist if a screen of thickness *at least* k *does* exist. (Seymour

[8]Notice that even if σ was monotone, a tree-decomposition could not be obtained in this way because T is undirected, while plays of a game are inherently directed and the monotonicity condition holds only in the direction in which the plays proceed.

[9]Screens are also often called bramble in the literature.

and Thomas called it min-max theorem.) Clearly, this is a most valuable tool for showing *lower bounds* on the tree-width of certain classes of graphs. This has also been used on the algorithmic side where the task is to *compute lower bounds* for the tree-width of a given graph [37].[10]

We use the tree-width duality theorem to demonstrate a basic property of partial grids. It is well known that the full undirected grid $\mathfrak{G}_m$ has tree-width m, where $\mathfrak{G}_m = ([m] \times [m], E_m)$ with edges

$$(j,l) \longleftrightarrow (j{+}1,l) \text{ for } j \in \{0,\dots,m{-}2\} \text{ and } l \in [m]$$

$$(j,l) \longleftrightarrow (j,l{+}1) \text{ for } j \in [m] \text{ and } l \in \{0,\dots,m{-}2\}$$

For $h, v \in [m]$, we call $H_h := [m] \times \{h\}$ a *row* or *horizontal level* of $\mathfrak{G}_m$ and analogously, $V_v := \{v\} \times [m]$ is called a *column* or *vertical level* of $\mathfrak{G}_m$. Moreover, we define $C_{h,v} = H_h \cup V_v$, that is, $C_{h,v}$ is a *cross* in $\mathfrak{G}_n$. Now consider the set $\mathcal{C} = \{C_{h,v} \mid h, v \in [m]\}$, consisting of all possible crosses in $\mathfrak{G}_m$ It is not hard to see that $\mathcal{C}$ is in fact a screen in $\mathfrak{G}_m$. Moreover, for any set $U \subseteq [m] \times [m]$ with $|U| < m$ we can find a cross $C_{h,v}$ such that $C_{h,v} \cap U = \emptyset$, so $\mathcal{C}$ has thickness at least m. Using the tree-width duality theorem this shows that $\text{tw}(\mathfrak{G}_m) \geq m{-}1$.

In fact, as we have mentioned, $\text{tw}(\mathfrak{G}_m) = m$. It is quite easy to see that $m{+}1$ cops can capture a robber on $\mathfrak{G}_m$, so $\text{tw}(\mathfrak{G}_m) \leq m$. To show that $\text{tw}(\mathfrak{G}_m) \geq m$, one has to use a slightly more elaborate screen consisting of pruned crosses and additional columns and rows (see for example [37]). Although we don't need either one of these results – $\text{tw}(\mathfrak{G}_m) \geq m{-}1$ is quite sufficient for our concerns – we prefer to specify the tree-width of $\mathfrak{G}_m$ exactly as $\text{tw}(\mathfrak{G}_m) = m$.)

We also need a corresponding result for the following less common class of *partial grids*: For even $m \in \mathbb{N}$, let $\mathfrak{G}_{\frac{1}{2},m} = ([m] \times [m], E_{\frac{1}{2},m})$ with $(j,h) \leftrightarrow (j{+}1,h)$ as before and $(v,j) \leftrightarrow (v,j{+}1)$ if v and j are both odd or both even. So $\mathfrak{G}_{\frac{1}{2},m}$ is obtained from $\mathfrak{G}_m$ by deleting, on each *even* vertical level V_v, every *even* edge $(v,j) \leftrightarrow (v,j{+}1)$, $j = 0, 2, \dots, m$ and, on each *odd* vertical level V_v, every *odd* vertical edge $(v,j) \leftrightarrow (v,j{+}1)$, $j = 1, 3, \dots, m{-}1$.

Proposition 4.3. $\text{tw}(\mathfrak{G}_{\frac{1}{2},m}) = m/2$.

Proof. The proof given above for $\mathfrak{G}_m$ can be adapted to $\mathfrak{G}_{\frac{1}{2},m}$ easily: For all *even* $h, v \in [m]$, define the double row $H^2_h := [m] \times \{h, h{+}1\}$ and the double column $V^2_v := \{v, v{+}1\} \times [m]$. This yields the double crosses $C^2_{h,v} = H^2_h \cup V^2_v$ as before, for h and v both even. As before, it is not hard to see that the set $\mathcal{C}^2 = \{C^2_{h,v} \mid h, v \text{ even }\}$ is a screen in $\mathfrak{G}_{m,\frac{1}{2}}$. Moreover, for any set U of vertices in $\mathfrak{G}_{\frac{1}{2},m}$ with $|U| < m/2$, we can find at least one $C^2_{h,v}$ for certain even $h, v \in [m]$ such that $U \cap C^2_{h,v} = \emptyset$, so $\mathcal{C}^2$ has thickness at least $m/2$. Hence, by Theorem 4.2, we obtain $\text{tw}(\mathfrak{G}_{\frac{1}{2},m}) \geq m/2{-}1$. The precise complexity of $\text{tw}(\mathfrak{G}_{\frac{1}{2},m}) = m/2$ can again be shown by a refinement of this proof. □

We like to note that $\text{tw}(\mathfrak{G}_{\frac{1}{2},m}) \geq m/2$ can also be obtained as a corollary of the corresponding result for $\mathfrak{G}_{m/2}$, using the concept of minors: It is not hard to see that, in fact, $\mathfrak{G}_{m/2}$ is a minor of $\mathfrak{G}_{\frac{1}{2},m}$ and it is well known that the tree width of a minor of a graph is at most the tree-width of the graph itself, see for example [67]. Hence, $\text{tw}(\mathfrak{G}_{\frac{1}{2},m}) \geq \text{tw}(\mathfrak{G}_{m/2}) = m/2$.

4.1.2 Complexity Measures for Directed Graphs

It is widely accepted that tree-width is a good measure for the complexity of an undirected graph. Tree-width is well understood, has several nice properties like monotonicity and a characterization by screens, and many hard problems are known to be solvable on classes of graphs with bounded tree-width.

On the other hand, for the complexity of graphs with respect to *directed* edges, there is is no canonical notion which would have proven similar significance as tree-width and it is most controversial what would be a good measure for the complexity of a directed graph. We will

[10]For more information on computing the tree-width of a graph we refer to [36].

mostly work with the characterizations of measures by cops and robber games and, in this respect, DAG-width is particularly nice.

DAG-width. DAG-width has been introduced independently in [158] and [24] and it has been characterized via the monotone cops and *directed* robber game. In fact, this game is exactly the same as the (monotone) cops and robber game as described above for tree-width with the only exception that the robber has to respect the direction of the edges which is a particularly straightforward and intelligible generalization of the game to directed graphs.

All definitions and observations carry over from the cops and robber games to the cops and directed robber games without any changes. We define the *DAG-width* of a graph $G = (V, E)$, denoted $\mathrm{dw}(G)$, to be the minimal number $k \in \mathbb{N}$ such that k cops *monotonously* capture a *directed* robber on G.[11] Notice that for any undirected graph G we have $\mathrm{tw}(G) = \mathrm{dw}(G) - 1$ and moreover, the graphs of DAG-width one are those where all cycles are selfloops.

Just as for the cops and (undirected) robber game, if the cop player has a winning strategy for the k cops and directed robber game which is robber-monotone or cop-monotone, then he also has a a winning strategy for the k cops and directed robber game which is both. In the following, we usually call the directed robber simply robber and speak of the cops and robber game on G. We will make the distinction explicit only when necessary.

It has been shown [158, 24] that DAG-width gives rise to a decomposition of directed graphs, similar to the tree-decomposition of undirected graphs. Where now, the decomposition is not an undirected tree, but a directed acyclic graph.

To define this notion of decomposition formally, for a directed acyclic graph $G = (V, E)$, let $\preceq$ denote the transitive and reflexive closure of E. (Which is analog to the partial order $\preceq$ defined above for trees.) Clearly, $\preceq$ is a partial ordering on V. We say that a set $U \subseteq V$ *guards* a set $W \subseteq V$ if for each edge $(w, v) \in E$ with $w \in W$ and $v \notin W$ we have $v \in U$. A *DAG-decomposition* of a directed graph $G = (V, E)$ is a pair $(D, \mathcal{X})$, where D is a directed acyclic graph and $\mathcal{X} : D \to \mathcal{P}(V)$ is a function such that the following conditions hold.

- $\bigcup\{\mathcal{X}(d) \mid d \in D\} = V$
- If $d \preceq e \preceq f$ then $\mathcal{X}(d) \cap \mathcal{X}(f) \subseteq \mathcal{X}(e)$.
- If (d, e) is an edge in D, then $\mathcal{X}(d) \cap \mathcal{X}(e)$ guards $\mathcal{X}(e \preceq) \setminus \mathcal{X}(d)$. Moreover, any root is guarded by the empty set.

We define the width of $(D, \mathcal{X})$ as $\max\{|\mathcal{X}(d)| \mid d \in D\}$.

Theorem 4.4. *[158, 24] A graph has DAG-width at most k if, and only if, it has a DAG decomposition of width at most k.*

Given a DAG-decomposition $(D, \mathcal{X})$ of width k we can play along D with k cops, just as we have played along the tree of a tree-decomposition. Only now, we have to start at one of the roots of D and play in the direction that is determined by the edges of D. The properties of the decomposition guarantee that the strategy is monotone and winning. Moreover, since now the edges of D do have a direction, the converse is more immediate as well, cf. [158, 24].

However, the correspondence between DAG-decompositions and *arbitrary* winning strategies for the cops fails in this case! In [123] it has been shown that there are graphs G_m, $m \in \mathbb{N}$ such that $3m-1$ cops capture a robber on G_m but $\mathrm{dw}(G_m) = 4m-2$. Hence, DAG-width has monotonicity cost at least $m-1$. Of course this is an inconvenient property, because to prove upper bounds on DAG-width we have to use monotone strategies or construct DAG-decompositions explicitly which is often quite complicated (or at least a tedious exercise).

On the other hand, whether DAG-width has *bounded* monotonicity cost is still open. Where bounded means the existence of a function $\kappa : \mathbb{N} \to \mathbb{N}$ such that whenever k cops capture a robber on a graph G then $k+\kappa(k)$ cops monotonously capture a robber on G. This would give a

[11] In [158, 24], DAG-width has been defined via DAG-decompositions. However, we like to start with the game.

bound on the monotonicity cost that does not depend on the graph, so in particular, if for a class of graphs the number of cops needed to capture a robber is bounded, then DAG-width would be bounded as well. In [25] it has been conjectured that DAG-width has linear monotonicity cost, that is, there is such a function $\kappa \in O(k)$.

Moreover, in [24] (see also [25]) it has been shown that the properties of DAG decompositions are sufficient in order for DAG-decompositions to be used for the construction of dynamic algorithms for parity games with full information:

Theorem 4.5. *[24] The strategy problem for parity games with full information can be solved in polynomial time for classes of graphs with bounded DAG-width.*

This is the most important algorithmic aspect about DAG-width for our work. We do not discuss the corresponding algorithm here but merely use this result as a black box. Since all game graphs to which we shall apply this result result from applying the powerset construction to some game graph with partial information, it might be interesting to see whether the method could be optimize for such powerset graphs, cf. Section 3.5.3. However, we do not consider this.

Normal Forms of Winning Strategies. A possible step towards proving that DAG-width has bounded monotonicity cost would be to prove certain normal forms of winning strategies for the cops which are weaker than monotone winning strategies but still have some nice properties that could be used to transform them into monotone winning strategies. Where by normal form we mean that any (monotone) winning strategy for the cops can be translated into a (monotone) winning strategy of this form using only *boundedly* many additional cops (that is, $\kappa(k)$ many, as defined above). However, as it turns out, winning strategies for the cops on directed graphs are quite resistant to normal forms.

Among the very few normal forms that are known for cops' winning strategies on directed graphs are the following simple ones:

(1) They need to depend only on the strongly connected connected component of the robber and not on his precise position.

(2) They do not need to move more than one cop per move (so in particular they don't need to remove and place cops in the same move).

(3) If monotone, they do not need to place cops on positions that are not reachable for the robber.

None of these normal forms require any additional cops. However, while the normal form (2) is quite useful for developing dynamic algorithms on DAG-decompositions[12], neither (1) nor (2) have been very helpful yet towards the monotonicity problem and neither will they do us any good during this work.

We will use (3) in the proof of our main technical result in Section 4.4. The proof of (3) is as trivial as for tree-width: Just omit placing cops that would be placed on vertices that are not reachable for the robber. Since the strategy is monotone, the robber will never be able to reach any of those positions later on. On the other hand, as we have mentioned, for *undirected* graphs, the area that is reachable for the robber coincides with his connected component. So for undirected graphs, property (3) guarantees that a monotone strategy for the cops needs to place cops only inside the connected component of the robber. This normal form, however, does *not* hold for *strongly connected components* in directed graphs! We will show that there is a class of graphs G_m, $m \in \mathbb{N}$ such that three robbers can capture a robber on G_m monotonously for any m, but at least $m+1$ cops are needed to capture a robber componentwisely on G_m (even non-monotonously), that means, the cops are only allowed to be placed inside the strongly connected component of the robber.

[12]Normal form (2) corresponds to *nice DAG decompositions* [24] – the analogue of nice tree decompositions, yielding the same normal form for cops' strategies on undirected graphs.

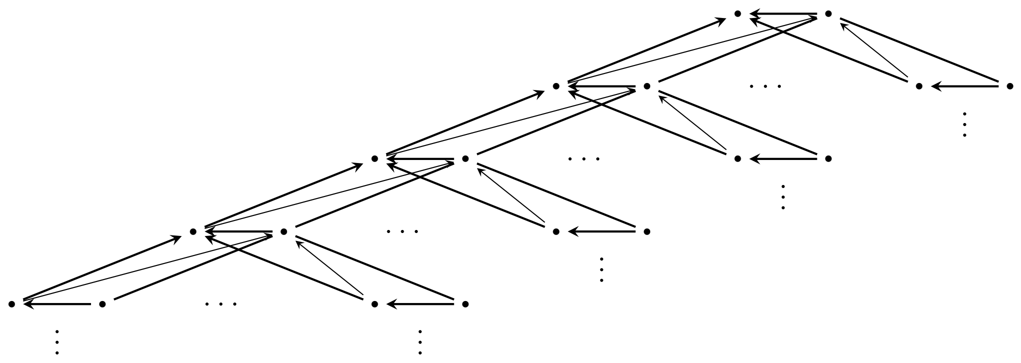


Figure 4.1: $\mathrm{dw}(G_m) = 4$, but m cops are needed to capture a robber componentwisely.

Before we prove this result, let us comment on some consequences. There are certain indications that it might be easier to transform cops' winning strategies into monotone ones if they are indeed restricted to place cops only inside the strongly connected component of the robber, see also the example in [123]. It would also be interesting to analyze the resulting notion as a graph complexity measure itself. However, our result on componentwise capturing a robber shows that this would not help to prove bounded monotonicity cost for DAG-width. Moreover, if this normal form existed, at least for *monotone* winning strategies, then the main result from Section 4.4 would be much easier to proof. We will have a closer look at this issue in Section 4.4.

For $m \in \mathbb{N}$, consider the full, undirected tree T_m of degree and depth m, that is, T_m has nodes $\{1, \ldots, m\}^{\leq m}$ and, for any $u \in \{1, \ldots, m\}^{<m}$ and any $j \in \{1, \ldots, m\}$, there is an edge from u to $u \cdot j$ and from $u \cdot j$ to u. Moreover, let $T_m^{\uparrow}$ be obtained from T_m by directing any edge in T_m to the root. Assume that $T_m^{\uparrow}$ has vertices $\{\overline{1}, \ldots, \overline{m}\}^{\leq m}$ so that T_m and $T_m^{\uparrow}$ are disjoint and for any node $u = u_0 \ldots u_l$ in T_m, let $\overline{u} = \overline{u}_0 \ldots \overline{u}_l$ denote the corresponding node in $T_m^{\uparrow}$. Notice that if we speak of a prefix $u \sqsubseteq v$ of a node v in the tree T_m we refer to the usual prefix relation on words and not to the edge relation of the tree. (Analogously for nodes $\overline{u} \sqsubseteq \overline{v}$ in $T_m^{\uparrow}$.) Now let G_m consist of the union of T_m and $T_m^{\uparrow}$ together with the following additional *directed* edges:

- $u \longrightarrow \overline{u}$ for any u
- $\overline{u} \longrightarrow u^{-1}$ for any $\overline{u} \neq \varepsilon$.

The shape of the graphs G_m is delineated in Figure 4.1. It is easy to see that $\mathrm{dw}(G_m) = 3$ for any $m \in \mathbb{N}$: The cops can start by occupying ε and $\overline{\varepsilon}$ and then chase the robber in a top-down manner, eventually capturing him in a leaf. This is monotone, if the cops bear in mind that they always have to occupy $\overline{vj}$, then take the cop from $\overline{v}$ and put it on vj. Only then, the cop from v can be taken as well. On the other hand, unboundedly many cops are needed to capture a robber componentwisely on the graphs G_m, $m \in \mathbb{N}$.

Proposition 4.6. *At least m cops are needed to capture a robber on G_m if cops may be placed only inside the strongly connected component of the robber.*

Proof. Let $l < m$ and consider any position (U, v) of the l cops and robber game on G_m where it is the cops' move. So the set of vertices occupied by cops is U and the robber is on vertex v in T_m. We define $\mathrm{Pre}(v) := \{u \in T_m \mid u \sqsubseteq v\}$ as the set of all prefixes of v (in T_m) and, accordingly, $\mathrm{Pre}(\overline{v})$ as the set of all prefixes of $\overline{v}$ (in $T_m^{\uparrow}$). Now assume that the following invariants hold.

(1) Any strict prefix $u \in \mathrm{Pre}(v) \setminus \{v\}$ of v in T_m is occupied by a cop.

(2) Any prefix $\overline{u} \in \mathrm{Pre}(\overline{v})$ of the corresponding vertex $\overline{v}$ in $T_m^{\uparrow}$ is cop-free.

We show that for any subsequent move of the cops, the robber can answer this move in such a way that, afterwards, the invariants (1) and (2) hold again. Clearly, the initial cops' position where the robber occupies the root ε of T_m and no cops are placed on the graph yet fulfills the invariants. Moreover, the conditions (1) and (2) together ensure that the robber is not captured, so by playing in this way, the robber will escape l componentwise hunters.

Consider any next move $(U, v) \to (U, U', v)$. It is easy to see that due to condition (1), none of the prefixes $\overline{u} \in \text{Pre}(\overline{v})$ is in the strongly connected component $C_U(v)$ of the robber in $G{-}U$. So, since the cops are playing componentwisely and, by condition (2), all these ancestors $\overline{u} \in \text{Pre}(\overline{v})$ are cop-free according to U, they are also cop-free according to U', that is, $\text{Pre}(\overline{v}) \cap U' = \emptyset$.

For the move of the robber we distinguish two cases. First, if a cop has been taken from some of the vertices $\text{Pre}(v)$, that means, there is a strict prefix $u \sqsubseteq v$ with $u \notin U'$, then the robber runs to the $\sqsubseteq$-minimal such u via the cop-free path $v \to \overline{v} \to \overline{v}^{-1} \to \ldots \to \overline{u} \to u$. Due to the choice of u and the fact that $\text{Pre}(\overline{v}) \cap U' = \emptyset$, conditions (1) and (2) both hold after this move of the robber.

Second, if no cop has been taken from any vertex $\text{Pre}(v)$ then, due to condition (1) and the fact that there are at most l cops, we have $|v| \leq l < m$, so the robber is not in a leaf. Now, if $v \notin U'$ then the robber just stays on v which guarantees that conditions (1) and (2) hold after the move. (Recall that we have already demonstrated that $\text{Pre}(\overline{v}) \cap U' = \emptyset$.) If, on the other hand, $v \in U'$ then at least one the $m > l$ many subtrees below v must be cop-free, so there is some $j \in \{1, \ldots, m\}$ such that the whole subtree rooted in $v \cdot j$ is cop-free. The robber now moves to $v \cdot j$ which again ensures that conditions (1) and (2) hold. $\square$

Remark. So far we have only talked about normal form for cops' strategies. Remind that the proof of monotonicity of tree-width goes through a normal form of strategies for the robber: If k cops cannot capture a robber monotonously then, by determinacy, the robber has a strategy that is winning against all monotone strategies for k cops. But then, Theorem 4.2 yields a screen of thickness at least k which, in turn, can be used as a strategy for the robber which is then strong enough to win against *any* strategy of k cops. However, normal forms of strategies for the robber are few and far between as well. In [113], a discussion on notions like screens for directed graph can be found. Notice, however, that for the notion of *D-haven* as suggested there, the correspondence to robber's strategies that can defeat any *monotone* strategy of the cops remains unclear. A normal form for robber's strategies that we will use later one is that the robber needs to actually run in his move only if he wants to go to a position that would be unavailable for him after the cops have landed. (We call such strategies *prudent*.) However, this normal form is again rather straightforward.

Directed Path-width. We also consider directed path-width, which is a more restrictive measure than DAG-width. The *directed path-width* of a graph G, denoted $\text{dpw}(G)$, can be characterized as the minimal number k such that $k{+}1$ cops monotonously capture a directed *invisible* robber on G. So the cops and robber games for directed path-width are games of *imperfect information* where the cops have partial information about the past moves of the robber. In fact, all the cops know about the moves of the robber is what they can deduce from their own actions and the edge relation of the graph (since the robber is still bound to run along directed edges). We will see that this makes analyzing the directed path-width of powerset graphs.

Formally, the cops and directed invisible robber game is played on the game graph with partial information that is obtained from the game graph of the cops and directed (visible) robber game by setting $(U, v) \sim^V_{\text{cops}} (U', v')$ if $U = U'$.[13] So, as we have mentioned, the cops knowledge about the possible positions of the robber is limited to what they can deduce from their own actions and the edge relation of the graph, that means, all they know is that the

[13]Actions can be defined on the game graph in the obvious way: $A_{\text{robber}} = V$ and $A_{\text{cops}} = 2^V$.

robber is not on some position of the graph which has already been searched and has henceforth been blocked for the robber.[14] But this also implies that the actual move of the robber does not really matter, because independently to which position he moves, the cops consider *any* position to which the robber *can* move possible to be his actual next position. Therefore, this game can also be viewed as a *solitaire* game played only by the cops. The positions in the corresponding game graph are of the form (U, R), where $R \subseteq V$ is a set of positions such that for any move of the cops from (U, R) to (U', R') we have $R' = \mathrm{Reach}_{G-U\cap U'}(R) \setminus U'$. (Notice that such a set R is not necessarily a strongly connected component of $G{-}U$ but merely a *union* of (*weakly*) connected components of $G{-}U$.)

Strategies can be translated from one representation of the game graph to the other in the obvious way. Notice that, while the original game is a game with imperfect information and the cops need memory to capture the robber (namely, the sets R), the new representation yields a game with full information which is positionally determined.[15] Now, given a (positional) strategy σ for the cops, we obtain a *unique* play of the game (in this new representation of the game graph) which is compatible which σ: $\pi_\sigma = (U_0 = \emptyset, R_0 = V)(U_1, R_1)(U_2, R_2)\dots$ with $U_{j+1} = \sigma(U_j, R_j)$ and $R_{j+1} = \mathrm{Reach}_{G-U_j\cap U_{j+1}}(R_j) \setminus U_{j+1}$. We will use this representation in Section 4.3 which is inessential for the argumentation, but technically convenient.

Obviously, $\mathrm{dw}(G){-}1 \leq \mathrm{dpw}(G)$.[16] So in particular, the strategy problem for parity games with full information is in PTIME for graphs of bounded directed path-width. On the other hand, directed path-width is *not* bounded by DAG-width, that is, there are classes of (directed) graphs with bounded DAG-width and unbounded directed path-width. For example, in [120] is has been shown that any complete undirected ternary tree of depth m has (undirected) path-width exactly m. (Notice that, for any undirected graph, its directed path-width equals its path-width.) In fact, bounded directed path-width is a very strong assumption, so it is often desirable to generalize results for bounded directed path-width to bounded DAG-width. Observe, however, that any directed acyclic graph has directed path-width zero! (A single cop can capture an invisible robber on an acyclic graph by searching the graph systematically, for instance, in a breadth-first order level by level.)

In contrast to DAG-width, directed path-width is monotone: In [11] it has been shown that directed path-width has monotonicity cost at most 1 and in [112] it has been noted that, in fact, the additional cop is not necessary.[17] Our proof that the powerset construction preserves boundedness of directed path-width does not rely on monotonicity which, however, can be used to give a simpler (though less insightful) proof of this result. We will comment on this in Section 4.3.

Entanglement. *Entanglement* was introduced in [26] where it has been shown that the structural complexity of a finite graph determining the number of fixed point variables needed to describe the graph up to bisimulation in the μ-calculus can be characterized by a cops and robber game. The basic concept is the same as for other complexity measures, but the particular rules are quite distinct.

Initially, the robber chooses an arbitrary vertex of the graph. Then, in each round, the cop player may remain idle or place a cop on a vertex v, but only if v is the current location of the robber. In either case, the robber must go from his current position v to a position w, which is cop-free, along an edge $(v, w) \in E$. If the robber cannot move, he loses. So, as before, we have cops' positions (U, v) and robber's positions (U, U', v). From (U, v), the cops can move to (U, U', v) if $U' = U$ (the cops remain idle) or $U' = U \cup \{v\}$ (a fresh cop is placed on v) or

[14] In the terminology of games with imperfect information this is a *blindfold* game.

[15] Essentially, this representation of the cops and directed invisible robber game can be obtained by applying the powerset construction to the original game graph and then choosing a succinct representation of the positions.

[16] Notice that directed path-width has again an offset of one, so in this case, unfortunately, directed paths have directed path-width zero, instead of one.

[17] Notice that for (undirected) path-width the monotonicity result was known already much earlier, see [120, 34, 132].

$U' = (U \setminus \{u\}) \cup \{v\}$ (the cop from u is moved to v). From (U, U', v) the robber can move to (U', w) if $(v, w) \in E$ and $w \notin U'$. The entanglement of a graph G, denoted $\mathrm{ent}(G)$ is the minimal number k such that k cops win the entanglement game on G. Notice that, quite contrary to the other measures, entanglement is *not* defined by way of *monotone* strategies.

As we have mentioned, the source of these particular rules is the variable hierarchy of the modal μ-calculus. Graphs of arbitrarily large entanglement form an integral part of the proof that this hierarchy is, indeed, strict [28, 32, 29]. [18]

In [26] it has also been shown that the strategy problem for parity games with full information can be solved in polynomial time on graphs of bounded entanglement. This, however, has not been done by means of a decomposition as in the case of tree-width and DAG-width. Quite contrary, in general, no decomposition is known that would correspond to entanglement. In [99], a decomposition has been constructed for graphs of entanglement two, but this particular decompositions fails already for graphs of entanglement three.[19] The proof of the PTIME result for parity games on graphs of bounded entanglement directly exploits the game definition of entanglement. Roughly speaking, it proceeds by combining the players in the parity game with the players in the entanglement game in an appropriate way which allows to detect, in an interactive way, whether a cycle has been reached during a play of the parity game.
Entanglement is a quite interesting notion as suggested, especially, by its connection to both, the descriptive as well as the computational complexity of parity games and the modal μ-calculus. Moreover, due to the remarkable rules of the game, it plays a special role as a complexity measure in the structure theory of directed graphs. The precise relation between entanglement and several other complexity measures for directed graphs is still unclear. At least it seems as though entanglement is a rather restrictive measure. In particular, there exist classes of graphs with tree-width three but unbounded entanglement. For an in-depth treatment of entanglement and a state-of-the-art survey see [27].

We conclude our discussion of entanglement with a proof of the result that DAG-width is not bounded by entanglement, originally proved in [26]. We use, however, a different (yet very similar) class of graphs that is specifically suited to show that the powerset construction does not preserve boundedness of entanglement which we prove in Section 4.3.

For any even $m \in \mathbb{N}$, consider the full undirected binary tree $T_{m,2}$ of depth m with nodes $\{0,1\}^{\leq m}$ and, for all $u \in \{0,1\}^{<m}$ and $j \in \mathbb{B}$, an edge from u to $u \cdot j$ and from $u \cdot j$ to u. Moreover, let $\overline{T}_{m,2}$ be a disjoint copy of $T_{m,2}$ with nodes $\{\overline{0}, \overline{1}\}^{\leq m}$. For any $u = u_1 \dots u_l$ in $T_{m,2}$, by $\overline{u} = \overline{u}_1 \dots \overline{u}_l$ we denote the corresponding node in $\overline{T}_{m,2}$. Remind that $\sqsubseteq$ denotes the prefix relation on words.

Now, let G_m consist of the union of $T_{m,2}$ and $\overline{T}_{m,2}$ together with additional edges $u \to \overline{u}$ and $\overline{u} \to u$ for all $u \in \{0,1\}^{\leq m}$. So each node u in $T_{m,2}$ is connected to its copy $\overline{u}$ in $\overline{T}_{m,2}$ via an undirected edge. The shape of the graphs G_m is depicted in Figure 4.3. It is easy to see that $\mathrm{dw}(G_m) = 3$ for all $m \in \mathbb{N}$.

Proposition 4.7. $\mathrm{ent}(G_m) \geq m/2-1$.

Proof. We describe how the robber can escape any number $l < m/2-1$ of cops. For this, assume that the robber is in some leaf $v \in \{0,1\}^m$ of $T_{m,2}$ such that the unique path in $\overline{T}_{m,2}$ from its duplicate node $\overline{v} \in \{\overline{0}, \overline{1}\}^m$ to the root $\overline{\varepsilon}$ is cop-free. (In particular, $\overline{v}$ and $\overline{\varepsilon}$ are cop-free.) Since v has m strict prefixes in $T_{m,2}$ but only $l \leq m/2-1$ cops are available, there is at least one strict prefix $u_{\bullet} \sqsubseteq v$ such that for the unique predecessor $u^{\bullet} \sqsubseteq u_{\bullet}$ of $u_{\bullet}$ in $T_{m,2}$ the following holds:

- The square $\{u_{\bullet}, \overline{u}_{\bullet}, u^{\bullet}, \overline{u}^{\bullet}\}$ in G_m with right lower corner $u_{\bullet}$ is cop-free.

[18] Notice that logics like CTL* and PDL can all be embedded into the two-variable fragment of the μ-calculus. They also do not exploit the full power of the alternation hierarchy which, nevertheless, has been shown to be strict [39].

[19] The structure of graphs of a certain entanglement was first analyzed for *undirected* graphs of entanglement two [13] as well as entanglement three [12].

- There is a cop-free path $P = u^{\bullet} \to \ldots \to w$ in $T_{m,2}$ to a leaf $w \in \{0,1\}^m$ such that the duplicate path $\overline{P} = \overline{u}^{\bullet} \to \ldots \to \overline{w}$ in $\overline{T}_{m,2}$ is cop-free as well.

We describe a sequence of moves for the robber to perform, independently of the moves that the cops might perform inbetween. Remind that, in each step, at most one cop can be actually moved which is then placed on the current position of the robber. The robber steps over from v to the vertex $\overline{v}$ in the other tree and from there he climbs up the unique path to $\overline{u}_{\bullet}$. Then the robber switches trees again, moving from $\overline{u}_{\bullet}$ to $u_{\bullet}$, and he moves up to $u^{\bullet}$. Finally, from $u^{\bullet}$, the robber strolls all the way down the cop-free path P to the leaf w of $T_{m,2}$.

Now, the unique path in $\overline{T}_{m,2}$ from $\overline{w}$ to the root $\overline{\varepsilon}$ is composed of the path $\overline{P}$ from w to $\overline{u}^{\bullet}$ and the path from $\overline{u}^{\bullet}$ to $\overline{\varepsilon}$. Since the path from $\overline{u}^{\bullet}$ to $\overline{\varepsilon}$ is contained in the path from $\overline{v}$ to $\overline{\varepsilon}$, both these paths were cop-free when the robber was in v. Moreover, during the sequence of moves that we have just described, the robber did not enter any node of either one of these paths. So, the unique path in $\overline{T}_{m,2}$ from $\overline{w}$ to the root $\overline{\varepsilon}$ is still cop-free. Hence, by starting in 0^m and following the intended strategy, the robber will always be able to maintain this invariant and escape l cops. □

Directed Tree-width and Kelly-width. We have mentioned two other complexity measures for directed graphs, directed tree-width and Kelly-width. We will not use these measures, but since they are both interesting (though less intuitive) generalizations of tree-width, we give a short description of these measures. We will make some further comments about difficulties that arise when using these measures in the context of the powerset construction in Section 4.3.

The *directed tree-width* of a graph G, denoted $\mathrm{dtw}(G)$, has been introduced in [116] by means of so called *arboreal* decompositions where the underlying structure of the decomposition is a directed tree. It has been shown that directed tree-width is in fact useful in that it is less restrictive for directed graphs than tree-width and still, several interesting combinatorial problems which are computationally hard in general, can be solved efficiently on classes of graphs with bounded directed tree-width, see [116].

However, the definition of directed tree-width is a little intricate and, in particular, the corresponding cops and robber games involve a condition which makes them somehow less intuitive than the games for DAG-width: They are played like the directed cops and robber games but, additionally, the robber is not allowed to leave his strongly connected component. So for any position (U, U', v), the robber may run only to a vertex w if v and w are in the same strongly connected component of $G - (U \cap U')$. Let us call such a robber a *homebody*. These games do not exactly characterize the measure. Rather, the situation is as follows [116]: If $\mathrm{dtw}(G) < k$, then k cops capture a directed homebody on G. Conversely, if k cops capture a directed homebody on G then $\mathrm{dtw}(G) \leq 3k{+}1$. Notice that this immediately yields $\mathrm{dtw}(G) \leq 3 \cdot \mathrm{dw}(G){+}1$ for any graph G. It has also been shown [116, 2] that directed tree-width is not monotone and, unlike DAG-width, cop-monotonicity and robber-monotonicity do not coincide.

It seems, in fact, as though directed tree-width was somewhat difficult to handle. In particular, no polynomial time algorithm for solving the strategy problem for parity games with full information on graphs of bounded directed tree-width is known.[20] Therefore, even if we could prove that the powerset construction preserves boundedness of directed tree-width on graphs of bounded partial information this would still not demonstrate that the strategy problem for parity games on graphs with bounded partial information can be solved in polynomial time (which makes it less worthwhile to try hard).

Kelly-width has been defined by Hunter and Kreutzer in [115] on the basis of various characterizations of tree-width like partial k-trees and elimination orderings. They proved that their generalizations of these notions to directed graphs are again all equivalent which suggests that the resulting graph complexity measure might be a natural and robust generalization of

[20]In [158], Obdrzálek suggests that for this particular problem, directed tree-width appears to be particularly cumbersome.

tree-width to undirected trees. Hunter and Kreutzer called this measure Kelly-width. They also demonstrated that this measure can again be characterized in terms of a cops and robber games which is, however, again less intuitive than the usual cops and directed robber game. In the cops and directed invisible inert robber game (Kelly-game, for short) the robber is also invisible for the cops but this time, he is also inert, which means that he can only leave his current position when a cop is about to land at this position.

Notice that these two conditions are antithetic in that being inert is a restriction for the robber and not seeing the robber is a restriction for the cops. Notice that if we removed inertness from the robber, we would obtain directed path-width. The Kelly-game can also be represented as a solitaire game with positions (U, R) analogously to directed path-width where, this time, if the cops move from U to U' we have $R' = (R \cup \mathrm{Reach}_{G-U\cap U'}(R \cap U')) \setminus U'$, cf. [115]. It has been shown in [115] that the strategy problem for parity games with full information can be solved in polynomial time on graphs of bounded Kelly-width. Since the precise relation between Kelly-width and DAG-width is unknown it would be interesting to know whether the powerset construction preserves Kelly-width on graphs with bounded partial information.

4.2 Unbounded Partial Information

We start our investigation of parity games with imperfect information on graphs of bounded complexity with the case of unbounded partial information, that means, we consider arbitrary game graphs $\mathcal{G} = (V, \delta, \sim^V, \sim^A)$ with partial information where we have no restrictions on $\sim^V$ (or $\sim^A$). In particular, $\sim^V$-equivalence classes may be arbitrarily large. The main result of this section is that the strategy problem for parity games with imperfect information on graphs of bounded complexity is, essentially, as hard as the general problem on arbitrary graphs. So in a sense, the intrinsic complexity caused by partial information is very high, even on simple graphs.

Let us approach this by reviewing how we would solve the strategy problem for parity games with full information on simple graphs. If the complexity of the graphs with respect to, say, DAG-width is bounded by k, then we start by computing a DAG-decomposition of a given graph, which takes only polynomial time, if k is *fixed.* Then we compute certain partial solutions of the strategy problem on the individual bags and combine them in a bottom-up manner along the DAG. Of course, the fully developed solution is much more sophisticated as suggested by this rough description, cf. [25].[21] In particular, the fragmentary solutions on the bags are not completely independent of each other because they do not yet take into account how the game may proceed outside the bags. The point is, however, that the dependencies between the partial solution are smooth enough so that they *can* be handled efficiently, cf. [25].

For parity games with *imperfect* information, the situation is quite different. Here, the bags have much stronger dependencies on each other: Since strategies for player 1 have to satisfy the consistency condition (S2), the actions that player 1 should choose in a bag $\mathcal{X}(d)$ may depend on the decisions in the other bags. Moreover, as we know, imperfect information also entails that strategies for player 1 are not positional, so when we are in some bag $\mathcal{X}(d)$, the actions that player 1 should choose may depend on how the positions in $\mathcal{X}(d)$ have been reached – in general via positions from other bags. So the partial information on the game graphs inhibits a direct dynamic approach!

On the other hand, we know an approach to turn a game graph with partial information effectively into one with full information: The powerset construction. The strategy problem for the game played on this new game graph can then again be solved using the dynamic approach. The drawback is, of course, that the new game graph is exponentially larger than the original one in general. But we might hope that if the complexity of the original game graph is low, then the powerset construction will yield a game graph that has low complexity again and, additionally, is also only polynomially large. Our first result, however, leaves us truly disenchanted: There are

[21] An invaluable tool to reduce some of the merely technical problems are so called *nice* DAG-decompositions, cf. [25], which have particularly easy interfaces between the bags.

game graphs $\mathcal{G}_m$ with partial information and $\chi(\mathcal{G}_m) \leq 2$ for any measure $\chi \in \{\mathrm{tw}, \mathrm{dw}, \mathrm{dpw}, \mathrm{ent}\}$ such that the powerset graphs $2^{\mathcal{G}_m}$ have DAG-width *exponential* in the number of positions of $\mathcal{G}_m$.

The definition of $\chi(\mathcal{G})$ for a game graph $\mathcal{G} = (V, \delta, \sim^V, \sim^A)$ with partial information is straightforward: We set $\chi(\mathcal{G}) = \chi(G)$ where G is the underlying graph structure of $\mathcal{G}$, that is, the directed graph $G = (V, E)$ with

$$E = \{(u, v) \in V \times V \mid \delta(u, a) = v \text{ for some } a \in \mathrm{act}(u)\}.$$

To prove that the powerset construction may blow up the complexity of the graphs exponentially in their size, we reuse the game graphs from Section 3.5.2 showing the exponential lower bound on the size of the memory. Here, we need only the gadget allowing player 0 to generate the digital numbers, that is, the subgraph induced by the positions of player 0, cf. Figure 3.2. Moreover, we double the number of digits which is inessential for the result but technically convenient. Formally, we have $\mathcal{G}_m = (V_m, \delta_m, (\sim^V_m), (\sim^A_m))$, where:

$V_m = V_m^1 = \{v_0\} \cup \{v_j^0, v_j^1 \mid j = 1, \dots, 2m\}$

$A = \{\downarrow_1, \dots, \downarrow_{2m}\} \cup \{\neg_1, \dots, \neg_{2m}\}$.

$v_0 \xrightarrow{\downarrow_j} v_j^0$

$v_j^0 \xrightarrow{\neg_j} v_j^1, \quad v_j^1 \xrightarrow{\neg_j} v_j^0 \quad \text{and} \quad v_j^\iota \xrightarrow{\neg_l} v_j^\iota \quad \text{for} \quad l \neq j$

It is easy to see that $\chi(\mathcal{G}_m) \leq 2$ for all $\chi \in \{\mathrm{tw}, \mathrm{dw}, \mathrm{dpw}, \mathrm{ent}\}$. To prove $\mathrm{dw}(2^{\mathcal{G}_m}) \geq 2^m$ we demonstrate that $2^{\mathcal{G}_m}$ contains a subgraph H that is isomorphic to the full undirected grid $\mathfrak{G}_{2^m}$ of size $2^m \times 2^m$. Obviously, the DAG-width of a subgraph is not greater than the DAG-width of the whole graph and since DAG-width and tree-width coincide on undirected graphs, the result follows from $\mathrm{tw}(\mathfrak{G}_{2^m}) \geq 2^m$. We denote any set

$$\{v_1^{\iota_1}, \dots, v_m^{\iota_m}, v_{m+1}^{\kappa_1}, \dots, v_{2m}^{\kappa_m}\} \subseteq \{v_j^0, v_j^1 \mid j = 1, \dots, 2m\}$$

by the unique corresponding digital number $(\iota_1, \dots, \iota_m, \kappa_1, \dots, \kappa_m) \in \{0, 1\}^{2m}$.

Proposition 4.8. $\mathrm{dw}(2^{\mathcal{G}_m}) \geq 2^m$.

Proof. The graph $2^{\mathcal{G}_m}$ contains the position $\{v_0\}$ with (unique) successor $(0, \dots, 0, 0, \dots, 0)$. From there, player 0 can generate any digital number $(\iota_1, \dots, \iota_m, 0, \dots, 0)$ by successively flipping exactly one of the first m bits in each step, using the actions $\neg_j$ for $j \in \{1, \dots, m\}$. This will generate the, say, top horizontal level of the grid-subgraph H. Notice that player 0 can reverse any flip-operation so that, indeed, any two digital numbers that differ by exactly one bit are connected via an *undirected* edge. Likewise, successively flipping the last m bits generates the elements $(0, \dots, 0, \kappa_1, \dots, \kappa_m)$ of the leftmost vertical level.

In the very same way, given any fixed $(\kappa_1, \dots, \kappa_m)$, from $(0, \dots, 0, \kappa_1, \dots, \kappa_m)$, player 0 can create any vertex $(\iota_1, \dots, \iota_m, \kappa_1, \dots, \kappa_m)$ on the $(\iota_1, \dots, \iota_m)$-th vertical level of H by successively applying the actions $\neg_j$ for $j \in \{1, \dots, m\}$ in the *same sequence* as before. Again, any two successive elements will be connected by an undirected edge. Analogously for the $(\kappa_1, \dots, \kappa_m)$-th horizontal level. All these levels together form the full undirected $2^m \times 2^m$-grid H. $\square$

Notice that, similar as for the memory lower bound from Section 3.5.2, for the exact lower bound of 2^m we use the equivalence relation $\sim^A$ on actions as well. However, similar as there, we could implement the actions into the positions of the game graph, see also Section 2.1.2, and obtain a $2^{\sqrt{|V|/c}}$ lower bound for the case where all actions of player 0 are indistinguishable for player 1.

Proposition 4.8 demonstrates that the powerset construction is not the right method to exploit simplicity of the game graphs if partial information is unbounded. The natural question is whether there is any possibility to do so. Again, we will answer this question negatively even for very simple graphs.

First, let us note that on *directed trees*, the strategy problem for parity games with imperfect information can actually be solved more easily. More precisely, we consider trees with a unique root and all edges directed away from this root. Notice that on directed trees, parity conditions are just reachability conditions. However, on arbitrary graphs, the strategy problem for reachability games with imperfect information is already EXPTIME-hard.

Now, if we perform the powerset construction on such a tree we again obtain a tree where the vertices on each level form a partition of the vertices on the same level of the original tree. This new tree can be computed in polynomial time in a top-down fashion and has at most as many positions as the original tree. So the strategy problem for reachability (and hence, parity) games, played on directed trees can be solved in polynomial time. We will see that as soon as we consider arbitrary acyclic graphs which are not necessary trees, the problem is already PSPACE-hard.

4.2.1 Lower Bounds on the Computational Complexity

To prove lower bounds on the computational complexity of strategy problems for parity games with imperfect information on graphs of bounded complexity we use alternating Turing machines. Those machines were introduced by Chandra, Kozen and Stockmeyer [49, 48] and have quickly become standard in the theory of computational complexity, see for example [161, 217]. In [49, 48] the relation between alternating and standard complexity classes has been studied. The following result is important for us.

Theorem 4.9. *[49, 48]* APSPACE = EXPTIME *and* APTIME = PSPACE.

We also use the following well-known result that allows us to restrict to alternating Turing machines with a *single tape*. For a proof see for example [217].

Proposition 4.10.

(1) For all $L \in$ ASPACE*$(S(n))$ with $S(n) \geq n$ there is an alternating Turing machine with a single tape and space bound $S(n)$ which accepts L.*

(2) For all $L \in$ ATIME*$(T(n))$ with $T(n) \geq n$ there is an alternating Turing machine with a single tape and time bound $O(T^2(n))$ which accepts L.*

Now let us fix our notation for such machines. An *alternating Turing machine* has the form

$$M = (\Sigma, \Gamma, Q, q_{\text{in}}, \delta)$$

where Σ is the input alphabet and $\Gamma \supseteq \Sigma$ is the tape alphabet. Moreover, Q is the finite set of states, $q_{\text{in}} \in Q$ is the initial state and $\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{l,r\}}$ is the transition function. So, when the machine is in some state $q \in Q$ and reads symbol $a \in \Gamma$ on the tape, then for any $(p, b, \iota) \in \delta(q, a)$, the computation may proceed as follows: change the state to $p \in Q$, write $b \in \Gamma$ to the tape (thereby deleting a) and move the head into direction $\iota \in \{l(\text{eft}), r(\text{ight})\}$

So far, M is just a nondeterministic Turing machine. An alternating Turing machine has as distinguishing feature a partition $Q = Q_\forall \cup Q_\exists \cup Q_+ \cup Q_-$ of the set of states into *universal*, *existential*, *accepting* and *rejecting* states. Given an input $u \in \Sigma^*$, we obtain a computation tree $T_{C_{\text{in}}(u)}$ whose nodes are configurations of M on u. The root of $T_{C_{\text{in}}(u)}$ is the initial configuration $C_{\text{in}}(u) = (q_{\text{in}}, u, 0)$ of M on input u and the successors of a node C in $T_{C_{\text{in}}(u)}$ are the successor configurations $C \vdash C'$ of C according to δ.

We define the semantics of M in terms of a game, played on $T_{C_{\text{in}}(u)}$. There are two players, the universal player 0, also called $\forall$, and the existential player 1, also called $\exists$. Universal configurations are the positions of player $\forall$ and existential configurations belong to player $\exists$. Accepting and rejecting configurations do not have any successors, and the goal of player $\exists$ is to reach an accepting configuration. The machine M accepts input u if player $\exists$ has a winning strategy for this game from the initial position $C_{\text{in}}(u)$. As usual, the language recognized by M is $L(M) = \{u \in \Sigma^* \mid M \text{ accepts } u\}$.

The main result of this section is that the strategy problem for reachability games with imperfect information is EXPTIME-hard on graphs of DAG-width at most three. Our main technical tools for the proof are Theorem 4.9 and Proposition 4.10 which demonstrate that for any language $L \in$ EXPTIME, there is an alternating Turing machine M with a single tape and space bound m^s for some $s \geq 1$ that recognizes L. We use this to show that L can be reduced to the strategy problem for reachability games with imperfect information in polynomial time, using graphs of DAG-width at most three.

So consider an arbitrary $L \in$ EXPTIME and let M be an alternating Turing machine M with a single tape and space bound m^s for some $s \geq 1$ that recognizes L. The main idea for the proof is the same as for the original proof by Reif [182] for the general case without any restrictions on the structural complexity of the game graphs, see also [52, 172]. We describe this idea before we provide a full technical proof of our result. Along the way we also describe some further technical tools. We first restrict to the case where M is deterministic, that is, each configuration of M has exactly one successor configuration. We will explain how the construction can be adapted to the general case after we have completed the technical proof for deterministic machines.

Let $\Theta = \Gamma \cup Q \times \Gamma$. Then a configuration of M can be described as a word

$$C = w_0 \dots w_{j-1}(q, w_j)w_{j+1} \dots w_{l-1} \in \Theta^*$$

where $w_0 \dots w_{l-1}$ is the tape contents, q is the current state and j is the position of the head on the tape. Since M has space bound m^s with $s \geq 1$, we can assume w.l.o.g. that $|C| = m^s$ for all configurations C of M on inputs of length m. (We use a representation of the tape which has a priori the maximum length occurring during a computation on inputs of length m.)

A very important observation for the proof is that the successor relation between configurations of M can be verified locally. More precisely, for a configuration C of M and some $0 \leq j \leq m^s - 1$ the symbol number j of the successor configuration $C' = \text{Next}(C)$ of C only depends on the symbols number $j-1$, j and $j+1$ of C. So there is a function $\kappa : (\Theta_\perp)^3 \to \Theta$ such that for any configuration C of M and any $0 \leq j \leq m^s - 1$ we have

$$C'_j = \kappa(C_{j-1}C_jC_{j+1})$$

where $\Theta_\perp = \Theta \cup \{\perp\}$ and we set $C_{j-1} = \perp$ if $j = 0$ and $C_{j+1} = \perp$ if $j = m^s - 1$.

Now consider any input $u = u_0 \dots u_{m-1} \in \Sigma^*$ of length m. We describe a reachability game with imperfect information such that player 1 has a winning strategy if, and only if, $u \in L$. The task of player 1 in this game is to construct the (unique) run of M on u symbol by symbol, that is, in each step, player 1 chooses a symbol from Θ and whenever he has completed a word $C \in \Theta^*$ of length $|u|^s$, the next configuration starts. The initial configuration will be implemented into the game graph entirely and will be skipped in the construction.

On the other hand, player 0 tries to detect an error in the construction that player 1 performs. For this he may, at any point during the play, jot down the current position $j \in \{0, \dots, |u|^s - 1\}$ within the present configuration as well as the symbols number $j-1$,j and $j+1$ chosen by player 1. If he does so we say, figuratively speaking, that player 0 turns red (at position j of the present configuration). Then, while player 1 constructs the next configuration, player 0 may check the symbol number j chosen by player 1 to be correct according to the symbols which he has memorized, using the function κ. If this symbol proves incorrect, player 1 loses. Moreover, as soon as player 1 writes a terminal configuration that is accepting, he wins.

So far we did not use the possibility to impose partial information on player 1 and, indeed, the construction is not correct in the stated form: If player 1 may notice when player 0 turns red, then he can cheat by adapting his construction locally (customized to the particular point when player 0 turns red) and keep the actual flaw in the construction away from player 0. So we have to hide the information when player 0 turns red from player 1. We do this by defining partial information on the game graph: The part of a position in which player 0 stores the position j and the three corresponding symbols chosen by player 1 is invisible for player 1. So player 0 makes his notes in private and hence, player 1 will be completely unaware when player 0 turns red.

Now the reduction is correct and can also be computed in polynomial time [182]. Our task is now to show that this game can be played on simple graphs. For this, an important refinement of the idea described above is to restrict player 0 so that he may turn red only *once* and, if he turns red but fails to detect and error in the next configuration that player 1 constructs, he *loses*. To see that this is sufficient, notice that player 0 knows the strategy that player 1 is using. Clearly, whenever player 1 uses a strategy that indeed writes down a run of M, then player 0 will not be able to detect a flaw, no matter how often he turns red. However, if player 0 uses a strategy σ that does not construct a run of M, player 0 can deduce the first moment when the construction fails from σ and turn red just in time to detect the failure.[22]

The idea is now that the construction of a configuration can be performed on a DAG D with a single root and with $2 \cdot m^s$ levels where the players alternate from level to level. To enable player 1 to construct a whole sequence of configurations, we have to add also back-edges from the last level of the DAG to the unique root. The game takes place on this DAG as long as player 0 has not yet decided to turn red. If player 0 turns red at position l with corresponding symbols $\overline{\theta} = \theta_1\theta_2\theta_3$ then, without player 1 noticing it, the game switches to a disjoint copy $D_l^{\overline{\theta}}$ of D where l and $\overline{\theta}$ are fixed. (So we have $m^s \cdot |\Theta|^3$ copies of D.) Since player 0 may turn red only once, there need not be any edges from $D_l^{\overline{\theta}}$ to nodes outside this DAG. (Which makes this restriction significant: If the DAGs were mutually connected, the graph complexity would be higher.) Moreover, since player 0 loses if he turns red spuriously, at level $2l$ of $D_l^{\overline{\theta}}$ a winner will be declared, so we shall delete all edges from level $2l$ to level $2l{+}1$ in $D_l^{\overline{\theta}}$. It is not hard to see that the whole game graph has DAG-width at most two. Notice that we are still in the case where M is deterministic. We will see, however, that handling arbitrary alternating machines increases the graph complexity by at most one.

Theorem 4.11. *The strategy problem for reachability games with imperfect information is* Exptime*-hard on graphs of DAG-width at most three.*

Proof. Let $L \subseteq \Sigma^*$ be any language in Exptime and let $M = (\Sigma, \Gamma, Q, q_{\text{in}}, \delta)$ be an alternating Turing machine with a single tape and space bound m^s for some $s \geq 1$ that recognizes L. First we assume that M is deterministic. Let $\Theta = \Gamma \cup Q \times \Gamma$, $\Theta_\perp = \Theta \cup \{\perp\}$ and let $\kappa : (\Theta_\perp)^3 \to \Theta$ be a function that can be used to check the successor relation between configurations of M locally.

Now consider any $u = u_0 \dots u_{m-1} \in \Sigma^*$. We construct a game graph $\mathcal{G}_u = (V, \delta, \sim^V, \sim^A)$ with partial information, using the idea described above, such that player 1 has a winning strategy if from initial position $v_0 \in V$ if, and only if, $u \in L$. We start with a formal description of the game graph and we demonstrate that the reduction is correct. Then we have a close look at the structural properties of the game graph and, finally, we describe how the construction can be adapted to the case where M is not necessarily deterministic.

Set of Positions

$$V = \{v_0\} \cup \{0,1\} \times \Theta_\perp \times [m^s] \times (Q \cup \{\perp\}) \times (\{\text{👁}\} \cup [m^s]) \times \Theta_\perp^3,$$

So a position is either the initial position v_0 or has the form $(i, \theta, j, q, l, \theta_1\theta_2\theta_3)$ where $i \in \{0,1\}$ the player whose turn it is and $\theta \in \Theta$ is the symbol written by player 1 in the last

[22]Intuitively, even though he knows that player 0 will be able to turn red only once, player 1 can still not find a good time to cheat: he is too intimidated by the power of player 0 to turn red secretly at any time. (We also call this the threat of being observed without observing it.)

step. Moreover, $j \in [m^s]$ is the current position within the present configuration and, if $q \neq \perp$ then $q \in Q$ is the last state that has been chosen by player 1 during the construction of the present configuration. Finally, the last two entries are used by player 0 to jot down the relevant information whenever he decides to turn red (so these last two components of a position will be hidden from player 1): If $l \neq 👁$ then $l \in [m^s]$ is the position at which player 0 has decided to turn red and $\theta_1\theta_2\theta_3 \in \Theta^3$ are the corresponding symbols number $l{-}1$,l and $l{+}1$ chosen by player 1 at this point. Recall that we let symbol number $l{-}1$ of a configuration be $\perp$ if $l = 0$ and analogously for symbol number $l{+}1$, if $l = m^s{-}1$. The initial position v_0 belongs to player 0. We now describe the moves that are available in the game.

Initial Moves of Player 0

(1) $v_0 \xrightarrow{\quad ☞ \quad} (1, \perp, 0, \perp, 👁, \perp\perp\perp)$

(2) $v_0 \xrightarrow{(l, \theta_1\theta_2\theta_3)} (1, \perp, 0, \perp, l, \theta_1\theta_2\theta_3)$

for $l \in [m^s]$, where $\theta_1\theta_2\theta_3$ are symbols $l{-}1$, l and $l{+}1$ of $C_{\text{in}}(u)$

As we have mentioned, $C_{\text{in}}(u)$ will not be constructed by player 1 but is implemented into the game graph. So if player 0 wants to check the first configuration that player 1 constructs (the intended second configuration in the run of M on u) then he notes down position l within $C_{\text{in}}(u)$ and the corresponding symbols $\theta_1\theta_2\theta_2$ without having seen them. The moves (2) serve this purpose. If player 0 does not want to turn red at the beginning then he uses move (1), going to position $(1, \perp, 0, \perp, 👁, \perp^3)$ where no information yet memorized yet.

Moves of Player 1

(3) $(0, \theta, j, q, 👁, \theta_1\theta_2\theta) \xrightarrow{\quad \theta' \quad} (1, \theta', j, q', 👁, \theta_2\theta\theta')$

for $\theta' \in \Theta$, where $q' = \text{Pr}_Q(\theta')$ if $\theta' \in Q \times \Gamma$ and $q' = q$ else.

(4) $(0, \theta, j, q, l, \theta_1\theta_2\theta_3) \xrightarrow{\quad \theta' \quad} (1, \theta', j, q', l, \theta_1\theta_2\theta_3)$

for $l \in [m^s]$ and $\theta' \in \Theta$ where $q' = \text{Pr}_Q(\theta')$ if $\theta' \in Q \times \Gamma$ and $q' = q$ else.

Moves (3) take place as long player 0 has not turned red yet: Then the sequence of symbols chosen by player 1 which is currently stored (as $\theta_1\theta_2\theta_3$ with $\theta_3 = \theta$) is updated according to the latest symbol θ'. However, if player 0 has turned red at position $l \in [m^s]$ then the information $\theta_1\theta_2\theta_3$ is noted down and does not change anymore. (Notice, however, that player 1 will not be able to distinguish these two situation since he sees only $(0, \theta, j, q)$.)

Moves of Player 0

(5) $(0, \theta, j, q, 👁, \theta_1\theta_2\theta) \xrightarrow{\quad ☞ \quad} (1, \theta, j{+}1, q, 👁, \theta_1\theta_2\theta)$

for $j < m^s{-}1$

$(0, \theta, m^s{-}1, q, 👁, \theta_1\theta_2\theta) \xrightarrow{\quad ☞ \quad} (1, \perp, 0, \perp, 👁, \perp\perp\perp)$

(6) $(0, \theta, j, q, 👁, \theta_1\theta_2\theta) \xrightarrow{\quad ✍ \quad} (1, \theta, j{+}1, q, l, \theta_1\theta_2\theta)$

for $1 \leq j < m^s{-}1$ where $l = j{-}1$

$(0, \theta, m^s{-}1, q, 👁, \theta_1\theta_2\theta) \xrightarrow{\quad ✍ \quad} (1, \perp, 0, \perp, m^s{-}2, \theta_1\theta_2\theta)$

$(0, \theta, m^s{-}1, q, 👁, \theta_1\theta_2\theta) \xrightarrow{\quad ✍ \quad} (1, \perp, 0, \perp, m^s{-}1, \theta_2\theta\perp)$

(7) $(0, \theta, j, q, l, \theta_1\theta_2\theta_3) \xrightarrow{\quad ☞ \quad} (1, \theta, j{+}1, q, l, \theta_1\theta_2\theta_3)$

for $l \in [m^s]$ and $j < l$

$$(0,\theta,m^s{-}1,q,l,\theta_1\theta_2\theta_3) \quad \xrightarrow{\text{☞}} \quad (1,\bot,0,\bot,l,\theta_1\theta_2\theta_3)$$

for $l < m^s{-}1$

Moves (5) - (6) are performed by player 0 in a situation where he has not turned red yet and, when player 0 performs move (5), then he has decided not to turn red in the current move as well. In performing move (6), on the other hand, player 0 turns red. There, we are in a situation where player 1 has written symbol j of the current configuration and has turned control over to player 0 so that he can decide whether to turn red or not.

If he does that means he wants to check symbol $l = j{-}1$ in the next configuration. (Notice that in order to check symbol $j{-}1$ in the next configuration, player 0 needs symbol j of the present configuration!) So he jots down $l = j{-}1$ and the last three symbols $\theta_1\theta_2\theta$, provided by player 1.

The second extra move in case (6) is needed to cover the special case where player 0 wants to check the last symbol in the next configuration. In this case, there is no symbol m^s, so player 0 jots down $l = m^s{-}1$ and adds an extra $\bot$ to the sequence of symbols that he memorizes.

Finally, in a situation where he has already turned red, player 0 can make moves (7), similar to (5), but only if $l \neq j$: At a position $(0,\theta,j,q,l,\theta_1\theta_2\theta_3)$ with $l = j$, a winner will be declared and, since player 1 has a reachability objective, we have to make sure that if player 1 is the loser, he will not get a second chance to reach a position that is winning for him. Moreover, when player 0 has turned red and he initializes the next configuration by moving to a position with $j = 0$, the symbols $\theta_1\theta_2\theta_3$ will, of course, not be reset.

Partial Information

$$(i,\theta,j,q,l,\theta_1\theta_2\theta_3) \;\sim^V\; (i,\theta,j,q,l',\theta_1'\theta_2'\theta_3')$$

So, as we have mentioned, player 1 observes the first four components of a position but is completely oblivious of the last two entries. So he can see whose turn it is, the last symbol he has chosen, the current position within the present configuration and also the last state that he has chosen, but he does not notice when player 0 turns red and memorizes the information l and $\theta_1\theta_2\theta_3$. Moreover, player 1 can distinguish any two of his own actions but all actions of player 0 are indistinguishable for him.

Reachability Objective

(A) $(0,\theta,m^s{-}1,q,\text{👁},\theta_1\theta_2\theta)$ with $q \in Q_+$

(B) $(0,\theta,j,q,l,\theta_1\theta_2\theta_3)$ with $l = j$ and $\theta = \kappa(\theta_1\theta_2\theta_3)$.

If player 1 reaches a position (A) then he has constructed an accepting configuration without being controlled, that means, player 0 has not turned red. In he reaches a position (B) then player 0 has turned red but has failed to detect a flaw in the construction of player 1. Notice that, since player 1 does not explicitly construct $C_{\text{in}}(u)$, to make this sound we have to assume, w.l.o.g., that q_{in} is neither accepting not rejecting.

Correctness The correctness of the reduction can be shown with similar arguments as in the proof of Theorem 3.20: Clearly, if $u \in L$ then player 1 has a winning strategy: He just plays the unique run of M on u, eventually reaching an accepting state or seeing player 0 embarrassing himself by trying to detect an error in the flawless construction.

Now let, conversely, σ be a winning strategy for player 1 for $\mathcal{G}_u$ from v_0 and consider the unique play π that results from σ when player 0 plays the action ☞ in each step. This play gives rise to a unique (infinite) sequence $\overline{C} = C_1C_2\ldots$ of words $C_j \in \Theta^{m^s}$. Since σ is a winning strategy, there must be some $t \in \mathbb{N}$ such that C_t is an accepting configuration. Let t be minimal with this property, let $C_0 := C_{\text{in}}(u)$ and assume that $C_0 \ldots C_t$ is *not* the unique run of M on u. (Remind that we have assumed that C_0 is neither accepting nor rejecting.) Then there is some $0 < s \leq t$ such that $C_0 \vdash \ldots \vdash C_{s-1}$ but $C_{s-1} \not\vdash C_s$. Let $l \in \mathbb{N}$ be minimal such that the symbol number l of C_s is incorrect, that is, $(C_s)_j = C_j$ for $j < l$ but $(C_s)_l \neq C_l$, where $C_{s-1} \vdash C$.

Now consider the play ρ obtained from π as follows. At position $j = l{+}1$ during the construction of C_{s-1} (or directly at the initial position v_0, if $s = 1$), player 0 replaces his action ☞ by ✍ (or by $(l, (C_0)_{l-1}(C_0)_l(C_0)_{l+1})$, respectively). Except for this change, player 0 and player 1 choose exactly the same actions in ρ as they do in π. Therefore, by construction, we have $\rho \sim^* \pi$, so the play ρ is consistent with σ as well and since σ is a winning strategy, ρ is won by player 1. However, by definition of $\mathcal{G}_u$, this play ρ will end up in a position

$$(0, (C_s)_l, j, q, l, (C_{s-1})_{l-1}(C_{s-1})_l(C_{s-1})_{l+1})$$

with $j = l$. Since $(C_s)_l \neq C_l$, player 1 loses at this position – a contradiction. So $C_0 \ldots C_t$ is the unique run of M on u and since C_t is accepting, $u \in L(M)$.

Structural Complexity Let G_u denote the graph structure of $\mathcal{G}_u$. We show that $\mathrm{dw}(G_u) \leq 2$. Notice that we are still in the case where M is deterministic.

Let $D_{\text{👁}}$ denote the subgraph of G_u induced by the vertices of the form $(i, \theta, j, q, \text{👁}, \overline{\theta})$. Then $D_{\text{👁}}$ is an augmented DAG of height $2 \cdot m^s$ with edges from any non-bottom level only to the level below and with additional back-edges from the bottom-level $2m^s$ to the root $r(\text{👁}) = (1, \bot, 0, \bot, \text{👁}, \bot\bot\bot)$. Moreover, for any fixed $(l, \overline{\theta}) \in [m^s] \times \Theta_\bot^3$, let $D_l^{\overline{\theta}}$ denote the subgraph of G_u induced by the positions of the form $(i, \theta, j, q, l, \overline{\theta})$. Such a graph is an augmented DAG of the same shape, that means, it has height $2 \cdot m^s$, edges from any non-bottom level only to the level below and additional back-edges from the bottom level to the root $r(l, \overline{\theta}) = (1, \bot, 0, \bot, l, \overline{\theta})$. Furthermore, the edges from level $2(l{+}1)$ to level $2(l{+}1){+}1$ of $D_l^{\overline{\theta}}$ are deleted (if $l = m^s{-}1$, then the back-edges to the root are deleted). It is easy to see that the union of $D_{\text{👁}}$ together with v_0 and the subgraphs $D_l^{\overline{\theta}}$ induces the whole graph G_u.

To see how these constituent parts of G_u are connected among each other, let us review how the game is played on G_u. From the initial position v_0, the game may proceed to the root $r(\text{👁})$ or to some root $r(l, \overline{\theta})$ for appropriate $\overline{\theta}$ (determined by $C_{\text{in}}(u)$). This entails exactly $m^s{+}1$ outgoing edges from v_0, but v_0 has no ingoing edges. As long as player 0 has not yet decided to turn red, the game takes place in $D_{\text{👁}}$. Notice that the only vertex in G_u that is outside $D_{\text{👁}}$ and from which there is an edge leading to $D_{\text{👁}}$ is the initial position v_0.

Now, if player 0 turns red in $D_{\text{👁}}$ at level $2(l{+}1)$ with $1 \leq l \leq m^s{-}2$, and $\overline{\theta} \in \Theta_\bot^3$ are the symbols that are currently stored, then the game proceeds to $D_{l-1}^{\overline{\theta}}$. So there are directed edges from level $2(l{+}1)$ of $D_{\text{👁}}$ to level $2(l{+}1){+}1$ of $D_{l-1}^{\overline{\theta}}$. Analogously, there are directed edges from level $2m^s$ of $D_{\text{👁}}$ to the root of $D_{m^s-2}^{\overline{\theta}}$ as well as the root of $D_{m^s-1}^{\overline{\theta}}$ for any $\overline{\theta}$. On the other hand, a subgraph $D_l^{\overline{\theta}}$ does not have any edges to vertices outside this subgraph.

It is now obvious how two cops can capture a directed robber on G_u: Occupy v_0, then free v_0. Since v_0 has no ingoing edges, the robber is expelled from there. Now occupy the root $r(\text{👁})$ of $D_{\text{👁}}$. If the robber is in $D_{\text{👁}}$, chase him down this subgraph with the second cop. (Otherwise, the robber is in some $D_l^{\overline{\theta}}$, where we will capture him in the next step.) Eventually, the robber will escape to some subgraph $D_l^{\overline{\theta}}$. There, a single cop is sufficient to chase the robber to level $2(l{+}1)$ where he is captured because at this level, there are only terminal vertices.

General Case To adapt the construction to the general case where M is not necessarily deterministic, the task is to handle multiple successor configurations and the fact that control over the successor relation is distributed among two players, $\forall$ and $\exists$. An important tool is again the locality of the successor relation, that means, for a configuration C of M, a position $0 \leq j \leq m^s{-}1$ and $\nu \in [d]$ symbol j of the ν-th successor configuration $C' = \mathrm{Next}_\nu(C)$ of C only depends on symbols $j{-}1$, j and $j{+}1$ of C. (We assume w.l.o.g. that each configuration of M

that is neither accepting nor rejecting has exactly d successor configurations for an appropriate $d \in \mathbb{N}$.) So for each $\nu \in [d]$ there is a function κ_ν that can be used to check the ν-th successor relation of M locally.

The idea is again the same as in [182]: In the game described above, player 0 simulates player $\forall$ and player 1 simulates player $\exists$. So before player 1 starts to construct a configuration, depending on whether the last configuration was existential or universal, either player 0 or player 1 determine a $\nu \in [d]$ meaning that player 1 must construct the ν-th successor configuration of the previous one. This information will be of course disclosed to player 1 and the correctness of the construction must then also be checked using κ_ν.

We implement this into the game graph $\mathcal{G}_u$ as follows. Let $D_{\circledast}$ and $D_l^{\overline{\theta}}$ for $(l, \overline{\theta}) \in [m^s] \times \Theta^3$ be the subgraphs of the game graph as defined above. Of each such subgraph we take d disjoint copies $D_{\circledast,\nu}$ and $D_{l,\nu}^{\overline{\theta}}$, $\nu \in [d]$, and we add distinct positions $\exists, \forall$ and $\exists_l^{\overline{\theta}}, \forall_l^{\overline{\theta}}$ to the game graph, where any two $\exists$-positions as well as any two $\forall$-positions are indistinguishable for player 1.

From $\forall$, player 0 can go to the root $r_\nu(\circledast) = (1, \nu, \perp, 0, \perp, \circledast, \perp^3)$ of $D_{\circledast,\nu}$ for any $\nu \in [d]$ and from $\exists$, player 1 can go to any such root. Analogously, from $\forall_l^{\overline{\theta}}$, player 0 can go to the root $r_\nu(l, \overline{\theta}) = (1, \nu, \perp, 0, \perp, l, \overline{\theta})$ of $D_{l,\nu}^{\overline{\theta}}$ for any $\nu \in [d]$ and from $\exists_l^{\overline{\theta}}$, player 1 can go to any such root. Moreover, if the initial configuration is existential, then all edges from v_0 are redirected to the corresponding $\exists$-nodes and if it is universal, these edges are redirected to the corresponding $\forall$-nodes. Finally, any back-edge from a node $(0, \nu, \theta, m^s{-}1, q, \circledast, \overline{\theta})$ at the bottom level of $D_{\circledast,\nu}$ to the root is redirected to $\exists$ if q is existential and to $\forall$, if q is universal. (Edges from positions with $q \in Q_+ \cup Q_-$ can be redirected to either $\exists$ or $\forall$, or even be deleted.) Analogously for the subgraphs $D_{l,\nu}^{\overline{\theta}}$.

Finally, the reachability condition is modified so that in a subgraph $D_{l,\nu}^{\overline{\theta}}$, we check $\theta = \kappa_\nu(\overline{\theta})$ with respect to the particular ν. (Recall that ν is observable for player 1.) Conceptually, it is not hard to see that the reduction is still correct, having the arguments for the deterministic case in mind. Moreover, it is also still computable in polynomial time. Structurally, the game graph is not much more complicated either: Instead of occupying the root $r(\circledast)$ of $D_{\circledast}$ with a single cop, we now occupy the vertices $\exists$ and $\forall$ at the same time using two cops, which has the same impact on the scope of the robber. For chasing the robber, a single additional cop is still sufficient, so we have $\mathrm{dw}(\mathcal{G}_u) \leq 3$. $\square$

Remark. To avoid any confusion about the graph structure G_u, notice that there is a bunch of positions in G_u that are not reachable from v_0 (in fact, more than $|V|/2$ positions are not reachable from v_0). For instance, positions of the form $(1, \theta, j, q, \circledast, \overline{\theta})$ with $j \leq 2$ and $\perp \notin \overline{\theta}$ are not reachable from v_0 because as long as player 0 has not turned red yet, the construction of a configuration always starts in $(1, \perp, 0, \perp, \circledast, \perp^3)$. Analogously, any position of the form $(i, \theta, j, q, l, \overline{\theta})$ with $l = 0$ and $\perp \notin \overline{\theta}$ is not reachable from v_0 and positions of form $(i, \theta, j, q, l, \perp^3)$ are also not reachable. Even more, any position $(0, \theta, j, q, \xi, \overline{\theta})$ where j is *even* is not reachable. So looking, for example, at $D_{\circledast}$, this graph looks in fact like a rectangle, rather than a pyramid, has several roots and contains, on each level, positions of both players. However, since the game on $\mathcal{G}_u$ is played from the fixed initial position v_0, all positions that are not reachable from v_0 are irrelevant. So we can simply ignore them and identify G_u with the subgraph induced by those positions that are reachable from v_0.

Theorem 4.11 demonstrates that the intrinsic complexity caused by partial information is very high, even for quite simple graphs. Next we show that even on acyclic graphs, the strategy problem for reachability games with imperfect information is still PSPACE-hard which substantiates this statement. Remind that we presume a partition of the leaves into positions where player 0 has won and those, where player 1 has won. Since we also prove PSPACE-membership, in this case we can do at least slightly better than for arbitrary graphs. [23]

Theorem 4.12. *The strategy problem for reachability games with imperfect information is* PSPACE*-hard on acyclic graphs.*

Proof. We start with PSPACE-membership. Let $\mathcal{G} = (V, \delta, \sim^V, \sim^A)$ be an acyclic game graph with partial information, let $R \subseteq V$ be a reachability objective on $\mathcal{G}$ and let $v_0 \in V$. Consider the powerset graph $2^{\mathcal{G}} = (\overline{V}, \overline{\Delta})$ and let $\overline{u}_0 \to \overline{u}_1 \to \ldots \to \overline{u}_l$ be any path in $2^{\mathcal{G}}$. According to Proposition 3.22, there is a path $u_0 \to u_1 \to \ldots \to u_l$ in $\mathcal{G}$ such that $u_j \in \overline{u}_j$ for all j, so we have $l \leq |V|$. In particular, $2^{\mathcal{G}}$ is acyclic.

The strategy problem for $(\mathcal{G}, R, v_0)$ can be solved by an alternating algorithm, performing an on-the-fly powerset construction, starting from $\{v_0\}$: Given a position $\overline{v} \in \overline{V}$, we proceed to some successor $\overline{u}$ of $\overline{v}$ where $\overline{u}$ is chosen universally, if $\overline{v}$ is a position of player 0 and $\overline{u}$ is chosen existentially, if $\overline{v}$ is a position of player 1. If the computation arrives at a position $\overline{v}$ with $\overline{v} \subseteq R$, it accepts. If the computation arrives at a terminal position $\overline{v}$, then it stops and accepts if $\overline{v} \subseteq R$, otherwise it rejects. The construction of a successor $\overline{u}$ of $\overline{v}$ can obviously be done in polynomial time. Moreover, as we have shown above, the number of steps until a terminal position is reached is at most $|V|$, so the algorithm has a polynomial time bound. Using Theorem 4.9, we get PSPACE-membership.

Let, conversely, $L \subseteq \Sigma^*$ be any language in PSPACE. According to Theorem 4.9 and Proposition 4.10, there is an alternating Turing machine $M = (\Sigma, \Gamma, Q, q_{\text{in}}, \delta)$ with a single tape and time bound m^s for some $s \in \mathbb{N}$ that recognizes L. To keep the notation simpler, we restrict to the case where M is deterministic. The changes that are necessary to handle the general case are exactly the same as in the proof of Theorem 4.11. Since M has time bound m^s and only one tape, M has space bound m^s as well. Hence, we can describe configurations of M in the same way as in the proof of Theorem 4.11. Let Θ, $\Theta_\perp$ and $\kappa : (\Theta_\perp)^3 \to \Theta$ be as there.

Now consider any $u = u_0 \ldots u_{m-1} \in \Sigma^*$. Conceptually, we carry out the same construction as in the proof of Theorem 4.11. The essential technical difference is that from positions $(0, \theta, m^s{-}1, q, \circledast, \overline{\theta})$ at the bottom-level of $D_{\circledast}$, instead of having back-edges to the root $r(\circledast)$ we have edges to the root of a *new copy* of $D_{\circledast}$. And analogously for the subgraphs $D_l^{\overline{\theta}}$ for $(l, \overline{\theta}) \in [m^s] \times \Theta_\perp^3$. In this way, the game graph will be acyclic by definition!

Since M has time bound m^s, if some input u is accepted by M, then player 1 can prove this by constructing at most m^s configurations. So we can bound the number of these copies by m^s, that means, we have m^s copies $D_{\circledast,\mu}$ and $D_{l,\mu}^{\overline{\theta}}$, $\mu \in [m^s]$ of the subgraphs $D_{\circledast}$ and $D_{l,\mu}^{\overline{\theta}}$. The edges from v_0 go to the first copies $D_{\circledast,0}$ and $D_{l,0}^{\overline{\theta}}$ as before. From positions at the bottom level of $D_{\circledast,\mu}$ we have edges to the root $r(\circledast, \mu{+}1)$ of $D_{\circledast,\mu+1}$ and analogously for the graphs $D_{l,\mu}^{\overline{\theta}}$. Moreover, the edges from level $2(l{+}1)$ of $D_{\circledast,\mu}$ go to level $2(l{+}1){+}1$ of the subgraphs $D_{\circledast,\mu}^{\overline{\theta}}$ as before, for any $\mu \in [m^s]$.

So in total we have m^s copies D_μ, $\mu \in [m^s]$ of the whole subgraph D induced by $D_{\circledast}$ and the subgraphs $D_l^{\overline{\theta}}$, where we start at v_0 from which we can go to the first copy D_0 just as we could go to D before. The overall size of the game graph is still polynomial in $|u|$. A winner will now be declared only at leaves of the DAG, that is, at terminal positions. (Notice that such positions do not only occur in the m^s-th copy of D, but in any subgraph $D_{l,\mu}^{\overline{\theta}}$.) At such positions, player 1 has won if the position belongs to the reachability set R as defined in the proof of Theorem 4.11, at all other terminal positions, player 0 has won. The correctness of the construction is immediate, since winning strategies carry over between the game constructed in the proof of Theorem 4.11 and the timeout-version constructed here in the obvious way. As we have mentioned, the game graph is acyclic which concludes the proof. ☐

[23] This result has been proved independently in [51]. There, for the PSPACE-hardness, a direct reduction from QBF is given while we prove this by adapting the proof of Theorem 4.11.

Notice that this does not completely settle the complexity of the strategy problem for reachability games with imperfect information on graphs of DAG-width at most one, because such graphs may contain arbitrary selfloops. Obviously, having selfloops at the leaves of an acyclic graph does not hurt the construction as given in the proof. Moreover, it is easy to show that the powerset graph of any graph of DAG-width at most one does not have cycles containing more than one vertex either (using again Proposition 3.22). On the other hand, the length of a path is not necessarily bounded by $|V|$, even if selfloops are not repeated: Consider a game graph $\mathcal{G} = ([n], \delta, \sim^V, \sim^A)$ with $A = \{\leftarrow, \rightarrow, \downarrow, \circlearrowleft\}$ where

- $\delta(0, \leftarrow) = 1, \ \ \delta(0, \rightarrow) = n$ and $\delta(0, \circlearrowleft) = 0$
- $\delta(j, \downarrow) = j + 1$ for $j = 1, \ldots, n-2$ and $\delta(n, \downarrow) = 1$
- $\leftarrow \ \sim^A \ \circlearrowleft$ and $0 \ \sim^V \ 1 \ \sim^V \ \ldots \ \sim^V \ n-1$

In $2^{\mathcal{G}}$ we have a path

$$\{0\} \rightarrow \{0,1\} \rightarrow \ldots \rightarrow \{0,1,\ldots,n{-}1\} \rightarrow \{n\} \rightarrow \{1\} \rightarrow \{2\} \rightarrow \ldots \rightarrow \{n\}$$

of length $2n$. So the argumentation as in the proof of Theorem 4.12 can not be applied directly.

Moreover, since Theorem 4.11 yields EXPTIME-hardness only for the strategy problem for reachability games with imperfect information on graphs of DAG-width at most three, the same problem for graphs of DAG-width at most two is still open. So it is not clear whether imperfect information causes an exponential blow up of the time complexity of the strategy problem for reachability games on game graphs of DAG-width at most three. We prove this for a slightly more complicated type of winning condition.

Let $\mathcal{G} = (V, \delta, \sim^V, \sim^A)$ be a game graph with partial information. We consider a winning condition that we call *sequence forcing* condition and which generalizes reachability conditions: A sequence forcing condition for $\mathcal{G}$ is given by a coloring $\mathrm{col} : V \to [c]$ for some $c \in \mathbb{N}$ and a set $S \subseteq [c]^l$ of sequences of length l for some $l \in \mathbb{N}$. An infinite play π in $\mathcal{G}$ is won by player 1 if there is a sequence of positions $v_1 \cdot v_l$ that occurs in π and satisfies $\mathrm{col}(v_1) \ldots \mathrm{col}(v_l) \in S$. As before, the winner of a finite play (ending in a terminal position) is determined by the corresponding partition of the terminal positions.

Sequence forcing games are determined, but obviously positional winning strategies do not suffice in general: To implement a winning strategy for player 1, we need a memory that stores the last l positions that have occurred. If l is fixed, the size of this memory structure is polynomial in the size of $\mathcal{G}$. On the game graph which is obtained by taking the product of the memory structure and the game graph, the sequence forcing condition is then just a simple reachability condition, so, for fixed l, sequence forcing games can be reduced to reachability games in polynomial time. In particular the strategy problem for sequence forcing games with full information can be solved in polynomial time for fixed l. On the other hand, the next theorem shows that the strategy problem for sequence forcing games with imperfect information is EXPTIME-hard, even for $l = 3$.

Theorem 4.13. *The strategy problem for sequence forcing games with imperfect information and $l = 3$ is* EXPTIME*-hard on graphs of DAG-width at most two.*

Proof. Let $L \subseteq \Sigma^*$ be any language in EXPTIME and let $M = (\Sigma, \Gamma, Q, q_{\mathrm{in}}, \delta)$ be an alternating Turing machine with a single tape and space bound m^s for some $s \geq 1$ that recognizes L and has branching degree d for an appropriate $d \in \mathbb{N}$. Let Θ, $\Theta_\perp$ and the functions $\kappa_\nu : (\Theta_\perp)^3 \to \Theta$ for $\nu \in [d]$ be as in the proof of Theorem 4.11 for the general case, where M is not necessarily deterministic. Now consider any $u = u_0 \ldots u_{m-1} \in \Sigma^*$ and let $\mathcal{G}_u = (V, \delta, \sim^V, \sim^A)$ be the game graph with partial information as constructed there (again, for the general case).

We modify $\mathcal{G}_u$ as follows. We introduce new positions $\diamondsuit$ and $\diamondsuit_l^{\overline{\theta}}$ for $(l, \overline{\theta}) \in [m^s] \times \Theta^3$ which belong to player 0 and are all indistinguishable for player 1. Moreover, we redirect any edge that leads to $\exists$ or $\forall$ to the vertex $\Diamond$ and analogously, we redirect any edge that leads to $\exists_l^{\overline{\theta}}$ or $\forall_l^{\overline{\theta}}$ to $\Diamond_l^{\overline{\theta}}$. Moreover, from $\Diamond$, player 0 can go to $\exists$ or to $\forall$ and likewise for $\Diamond_l^{\overline{\theta}}$. Finally, we delete all outgoing edges from any position $(0, \theta, m^s{-}1, q, \text{👁}, \overline{\theta})$ at the bottom level of $D_{\text{👁}}$, where q is either accepting or rejecting. So any position in $\mathcal{G}_u$ where a winner can be declared immediately is now a terminal position.

Now we define a sequence forcing winning condition $\mathcal{G}_u$ instead of the reachability condition R constructed in the proof of Theorem 4.11: We use the colors $\{0, 1\}$ and we start by coloring v_0 with 1. All $\Diamond$-positions as well as all $\forall$-positions get color 0 and all $\exists$-positions get color 1. (Where by $\Diamond$-position we refer to any position of the form $\Diamond$ or $\Diamond_l^{\overline{\theta}}$, likewise for $\exists$-positions and $\forall$-positions.) Moreover, any position $(i, \theta, m^s{-}1, q, \xi, \overline{\theta})$ at the bottom-level of one of the subgraphs $D_{\text{👁}}$ and $D_l^{\overline{\theta}}$ is colored with 1, if q is universal and the position is colored with 0, if q is existential. Finally, all positions that have not been colored yet get color 1. The set of sequences that player 1 wants to force is $\{000\}$.

Clearly, the whole construction can be carried out in polynomial time. Moreover, the correctness can be proved with essentially the same arguments as before. The only difference is that now, we have to take into account the fact that from a position at the bottom level of one of the subgraphs $D_{\text{👁}}$ and $D_l^{\overline{\theta}}$ with an existential state, the game does not automatically proceed to the corresponding $\exists$-positions: Player 0 might go to the corresponding $\forall$-position. But, in this case, he loses because he has played the sequence 000. Moreover, this is also the only possibility how this sequence can occur during any play of the game. Hence, player 0 will lose if, and only if, he tries to cheat at this point or if a terminal position is reached where he is declared the loser. $\square$

Notice that, although the PTIME-reduction from sequence forcing games (with fixed l) to reachability games does work for games with imperfect information as well, this cannot be directly used to strengthen Theorem 4.13 to reachability games: Taking the product of the original game graph and the memory structure may increase the DAG-width of the game graph!

In total, this analysis yields a fairly tight picture of the complexity of the strategy problem for games with imperfect information played on graphs of bounded DAG-width, but with unbounded partial information. We conclude this section with some remarks on other measures than DAG-width as well as other winning conditions, in particular, parity and safety conditions and on further restrictions of the game graphs that one might consider.

Other Measures. We start with a look at other measures. We will see that, while Theorem 4.11 does hold for directed path-width and entanglement, the particular bound proved there is still open for tree-width. In the following, we consider only the graphs $\mathcal{G}_u$ as constructed in the proof of Theorem 4.11 (for the general case). It is not hard to see that, as for DAG-width, the graphs $\mathcal{G}'_u$ from the proof of Theorem 4.13 satisfy $\chi(\mathcal{G}'_u) = \chi(\mathcal{G}_u){-}1$ for any $\chi \in \{\mathrm{tw}, \mathrm{dpw}, \mathrm{ent}\}$.

We start with directed path-width and we show $\mathrm{dpw}(\mathcal{G}_u) \leq 2$. Remind that directed path-width and DAG-width differ by one due to the definition. To capture an invisible directed robber on $\mathcal{G}_u$, we use exactly the same search strategy as in the proof of Theorem 4.11, except that we do not chase the robber down the subgraphs $D_{\text{👁}}$ and $D_l^{\overline{\theta}}$, respectively, by placing a cop on the robber in each step which we can't, but by conquering the levels systematically, in a breadth-first fashion (cf. Section 4.1). For this, a single cop is sufficient as well.

For entanglement, the scheme is also the same, but we have to take into account the fact that we can place a cop only on the current vertex of the robber. Therefore, we cannot obstruct the two vertices $\exists$ and $\forall$ in $\mathcal{G}_u$ at once but we first place a cop on the vertex through which the robber enters $D_{\text{👁}}$ (if he does) and if he enters this subgraph a second time, we obstruct the other vertex. Chasing the robber is not necessary in the case of entanglement because in the entanglement game, the robber has to move in each step. So, once $\exists$ and $\forall$ are blocked, the robber is actually captured Therefore, $\mathrm{ent}(\mathcal{G}_u) \leq 2$.

The most fiddly case is that of tree-width, as the robber can move along edges in any direction. So, in particular, chasing the robber down the subgraphs $D_{\odot}$ and $D_l^{\overline{\theta}}$, respectively, cannot be done with a single cop. Rather, we have to use $2 \cdot b(\odot)$ cops, where $b(\odot)$ is the breadth of $D_{\odot}$. (Notice that the breadth $b(l, \overline{\theta})$ of the subgraphs $D_l^{\overline{\theta}}$ satisfies $b(l, \overline{\theta}) \leq b(\odot)$.) Having that many cops, we can safely chase the robber down $D_{\odot}$ (and $D_l^{\overline{\theta}}$) by occupying two successive levels completely in each step and gradually moving the cops from the upper one of these two levels to the next unoccupied level. Notice that, for each $l, \overline{\theta}$, exactly one level of $D_l^{\overline{\theta}}$ is connected to exactly one level of $D_{\odot}$. Hence, if the robber goes from $D_{\odot}$ to one of the DAGs $D_l^{\overline{\theta}}$, we can guarantee that the robber does not go back to $D_{\odot}$ with $b(\odot)$ many cops and we still have $b(\odot)$ cops at our disposal to occupy the corresponding level in $D_l^{\overline{\theta}}$. Then, we can take all the cops from $D_{\odot}$ and continue to chase the robber down $D_l^{\overline{\theta}}$. Additionally, we need one cop to keep v_0 occupied the whole time and we also need two more cops to occupy the nodes $\exists$, $\forall$ and $\exists_l^{\overline{\theta}}$,$\forall_l^{\overline{\theta}}$, respectively. So $\mathrm{tw}(\mathcal{G}_u) \leq 2b(\odot){+}2$.

Now, $b(\odot) = |\Theta|^3 \cdot |Q|$ and both these values depend only on the Turing machine M but not on the input u. It also not hard to modify the construction so that keeping track of the state q is not necessary so that we obtain an upper bound of $\mathrm{tw}(\mathcal{G}_u) \leq 2|\Theta|^3 + 2$. (Essentially, we just delete the Q-component from the positions and, in turn, we make any position $(i, \theta, l, \odot, \overline{\theta})$ where $\theta = (q, \gamma)$ with $q \in Q_+ \cup Q_-$ a terminal position where player 1 has won if $q \in Q_+$ and otherwise, player 0 has won.) However, since $|Q|$ is also contained in $|\Theta|$, this is not really important. The point is that we can conclude: The strategy problem for reachability games with imperfect information is EXPTIME-hard on graphs of tree-width at most k; Where k is the minimal number such that there is an EXPTIME-complete language L which can be recognized by an alternating Turing-machine with polynomial space bound that satisfies $2|\Theta|^3{+}2 \leq k$. On the other hand, we do not know any better bounds. In particular, it is quite conceivable that on graphs of tree-width at most two, the strategy problem for reachability games with imperfect information can be solved in polynomial time.

Finally, let us note that $\mathrm{dpw}(\mathcal{G}_u) \leq 2$ implies that the graphs $\mathcal{G}_u$ also have Kelly-width at most two and directed tree-width at most seven. We do not examine these bounds closer.

Winning Conditions. We have formulated Theorem 4.11 and Theorem 4.12 for reachability conditions. However, reachability conditions can be easily reduced to parity conditions. In this case, we color all positions where player 1 has not won yet with color 1 and we delete all outgoing edges from positions in $D_{\odot}$, where the state q is accepting, and make them winning for player 1. Moreover, in the context of Theorem 4.11 where we do not consider necessarily acyclic graphs, we can also easily get rid of the terminal positions and obtain a pure parity condition: We simply add selfloops to all terminal positions and color them with $1{-}i$, if they are winning for player i. Even more, if we don't like selfloops, we can add, for each terminal position u, an undirected edge to a position $\tilde{u}$ and color both positions with the appropriate color. In this case, the DAG-width (or any other measure) of the game graph will not be affected by this construction.

Notice, though, that the parity condition defined in this way is not observable. On the other hand, it is also still a reachability condition defined by the reachability set $R \subseteq V$. As we have already observed (cf. Section 2.2.3), we can turn a reachability condition into an observable one by simply making all positions in R distinguishable for player 1 from all other positions. In this way, the coloring as defined above will also be observable. Moreover, the coloring as defined in the proof of Theorem 4.13 already is observable. To replace, additionally, terminal positions by selfloops (or undirected edges), and still preserve observability, we first have to make the positions where player 0 has won distinguishable for him from all other positions just as for the reachability condition. Then we can color all positions where player i has won with color $1{-}i$, thereby obtaining an observable (pure) sequence condition with two colors and the same set of sequences that player 1 wants to force as before.

Furthermore, the result also holds for (observable) safety winning conditions. To show this, we have to go back to alternating Turing machines. It is not hard to see, that for any alternating Turing machine M with a single tape and space bound m^s for some $s \geq 1$, there is an alternating Turing machine N with a single tape and space bound m^s such that $L(N) = L(M)$ and, additionally, N has time bound $2^{O(m^s)}$, cf. [48]. So in particular, any possible run of N is finite. Hence, we can make all positions in $\mathcal{G}_u$ where player 1 has not lost yet *safe* for player 1. Moreover, as before, we delete all outgoing edges from positions in $D_{\text{👁}}$, where q is accepting, and make them winning for player 1. (Or turn the edges into selfloops or undirected edges.) If player 1 bums around safe positions in $D_{\text{👁}}$ for more than $2^{O(m^s)}$ steps, then he has written at least one configuration that contains a flaw (with respect to $\vdash$), so player 0 can make him lose.

Altogether, the hardness results from Theorem 4.11 and Theorem 4.12 hold for observable parity, reachability and safety winning conditions. On the other hand, the result from Theorem 4.13 holds for observable sequence conditions but is still open for any winning condition that allows positional winning strategies.

Remarks. As we have mentioned, the results that we have proved in this section demonstrate that the intrinsic (computational) complexity caused by partial information is very high, even for simple graphs. Due to the fundamental character of this statement, it is expedient to consider certain other aspects and parameters of our setting, and have a look at their impact on the complexity of the strategy problem. We shall see that the computational hardness of the strategy problem also withstands certain other natural restrictions of the setting.

A very important aspect is the size of the equivalence classes of positions. Indeed, the impact of this parameter on the complexity of the strategy problem is quite strong: If we bound the size of equivalence classes of positions, the strategy problem for parity games with imperfect information and a fixed number of colors can be solved in polynomial time. Moreover, in Section 4.3 we prove that the problem can be solved in polynomial time for arbitrarily many colors on graphs of bounded DAG-width. On the other hand, the size of the equivalence classes of actions is much less important: All hardness-results also hold if we bound the size of equivalence classes of actions to a maximum of two. Even more, we can assume that player 0 has exactly two actions available in total.

To see how this improvement can be obtained, let 0 and 1 be two distinct actions which are indistinguishable for player 1 and replace the actions ☞ and ✍ by 0 and 1, respectively. Unboundedly many actions of player 0 (which are all indistinguishable for player 1) occur at the initial position v_0, allowing player 0 to check the initial configuration. To avoid this, we replace v_0 by a binary directed tree with root v_0 in which any two positions belong to player 0 and are indistinguishable for player 1. At any node of the tree, player 0 has 0 and 1 available. The leaves of the tree are positions ☞ and $(l, \theta_1\theta_2\theta_3)$, corresponding to the actions that player 0 could choose at v_0. At any such position a, player 0 then just needs the single action 0, leading to $\delta(v_0, a)$. Notice that any action available at v_0 can be uniquely identified by a bit-sequence of length $O(\log(m^s))$, so the tree has polynomial size. (Notice that this construction can also be applied to the game graphs in the proofs of Proposition 3.27 and Proposition 4.8, demonstrating a lower bound of $2^{\sqrt{|V|}/c}$ in both cases.)

Moreover, the game graphs constructed in the hardness-proofs have the property that each edge is labeled by exactly one action. Also, as we have mentioned before, the results still hold as stated, if we discard selfloops: We can use either terminal positions or directed edges at the leaves. Another natural restriction of the game graph would be that they are strongly connected. This is not the case, but it is very simple to modify the construction in such a way that the graph is strongly connected but only slightly more complex: We just add a position $\diamondsuit$ to the game graph that belongs to player 1 and is colored with 0 and which is connected via an undirected edge with each position where it is player 0's turn. Clearly, the resulting graph is strongly connected and the correctness of the construction is not harmed, because whenever player 0 decides to go to $\diamondsuit$, he immediately loses. Moreover, the DAG-width (and any other measure) are obviously increased by at most one. Hence, Theorem 4.11 holds at least for strongly connected game graphs of DAG-width at most four.

Finally, let us note that the hardness results also hold for asynchronous observability, cf. Section 2.1.4 and Section 3.5.4): Since in the game graphs constructed in the proofs, there are no private moves of player 0, the strategy problem and the asynchronous strategy problem coincide for all these graphs.

4.3 Bounded Partial Information

In Section 4.2 we have seen that the intrinsic complexity caused by partial information is very high, even on simple graphs. We have also seen that this complexity withstands several further restrictions like bounded size of equivalence classes of actions. On the other hand, if we bound the size of the equivalence classes of positions, then the powerset construction clearly yields a graph that has only polynomial size (independent of the size of equivalence classes of actions). In this Section we consider the case of equivalence classes of positions which have only bounded size. We also simply call this *bounded partial information.*

The main approach that we take is the following: For a class of parity game graphs $\mathcal{G}$ with bounded partial information, the powerset graphs $2^{\mathcal{G}}$ have polynomial size, so the strategy problem on the class of powerset graphs can be solved in polynomial time, if the graph complexity is bounded with respect to some $\chi \in \{\mathrm{tw}, \mathrm{dw}, \mathrm{dpw}, \mathrm{ent}\}$. Hence, to show that the strategy problem for parity games played on game graphs with bounded partial information and bounded complexity with respect to some χ is in PTIME, it suffices to prove that the powerset construction preserves boundedness of χ.

Our first results, however, are negative: For tree-width and for entanglement this is not the case. For tree-width, of course, this is not surprising, because tree-width measures the complexity of the underlying undirected graph: If we perform the powerset construction and then simply neglect the direction of the edges of the powerset graph then much graph structure can be created. For entanglement, on the other hand, the reason is that the measure is too delicate: If we replace each node in a full undirected (binary) tree of depth $2m$ by an undirected edge, then the entanglement jumps from two to at least $m - 1$, cf. Proposition 4.7. Such small gadgets, however, can be created from appropriate graphs by applying the powerset construction.

As it turns out, for our concerns, DAG-width is a much more appropriate measure. We start by showing that strategies for the cops and directed robber game can be translated from a game graph with bounded partial information to its powerset graph using a concept that we call concurrent graph searching. However, our straightforward translation does not preserve monotonicity. We will analyze this problem and prove, as an intermediate step, that the powerset construction preserves boundedness of directed path-width. Refining this approach leads us to the concept of graph searching with multiple robbers which we can, finally, use to demonstrate that, indeed, boundedness of DAG-width is preserved by the powerset construction. From this we obtain the main result of this section: The strategy problem for parity games on game graphs with bounded partial information and bounded DAG-width can be solved in polynomial time.[24] The main technical result about the concept of graph searching with multiple robbers that we need for this is proved in Section 4.4.

Tree-width. First, we construct a class of game graphs with bounded partial information and bounded measure χ for any $\chi \in \{\mathrm{tw}, \mathrm{dw}, \mathrm{dpw}, \mathrm{ent}\}$ which are, essentially, turned into partial grids by the powerset construction. These partial grids have already been shown to have large tree-width, cf. Section 4.1.

For any even $m \in \mathbb{N}$, consider the game graph $\mathcal{G}_m = (V_m, \delta_m, \sim_m^V, \sim_m^A)$ defined as follows.

[24]We will see that, while the negative results on tree-width and entanglement hold also for small equivalence classes of actions, this positive results holds for arbitrarily large equivalence classes of actions.

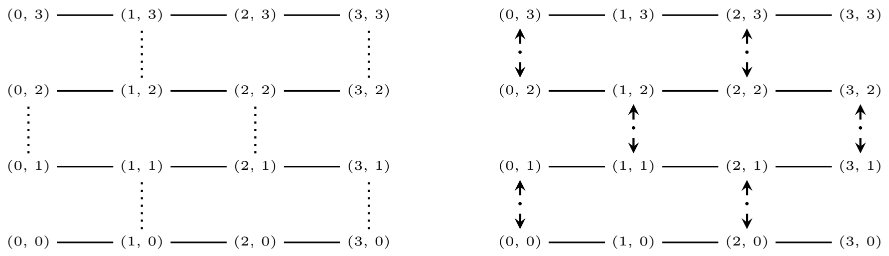


Figure 4.2: Abstract shape of $\mathcal{G}_4$ (without v_0) and a subgraph of its powerset graph.

- $V_m = \{v_0\} \cup [m] \times [m]$ and $A_m = \{a_{j,l} \mid j,l \in [m]\} \cup \{\rightarrow, \leftarrow\}$
- $v_0 \xrightarrow{a_{j,l}} (j,l)$ for all $j,l \in [m]$
- $(j,h) \xleftrightarrow{\rightarrow} (j+1,h)$ for $h \in [m]$ and $j \in [m{-}1]$.
- $(j,h) \sim^V_m (j,h+1)$ and $a_{j,h} \sim^A_m a_{j,h+1}$

  for all $j \in [m]$ and $h \in [m{-}1]$ with different parity

Notice that in $\mathcal{G}_m$, any equivalence class (of positions as well as of actions) has size at most two. Structurally, $\mathcal{G}_n$ is an undirected grid with all vertical edges deleted, together with the initial node v_0 that has a directed edge to each vertex in the graph. Clearly, $\chi(\mathcal{G}_m) \leq 2$ for all $\chi \in \{\text{tw}, \text{dw}, \text{dpw}, \text{ent}\}$. (Notice that $\mathcal{G}_n$ can also be seen as a disjoint union of m undirected paths of length m, together with v_0 and the directed edges from there.)

Now consider the powerset graph $\overline{\mathcal{G}}_m$, cf. Figure 4.2. We describe only a subgraph of $\overline{\mathcal{G}}_m$ that contains enough structure to prove large tree-width. First, the graph contains an isomorphic copy of $\mathcal{G}_m$, where each vertex (i,j) is replaced by $\{(i,j)\}$. Additionally, for all $j \in [m]$ and $h \in [m{-}1]$ which have a different parity we have a vertex $\{(j,h),(j,h+1)\}$ from which there is a directed $\leftarrow$-edge to $(j{-}1,h)$ and a directed $\leftarrow$-edge to $(j{-}1,h+1)$. Clearly, $\text{tw}(\overline{\mathcal{G}}^{\text{sym}}_m) = \text{tw}(\overline{\mathcal{G}}_m)$, and it is easy to see that $\text{tw}(\overline{\mathcal{G}}^{\text{sym}}_m) \geq \text{tw}(\mathfrak{G}_{m,\frac{1}{2}})$ where $\overline{\mathcal{G}}^{\text{sym}}_m$ denotes the symmetric closure of $\overline{\mathcal{G}}_m$ and $\mathfrak{G}_{m,\frac{1}{2}}$ denotes the partial grid as defined in Section 4.1. (For example, by contracting each path $\{(j{-}1,h)\}$ — $\{(j,h),(j,h{+}1)\}$ — $\{(j{-}1,h{+}1)\}$ to a single edge $\{(j{-}1,h)\}$ — $\{(j{-}1,h+1)\}$ we can see that $\overline{\mathcal{G}}^{\text{sym}}_m$ is a minor (in fact, even a topological one) of $\mathfrak{G}_{m,\frac{1}{2}}$.) We have shown there that $\text{tw}(\mathfrak{G}_{m,\frac{1}{2}}) \geq m/2$, so $\text{tw}(\overline{\mathcal{G}}_m) \geq m/2$ as well.

Proposition 4.14. *Boundedness of tree-width is not preserved.*

Notice that the game graphs $\mathcal{G}_m$ can also easily be made undirected by simply adding edges $(j,l) \xrightarrow{a_{j,l}} v_0$ for all j,l. This does not affect the tree-width of $\mathcal{G}_m$ but merely increases its DAG-width and entanglement from two to three. Moreover, the tree-width of the resulting powerset graph is at least $m/2$ as well.

Entanglement. Now we turn to entanglement. We construct a class of game graphs with bounded partial information and bounded measure χ for any $\chi \in \{\text{tw}, \text{dw}, \text{ent}\}$ which are, essentially, turned into the double binary trees from Section 4.1 by the powerset construction that we have shown to have large entanglement. For some particular notation that we use here, we refer to the construction provided there.

For any even $m \in \mathbb{N}$ consider the game graph $\mathcal{G}_m = (V_m, \delta_m, \sim^V_m, \sim^A_m)$ defined as follows.

- $V_m = \{0,1\}^{\leq m} \cup \{\bar{0},\bar{1}\}^{\leq m} \cup \{\tilde{0},\tilde{1}\}^{\leq m}$ and $A_m = \{0, 1, \rightarrow, \circlearrowleft\}$
- $u \xrightarrow{\circlearrowleft} u \qquad u \xleftrightarrow{0} u0 \qquad u \xleftrightarrow{1} u1$

  $\overline{u} \xleftrightarrow{0} \overline{u}\overline{0} \qquad \overline{u} \xleftrightarrow{1} \overline{u}\overline{0} \qquad$ for all $u \in \{0,1\}^{\leq m-1}$

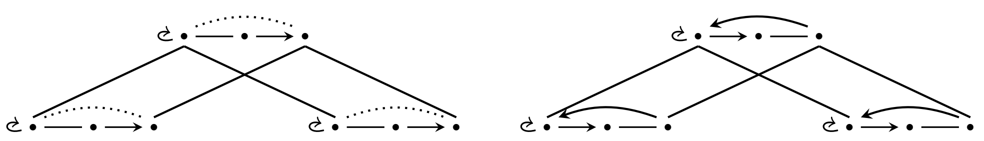


Figure 4.3: Abstract shape of $\mathcal{G}_2$ and its powerset graph.

- $u \overset{\rightarrow}{\longleftrightarrow} \tilde{u}$ and $\tilde{u} \overset{\rightarrow}{\longrightarrow} \overline{u}$ for all $u \in \{0,1\}^{\leq m}$.
- $u \sim_m^V \overline{u}$ for all $u \in \{0,1\}^{\leq m}$

So $\mathcal{G}_m$ is obtained from the double binary tree defined in Section 4.1 by adding selfloops to any $u \in \{0,1\}^{\leq m}$ and by making any vertex u equivalent to its duplicate vertex $\overline{u}$, cf. Figure 4.3. (All actions can be made distinguishable in this case.) Moreover, we paste an additional vertex $\tilde{u}$ on each edge that connects a vertex u with its duplicate $\overline{u}$. The important point is now that from such a vertex we have an undirected edge connecting it with u but only a *directed* edge from $\tilde{u}$ to $\overline{u}$ and no back-edge. So, whenever the robber in the entanglement game decides to move over to the copy $\overline{T}_{m,2}$ of the binary tree $T_{m,2}$, he cannot leave this copy anymore. So on the whole graph, two cops can win the entanglement game by playing a strategy for the full undirected binary tree until the robber decides to go to the other copy and then apply the strategy there. Hence, $\text{ent}(\mathcal{G}_m) \leq 2$ and, obviously, $\text{tw}(\mathcal{G}_m) = \text{dw}(\mathcal{G}_m) = 2$.

Now consider the powerset graph $\overline{\mathcal{G}}_m$, cf. Figure 4.3. This graph is essentially an isomorphic copy of $\mathcal{G}_m$, where each vertex u and $\tilde{u}$ is replaced by $\{u\}$ and $\{\tilde{u}\}$, respectively. Moreover, $\overline{u}$ is replaced by $\{u, \overline{u}\}$. Additionally, we direct the edge between $\{u\}$ and $\{\tilde{u}\}$ towards $\{\tilde{u}\}$ and we make the edge between $\{\tilde{u}\}$ and $\{u, \overline{u}\}$ undirected. Finally, and most importantly, we have a *back-edge* from $\{u, \overline{u}\}$ to $\{u\}$. It is now very easy to see that the entanglement of $\overline{\mathcal{G}}_m$ is at least as large as that of the double binary tree from Proposition 4.7, that means, $\text{tw}(\overline{\mathcal{G}}_m) \geq m/2 - 1$.

Proposition 4.15. *Boundedness of entanglement is not preserved.*

4.3.1 DAG-width and Concurrent Graph Searching

After these two negative results, we now turn to analyzing DAG-width. In the following, let

$$\mathcal{G} = (V, \delta, \sim^V, \sim^A)$$

denote a game graph with partial information and let

$$(\overline{V}, \overline{\Delta}) = \overline{\mathcal{G}} = 2^{\mathcal{G}} = (2^V, \overline{\Delta})$$

denote the corresponding powerset game graph with full information.

First we demonstrate that if k cops capture a robber on the graph $\mathcal{G}$, then $k \cdot r \cdot 2^{r-1}$ cops capture a robber on $2^{\mathcal{G}}$. This result holds, in fact, also for $r = |V|$, that means, for the case of unbounded partial information. On the other hand, if r is unbounded, then $k \cdot r \cdot 2^{r-1}$ is unbounded as well.

To see an example of how the robber can move on a powerset graph, consider the game graph $\mathcal{G}$ and its powerset graph in Figure 4.4. To catch a robber on $\mathcal{G}$ with two cops, we place a cop on v_1 and on v_2 because they form a clique. Then the robber can go to v_3 or to v_4. In the first case, we take the cop from v_2 and put it on v_3, in the second case we take the cop from v_1 and put it on v_4. On the powerset graph, however, after we have put cops on $\{v_1\}$ and on $\{v_2\}$, the robber goes to $\{v_3, v_4\}$. In order to catch the robber, neither the cop from $\{v_1\}$ nor from $\{v_2\}$ can be taken because now, $\{v_1\}$, $\{v_2\}$ *and* $\{v_3, v_4\}$ form a clique. The reason is that if the robber goes from $\{v_1\}$ to $\{v_3, v_4\}$ then this corresponds to *two* possible plays in the original game: The one where the robber goes from v_1 to v_3 and the one where he goes from v_1 to v_4.

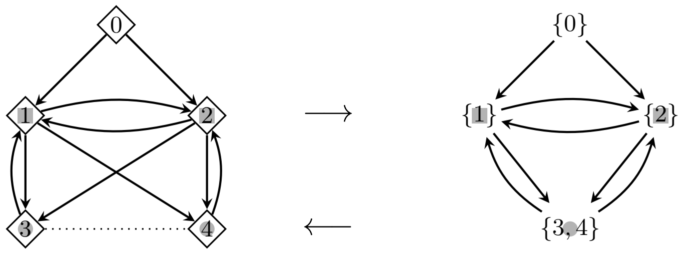


Figure 4.4: Concurrent Graph Searching

This example suggests that, to be sure to catch the robber on $2^{\mathcal{G}}$, we have to trace all plays in the cops and robber game on $\mathcal{G}$ that are induced by a particular play in the cops and robber game on $2^{\mathcal{G}}$, that means, which correspond to the possible positions of the robber. Since the positions in $2^{\mathcal{G}}$ have size at most r, at each point, there are at most r different vertices where the robber might be, so we have to trace at most r different plays. Clearly, for each such play, we have to allocate an individual team of cops. We call this *concurrent graph searching.*

The idea for the strategy translation is as follows. We translate the moves of the cops from $\mathcal{G}$ to $2^{\mathcal{G}}$ and the moves of the robber in the opposite direction. Consider positions in games on both graphs. When the robber makes a move on $2^{\mathcal{G}}$ to a vertex $\{v_1, \dots, v_l\}$ we consider l plays in the game on G where he moves to v_1, v_2,..., v_l. For each of these moves, the strategy for the cops for the game on $\mathcal{G}$ supplies an answer, moving the cops from U to U'. All these moves are translated into a move in which the cops occupy precisely the vertices of $2^{\mathcal{G}}$ that include a vertex from some U'. These moves of the cop player on $2^{\mathcal{G}}$ can be realized with $k \cdot r \cdot 2^{r-1}$ cops and guarantee that moves of the robber can always be translated back to the game on G. The key argument for this reverse translation is that, by Proposition 3.22, for any path

$$\overline{u}_0 \to \overline{u}_1 \to \dots \to \overline{u}_\mu$$

in $2^{\mathcal{G}}$ and for any $u_\mu \in \overline{u}_\mu$, there is a path

$$u_0 \to u_1 \to \dots \to u_\mu$$

in $\mathcal{G}$ such that $u_\lambda \in \overline{u}_\lambda$ for any $\lambda \in \{0, \dots, \mu\}$.

Proposition 4.16. *If k cops capture a robber on $\mathcal{G}$ then $k \cdot r \cdot 2^{r-1}$ cops capture a robber on $2^{\mathcal{G}}$.*

Proof. Let σ be a winning strategy for the cop player for the k cops and robber game on $\mathcal{G}$. We define the strategy $\overline{\sigma}$ for the cop player for the cops and robber game on $\overline{\mathcal{G}} = 2^{\mathcal{G}}$ by induction on the length of histories

$$\overline{\pi} = (\overline{U}_0, \overline{v}_0)(\overline{U}_0, \overline{U}_1, \overline{v}_0)(\overline{U}_1, \overline{v}_1) \dots (\overline{U}_l, \overline{v}_l)$$

where it is the cops' turn. At the same time, with each such history that is consistent with $\overline{\sigma}$, we associate a finite set $\zeta(\overline{\pi})$ that consists of histories

$$\pi = (U_0, v_0)(U_0, U_1, v_0)(U_1, v_1) \dots (U_l, v_l)$$

in the k cops and robber game on $\mathcal{G}$ such that the following conditions hold. To formulate the conditions, let $U_{\zeta(\overline{\pi})}$ denote the union of all the sets U_l of vertices occupied by cops in one of these histories and let $R_{\zeta(\overline{\pi})}$ denote the set of all the vertices v_l occupied by the robber in one of these histories.

(1) Each history in $\zeta(\overline{\pi})$ is consistent with σ

(2) $\overline{v}_l = R_{\zeta(\overline{\pi})}$

(3) $\overline{v} \in \overline{U}_l \iff \overline{v} \cap U_{\zeta(\overline{\pi})} \neq \emptyset$

(4) if $\overline{\pi} \sqsubseteq \overline{\rho}$ then $\zeta(\overline{\pi}) \sqsubseteq \zeta(\overline{\rho})$

Notice that $\zeta(\overline{\pi})$ can be viewed as a tree which has $\perp$ as root and all the histories π as leaves. We maintain this view of $\zeta(\overline{\pi})$ as a tree but by writing $\pi \in \zeta(\overline{\pi})$ we still refer to the leaves only. $\zeta(\overline{\pi}) \sqsubseteq \zeta(\overline{\rho})$ means that if $\zeta(\overline{\pi})$ has depth d then $\zeta(\overline{\rho})$ has depth $e \geq d$ and, up to level d, the two trees coincide.

First, with any history $\overline{\pi} = (\emptyset, \overline{v})$ we associate the tree $\zeta(\overline{\pi})$ that consists of all histories $\pi = (\emptyset, v)$ with $v \in \overline{v}$. Now consider any history $\overline{\pi} = (\overline{U}_0, \overline{v}_0)(\overline{U}_0, \overline{U}_1, \overline{v}_0)(\overline{U}_1, \overline{v}_1) \ldots (\overline{U}_l, \overline{v}_l)$ that is consistent with $\overline{\sigma}$ such that $\zeta(\overline{\pi})$ is constructed and satisfies (1) - (4). We define $\overline{\sigma}(\overline{\pi}) = \overline{U}_{l+1}$ by $\overline{v} \in \overline{U}_{l+1}$ if there is some $\pi \in \zeta(\overline{\pi})$ such that $\overline{v} \cap U_{l+1} \neq \emptyset$ where $U_{l+1} = \sigma(\pi)$.

Now consider any possible next move of the robber from $\overline{v}_l$ to $\overline{v}_{l+1}$. We show how this move can be translated back into several moves of the robber in the cops and robber game on $\mathcal{G}$, using only the current positions of the robber in the histories that we have traced. Since the robber has moved from $\overline{v}_l$ to $\overline{v}_{l+1}$ in the cops and robber game on $\overline{\mathcal{G}}$, there is a path

$$\overline{v}_l = \overline{w}_0 \to \overline{w}_1 \to \ldots \to \overline{w}_\mu = \overline{v}_{l+1}$$

in $\overline{\mathcal{G}} - \overline{U}_l \cap \overline{U}_{l+1}$. Let $v_{l+1} \in \overline{v}_{l+1}$. According to Proposition 3.22, there is a path

$$v_l = w_0 \to w_1 \to \ldots \to w_\mu = v_{l+1}$$

in $\mathcal{G}$ with $w_j \in \overline{w}_j$ for all $0 \leq j \leq \mu$. So $v_l \in \overline{v}_l$ and by (2) this yields $v_l \in R_{\zeta(\overline{\pi})}$, so there is some history $\pi \in \zeta(\overline{\pi})$ that ends in a position (U_l, v_l). Let $\pi' = \pi(U_l, U_{l+1}, v_l)$ with $U_{l+1} = \sigma(\pi)$. Since $\overline{v}_{l+1} \notin \overline{U}_{l+1}$ and $\overline{w}_j \notin \overline{U}_l \cap \overline{U}_{l+1}$, by (3) and the definition of $\overline{U}_{l+1}$ it follows that $v_{l+1} \notin U_{l+1}$ and $w_j \notin U_l \cap U_{l+1}$ for $0 \leq j \leq \mu$.

Therefore, $v_{l+1} \in \mathrm{Reach}_{G - U_l \cap \sigma(U_l)}(v_l) \setminus U_{l+1}$ which demonstrates that, given the history π', the robber can move from v_l to v_{l+1} in the cops and splitting robbers game on $\mathcal{G}$. Hence, $\rho := \pi(U_l, U_{l+1}, v_l)(U_{l+1}, v_{l+1})$ is a history in the cops and robber game on $\mathcal{G}$ and with $\overline{\rho} = \overline{\pi}(\overline{U}_l, \overline{U}_{l+1}, \overline{v}_l)(\overline{U}_{l+1}, \overline{v}_{l+1})$ we associate the tree $\zeta(\overline{\rho})$ that consists of such histories ρ extended in this way from histories in $\zeta(\overline{\pi})$, such that all $v_{l+1} \in \overline{v}_{l+1}$ are covered. (Notice that not all histories in $\zeta(\overline{\pi})$ are actually extended in this way because not all those histories are necessary to cover the possible robber's positions in $\overline{v}_{l+1}$. Feckless histories die out.) (1) - (4) all hold by construction.

Now assume that $\overline{\sigma}$ is not a winning strategy for the cop player. Then there is an infinite play $\overline{\pi}$ in the cops and robber game on $\overline{\mathcal{G}}$ that is consistent with $\overline{\sigma}$. Using (4) we obtain an infinite tree $\zeta(\overline{\pi})$ with $\zeta(\overline{\pi}(\leq l)) \preceq \zeta(\overline{\pi})$ for all $l \in \mathbb{N}$. Clearly, $\zeta(\overline{\pi})$ is finitely branching so, by König's Lemma, there is some infinite path π through $\zeta(\overline{\pi})$. Since, using (1), π is a play in the k cops and robber game on $\mathcal{G}$ which is consistent with σ this contradicts the premise that σ is a winning strategy for the cop player in this game. Finally, for any $\overline{\sigma}$-history $\overline{\pi}$ with $\mathrm{last}(\overline{\pi}) = (\overline{U}, \overline{v})$, by (3) we have $|\overline{U}| \leq |U_{\zeta(\overline{\pi})}| \cdot 2^{r-1}$. Moreover, $|U_{\zeta(\overline{\pi})}| \leq k \cdot |\zeta(\overline{\pi})|$ and, by (2) and the construction of $\zeta(\overline{\pi})$, $|\zeta(\overline{\pi})| \leq |U_{\zeta(\overline{\pi})}| = |\overline{v}| \leq r$. Hence, $|\overline{U}| \leq k \cdot r \cdot 2^{r-1}$. □

The proof of Proposition 4.16 shows that for a very plain (yet sensible) strategy for the cops on $2^{\mathcal{G}}$, moves of the robber can always be translated back to the graph $\mathcal{G}$. This suggests that, in the case of *bounded* partial information, the effects of the powerset construction on the graph structure are containable, if we measure the structural complexity of a graph in terms of DAG-width.

Remark. For directed path-width and Kelly-width, this is not quite clear. We do not know any examples that would show unbounded growth of these measures when the powerset construction

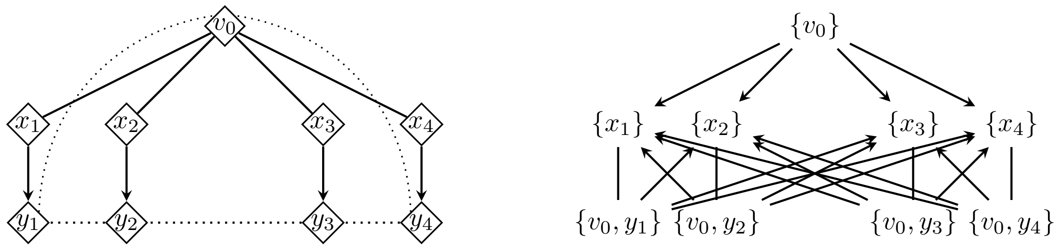


Figure 4.5: The game graph $\mathcal{G}_4$ and its powerset graph $\overline{G}^4_{v_0}$ from v_0.

is applied, but the strategy translation as described above cannot be directly applied to the games that characterize these measures. The problem is that, given the translation of the moves of the cops that we have defined, there is not such a nice correspondence between the moves of the robber on $\mathcal{G}$ and on $2^{\mathcal{G}}$: If a directed homebody moves from some $\overline{v}$ to $\overline{v}'$ then $\overline{v}$ and $\overline{v}'$ must be in the same strongly connected component of $2^{\mathcal{G}} - \overline{U} \cap \overline{U}'$. But that does not imply in general that for any $v' \in \overline{v}'$ there is some $v \in \overline{v}$ that is in the same connected component of $G - U \cap U'$ (for the appropriate U and U'). Moreover, for Kelly-width, the problem is that an inert robber can move from $\overline{v}$ to $\overline{v}'$ only if a cop is placed on $\overline{v}$. But this does not necessarily mean that each $v \in \overline{v}$ is occupied by a cop in some play: This need only be the case for at least one v. We do not consider this here further.

On the other hand, if the partial information is unbounded, then the rather small effects of the powerset construction may be accumulated. To make this a little more precise consider, for $m \in \mathbb{N}$, the graph $\mathcal{G}_m = (V_m, \delta_m, \sim^V_m, \sim^A_m)$, defined as follows.[25]

- $V_m = \{v_0, x_1, \dots, x_m, y_1, \dots, y_m\}$ and $A_m = \{a_1, \dots, a_m, \uparrow, \downarrow\}$
- $v_0 \xrightarrow{a_j} x_j \quad x_j \xrightarrow{\uparrow} v_0 \quad x_j \xrightarrow{\downarrow} y_j$
- $v_0 \sim^V_m y_j$

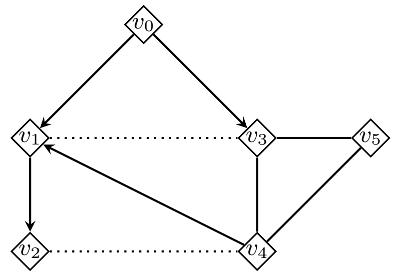
The graph $\mathcal{G}_4$ and its powerset graph are delineated in Figure 4.5. Clearly, $\chi(\mathcal{G}_m) \leq 2$ for all $\chi \in \{\text{tw}, \text{dpw}, \text{dw}, \text{ent}\}$. However, although $2^{\mathcal{G}_m}$ has no more positions than $\mathcal{G}_m$ has and, even more, any position in $2^{\mathcal{G}_m}$ has size at most two, we have $\chi(\mathcal{G}_m) \geq m$ for all $\chi \in \{\text{tw}, \text{dpw}, \text{dw}, \text{ent}\}$! (Notice that even to capture a robber nonmonotonously on $2^{\mathcal{G}_m}$, at least m cops are needed.) The reason is that the node v_0, which is highly connected with the rest of the graph, is contained in unboundedly many positions in $2^{\mathcal{G}_m}$. In this sense, v_0 accumulates the small effects of the powerset construction on the graph structure so that, in total, $\mathcal{G}_m$ has large complexity.[26]

In particular, this example reveals that the requirement that the positions in the powerset graph $2^{\mathcal{G}}$ have size at most r (which also implies that the powerset graph has polynomial size in total) is not sufficient for proving Proposition 4.16. On the other hand, it is not quite clear whether to prove Proposition 4.16, it is sufficient to assume merely that each position from $\mathcal{G}$ is contained in at most r positions in $2^{\mathcal{G}}$ (while the positions in $2^{\mathcal{G}}$ may have arbitrary size). This implies that the powerset graph has polynomial size in total as well, but clearly, the approach used in the proof given above cannot be readily applied: There, we have a second factor that potentially causes a need for unboundedly many cops: The number of plays that we trace simultaneously, which is also r. We will not pursue this question here further.

Instead, we shall stick to the assumption of bounded partial information and return to the fact that even under this assumption, Proposition 4.16 in the stated form is not a big help in proving

[25]This example is taken from [172].

[26]Examples like that in Proposition 4.8 show that these accumulation effects can multiply: If unboundedly many positions from $\mathcal{G}$ are contained in unboundedly many positions from $2^{\mathcal{G}}$, then the graph complexity can become exponential.



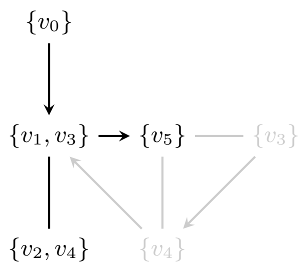


Figure 4.6: Monotone strategy is translated to a non-monotone one.

that the strategy problem for parity games with bounded partial information can be solved in polynomial time on graphs of bounded DAG-width: We were not able to prove that our strategy translation preserves monotonicity and since it is unclear whether DAG-width has bounded monotonicity cost, Proposition 4.16 does not yield that $\mathrm{dw}(2^{\mathcal{G}}) \leq k \cdot r \cdot 2^{r-1}$. However, we will see that this is really not a matter of structural complexity of the graph but of strategic reasoning.

Let us start by having a look at some concrete example where the strategy translation in the proof of Proposition 4.16 fails to preserve monotonicity. Consider the game graph $\mathcal{G}$ with partial information as depicted in Figure 4.6. We describe a (partial) monotone winning strategy σ for three cops, assuming the robber starts in v_0. First, put a cop on v_0. If the robber goes to v_1 then put a cop on v_1, take the cop away and put it on v_2. If the robber goes to v_3, place a cop on v_5. If the robber answers this move by going to v_4, take the cop from v_0 and put it on v_4. Then place a third cop successively on v_1, v_2, v_3. Now consider the translation $\overline{\sigma}$ of σ to $2^{\mathcal{G}}$ from the proof of Proposition 4.16. Assume the robber starts, analogously, in $\{v_0\}$. Then we put a cop on this position and, if the robber moves to $\{v_1, v_3\}$, we place a cop on $\{v_1, v_3\}$ due to the possibility that the robber might be on v_1 and we place a cop on $\{v_5\}$ due to the possibility that he might be on v_3. Now consider the move of the robber to $\{v_2, v_4\}$. Then the cop from $\{v_1, v_3\}$ will be taken because the cop from v_1 is taken in the play where the robber initially fled to v_1 and the vertex v_3 is still cop-free in the play where the robber initially fled to v_3. At this moment, $\{v_1, v_3\}$ will be available for the robber again, so the strategy is not monotone.

On the other hand, the there are several possibilities to change the strategy σ, so that the resulting strategy is a monotone winning strategy for two cops on $\mathcal{G}$ *and* the translated strategy $\overline{\sigma}$ is monotone as well. For example, if we don't free position v_1 after we have expelled the robber from there but, instead, take the cop from v_0 to occupy v_2, then the position $\{v_1, v_3\}$ would still be occupied by a cop, after the robber has moved to $\{v_2, v_4\}$. Moreover, if we placed three cop on v_3, v_4, v_5 immediately if the robber goes to one of those positions then the described nonmonotonicity would also not arise. However, starting from an *arbitrary* monotone winning strategy for three cops does include, in particular, strategies which make such moves that are, for someone who looks at them from the outside, unreasonable but still fulfill anything that we can expect from a monotone winning strategy.

Let us analyze the generic reason why this kind of nonmonotonicity arises. When the robber moved to $\{v_1, v_3\}$, we traced two plays in the original cops and robber game: A play π where the robber moves from v_0 to v_1 and a play ρ, he moves to v_3. Then we place a cop on $\{v_1, v_3\}$ in the game on $2^{\mathcal{G}}$ but we do this merely due to the play π, where a cop is placed on v_1. Then, when the robber moves to $\{v_2, v_4\}$, in the play π he moves to v_2, so the cop from v_1 is removed. But at this point, in the game on $2^{\mathcal{G}}$ we remove the cop from $\{v_1, v_3\}$ and this position is still available for the robber, because according to the play ρ, he might also be on vertex v_4. So the two plays that we trace simultaneously *interfere* with each other.

In general, these interferences are unavoidable because the actions of a strategy for the cops may depend on the position of the robber in the graph. (Or at least, on his strongly connected component, which wouldn't make any difference in the example above.) A natural and very nice way to deal with this problem would be to prove, that there is a normal form for cops' winning strategies which guarantees that such interferences do not appear. More precisely:

? Is there a normal form for monotone cops' winning strategies, such that the translation as described in the proof of Proposition 4.16 is monotone?

Remind that by normal form we mean that an appropriate (monotone) winning strategy would use only $\kappa(k)$ more cops than a given arbitrary one. However, we do not know whether this holds. We make a first approach to solve the interference problem by restricting the graph structure further.

Since the problem about preserving monotonicity when translating strategies is that a robber can cause nonmonotonicity in another play, it would be helpful to assume that any vertex that can be reached by one of the robbers can also be reached by the robbers in all the other plays. Then, if a robber would cause nonmonotonicity in a play, the robber that actually belongs to this play would cause nonmonotonicity as well which would contradict monotonicity of the original winning strategy. A simple way to guarantee this is to assume that all any equivalence class of positions are strongly connected and to adapt the original monotone winning strategy σ for k cops on $\mathcal{G}$ so that it is *compatible* with equivalence classes: For any σ-position (U, v) of the cop and directed robber game and any vertex u, either $[u] \subseteq U$ or $[u] \cap U = \emptyset$. This adaption can be easily done, using at most $r \cdot k$ cops: We just occupy the whole set $[u]$ as long as at least one $u \in [u]$ is occupied according to σ and, as soon as this is not the case anymore, we remove all cops from $[u]$.

Now, consider the strategy $\overline{\sigma}$ from the proof of Proposition 4.16 and assume at some point during a $\overline{\sigma}$-play in the cops and robber game on $2^{\mathcal{G}}$ nonmonotonicity occurs. That means, the cops move from $(\overline{U}, \overline{v})$ to $(\overline{U}, \overline{U}', \overline{v})$ and the robber can reach some $\overline{u} \in \overline{U} \setminus \overline{U}'$ in $G - \overline{U} \cap \overline{U}'$. Then, using Proposition 3.22 similar as in the proof of Proposition 4.16, there is some robber $v \in \overline{v}$ that can reach some $u \in U \setminus U'$ in $G - U \cap U'$, where the sets U and U' belong to some play π that we have traced. This play also has a robber, say at w, and, since $[v]$ is strongly connected and completely cop-free, the robber at w can reach v in $G - U \cap U'$ and from there, cause nonmonotonicity in π – a contradiction.

Proposition 4.17. *If* $\mathrm{dw}(\mathcal{G}) \leq k$ *and each equivalence class of positions is strongly connected, then* $\mathrm{dw}(2^{\mathcal{G}}) \leq k \cdot r^2 \cdot 2^{r-1}$.

Clearly, the quadratic overhead that we get is negligible compared to the exponential factor that we already have. On the other hand, having strongly connected equivalence classes is a very restrictive assumption. This might be appropriate for some very special situations where, for example, the partial information of player 1 concerns only certain private variables of player 0 that he can change arbitrarily without giving control to player 1. For most scenarios, however, this assumption is completely ill-suited.

Instead, we look for an assumption which also guarantees that no malign interferences between different plays occur and which can be expressed merely in terms of structural complexity of the game graph. Here, the concept of directed path-width comes into play. We know that the cops' actions in this game do *not* depend on what the robber does because they don't see him. So it's quite reasonable that the interferences as described above do not occur in this case so that a strategy translation similar to that in the proof of Proposition 4.16 will be actually monotone. For this, recall the representation of the cops and invisible robber by means of positions (U, R), where R is the *set* of all vertices in the graph on which the robber might still be. Since the cops and robber game that we are interested in is played on the *powerset graph* of $\mathcal{G}$, this powerset representation of the cops and invisible robber game will be most convenient.[27]

[27]Also notice that this game is a *solitaire* game, so we do not even have to consider multiple plays that are

Proposition 4.18. *If* $\mathrm{dpw}(\mathcal{G}) \leq k$ *then* $\mathrm{dpw}(2^{\mathcal{G}}) \leq (k+1) \cdot 2^{r-1}$.

Proof. Let σ be a monotone winning strategy for $(k+1)$ cops for the cops and invisible robber game on $\mathcal{G}$ and let

$$\pi = (U_0, R_0)(U_1, R_1)\ldots(U_s, R_s)$$

be the unique play in the game that is consistent with σ. Notice that $R_j \neq \emptyset$ for $j < s$ and $R_s = \emptyset$. We construct a monotone play

$$\overline{\pi} = (\overline{U}_0, \overline{R}_0)(\overline{U}_1, \overline{R}_1)\ldots(\overline{U}_s, \overline{R}_s)$$

in the $(k+1) \cdot 2^{r-1}$ cops and invisible robber game on $\overline{\mathcal{G}} = 2^{\mathcal{G}}$ with $\overline{R}_s = \emptyset$. A monotone winning strategy $\overline{\sigma}$ for $(k+1) \cdot 2^{r-1}$ cops for this game can then be directly obtained from $\overline{\pi}$. (That is, we construct a monotone winning strategy $\overline{\sigma}$ for $(k+1) \cdot 2^{r-1}$ cops for the cops and invisible robber game on $\overline{\mathcal{G}}$ by constructing the unique monotone, finite play that is consistent with $\overline{\sigma}$.) Remind that the play $\overline{\pi}$ has to satisfy $\overline{U}_0 = \emptyset$, $\overline{R}_0 = \overline{V}$ and $\overline{R}_{j+1} = \mathrm{Reach}_{\overline{\mathcal{G}} - \overline{U}_j \cap \overline{U}_{j+1}}(\overline{R}_j) \setminus \overline{U}_{j+1}$.

We construct histories $\overline{\pi}_l = (\overline{U}_0, \overline{R}_0)\ldots(\overline{U}_l, \overline{R}_l)$, $l \leq s$, in the cops and invisible robber game on $\overline{\mathcal{G}}$ such that the following conditions holds.

(1) $\bigcup \overline{R}_l \subseteq R_l$

(2) $\overline{u} \in \overline{U}_l \iff \overline{u} \cap U_l \neq \emptyset$

Notice that from $R_s = \emptyset$ and (1) it follows that $\overline{R}_s = \emptyset$. So once the play $\overline{\pi}$ is constructed such that (1) holds, $\overline{\pi}$ is won by the cops.

The history $\overline{\pi}_0 = (\overline{U}_0, \overline{R}_0) = (\emptyset, \overline{V})$ is uniquely determined. Now let the history $\overline{\pi}_l = (\overline{U}_0, \overline{R}_0)\ldots(\overline{U}_l, \overline{R}_l)$ with $l < s$ be constructed such that (1) and (2) are satisfied. We set $\overline{U}_{l+1} = \{\overline{v} \in \overline{V} \mid \overline{v} \cap U_{l+1} \neq \emptyset\}$ and $\overline{R}_{l+1} := \mathrm{Reach}_{\overline{\mathcal{G}} - \overline{U}_l \cap \overline{U}_{l+1}}(\overline{R}_l) \setminus \overline{U}_{l+1}$. With these definitions, $\overline{\pi}_{l+1} = \overline{\pi}_l(\overline{U}_{l+1}, \overline{R}_{l+1})$ is a history in the cops and invisible robber game on $\overline{\mathcal{G}}$ by construction. Moreover, (2) also holds by definition. To prove that (1) holds as well, we have to demonstrate that all possible (implicit) moves of the robber in the game on $\overline{\mathcal{G}}$ can be translated back to the game on $\mathcal{G}$. This can be shown by the same arguments as in the proof of Proposition 4.16 (using Proposition 3.22, condition (1) for $\overline{R}_l$ and condition (2) for $\overline{U}_l$ as well as the definition of $\overline{U}_{l+1}$.)

Let us show that the cops' move from $(\overline{U}_l, \overline{R}_l)$ to $(\overline{U}_l, \overline{U}_{l+1}, \overline{R}_l)$ is monotone. Assume, towards a contradiction, that there is some $\overline{u} \in \overline{U}_l \setminus \overline{U}_{l+1}$ such that $\overline{u}$ is reachable from $\overline{R}_l$ in $\overline{\mathcal{G}} - \overline{U}_l \cap \overline{U}_{l+1}$. So there is a path

$$\overline{v} = \overline{v}_0 \to \overline{v}_1 \to \ldots \to \overline{v}_\mu = \overline{u}$$

in $\overline{\mathcal{G}} - \overline{U}_l \cap \overline{U}_{l+1}$ from some $\overline{v} \in \overline{R}_l$ to $\overline{u}$. Since $\overline{u} \in \overline{U}_l$, by condition (2), there is some $u \in \overline{u}$ with $u \in U_l$ and as $\overline{u} \notin \overline{U}_{l+1}$, condition (2) yields $u \notin U_{l+1}$. According to Proposition 3.22, there is some path

$$v = v_0 \to v_1 \to \ldots \to u_\mu = u$$

in $\mathcal{G}$ such that $v_j \in \overline{v}_j$ for all $0 \leq j \leq \mu$. Since $\overline{v}_j \notin \overline{U}_l \cap \overline{U}_{l+1}$, by condition (2) it follows that $v_j \notin U_l \cap U_{l+1}$ for $0 \leq j \leq \mu$. Moreover, by condition (1) we have $\overline{v} \subseteq R_l$, so $v \in R_l$. In total we have $u \in U_l \setminus U_{l+1}$ and u is reachable from $v \in R_l$ in $G - U_l \cap U_{l+1}$. Therefore, the cops' move from (U_l, R_l) to (U_l, U_{l+1}, R_l) in π is not monotone, which is a contradiction.

So, from $\overline{\pi}$ we obtain a monotone winning strategy for the cop player for the cops and invisible robber game on $\overline{\mathcal{G}}$ and it is easy to see that at most $(k+1) \cdot 2^{r-1}$ cops are used. $\square$

Remind that, in fact, the proof of monotonicity was redundant because directed path-width is known to be monotone. However, as we have mentioned, we are interested in the reason *why* the translation described in the proof above preserves monotonicity. We have already suggested

consistent with a given monotone winning strategy for the cops but only a single play. This makes the proof indeed somewhat easier but is not necessary for avoiding interferences, cf. Proposition 4.19.

the main reason before: If the robber could, at some point, from any $\overline{v}$ on which he might currently be, reach a position $\overline{u}$ from which a cop is just leaving, then there is some robber $v \in \overline{v}$ that could reach a position $u \in \overline{u}$ from which a cop is just leaving in the original game. In the case of Proposition 4.16, the problem was that v and u might belong to different plays that we are tracing and since the original strategy may make different decisions in these plays, we were not able to deduce non-monotonicity of the strategy. Here, the original strategy makes the same decision in all such plays, so non-monotonicity follows.

However, as bounded directed path-width is too restrictive, we are mainly interested in using this insight to transfer the result to DAG-width. To see how we can do this, recall that the fact that for directed path-width we have to trace only a single play is the imperfect information in the cops and robber game. The cops do not see the robber, so their actions may not depend on the position $v \in \overline{v}$ that the robber occupies. So in a sense, the partial information in the cops and robber game comprises the partial information on the game graph. However, since the robber in the game is completely invisible but positions in the powerset graph have size at most r, this is an overapproximation: A robber that might be on r possible vertices at any point would be sufficient.

As we have mentioned before, the powerset representation of the game graph is more suited for our concerns. In fact, the condition of the robber being on at most r possible vertices can also be expressed most naturally in terms of the sets R in this representation: We parametrize the powerset game graph with a bound $r \in \mathbb{N}$ on the size of R. This leads, in a very natural way, to the concept of graph searching with *multiple visible* robbers: The k cops and r directed robbers game has cops' positions (U, R) and robbers' positions (U, U', R) where U and U' are as before and $R \subseteq V$. The moves of the cops are also as in the usual game, while the robbers' moves are now from (U, U', R) to any (U', R') such that $R' \subseteq \mathrm{Reach}_{G-U\cap U'}(R) \setminus U'$ and $|R'| \leq r$. So, in any move, the robbers can decide which r many of all the possible positions they want to occupy without any further restrictions. In particular, this also includes the possibility to take robbers from certain positions of the graph away and *split* robbers at other positions into several copies. These robbers need not even be in the same connected component of $G - U \cap U'$ at this point, so in a sense, the robbers can *jump* in the graph, but not in an arbitrary fashion (which would it make impossible to catch them with less than $|V|$ cops) but only to other robbers that are currently in the graph. Since we do not distinguish between individual robbers but merely have a *set* R of vertices currently occupied by robbers, the moves of the robbers are, indeed, described more adequately by the concept of splitting. However, to highlight this additional power of the robbers (over mere movements along edges), we sometimes use the term of jumping.

We will discuss the properties of this game in greater detail in Section 4.4. It is, however, clear that we need this particular rule to obtain a game that corresponds to the cops and (single) directed robber game played on the powerset graph $2^{\mathcal{G}}$. Proposition 4.19 states that, if the partial information is bounded by r, then a strategy for $k(r)$ cops that guarantees to capture r robbers *simultaneously* in a *monotone* way on $\mathcal{G}$ can be used to construct a strategy for $k(r) \cdot 2^{r-1}$ cops that guarantees to capture a directed robber on $2^{\mathcal{G}}$ *monotonously.* Of course, to be able to infer from this that the powerset construction preserves boundedness of DAG-width, we have to show that if DAG-width is bounded, then for any fixed r, $k(r)$ is bounded as well. This is our main result in Section 4.4:

$$k(r) \leq k \cdot r$$

We will discuss related work on graph searching with multiple robbers for undirected graphs as well as the particular difficulties that arise in the directed case there. We conclude this section with showing that this result can in fact be used to prove that the powerset construction preserves boundedness of DAG-width. Notice that in this case, we do not have just a single play that is consistent with a given strategy for the cops because the cops can see the robbers. On the other hand, for any given play in the cops and robber game on $2^{\mathcal{G}}$, we have exactly one play in the cops and multiple robbers game on $\mathcal{G}$ that corresponds to it. Therefore, no interferences can occur.

Proposition 4.19. $\mathrm{dw}(2^{\mathcal{G}}) \leq k(r) \cdot 2^{r-1}$.

Proof. Let σ be a monotone winning strategy for the cop player for the $k(r)$ cops and r splitting robbers game on $\mathcal{G}$. We define a strategy $\overline{\sigma}$ for the cop player for the cops and robber game on $\overline{\mathcal{G}} = 2^{\mathcal{G}}$ inductively on the length of histories

$$\overline{\pi} = (\overline{U}_0, \overline{v}_0)(\overline{U}_0, \overline{U}_1, \overline{v}_0)(\overline{U}_1, \overline{v}_1) \dots (\overline{U}_l, \overline{v}_l)$$

where it is the cops' turn. At the same time, with each such history that is consistent with $\overline{\sigma}$, we associate a history

$$\zeta(\overline{\pi}) = (U_0, R_0)(U_0, U_1, R_0)(U_1, R_1) \dots (U_l, R_l)$$

in the $k(r)$ cops and r splitting robbers game on $\mathcal{G}$ such that the following conditions hold.

(1) $\zeta(\overline{\pi})$ is consistent with σ

(2) $R_l = \overline{v}_l$

(3) $\overline{u} \in \overline{U}_l \iff \overline{u} \cap U_l \neq \emptyset$

(4) If $\overline{\pi} \sqsubseteq \overline{\rho}$ then $\zeta(\overline{\pi}) \sqsubseteq \zeta(\overline{\rho})$

First, with any history $\overline{\pi} = (\emptyset, \overline{v})$ we associate the history $\zeta(\overline{\pi}) = (\emptyset, \overline{v})$. Now consider any history $\overline{\pi} = (\overline{U}_0, \overline{v}_0)(\overline{U}_0, \overline{U}_1, \overline{v}_0)(\overline{U}_1, \overline{v}_1) \dots (\overline{U}_l, \overline{v}_l)$ that is consistent with $\overline{\sigma}$ such that $\zeta(\overline{\pi})$ is constructed and satisfies (1) - (4). Let $\sigma(\zeta(\overline{\pi})) = (U_l, U_{l+1}, R_l)$ and define $\overline{\sigma}(\overline{\pi}) := \overline{U}_{l+1} = \{\overline{u} \in \overline{V} \mid \overline{u} \cap U_{l+1} \neq \emptyset\}$. With the same argument as in the proof of Proposition 4.18 (using Proposition 3.22, condition (2) for $\overline{v}_l$ and condition (3) for $\overline{U}_l$ as well as the definition of $\overline{U}_{l+1}$) it can be shown that the cops' move from $\overline{U}_l$ to $\overline{U}_{l+1}$ is monotone.

Now consider any possible next move of the robber from $\overline{v}_l$ to $\overline{v}_{l+1}$. With the same arguments as in the proof of Proposition 4.16 (using Proposition 3.22, condition (2) for $\overline{v}_l$ and condition (3) for $\overline{U}_l$ as well as the definition of $\overline{U}_{l+1}$) it can be shown that this move can be translated back to the cops and splitting robbers game on $\mathcal{G}$. That means, the robber can move from R_l to $\overline{v}_{l+1}$ and with $\overline{\rho} = \overline{\pi}(\overline{U}_l, \overline{U}_{l+1}, \overline{v}_l)(\overline{U}_{l+1}, \overline{v}_{l+1})$ we associate the history $\zeta(\overline{\rho}) = \zeta(\pi)(U_l, U_{l+1}, R_l)(U_{l+1}, \overline{v}_{l+1})$. Clearly, (1) – (4) all hold.

Obviously, $\overline{\sigma}$ is a winning strategy for the cops: Given an infinite $\overline{\sigma}$-play $\overline{\pi}$ in the cops and robber game on $\overline{\mathcal{G}}$, conditions (1) and (4) immediately yield an infinite σ-play $\zeta(\overline{\pi})$ in the $k(r)$ cops and r splitting robber game on $\mathcal{G}$ – a contradiction. Moreover, by construction, $\overline{\sigma}$ uses at most $k(r) \cdot 2^{r-1}$ cops and since all moves according to $\overline{\sigma}$ are monotone it follows that $\mathrm{dw}(\overline{\mathcal{G}}) \leq k(r) \cdot 2^{r-1}$. □

Theorem 4.20. *Boundedness of DAG-width is preserved.*

Proof. Proposition 4.19 and Theorem 4.22. □

So, for a given parity game graph $\mathcal{G} = (V, \delta, \sim^V, \sim^A)$ with equivalence classes of size at most r and DAG-width at most k, we can apply the powerset construction to obtain a parity game graph $\overline{\mathcal{G}} = (V, \overline{\Delta})$ with full information which has size at most $2^r \cdot |V|$ and DAG-width at most $k \cdot r \cdot 2^{r-1}$. The last step that is needed to be able to apply the algorithm developed in [24], it to transform $\overline{\mathcal{G}}$ into a purely position based, deterministic game graph $\mathcal{H} = (V, E)$, where the players choose successor positions instead of actions, cf. Section 2.1.2. For this, we just have to turn $\overline{\mathcal{G}}$ into a deterministic game graph $\overline{\mathcal{G}}^d = (V^d, \delta^d)$. As we have shown in Section 2.1.2, a purely position based game graph is then obtain by simply deleting the actions from the edges which clearly does not affect the complexity of the game graph and can therefore be neglected. We have described a determinization construction in Section 2.1.2, so it remains to convince ourselves that the construction is nonhazardous in terms of DAG-width. In fact, this is easy to see, we just give a short description.

Let σ be a monotone winning strategy for $\mathrm{dw}(\overline{\mathcal{G}})$ cops on $\overline{\mathcal{G}}$. If the game on $\overline{\mathcal{G}}^d$ is in a position (U, v), to be able to make the next move according to σ, the only problem is that we do not necessarily have $v \in V$ (if $v \in V$, then the cops just move to $\sigma(U, v)$). If $v \notin V$ then $v = (w, a)$ with $w \in V$ (recall that $V^d = V \cup V \times A$) and we distinguish two cases. If $w \notin U$, then the cops can assume that the robber is on w (notice that $\mathrm{Reach}_{G-U}(v) \subseteq \mathrm{Reach}_{G-U}(w)$) and move to $\sigma(U, w)$. If $w \in U$, then either $|U| < k$ or there is at least one $u \in U$ such that $u \notin \mathrm{Reach}_{G-(U\setminus\{u\})}(v)$. (Otherwise, no cop could be moved without causing nonmonotonicity, but the robber is not caught yet.) We place such a free cop on v, then remove the cop from v (and put it back to where it came from). Since v is only reachable from w which is occupied by a cop, this move is monotone. If the robber is not caught yet, then he runs along some path $v \to w' \to \ldots$ for some $w' \in V$ with $(w, a, w') \in \overline{\Delta}$ and the cops move to $\sigma(U, w')$. In this way, any play π on $\overline{\mathcal{G}}^d$ that is consistent with this modified strategy, essentially coincides with a play on $\overline{\mathcal{G}}$ that is consistent with σ (and where the robber has always less or equal positions available). Thus, π is monotone and won by the cops, so $\mathrm{dw}(\overline{\mathcal{G}})$ cops monotonously capture a robber on $\overline{\mathcal{G}}^d$. Hence, $\mathrm{dw}(\overline{\mathcal{G}}^d) \leq k \cdot r \cdot 2^{r-1}$.

Corollary 4.21. *The strategy problem for parity games on graphs with bounded partial information and bounded DAG-width can be solved in polynomial time.*

4.4 Graph Searching with Multiple Robbers

This section is devoted to the following theorem. Throughout this section, let $G = (V, E)$ be a directed graph.

Theorem 4.22. $\mathrm{dw}_r(G) \leq r \cdot \mathrm{dw}(G)$.

The number $\mathrm{dw}_r(G)$ is the least $k(r) \in \mathbb{N}$ such that $k(r)$ cops *monotonously* capture r *directed splitting* robbers on G. We have already briefly introduced our concept of graph searching with multiple robbers in Section 4.3. We start with a detailed presentation of these games and some basic properties. In particular, we discuss the moves that the robbers can make in the game and the corresponding concept of monotonicity. In the second step, we present two simple normal forms for robbers' strategies, that are useful for the proof, and for analyzing graph searching with multiple robbers in general.

We approach the proof of Theorem 4.22 with the simpler cases of undirected graphs and robbers that do not jump, respectively. We will see that, in fact, much easier proofs of Theorem 4.22 can be given for these cases. We then discuss the main difficulties that arise in the general case of directed graphs and robbers that can jump and we give a rough blueprint of how we are going to solve these difficulties in the proof that we present at the end of this section. Before we begin, we make some general remarks about the concept of graph searching with multiple robbers and about related work on this topic.

First, from the definition of $\mathrm{dw}_r(G)$ it is quite clear that we have the following:

$$\mathrm{dw}(G) = \mathrm{dw}_1(G) \leq \mathrm{dw}_2(G) \leq \ldots \leq \mathrm{dw}_{|V|}(G) = \mathrm{dpw}(G)$$

Hence, the r-DAG-width is both, a refinement of directed path-width and a generalization of DAG-width. Clearly, we could have tackled the problem of concurrent graph searching also from this end, replacing several plays with a single robber by a single play with multiple robbers directly, instead of going through invisible robbers. However, in the context of game graphs with partial information, this detour via invisible robbers is a very natural approach.

Graph searching with multiple robbers has also been considered in [183] for *undirected* graphs and robbers that can only move along edges but not jump.[28] While Theorem 4.22 is quite easy

[28] The authors also consider mixed search (rather than node search as we consider it here): cops can also *slide* along edges. This, however, is insignificant.

to prove for undirected graphs, even for jumping robbers, in [183] it has been shown that the result can be improved in their setting: For an undirected graph G, if k cops monotonously capture a robber on G then $O(\log(r) \cdot k)$ cops monotonously capture r non-jumping robbers on G. On the other hand, as to directed graphs, we rather think that the bound from Theorem 4.22 is, essentially, optimal, even for non-jumping robbers.

Moreover, in [183], it has been shown that graph searching with non-jumping robbers on undirected graphs is not monotone. In fact, if we allow cops to play nonmonotone strategies without further restrictions, it is clear that they need less cops to capture non-jumping robbers: They can capture the robbers sequentially rather than concurrently using merely k cops. On the other hand, for jumping robbers, the situation is quite different: Such a straightforward sequential search strategy will not suffice, even if we allow nonmonotone cops' strategies, because once a robber is captured, he may still jump to other robbers. It is not quite clear, indeed, whether actually more cops are needed to capture multiple jumping robbers simultaneously, if monotonicity is required. While for $\mathrm{dw}_1 = \mathrm{dw}$ this is the case [123], for $\mathrm{dw}_{|V|} = \mathrm{dpw}$, this is not the case [11, 112]. In particular, the example constructed in [123] does not work anymore for dw_2: For any of their graphs $\mathcal{G}_m$ we have $\mathrm{dw}_2(\mathcal{G}_m) = \mathrm{dw}(\mathcal{G}_m)$, but to capture two jumping robbers simultaneously and nonmonotonously on $\mathcal{G}_m$, we also need $\mathrm{dw}(\mathcal{G}_m)$ many cops. Nevertheless, we conjecture that for any fixed $r \in \mathbb{N}$, dw_r is not monotone.

From the above, it is quite clear that jumping increases the power of the robbers against nonmonotone cops: Additional cops are necessary to capture jumping robbers instead of purely running ones. However, it is not clear, neither for directed nor for undirected graphs, whether jumping actually increases the power of the robbers against monotone cops. At any rate, the most general case where we consider directed graphs and jumping robbers appears to be exceptionally cumbersome. We do not know whether the rather intricate strategic reasoning from the proof of Theorem 4.22 can be avoided. However, we demonstrate that if we have at least undirected graphs or non-jumping robbers, much easier proofs can be obtained. We will explain why these approaches fail in the general case and discuss the underlying properties of jumping robbers on directed graphs that are crucial for concurrent graph searching.

The paper [183] also provides a lower bound of $\min\{\mathrm{dpw}(G), \lfloor \log(r) \rfloor + 1\} + 1$ on the number of cops needed to capture r robbers monotonously on any *undirected tree*. This lower bound clearly holds for our setting as well, demonstrating that Theorem 4.22 is not trivial in that $\mathrm{dw}_r(G)$ cannot be bounded in terms of $\mathrm{dw}(G)$ only, independent of r. Moreover, the $\mathrm{dw}_r(G)$-hierarchy between DAG-width and directed path-width described above does not collapse.

Another version of concurrent graph searching or, as it is called there, nondeterministic graph searching, has been considered in [84], also for *undirected* graphs. There, instead of having multiple robbers, we have a single invisible robber again, but the cops can query the position of the robbers q times for some $q \in \mathbb{N}$. This also induces a hierarchy between path-width and tree-width (in their setting of undirected graphs), but this time, $\mathrm{pw} = \mathrm{tw}_0(G) \geq \mathrm{tw}_1(G) \geq \ldots \geq \mathrm{tw}_{|V|}(G) = \mathrm{tw}(G)$. (Notice that $|V|$ queries are sufficient if we play a strategy that decreases the set of vertices available for the robber by at least one in each step.) In [84] a corresponding notion of graph decomposition has been provided (which is unknown for graph searching with multiple robbers) and also a lower bound has been demonstrated.

4.4.1 Cops and Directed Splitting Robbers

As we have mentioned, cops and directed splitting robbers games can be obtained by a very simple restriction of the powerset representation of the game graphs on which the cops and invisible robber games are played. There we have cops' positions (U, R) and robber's positions (U, U', R) with $U, R \subseteq V$ and, if the robber moves from (U, U', R) to (U', R') then

$$R' = \mathrm{Reach}_{G - U \cap U'}(R) \setminus U'.$$

Cops and directed splitting robbers games are played on a parametrized version of these game graphs where in each such move, the robber player can choose any $R'' \subseteq R'$ with $|R''| \leq r$. That means, positions (U, R) and (U, U', R) have to satisfy $|R| \leq r$ and, while the cops' moves are as before, the robber can move from (U, U', R) to any (U', R') with

$$R' \subseteq \mathrm{Reach}_{G-U\cap U'}(R) \setminus U'.$$

The terminal positions of the game graphs are cops' positions (U, R) with $R = \emptyset$. At these positions, all robbers are captured and the cops have won. As before, any infinite play is won by the robber player.

We have also mentioned the fact that these games can be viewed as a generalization of DAG-width which is obtained by allowing several robbers instead of just a single one. However, the above parametrization of the game graphs for the directed path-width game leads to splitting robbers in a very natural way: The transition from R to R' as defined above, in particular, allows for the possibility that several robbers run from a robber $b \in R$ to positions in R' that are not reachable from any other robber $b' \in R \setminus \{b\}$. So in a sense, the robber b splits into several copies while certain other robbers $b' \in R$ are simply taken away from the graph which can be seen as splitting into zero copies. We have also used the term of jumping to highlight this particular feature more distinctly, but since we do not distinguish between individual robbers we will never be interested in which robber jumps to which other one. So we shall stick to the view of the robber b splitting into several (possibly zero) copies.

On the other hand, starting from DAG-width it might seem more appropriate to restrict the robbers so that they can only run along directed edges from their current position but not split into several copies. This, however, is not sufficient for our concerns: In the cops and robber game on the powerset graph, if the robber moves from some position $\overline{v}$ to a position $\overline{w}$ then each $w \in \overline{v}$ is reachable from some $v \in \overline{w}$. However, the function $f : \overline{w} \to \overline{v}$ with $f(w) = v$ for some such v is neither injective nor surjective in general. Which is why, in the proof of Proposition 4.16, some of the plays that we trace die out while others are split into several plays. So, for Proposition 4.19 we need splitting robbers.

In this sense, the detour through directed path-width and the powerset representation of the game graphs has led to the right concept of graph searching with multiple robbers in a very natural way.

Monotonicity. The notion of monotonicity in these games is the straightforward generalization of monotonicity in the cops and single robber games: A play π in the cops and splitting robbers game is monotone if for all positions (U, U', R) we have $\mathrm{Reach}_{G-U\cap U'}(R) \subseteq \mathrm{Reach}_{G-U}(R)$, that is, the area that is reachable from all the robbers in total may never increase. As before, this is equivalent to the requirement that no robber can ever reach a position from which a cop is just leaving, that means, the play never reaches a position (U, U', R) such that $(U \setminus U') \cap \mathrm{Reach}_{G-U\cap U'}(R) \neq \emptyset$. So a robber may never reconquer a vertex that way previously unavailable for the robbers. Notice that in our context with splitting robbers it does not make sense to consider the seemingly weaker notion where a robber may not reconquer a vertex that was previously unavailable for him. However, even in the games with merely running robbers, the two intuitions of monotonicity lead to the same definition: If a robber reconquers a vertex that was previously unavailable for him then he can reach a vertex from which a cop is just leaving but this vertex was unavailable for all the robbers.

Again, this notion of monotonicity is what would be called robber-monotonicity. The notion of cop-monotonicity is defined as before and as for tree-width and DAG-width, robber-monotonicity and cop-monotonicity coincide which can be shown with the same arguments as there. (In the sense that whenever k cops have a winning strategy to capture r robbers that is robber-monotone or cop-monotone then they also have one that is both.)

Strategy Normal Forms. To outsource at least some of the merely technical difficulties in the proof of Theorem 4.22, we prove two plain normal forms for robbers' strategies separately: Isolating strategies and prudent strategies. Moreover, as we have already mentioned, it is easy to see that if k cops have a (positional) monotone winning strategy then k cops have a (positional) monotone winning strategy that never places cops outside the area that is reachable for the robber. We call such strategies *thrifty*. Formally, a strategy for the cops is thrifty, if for any move $(U, r) \to_\sigma (U, U', r)$ according to σ, any vertex in $U' \setminus U$ is reachable for the robber:

$$(U, r) \to_\sigma (U, U', r) \;\;\Rightarrow\;\; U' \setminus U \subseteq \mathrm{Reach}_{G-U}(r).$$

Notice that in the proof of Theorem 4.22 we start with a strategy for k cops for the cops and single robber game so it is sufficient for us to have this normal form for cops' strategies against a single robber. Nevertheless, the normalform obviously holds for several robbers as well.

We have already mentioned prudent strategies for single robber in Section 4.1: A strategy for the robber is prudent if he moves only to positions that would otherwise be unavailable for him after the cops have landed. Formally, the moves $(U, U', b) \to (U', b')$ of the robber are restricted by the requirement that $b' \notin \mathrm{Reach}_{G-U'}(b)$. The generalization to multiple splitting robbers is straightforward: A strategy for the robber player for the cops and r splitting robbers game is called *prudent* if for each move $(U, U', R) \to_\sigma (U', R')$ that is consistent with σ any new robber $b \in R' \setminus R$ is not reachable from R in $G - U'$:

$$(U, U', R) \to_\sigma (U', R') \;\;\Rightarrow\;\; (R' \setminus R) \cap \mathrm{Reach}_{G-U'}(R) = \emptyset.$$

Notice that, in particular, this restricts a robber $b \in R$ to move only if at least one cop is placed in the area that is reachable for him. However, even if this is the case, all emissaries of b must go to positions that would be unavailable for b after the cops have landed. Intuitively it is quite clear that this is not a serious restriction of the robbers: Moving from a vertex b to another vertex b' that is still reachable from b in $G - U'$ can never be better for the robber because he can still go from b to b' in a later move. (While it might be possible that he cannot return from b' to b in a later move.)

Isolating strategies, on the other hand, are particular to games with multiple robbers. A strategy for the robber player for the cops and r splitting robbers game is called *isolating* if it splits robbers only into instances that are topologically incomparable. More precisely, for any position $\to_\sigma (U, R)$ that is consistent with σ, any two robbers from R are topologically incomparable in $G - U$:

$$\to_\sigma (U, R) \;\;\Rightarrow\;\; \text{For all } b, b' \in R: \;\; b' \notin \mathrm{Reach}_{G-U}(b).$$

In particular, b and b' are not in the same strongly connected component of $G - U$. Again, intuitively, it is quite clear that restricting the cops to isolating strategy does not restrict their competitive capacity: If there are two robbers $b, b' \in R$ such that $b' \in \mathrm{Reach}_{G-U}(b)$ then in fact, b' is redundant because without the robber b' the robbers still have the same scope. More precisely, $\mathrm{Reach}_{G-U}(R \setminus \{b'\}) = \mathrm{Reach}_{G-U}(R)$. So the robbers can simply remove b' from the graph and at a point where b and b' would become topologically incomparable, they split b into two copies that move to the appropriate positions.

It should be mentioned that both these normalforms are not a substantial contribution to the proof of Theorem 4.22. In the proof we will see that, conceptually, it is rather easy to deal with topologically comparable robbers as well as with non-prudent robbers, when designing cops' strategies. Technically, however, this makes it much more expensive to describe a winning strategy, particularly in the proof of Theorem 4.22. Moreover, assuming that the robbers play an isolating prudent strategy makes it easier to comprehend and analyze the dynamics in the cops and splitting robber game.

Proposition 4.23. *If k cops have a monotone strategy that is winning against all isolating prudent strategies for r robbers then they have a monotone winning strategy against r robbers.*

Proof. Let σ be a monotone strategy for k cops that is winning against all isolating prudent strategies for r robbers. We construct the strategy $\tilde{\sigma}$ for k cops by induction on the length of histories

$$\pi = (U_0, R_0)(U_0, U_1, R_1)\ldots(U_l, R_l)$$

where it is the cops' turn and at the same time, with each such history that is consistent with $\tilde{\sigma}$ we associate a history

$$\zeta(\pi) = (U_0, \tilde{R}_0)(U_0, U_1, \tilde{R}_1)\ldots(U_l, \tilde{R}_l)$$

such that the following conditions hold.

(1) $\zeta(\pi)$ is consistent with σ

(2) $\zeta(\pi)$ is consistent with some isolating prudent strategy for r robbers

(3) $\mathrm{Reach}_{G-U_l}(R_l) \subseteq \mathrm{Reach}_{G-U_l}(\tilde{R}_l)$

(4) If $\pi \sqsubseteq \rho$ then $\zeta(\pi) \sqsubseteq \zeta(\rho)$

First, with any history $\pi = (\emptyset, R)$ we associate the history $\zeta(\pi) = (\emptyset, \min_\emptyset(R))$ where $\min_U(R)$ consists of the topologically minimal elements in R with respect to reachability in $G - U$, that is,

$$\min_U(R) = \{b \in R \mid \text{ there is no } b' \in R \setminus \{b\} \text{ with } b \in \mathrm{Reach}_{G-U}(b')\}.$$

Now consider any history $\pi = (U_0, R_0)(U_0, U_1, R_1)\ldots(U_l, R_l)$ that is consistent with $\tilde{\sigma}$ such that $\zeta(\pi)$ is constructed and satisfies (1) - (4). We define $U_{l+1} = \tilde{\sigma}(\pi) := \sigma(\zeta(\pi))$.

Since σ is monotone, the move from $(U_l, \tilde{R}_l)$ to $(U_l, U_{l+1}, \tilde{R}_l)$ is monotone, that means, $U_l \setminus U_{l+1} \cap \mathrm{Reach}_{G-U_l\cap U_{l+1}}(\tilde{R}_l) = \emptyset$. Moreover, due to (3), we have $\mathrm{Reach}_{G-U_l}(R_l) \subseteq \mathrm{Reach}_{G-U_l}(\tilde{R}_l)$, so $\mathrm{Reach}_{G-U_l\cap U_{l+1}}(R_l) \subseteq$

$$\mathrm{Reach}_{G-U_l\cap U_{l+1}}(\mathrm{Reach}_{G-U_l}(R_l)) \subseteq \mathrm{Reach}_{G-U_l\cap U_{l+1}}(\mathrm{Reach}_{G-U_l}(\tilde{R}_l)) \subseteq$$

$$\mathrm{Reach}_{G-U_l\cap U_{l+1}}(\mathrm{Reach}_{G-U_l\cap U_{l+1}}(\tilde{R}_l)) =$$

$\mathrm{Reach}_{G-U_l\cap U_{l+1}}(\tilde{R}_l)$. Therefore, $U_l \setminus U_{l+1} \cap \mathrm{Reach}_{G-U_l\cap U_{l+1}}(\tilde{R}_l) = \emptyset$, so the move from (U_l, R_l) to (U_l, U_{l+1}, R_l) is also monotone.

Now consider any possible next move of the robber from (U_l, U_{l+1}, R_l) to (U_{l+1}, R_{l+1}). First, for any $b' \in R_{l+1} \cap \mathrm{Reach}_{G-U_{l+1}}(\tilde{R}_l)$ choose some $b \in \tilde{R}_l$ such that $b' \in \mathrm{Reach}_{G-U_{l+1}}(b)$ and let R consist of all these b. Moreover, let $R' := R_{l+1} \setminus \mathrm{Reach}_{G-U_{l+1}}(\tilde{R}_l)$ and let $\tilde{R}_{l+1} := \min_{U_{l+1}}(R \cup R')$. Obviously, $|\tilde{R}_{l+1}| \leq |R \cup R'| \leq |R_{l+1}| \leq r$. Moreover, for each $b \in \tilde{R}_{l+1}$, either $b \in R'$ which implies $b \in R_{l+1} \subseteq \mathrm{Reach}_{G-U_l\cap U_{l+1}}(R_l) \setminus U_{l+1}$ and, as we have seen above, $\mathrm{Reach}_{G-U_l\cap U_{l+1}}(R_l) \subseteq \mathrm{Reach}_{G-U_l\cap U_{l+1}}(\tilde{R}_l)$, so $b \in \mathrm{Reach}_{G-U_l\cap U_{l+1}}(\tilde{R}_l) \setminus U_{l+1}$. Or $b \in R$ which implies $b \in \tilde{R}_l$ and $\mathrm{Reach}_{G-U_{l+1}}(b) \neq \emptyset$, so we have $b \in \mathrm{Reach}_{G-U_l\cap U_{l+1}}(\tilde{R}_l) \setminus U_{l+1}$ as well. Hence, $\tilde{R}_{l+1} \subseteq \mathrm{Reach}_{G-U_l\cap U_{l+1}}(\tilde{R}_l) \setminus U_{l+1}$.

Therefore, going from $(U_l, U_{l+1}, \tilde{R}_l)$ to $(U_{l+1}, \tilde{R}_{l+1})$ is a legal move for the robbers and with $\rho = \pi(U_l, U_{l+1}, R_l)(U_{l+1}, R_{l+1})$ we associate $\zeta(\rho) = \zeta(\pi)(U_l, U_{l+1}, \tilde{R}_l)(U_{l+1}, \tilde{R}_{l+1})$. (1) and (4) obviously hold by construction. Moreover, by definition, $R \cup R'$ does not contain any vertices that are reachable from $\tilde{R}_l$ in $G - U_{l+1}$ and, additionally, $\tilde{R}_{l+1} \subseteq R \cup R'$ does not contain any topologically comparable robbers. Hence, going from $(U_l, U_{l+1}, \tilde{R}_l)$ to $(U_{l+1}, \tilde{R}_{l+1})$ is an isolating prudent move for the robbers which demonstrates (2). Finally, $R_{l+1} \subseteq \mathrm{Reach}_{G-U_{l+1}}(R \cup R')$ and thus $\mathrm{Reach}_{G-U_{l+1}}(R_{l+1}) \subseteq \mathrm{Reach}_{G-U_{l+1}}(R \cup R') = \mathrm{Reach}_{G-U_{l+1}}(\min_{U_{l+1}}(R \cup R'))$ which shows that (3) holds as well. (Notice that, in general, whenever $U = U'$, we have $\mathrm{Reach}_{G-U}(W) = \mathrm{Reach}_{G-U}(\min_{U'}(W))$, while this is not necessarily the case if $U \neq U'$.)

Now consider any play $\pi = (U_0, R_0)(U_0, U_1, R_0)(U_1, R_1)\ldots$ that is consistent with $\tilde{\sigma}$. By (4) we obtain a play $\zeta(\pi) = (U_0, \tilde{R}_0)(U_0, U_1, \tilde{R}_0)(U_1, \tilde{R}_1)\ldots$ which, by (1) and (2), is consistent with σ and with also with some isolating prudent strategy for r robbers. Since σ is winning against all isolating prudent strategies for r robbers, $\zeta(\pi)$ is won by the cops which means that it is finite and ends in a position $(U_l, \tilde{R}_l)$ with $\tilde{R}_l = \emptyset$. Therefore, by (3), $R_l = \emptyset$, so π is won by the cops as well. Since we have already shown that any move according to $\tilde{\sigma}$ is monotone and, obviously, $\tilde{\sigma}$ uses at most k cops, the proof is done. □

Notice that, to transform the strategy σ into a winning strategy $\tilde{\sigma}$, we did not use the assumption of monotonicity. In fact, if we just skip monotonicity, the proof of Proposition 4.23 yields that if k cops have a strategy that is winning against all isolating prudent strategies for r robbers, then k cops have a winning strategy against r robbers. Moreover, independently of monotonicity, if σ is a positional strategy, then so is $\tilde{\sigma}$.

Also notice that, contrary to our announcement, Proposition 4.23 does not directly yield a normal form for robbers' strategies. Indeed, in the proof of Theorem 4.22, we start from a monotone winning strategy for k cops against a single robber and we construct a monotone strategy for $k \cdot r$ cops that is winning against all isolating prudent strategies for r robbers, so Proposition 4.23 is more helpful in the stated form. Clearly, Proposition 4.23 would not be harder to prove if it was formulated to yield a respective normal form for robbers' strategies. But then, to apply it in the proof of Theorem 4.22 we had to use determinacy. Since determinacy is a rather intense property, we like to avoid it if it is not necessary.

Finally, it should be mentioned that, while prudent strategies are also sufficient for robbers that cannot split, isolating strategies are obviously not: If a bunch of non-splitting robbers has to use an isolating strategy on a graph that is strongly connected, then they have to start with a single robber but since they cannot split, the robber will remain a lone wolf throughout the game. Therefore, the cops just play their strategy against a single robber. Of course, one could think of adaptions of the rules of the game in order to make isolating strategies non-trivial like, for example, allowing the robbers to split, but once a robber is on the graph, he cannot be taken away and as soon as he is captured, he cannot be re-used. This, however, will not be of any further relevance to us.

4.4.2 Undirected Graphs and Non-Splitting Robbers

As we have mentioned, we approach the proof of Theorem 4.22 by considering the simpler cases of undirected graphs and non-splitting robbers, respectively. While we discuss the case of non-splitting robbers on directed graphs only briefly, we give a more detailed proof for the case of splitting robbers on undirected graphs. The main scheme and some of the patterns and constructs from the proof also form the foundation for the proof of Theorem 4.22. However, there are certain essential properties of the concept of splitting robbers on directed graphs that inhibit a direct adaption of the proof to the directed case. Likewise, it seems not possible to transfer the proof for non-splitting robbers on directed graphs readily to the case of splitting robbers. We will discuss these properties and outline how we solve the resulting problems in the proof of Theorem 4.22.

Let $G = (V, E)$ be an undirected graph and assume that k cops have a thrifty monotone winning strategy σ against a single robber. The very basic idea to catch r splitting robbers simultaneously (and monotonously) is as follows. We assume that the robbers play an isolating prudent strategy. As long as there is only a single robber on G, we play according to σ. As soon as the robbers split into several copies, they are topologically incomparable. But since G is undirected, this also means that the areas available for the different robbers are pairwise disjoint! So, as σ is thrifty, we can play against the individual robbers completely independently, according to the given strategy σ. As soon as a robber b splits once again, the several emissaries have to move in such a fashion that afterwards, they are completely separated in the current graph as well. So we can continue to play the play that we have played so far against b, against all these different robbers completely independently.

Proposition 4.24. $\mathrm{dw}_r(G) \leq r \cdot \mathrm{dw}(G)$ *for any undirected graph* G.

Proof. Let σ be a thrifty monotone winning strategy for k cops against a single robber. We construct a monotone strategy $\otimes_r\sigma$ that is winning against all isolating prudent strategies for r robbers which is sufficient by Proposition 4.23. We define the strategy as a memory strategy

$$\otimes_r\sigma : \mathcal{M} \times (2^V \times 2^V) \to 2^V$$

which takes current memory state ζ and position (U, R) and yields a next set of vertices U' to be occupied by cops. The memory states have the form

$$\zeta = \{\rho_1, \dots, \rho_s\}$$

with $s \leq r$, where each ρ_j is a history in the k cops and single robber game on G that ends in a cops' position $\mathrm{last}(\rho_j) = (W_j, b_j)$. During the construction, we maintain the following invariants on the memory states and the corresponding positions, that means, we construct the strategy $\otimes_r\sigma$ and the memory update function δ_{up} in such a way, that for each memory state ζ and each position (U, R) which are consistent with $\otimes_r\sigma$ and δ_{up}, the following conditions hold.

(Cons) Any history ρ_j is consistent with σ

(Robs) The b_j are pairwise distinct and $R = \{b_1, \dots, b_s\}$

(Cops) $U \supseteq \bigcup_{j=1}^{s} W_j$ and $|U| \leq r \cdot k$

(Area) $\mathrm{Reach}_{G-U}(b_j) = \mathrm{Reach}_{G-W_j}(b_j)$

Notice that the property which allows us to actually play against the several robbers independently is **(Area)**: For robber b_j the situation is just like he was alone on the graph and history ρ_j has been played against him. Of course, he can split and spawn several copies, but in this case, all these copies have to move in such a way that the robbers will be separated again afterwards. Also notice that the inclusion $\subseteq$ in **(Area)** is trivial by **(Cops)**.

First, for any move $\perp \to (\emptyset, R)$ of the robbers to a set $R = \{b_1, \dots, b_s\}$, we set $\delta_{\mathrm{up}}(\perp, (\emptyset, R)) = \{(\emptyset, b_1), \dots, (\emptyset, b_s)\}$. (Remind that we consider plays here from a predetermined initial position $\perp$ and we use $\perp$ also as the unique initial memory state.) Now consider some position (U, R) where it is the cops' turn and some memory state $\zeta = \{\rho_1, \dots, \rho_l\}$ such that all the invariants hold. We define the next move of the cops in such a way, that they play only against robber b_s. We could, completely analogously, play against all robbers $b_1, \dots, b_s$ simultaneously, but it might help to picture the dynamics in the game more clearly if we play only against a single robber at each time.

Let $W'_s := \sigma(\rho_s)$. We define

$$U' = \otimes_r\sigma(\zeta, (U, R)) := \bigcup_{j=1}^{s-1} W_j \cup W'_s.$$

Using monotonicity of σ and **(Area)**, we have

$$\mathrm{Reach}_{G-U\cap U'}(b_s) \subseteq \mathrm{Reach}_{G-W_s\cap W'_s}(b_s) = \mathrm{Reach}_{G-W_s}(b_s) = \mathrm{Reach}_{G-U}(b_s).$$

Moreover, since for any $j < s$ we have $W_j \subseteq U \cap U'$ it follows by **(Area)** that

$$\mathrm{Reach}_{G-U\cap U'}(b_j) \subseteq \mathrm{Reach}_{G-W_j}(b_j) = \mathrm{Reach}_{G-U}(b_j).$$

Therefore, the move $(U, R) \to (U, U', R)$ is monotone.

Now consider any possible next move $(U, U', R) \to (U', R')$ of the robbers. Notice that since the robbers use an isolating prudent strategy, for any $j < s$ we either have $b_j \in R'$ or there is no $b \in R'$ with $b \in \mathrm{Reach}_{G-U\cap U'}(b_j)$, that means, the robber b_j has either remained motionless or he has been taken from the graph. As a consequence, any robber $b \in R' \setminus \{b_1, \dots, b_s\}$ has been spawned from b_s. We define the new memory state $\zeta' = \delta_{\mathrm{up}}(\zeta, (U', R'))$ as follows.

Consider any robber $b \in R' \setminus \{b_1, \ldots, b_{s-1}\}$. Then $b \in \mathrm{Reach}_{G-U\cap U'}(b_s) \subseteq \mathrm{Reach}_{G-W_s\cap W'_s}(b_s)$, so $\rho = \rho_s(W_s, W'_s, b_s)(W'_s, b)$ is a history in the cops and single robber game on G and, by construction, ρ is consistent with σ. The memory state ζ' is obtained from ζ by two operations. First, we add such histories ρ extended in this way from ρ_s, such that all $b \in R' \setminus \{1, \ldots, s-1\}$ are covered. Notice that the set of all these histories is empty if the robbers' answer to the cops' move is to simply *remove* b_s from the graph without spawning any copies from it. (Which, in particular, is the case if b_s is captured.) Second, we delete, for any $j \leq s$ with $b_j \notin R$ the history ρ_j. Let

$$\zeta' = \{\rho'_1, \ldots, \rho'_{s'}\} \quad \text{with} \quad \mathrm{last}(\rho'_j) = (W'_j, b'_j).$$

Now we have to show that all invariants hold. We have already seen that **(Cons)** holds for all newly inserted histories and it holds by induction for all ρ_j with $j < s$ that are still in ζ'. Moreover, **(Robs)** and **(Cops)** hold by definition of U' and ζ'. To prove **(Area)**, first consider any b_j with $j < s$ and $b_j \in R'$ and let b_j belong to history number j' in ζ'. As the robbers play an isolating strategy and G is *undirected*, we have $\mathrm{Reach}_{G-U}(b_j) \cap \mathrm{Reach}_{G-U}(b_s) = \emptyset$ and since σ is thrifty, using **(Area)** we obtain $U' \setminus U = W'_s \setminus W_s \subseteq \mathrm{Reach}_{G-W_s}(b_s) = \mathrm{Reach}_{G-U}(b_s)$, so $\mathrm{Reach}_{G-U}(b_j) \cap (U' \setminus U) = \emptyset$. Hence, $\mathrm{Reach}_{G-U'}(b_j) \supseteq \mathrm{Reach}_{G-U}(b_j) = \mathrm{Reach}_{G-W_j}(b_j) = \mathrm{Reach}_{G-W'_{j'}}(b'_{j'})$.

Now consider any robber $b \in R' \setminus \{b_1, \ldots, b_{s-1}\}$ and let b belong to history number j in ζ' and assume that there is some $u \in U' \cap \mathrm{Reach}_{G-W'_j}(b) = U' \cap \mathrm{Reach}_{G-W'_s}(b)$. First assume that $u \in U$. Due to monotonicity of σ we have $\mathrm{Reach}_{G-W'_s}(b) \subseteq \mathrm{Reach}_{G-W_s}(b_s)$, so $u \in U \cap \mathrm{Reach}_{G-W_s}(b_s)$ which contradicts **(Area)**. Therefore, $u \in U' \setminus U = W'_s \setminus W_s$ and hence, $u \in W'_s \cap \mathrm{Reach}_{G-W'_s}(b)$ which also is a contradiction. We conclude that $U' \cap \mathrm{Reach}_{G-W'_s}(b) = \emptyset$ which yields that $\mathrm{Reach}_{G-W'_j}(b'_j) = \mathrm{Reach}_{G-W'_s}(b) \subseteq \mathrm{Reach}_{G-U'}(b)$.

Since we obtain the maximum number $r \cdot k$ of cops that are used immediately from **(Cops)**, it only remains to show that $\otimes_r\sigma$ is actually winning for the cops. Consider any play $\pi = (U_0, R_0)(U_0, U_1, R_1)(U_1, R_2)\ldots$ that is consistent with $\otimes_r\sigma$. It suffices to show that for all $\lambda \in \mathbb{N}$ with $R_\lambda \neq \emptyset$, there is some $\gamma > \lambda$ with $\mathrm{Reach}_{G-U_\gamma}(R_\gamma) \subsetneq \mathrm{Reach}_{G-U_\lambda}(R_\lambda)$. If this is not the case, then, since $\otimes_r\sigma$ is monotone, there is a $\lambda \in \mathbb{N}$ such that $R_\lambda \neq \emptyset$ and $\mathrm{Reach}_{G-U_\gamma}(R_\gamma) = \mathrm{Reach}_{G-U_\lambda}(R_\lambda)$ for all $\gamma > \lambda$. Since the robbers follow a prudent strategy, they do not move after round λ, that is, $R_\gamma = R_\lambda$ for all $\gamma > \lambda$. By **(Robs)** it follows that $\zeta_\lambda \neq \emptyset$, where $\zeta_\lambda = \{\rho_1, \ldots, \rho_s\}$ denotes the memory state at round λ. According to the definition of $\otimes_r\sigma$ the cops play, in each round $\gamma > \lambda$, against the robber $b_s \in R$ according to σ. But since $\otimes_r\sigma$ never places a cop inside $\mathrm{Reach}_{G-U}(R) \supseteq \mathrm{Reach}_{G-U}(b_s)$ after round λ, by **(Area)** it follows that σ never places a cop inside $\mathrm{Reach}_{G-W_s}(b_s)$. Therefore, the unique extension of ρ_s to the σ-play where the robber makes himself comfortable on b_s forever, is won by the robber – this contradicts the premise that σ is a winning strategy. $\square$

It should be mentioned that the reason why the invariant **(Cops)** requires only $\supseteq$ instead of $=$ is merely that, after a move of the cops, some of the histories ρ_j are deleted from ζ. However, we cannot instantly take the corresponding cops in U away from the graph, because it is not the cops' turn. Instead, these cops will be taken from the graph during the next cops' move implicitly, where we define U only with respect to the histories that are still in ζ. Invariant **(Area)** guarantees that no harm is done by this. Of course, there are other technical possibilities to deal with this, but modifying **(Cops)** in this way seems to be the easiest. With regard to the proof of Theorem 4.22, however, one should keep in mind that the intention behind **(Cops)** is actually to have exactly the cops from the individual histories on the graph.

Let us discuss why the proof of Proposition 4.24 fails for directed graphs. We have already highlighted the argument where we have actually used that G is undirected: We were able to infer $\mathrm{Reach}_{G-U}(b_j) \cap \mathrm{Reach}_{G-U}(b_s) = \emptyset$ for all $j < s$ from the fact that the robbers use an isolating strategy. So, as long as the strategy σ is thrifty, we can play against the robber b_s without placing any cops inside the reachability areas of other robbers. This is not the case of directed graphs! Even if any two robbers $b_j, b_{j'}$ are topologically incomparable, they may share a common area that is reachable for them and the strategy σ may prescribe to place cops inside this area against these robbers. Of course, these placements are made outside the strongly connected component of b_j in $G - W_j$ but, as we have seen in Section 4.1, this is unavoidable in general!

Now, the problem that arises here is the following: Assume we place cops against a robber b_j on vertices that are also reachable from another robber $b_{j'}$. When we continue to play against b_j according to the given winning strategy σ, then at some point, σ may prescribe to take these cops from the graph away. Since σ is monotone, the robber b_j will not be able to cause nonmonotonicity at any of these vertices. However, σ does *not* guarantee that $b_{j'}$ (which corresponds to a different σ-play!) cannot cause nonmonotonicity. So, in the game against multiple robbers, nonmonotonicity may occur if we follow σ strictly.

On the other hand, it might be possible that monotone winning strategies for the cops can be turned into a normalform such that this problem does not occur. However, such a normalform would be very similar to the one that we have mulled over after the proof of Proposition 4.16 and as we have mentioned there, we do not any such normalform. The point here is that the multiple robbers scenario allows us to tackle this problem at a level where the interferences are present more directly. Before we show how this problem can be actually solved in this scenario, let us consider the case of non-splitting robber which also helps to clarify the particular properties and difficulties of the general case. However, the techniques used here will not be of much help for the general case, so we give only a brief sketch of the proof.

Proposition 4.25. $\mathrm{dw}_r(G) \leq r \cdot \mathrm{dw}(G)$ *for non-splitting robbers.*

Proof. (Sketch) We proof this by induction on r, simultaneously for all graphs G. The case $r = 1$ is trivial, so let $r > 1$ and let σ be a monotone winning strategy for $\mathrm{dw}(G)$ cops against a single robber on G. As we have mentioned, we cannot assume that the robbers play an isolating strategy but it helps to assume that σ depends only on the strongly connected component of the robber instead of his precise position. Then we can treat all robbers that are in the same strongly connected component of $G - U$ as a single robber, that is, we assume that all those robbers actually are on the same vertex. We describe a monotone winning strategy for $r \cdot \mathrm{dw}(G)$ cops against r non-splitting robbers on G.

We start as follows. Given a position (U, R) in the game, as long as all robbers are on the same vertex u, we just play according to σ. Now assume that, at a position (U, R), some robbers have separated and consider the topologically minimal vertex u in $G - U$ that is occupied by a robber. Moreover, let $\tilde{G}$ denote the strongly connected component of $G - U$ with $u \in \tilde{G}$. Since some robbers have split up, there are at most $r - 1$ robbers in $\tilde{G}$, so by induction hypothesis, $(r-1) \cdot \mathrm{dw}(\tilde{G}) \leq (r-1) \cdot \mathrm{dw}(G)$ cops have a monotone winning strategy $\tilde{\sigma}$ against all the robbers that are in $\tilde{G}$. We proceed as follows.

We play according to $\tilde{\sigma}$ on $\tilde{G}$ using at most $r \cdot k$ cops and while we do this, the at most k cops that occupy the vertices in U stand idle. Since we have chosen $\tilde{G}$ as a topologically minimal strongly connected component in $G - U$ and, moreover, the robbers cannot split, no robbers from outside $\tilde{G}$ will be able to reach $\tilde{G}$. So we can be sure that after a finite number of steps, $\tilde{G}$ will be conquered by the cops, that means, all robbers from $\tilde{G}$ are either captured or have left the component and are now on vertices outside $\tilde{G}$. At this point, we simply remove all cops except those on U from the graph. Notice that $\tilde{\sigma}$ is a winning strategy for the cops on $\tilde{G}$, so by playing $\tilde{\sigma}$ on $\tilde{G}$ no cops are placed anywhere outside of $\tilde{G}$. Moreover, by topological minimality of $\tilde{G}$ in $G - U$, taking all cops except those in U from the graph is a monotone move.

Now we continue to proceed as before: As long as all robbers are on the same vertex, we play according to σ and as soon as they split up, we repeat the procedure above. It is obvious that, in this way, finally all robbers are captured and as we have already seen, all moves are monotone. Since we have used only $r \cdot k$ cops in total, the proof is completed. □

In this case, it is rather obvious why the proof can not be readily adapted to the general case where the robbers can split: As soon as the robbers from outside $\tilde{G}$ are taken from the graph and used to split up the robbers inside $\tilde{G}$ into r many robbers, we cannot continue to play $\tilde{\sigma}$ on $\tilde{G}$ like this because it is a strategy against at most $r-1$ robbers. On the other hand, if we try to play on $\tilde{G}$ according to σ like before, then we run into the same problem that we have described above: σ may prescribe to place cops on vertices outside of $\tilde{G}$ which may still be reachable from robbers that are not in $\tilde{G}$.

4.4.3 General Case

Let $G = (V, E)$ be a graph and let σ be a thrifty monotone winning strategy for k cops against a single robber on G.[29]

We construct a monotone strategy $\ominus_r\sigma$ for $r \cdot k$ cops that is winning against all isolating prudent strategies for r splitting robbers and then use Proposition 4.23. As in the proof of Proposition 4.24, we describe $\ominus_r\sigma$ as a memory strategy

$$\ominus_r\sigma : \mathcal{M} \times (2^V \times 2^V) \to 2^V.$$

We first give a brief description of the proof. In particular, we introduce and explain the memory states and the associated invariants that we will maintain, which follow the same scheme as in the proof of Proposition 4.24 but are much less straightforward.

To catch r splitting robbers on G simultaneously and monotonously, we also apply the concept of concurrent graph searching of which we have used different incarnations in the proofs of Proposition 4.16 and Proposition 4.24. That means, we consider up to r different σ-histories $\rho_1, \ldots, \rho_s$ such that all robbers in R are covered by the robbers in these histories. So, we also have the invariant **(Cons)**:

(Cons) Any history ρ_i is consistent with σ.

Our previous applications of concurrent graph searching have been rather straightforward: In Proposition 4.16 we just neglected monotonicity, while in the setting of Proposition 4.24, the individual robbers where completely separated from each other so that no interferences did occur. However, it is helpful to keep the basic scheme of the proof of Proposition 4.24 in mind.

Remind what the situation is like in the directed case: To guarantee monotonicity, we cannot just follow the strategy σ directly in each play, because it is possible that we place cops against a robber b on vertices that are also reachable from another robber b'. When we continue to play against b according to σ, then at some point, σ may prescribe to take these cops from the graph away. Since σ is monotone, b cannot cause nonmonotonicity but b' may still be able to reach these vertices. So, in the game against multiple robbers, nonmonotonicity may occur if we follow σ strictly.

The leading idea of our solution is to *omit* placing cops on such vertices. Of course, it is immediately clear that this will not work either, but we start from this basic approach and we will refine it until it works. An obvious problem that occurs is the following: If we omit placing certain cops against a robber, then this robber may cause nonmonotonicity by going through these vertices because monotonicity of σ relies on the fact that these cops are placed. Moreover, by going through one of these vertices where cops have been omitted, the robber may cause a situation that is not consistent with σ. That means, the strategy σ will not have a good answer to this situation, so we do not know how to use σ to go on in the game against r robbers.

The most important step towards solving these problems is to *linearize* the σ-histories $\rho_1, \ldots, \rho_s$ that we maintain, that means, we write the memory states as sequences $\rho_1, \ldots, \rho_s$ and we have an additional invariant **(Lin)**:

(Lin) $\rho_1 \sqsubset \rho_2 \sqsubset \ldots \sqsubset \rho_s$.

[29] Just to simplify the notation a little, we sometimes use σ as positional strategy. For the argumentation, however, this is completely dispensable.

We maintain this by, primarily, always playing against the robber b_s from the latest history ρ_s (which is now an important feature of the proof rather than a technical simplification) and we place cops only on vertices that are not reachable from earlier robbers, that is, robbers from earlier histories ρ_j with $j < s$. Therefore, by prudence, only robbers that are spawned from b_s will respond to such a move and we will ensure two properties:

(1) for any of these robbers that moves in such a way that his position is not consistent with ρ_s, it is consistent with some earlier history ρ_i, $i < s$

(2) any vertex where b_s could cause non-monotonicity due to cops that we have omitted, is still occupied by a cop from some earlier history ρ_i, $i < s$

Notice that (2) is quite contrary to (Area) from the proof of Proposition 4.24: (Area) guaranteed us that the area available to an individual robber is completely determined by the cops from his own history, while condition (2) directly implies that this area will be restricted by cops from other histories as well. It will be important, however, that this is only the case for cops from *earlier* histories.

Since the assignment of the robbers from R to the histories $\rho_1, \dots, \rho_s$ is here not as canonical as in the proof of Proposition 4.24 but has to be hand-picked, it is more convenient not to have an individual history for each robber. Rather, for each $i < s$, we do not store the last move of the robber in ρ_i and we associate with ρ_i a set $R_i \subseteq R$ of robbers that can be consistently attached to ρ_i. That means, for any $b \in R_i$, extending the history ρ_i by the move of the robber to the vertex b still yields a σ-history. So we add the components R_i to the elements of the memory states and maintain the following invariant:

$$\textbf{(Robs)}\quad \text{The sets } R_i \text{ are pairwise disjoint and } R = \bigcup_{i=1}^{s} R_i.$$

Now, as to the relationship between the cops in the histories ρ_i and the cops U in the game against r robbers, we have to be particularly careful. We have mentioned that we avoid to place cops that are still reachable from earlier robbers, so we will not have $U = \bigcup_{i=1}^{s} W_i$ but rather $U = \bigcup_{i=1}^{s} U_i$ for sets $U_i \subseteq W_i$, where $W_i \setminus U_i$ are the vertices where we have omitted to place cops. However, the set of exactly those vertices that are reachable from earlier robbers is, in a sense, too *dynamic*: If certain robbers are taken from the graph, some vertices that have been omitted previously might then not be reachable from earlier robbers anymore, in which case they should be occupied by cops immediately. But placing these cops in such a situation does not correspond to a σ-move in general, so if (*later*) robbers respond to this move, their new position might not be consistent with any of our σ-histories. Instead, we add another component to the elements of the memory states: For each $i < s$, we have a set O_i of vertices that must be omitted due to the history ρ_i, and we will maintain two additional invariants which, in particular, guarantee the properties (1) and (2) as formulated above. The invariant for the cops' vertices is now clear:

$$\textbf{(Cops)}\;\; U = \bigcup_{i=1}^{s} \left(W_i \setminus O^{i-1}\right) \quad \text{where } O^i = \bigcup_{j=1}^{i} O_j.$$

So, a memory state is now a sequence

$$\zeta = (\rho_1, R_1, O_1), \dots, (\rho_{s-1}, R_{s-1}, O_{s-1}), \rho_s.$$

where the ρ_i are as before, and R_i, O_i are sets of vertices of G. Clearly, for ρ_s we don't have a set O_s and, moreover, as we pursue only one robber at a time, at most the robber b_s is associated with ρ_s. The invariants (Robs) and (Cops) establish the connection between what actually happens on the graph while playing against r robbers and the σ-histories which we maintain in the memory state. The technically most difficult part of the proof is now to define and maintain appropriate invariants for the sets O_i.

To see how the sets O_i will be defined, let us denote $\mathrm{last}(\rho_i) = (W_i^{-1}, W_i, b_i)$, and assume that a new history ρ_{s+1} arises which we have to maintain, that means, we have played against the latest robber b_s according to σ and subsequently, b_s has split into several copies R_s. Then we store ρ_s and associate all but one of the new robber copies from R_s with ρ_s, which is clearly consistently possible. The distinguished robber copy $b \in R_s$ which we do not associate with ρ_s is the one which we pursue further. The set O_s of vertices which we attach to ρ_s is precisely the set of vertices reachable by the robbers associated with ρ_s, but *in the graph $G - W_s$*. Essentially, the reason is that the set W_s is less dynamic than the set U of vertices actually occupied by cops in the game against r robbers, which makes it easier to handle the set O_s.

Notice that, since σ is monotone, reachability in $G - W_s$ is the same as if we added all cops from earlier histories (which have also been placed in ρ_s previously due to *(Lin)*, but have already been taken from the graph). Moreover, when we add a new history due to robbers that have split, it will always be the latest history, so there are no cops from later histories that would have to be considered. This indicates that $O_s = \mathrm{Reach}_{G-W_s}(R_s \setminus \{b\})$ is not too bad a choice. However, this set is still too dynamic to maintain equality in every move (for the reason explained above). However, what we actually need here is that $R_s \setminus \{b\} \subseteq O_s$ and, moreover, that O_s is closed under reachability in $G - W_s$. Then, if a robber from $R_s \setminus \{b\}$ leaves the set O_s, we can be sure that he uses a vertex that we have omitted due to an earlier history so that the set $O^s = \bigcup_{i=1}^{s} O_i$ forms a trap for the robber in the (actual) graph $G - U$. This makes it possible to guarantee condition (1) as formulated above. Hence, we use these as the constitutive properties of the set O_s and maintain them as invariant:

(Omit) For all $i \in \{1, \ldots, s-1\}, R_i \subseteq O_i = \mathrm{Reach}_{G-W_i}(O_i)$.

Finally, to guarantee that any robber which we have associated with ρ_s is in fact consistent with ρ_s, we add the following invariant:

(Ext) For all $i \in \{1, \ldots, s-1\}, O_i \subseteq \mathrm{Reach}_{G-W_i^{-1}}(b_i)$.

Now, assume that for some history ρ_i all the robbers from R_i associated with ρ_i move and have to be associated with other histories, so there are no robbers left that can be consistently associated with ρ_i, that is, $R_i = \emptyset$. In this case, if we would just continue playing against the latest history, we might end up with more than r histories in total, because the robbers that were previously in R_i may now be used to split the latest robber into several copies. Clearly, we have to avoid this to be sure to keep to our budget of $r \cdot k$ cops.

Instead, in this case, we continue to play the history ρ_i, that means, we extend ρ_i to $\tilde{\rho}_i$, by the next move according to σ. To determine this move, we need the current history ρ_i and the next vertex $\tilde{b}_i$ of the robber. Notice that $\tilde{b}_i$ is not necessarily occupied by a robber in the game against r robbers, but the information about the next vertex $\tilde{b}_i$ of the robber can be determined from the next history ρ_{i+1}: $\tilde{b}_i$ is the unique vertex such that there is a suffix of ρ_{i+1} with $\rho_{i+1} = \rho_i(W_i, \tilde{b}_i)\eta$. Notice, on the other hand, that playing against a robber from R_{i+1} which is *associated* with ρ_{i+1} would not necessarily preserves linearity of the histories that we maintain.

Clearly, to maintain (Omit) and (Ext), we have to update the set O_i. In particular, we have to ensure, that it is still a subset of the area $\mathrm{Reach}_{G-W_i}(\tilde{b}_i)$ available to $\tilde{b}_i$ in the graph $G - W_i$, before the cops move to $\sigma(W_i, \tilde{b}_i)$. For this, we simply intersect the set O_i with $\mathrm{Reach}_{G-W_i}(\tilde{b}_i)$ and remove all vertices from $\sigma(W_i, \tilde{b}_i)$. In the more technical part of the proof we will see that, since σ is monotone, this also guarantees that (Omit) holds again. Now, dependent on the response of the robbers, afterwards there may be robbers associated with $\tilde{\rho}_i$ again. If not, then we extend $\tilde{\rho}_i$ by another move, and so on, until some robbers are associated with the current history or we reach the next history, that is, $\tilde{\rho}_i = \rho_{i+1}$. In that case, $(\tilde{\rho}_i, \emptyset, \tilde{O}_i)$ and $(\rho_{i+1}, R_{i+1}, O_{i+1})$ are merged. Again, the details of this will be presented in the technical proof.

It is helpful to notice the following: If we continue earlier histories ρ_i like we have just described, then we possibly fill certain gaps, by which we mean that we place cops on vertices that have been omitted in *later* histories ρ_j, $j > i$. On the other hand, the update of O_i excludes only vertices from $\sigma(W_i, \tilde{b}_i)$ and vertices not in $\sigma(W_i, \tilde{b}_i)$ which are also not reachable from $\tilde{b}_i$ in $G - W_i$. But on those vertices, by monotonicity and thriftiness, no cop is placed in any extension of ρ_i. So, all the gaps that can be filled in later histories, are filled while we continue ρ_i. Therefore, when playing any history ρ_j, $j \le s$, we will never place cops on vertices that have previously been omitted while playing ρ_j!

Altogether, it can be shown that by following this strategy (under the assumption that the robbers use an isolating prudent strategy) all robbers are finally caught. Notice that in the description above, we did not mention this assumption on the robbers' strategy and in fact, prudence is utilized only in the more technical part of the proof and is not essential for the strategy $\ominus_r\sigma$. On the other hand, the strategy $\ominus_r\sigma$ as given above relies on the assumption that any two different robbers are topologically incomparable: If b_s and a robber b from, say, R_{s-1} were topologically comparable, then any vertex that is reachable for b_s would also be reachable from b, so we would not place any cops against b_s. (Notice, however, that this is clearly just a matter of *description* of the strategy and not a conceptual issue, cf. the proof of Proposition 4.23.)

Assuming an isolating strategy for the robbers guarantees that such a situation cannot occur. However, since the sets O_i are not simply the sets of vertices that are reachable from earlier robbers but just fulfill certain properties formulated as (Omit) and (Ext), we have to maintain the appropriate property as an additional invariant. Conveniently, this one can be considered separately, as it can be proved from the other invariants but is not further intertwined with them.

(Progress) For $i \in \{2, \dots, s-1\}, R_i \cap O^{i-1} = \emptyset$ and $b_s \notin O^{s-1}$.

Notice that since σ is winning, if the robber does not move, it will finally prescribe to place a cop on the vertex currently occupied by the robber. (Progress) tells us that we will not omit this cop-placement so, as long as the robbers use a prudent strategy, it guarantees progress of the strategy $\ominus_r\sigma$ against r robbers.

Now we provide a technical proof of Theorem 4.22 that follows the description above. In particular, we start from a thrifty monotone winning strategy σ for k cops against a single robber and we construct a monotone strategy $\ominus_r\sigma$ for $r \cdot k$ cops that is winning against all isolating prudent strategy for at most r splitting robbers. First, let us assemble all the invariants that we have introduced above, along with some further notation that we need.

We define the memory strategy

$$\ominus_r\sigma : (2^V \times 2^V) \times M \to 2^V$$

with memory states

$$\zeta = (\rho_1, R_1, O_1), \dots, (\rho_{s-1}, R_{s-1}, O_{s-1}), \rho_s$$

where ρ_s is a history of the cops and (single) robber game on G. Moreover, for $i \in \{1, \dots, s-1\}$, ρ_i is a history of the cops and (single) robber game on G which ends in a robbers' position and R_i, O_i are subsets of V. We construct the strategy $\ominus_r\sigma$ and the memory such that, for any memory state $\zeta = (\rho_1, R_1, O_1), \dots, (\rho_{s-1}, R_{s-1}, O_{s-1}), \rho_s$ and any position (U^{-1}, U, R) (where $U^{-1} = U$, if (U^{-1}, U, R) is a cops' position) which are consistent with $\ominus_r\sigma$, the following invariants hold. To describe them, let

- $\mathrm{last}(\rho_i) = (W_i^{-1}, W_i, b_i)$, for $i \in \{1, \dots, s-1\}$
- $\mathrm{last}(\rho_s) \in \{(W_s, b_s), (W_s^{-1}, W_s, b_s)\}$
- $X^i = \bigcup_{j=1}^{i} X_j$ for $X \in \{O, W, U, R\}$
- $U_i = W_i \setminus O^{i-1}$ for $i \in \{1, \dots, s\}$
- $R^{>i} = \bigcup_{j=i+1}^{s-1} R_j \cup \{b_s\}$ for $i \in \{1, \dots, s-1\}$

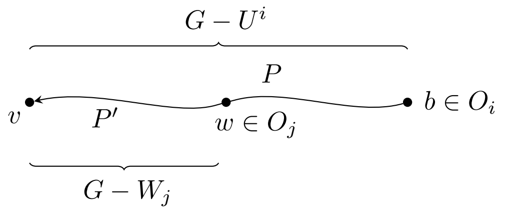


Figure 4.7: $v \in \mathrm{Reach}_{G-W_j}(O_j)$ implies $v \in O_j$ by (Omit)

- $R_s = \{b_s\}$, if $b_s \in R$ and $R_s = \emptyset$, else.

Invariants.

(Cons) Any history from ζ is consistent with σ.

(Lin) $\rho_1 \sqsubset \rho_2 \sqsubset \ldots \sqsubset \rho_s$.

(Robs) The sets R_i are pairwise disjoint and $R = \bigcup_{i=1}^{s} R_i$.

(Cops) $U = \bigcup_{i=1}^{s} U_i$.

(Omit) For all $i \in \{1, \ldots, s-1\}$, $R_i \subseteq O_i = \mathrm{Reach}_{G-W_i}(O_i)$.

(Ext) For all $i \in \{1, \ldots, s-1\}$, $O_i \subseteq \mathrm{Reach}_{G-W_i^{-1}}(b_i)$.

In addition to (Cops), our construction also relies on the property that, if (U^{-1}, U, R) is a cops' position and $b_s \in R$, then $\mathrm{last}(\rho_s) = (W_s, b_s)$. However, this is merely a technical detail, so we do not treat this as an invariant explicitly. It will always be clear from the construction that this holds.

Basic Implications. We start with some properties that can be easily derived from the invariants and which we will use frequently in the proof.

The first part of (Omit) together with (Cons) and (Ext) guarantees that each robber that is associated with ρ_i is also consistent with ρ_i which is formulated more precisely in the following lemma.

Lemma 4.26. *For all $b \in R_i$, $\rho_i \cdot (W_i, b)$ is a σ-history.*

Proof. By (Omit), $b \in O_i$ so, by (Ext), b is reachable from b_i in $G - W_i^{-1}$ and as ρ_i is a σ-history according to (Cons), $\rho_i \cdot (W_i, b)$ is a σ-history as well. □

Given the (simple but essential) property from Lemma 4.26, the following lemma can be easily obtained from monotonicity of σ, using (Cons) and (Lin). It is important to observe this as it ensures that the sets W_j of vertices occupied by cops according to earlier histories ρ_j, $j < i$ do not affect the set O_i.

Lemma 4.27. $\mathrm{Reach}_{G-W_i}(b) = \mathrm{Reach}_{G-W^i}(b)$ *for all $b \in R_i$.*

Proof. Let $b \in R_i$ and assume that $\mathrm{Reach}_{G-W_i}(b) \subseteq \mathrm{Reach}_{G-W^i}(b)$ doe not hold. Then there must be some vertex $u \in W^{i-1} \setminus W_i$ such that $u \in \mathrm{Reach}_{G-W_i}(b)$. Now for $j \in \{1, \ldots, i-1\}$ with $u \in W_j$ we have $\rho_j \sqsubset \rho_i$ due to (Lin). Moreover, by Lemma 4.26, $\rho_i \cdot (W_i, b_i)$ is consistent with σ. But as ρ_j is consistent with σ as well due to (Cons), $\mathrm{Reach}_{G-W_i}(b) \cap W_j \neq \emptyset$ contradicts monotonicity of σ. □

The following lemma is one of the key arguments for monotonicity and can be derived from (Omit) without using other invariants. We have already mentioned this very important idea in the description above: (Omit) guarantees that O_i is closed under reachability in $G - W_i$. So, when a robber is in O_i and leaves the set in the graph $G - U^i$ then he has to use a vertex that we have omitted due to some earlier history, that is, a vertex from the set O^{i-1}. Hence, O^i is a trap for the robbers in $G - U^i$, and thus in $G - U$.

Lemma 4.28. $\mathrm{Reach}_{G-U^i}(O_i) \subseteq O^i$.

Proof. Let $v \in \mathrm{Reach}_{G-U^i}(O_i)$ and let P be a path from some $b \in O_i$ to v in the graph $G - U^i$, cf. Figure 4.7. If $v \in \mathrm{Reach}_{G-W_i}(O_i)$, by (Omit) we immediately have $v \in O_i \subseteq O^i$, so assume that $v \notin \mathrm{Reach}_{G-W_i}(O_i)$. Then there must be some vertex from W_i that lies on P and we consider the minimal $l \leq i$ such that $P \cap W_l \neq \emptyset$ and some vertex $w \in P \cap W_l$. Since P is a path in $G - U^i$ we have $P \cap U^i = \emptyset$, so we obtain $w \notin U^i \supseteq U_l$ and, by definition of U_l, this yields $w \in O^{l-1}$. (Notice that $w \in W_l \setminus U_l$ yields $l > 1$!) Let $j < l$ with $w \in O_j$. Due to minimality of l we have $P \cap W_j = \emptyset$ and since $w \in P$, the vertex v is reachable from w in G via some subpath $P' \subseteq P$. So $v \in \mathrm{Reach}_{G-W_j}(w) \subseteq \mathrm{Reach}_{G-W_j}(O_j)$ and by (Omit), $\mathrm{Reach}_{G-W_j}(O_j) = O_j \subseteq O^i$. □

Notice that with the same arguments we can show that $\mathrm{Reach}_{G-U}(O^{s-1}) = O^{s-1}$ which demonstrates the following: If $b_s \in R$, then clearly the robber b_s may have vertices available in $G - U$ that do not belong to O^{s-1} but, as soon as he runs into O^{s-1}, he is also trapped there.

Finally, the fact that the reachability area of a robber is not restricted by cops from later robbers can be easily obtained from Lemma 4.28.

Lemma 4.29. $\mathrm{Reach}_{G-U^i}(b) = \mathrm{Reach}_{G-U}(b)$ *for all* $b \in R_i$.

Initial Move. To simplify the first step a little, let us assume, that the graph G is strongly connected, so the robbers do not split in the first move. That is, the first move is of the form $\perp \to (\emptyset, \{b\})$ for some $b \in V$ and the associated memory state is $\zeta = (\emptyset, b)$. All invariants are obviously satisfied.

Now consider some game position (U, R) where it is the cops player's turn and some memory state ζ such that all invariants are satisfied. We will define the cops next move according to (U, R) and ζ and then we consider any possible answer of the robbers just like in the proof of Proposition 4.24. However, to disentangle the rather technical proof of the invariants, we make a memory update after the cops move as well as after the robbers move and we prove that the invariants hold in both cases separately.

Move of the Cops. In the following, we define the new set

$$U' = \ominus_r \sigma((U, R), \zeta)$$

of vertices occupied by cops and the new memory state

$$\zeta' = (\rho'_1, R'_1, O'_1), \ldots, (\rho'_{s'-1}, R'_{s'-1}, O'_{s-1}), \rho'_{s'}.$$

Case I: $b_s \notin R$
That means, the latest robber which we had chosen to play against has been either caught or has returned to an earlier situation. Hence, if $s = 1$, all the robbers are caught and the cops have won. Otherwise, we set $U' = U^{s-1}$, that means we remove all the cops from the latest history. For the memory update, we distinguish two cases:

- $R_{s-1} = \emptyset$:

  $\zeta \to \zeta'$: delete ρ_s;
  replace $(\rho_{s-1}, R_{s-1}, O_{s-1})$ by $\rho_{s-1} \cdot (W_{s-1}, b_s)$.

  Notice that $R_{s-1} = \emptyset$ implies that ρ_{s-1} could in fact also be deleted but to make the proof of the invariants somewhat easier, we don't delete several histories at the same time. So we delete only ρ_s and we extend ρ_{s-1} by (W_{s-1}, b_s), adding information about a (consistent) last robber's move. After that, $\rho_{s-1} \cdot (W_{s-1}, b_s)$ is the new latest history and since the robbers use a prudent strategy, they will not react to this cops' move. So after the next cops' move, this history will also be deleted.

- $R_{s-1} \neq \emptyset$:

  Choose some robber $b \in R_{s-1}$ and define

  $$\tilde{O}_{s-1} := \mathrm{Reach}_{G-W_{s-1}}(R_{s-1} \setminus \{b\}).$$

  $\zeta \to \zeta'$: replace $(\rho_{s-1}, R_{s-1}, O_{s-1})$ by $(\rho_{s-1}, R_{s-1} \setminus \{b\}, \tilde{O}_{s-1})$;
  replace ρ_s by $\rho_{s-1} \cdot (W_{s-1}, b)$.

  So we select one of the robbers from R_{s-1} that we want to pursue next and we extend the history ρ_{s-1} by this robbers' move (W_{s-1}, b). Moreover, with the history ρ_{s-1} we associate the remaining $R_{s-1} \setminus \{b\}$. Consequently, we also have to define a new set O_{s-1} which should now contain only vertices reachable from the robbers in $R_{s-1} \setminus \{b\}$ (in the graph $G - W_{s-1}$).

Case II: $b_s \in R$.

Case II.1: There is some $i \in \{1, \ldots, s-1\}$ such that $R_i = \emptyset$.
As explained in the description above, in this case we do not play against the latest robber, as this might be critical to the total number of cops we use. Instead, we continue to play ρ_i, or rather, to make the choice definite, the latest one of all histories which have no robbers associated with them. For this, consider the next robbers' move according to ρ_{i+1} (notice that $i < s$) which is given by the unique vertex $\tilde{b}_i$ such that there is a suffix η of ρ_{i+1} with $\rho_{i+1} = \rho_i(W_i, \tilde{b}_i)\eta$. Now we distinguish three more cases.

(a) $\rho_{i+1} = \rho_i \cdot (W_i, \tilde{b}_i) = \rho_s$

   We set $U' := U$ and define the memory update as follows:
   $\zeta \to \zeta'$: delete (ρ_i, R_i, O_i)
   Since extending ρ_i by one move of the robber already yields the latest history and there are no robbers associated with $\rho_i = \rho_{s-1}$, the history can be deleted. Notice that in this case, $W_{s-1} = W_s$.

Otherwise, we set

- $\tilde{W}_i := \sigma(W_i, \tilde{b}_i)$ and
- $U' := \bigcup_{j \neq i} U_j \cup (\tilde{W}_i \setminus O^{i-1})$.

So, in the game against r robbers we make the next move according to σ, given the history $\rho_i \cdot (W_i, \tilde{b}_i)$, but we omit all the vertices from O^{i-1}. Cops from other histories than ρ_i are left unchanged. To define the memory update, we make the following definitions and distinguish two other cases.

- $\tilde{O}_i = (O_i \cap \mathrm{Reach}_{G-W_i}(\tilde{b}_i)) \setminus \tilde{W}_i$ and
- $\tilde{\rho}_i = \rho_i \cdot (W_i, \tilde{b}_i) \cdot (W_i, \tilde{W}_i, \tilde{b}_i)$

The update of the set O_i to $\tilde{O}_i$ is as explained in the description above: only vertices which are in the reachability area of $\tilde{b}_i$ in $G - W_i$ represent robber positions which can be consistently associated with $\tilde{\rho}_i$. Notice that if we associate certain robbers with the history $\tilde{\rho}_i$ then their reachability area in $G-\tilde{W}_i$ is also a subset of $\mathrm{Reach}_{G-W_i}(\tilde{b}_i)$ due to monotonicity of σ. Moreover, the set $\tilde{O}_i$ should clearly not prescribe to exclude vertices from placing cops that are already occupied by cops according to $\tilde{\rho}_i$, so we remove $\tilde{W}_i$.

(b) $\tilde{\rho}_i \neq \rho_{i+1}$.

$\zeta \to \zeta'$: Replace (ρ_i, R_i, O_i) by $(\tilde{\rho}_i, R_i, \tilde{O}_i)$

(c) $\tilde{\rho}_i = \rho_{i+1}$.

$\zeta \to \zeta'$: replace $(\rho_{i+1}, R_{i+1}, O_{i+1})$ by $(\rho_{i+1}, R_{i+1}, O_{i+1} \cup \tilde{O}_i)$;
remove (ρ_i, R_i, O_i)

Clearly, if we reach the next history, then we do not maintain both histories further. So we delete $\tilde{\rho}_i$ but we have to take into account the set $\tilde{O}_i$ that $\tilde{\rho}_i$ brings along. (Clearly, $\tilde{O}_i \not\subseteq O_{i+1}$ in general due to different robbers which originally induced these sets.) So, we join the sets to obtain the new set of vertices which ρ_{i+1} now induces to be excluded from placing cops.

Case II.2: For all $i \in \{1, \dots, s-1\}$ we have $R_i \neq \emptyset$.
We set

- $\tilde{W}_s = \sigma(W_s, b_s)$ and
- $U' := \bigcup_{j<s} U_j \cup (\tilde{W}_s \setminus O^{s-1})$.

So, in the game against r robbers we make the next move according to σ, given the history ρ_s, but we omit all the vertices from O^{s-1}. Cops from other histories than ρ_s are left unchanged. The memory update is defined as follows: $\zeta \to \zeta'$: replace ρ_s by $\rho_s \cdot (W_s, \tilde{W}_s, b_s)$

Now we prove that the move of the cops from U to U' is monotone.

Lemma 4.30. $(U \setminus U') \cap \mathrm{Reach}_{G-(U \cap U')}(R) = \emptyset$.

Proof. First consider Case I. Then $U' = U^{s-1}$ so, by (Cops), $U \setminus U' \subseteq U_s$ and $U^{s-1} \subseteq U \cap U'$. Moreover, by (Robs), $R = \bigcup_{i=1}^{s-1} R_i$. Therefore, by Lemma 4.28, we have $\mathrm{Reach}_{G-(U\cap U')}(R) \subseteq O^{s-1}$ and since $O^{s-1} \cap U_s = \emptyset$ this yields $\mathrm{Reach}_{G-(U\cap U')}(R) \cap U_s = \emptyset$.

Now consider Case II.1. In Subcase (a) the cops don't move, so the move is monotone. Otherwise we have $U' = \bigcup_{j \neq i} U_j \cup \tilde{U}_i$. Assume that this move is not monotone, that means, there is some $v \in U \setminus U'$ with $v \in \mathrm{Reach}_{G-(U\cap U')}(R)$. Notice that, by definition of U' and (Cops), $v \in U_i \setminus \tilde{U}_i$. We distinguish, which robbers can reach v. Robbers from earlier histories than ρ_i, that means, from the set R^{i-1}, are treated as in Case I: By (Cops) we have $U^{i-1} \subseteq U \cap U'$ so using Lemma 4.28 we obtain $\mathrm{Reach}_{G-(U\cap U')}(R^{i-1}) \subseteq O^{i-1}$ and since $O^{i-1} \cap U_i = \emptyset$ we have $v \notin \mathrm{Reach}_{G-(U\cap U')}(R^{i-1})$. So, as $R_i = \emptyset$, we have $v \in \mathrm{Reach}_{G-(U\cap U')}(R^{>i})$, that means, some robber from a later history than ρ_i is the villain.

Essentially, the argument for the robbers from $R^{>i}$ is the following: Due to (Lin) and (Cons), robbers from $R^{>i}$ are associated with σ-histories which extend the history ρ_i. Hence, as σ is monotone, $\tilde{b}_i$ can still reach the robbers from $R^{>i}$. So if one of them could reach v, so could b_i, which contradicts monotonicity of σ. The tricky part is that for this last argument we have to transfer reachability statements between the graphs $G - U \cap U'$ and $G - W_i \cap \tilde{W}_i$ which is similar as in the proof of Lemma 4.28, involving the sets O_j. Let us detail this a little.

First, we show that $v \notin \mathrm{Reach}_{G-(W_i \cap \tilde{W}_i)}(R^{>i})$. For $l \in \{i+1, \dots, s\}$ and $b \in R_l$, by (Lin) and our case distinction, the history $\rho_i \cdot (W_i, \tilde{b}_i)$ is a strict prefix of the history $\rho_l \cdot (W_l, b)$ (of the history ρ_l, if $l = s$). Moreover, by Lemma 4.26, both these histories are consistent with σ so, by monotonicity of σ, any robber $b \in R^{>i}$ is reachable from $\tilde{b}_i$ in $G - W_i$ and hence in $G - (W_i \cap \tilde{W}_i)$. Therefore, if $v \in \mathrm{Reach}_{G-(W_i\cap\tilde{W}_i)}(R^{>i})$ then $v \in \mathrm{Reach}_{G-(W_i\cap\tilde{W}_i)}(\tilde{b}_i)$. But as $v \in U_i \subseteq W_i$ this contradicts monotonicity of σ.

Now consider some path P from $R^{>i}$ to the vertex $v \in U_i \setminus \tilde{U}_i$ in the graph $G - (U \cap U')$. Since $v \notin \text{Reach}_{G-(W_i \cap \tilde{W}_i)}(R^{>i})$, there is a vertex from $W_i \cap \tilde{W}_i$ that lies on P and we consider the minimal $l \leq i$ with $P \cap \widehat{W}_l \neq \emptyset$ where we set $\widehat{W}_j = W_j$ for $j < i$ and $\widehat{W}_i = W_i \cap \tilde{W}_i$, likewise $\widehat{U}_j$. Now let $w \in P \cap \widehat{W}_l$. First, as $w \in P$ we have $w \notin U \cap U'$ so (Cops) and the definition of U' yield $w \notin \widehat{U}_l$. Therefore, $w \in \widehat{W}_l \setminus \widehat{U}_l$ and, using the definition of U_l ($\tilde{U}_l$, if $l = i$), we obtain $w \in O^{l-1}$, that means, $w \in O_j$ for some $j < l$. Due to the minimal choice of l we have $P \cap \widehat{W}_j = P \cap W_j = \emptyset$, so v is reachable from w in $G - W_j$ via some path $P' \subseteq P$. Moreover, using (Omit), $\text{Reach}_{G-W_j}(w) \subseteq \text{Reach}_{G-W_j}(O_j) = O_j \subseteq O^{i-1}$ but as $O^{i-1} \cap U_i = \emptyset$ this contradicts $v \in U_i$.

Finally, consider Case II.2. For robbers other than b_s the same arguments as in Case I, using (Robs) and Lemma 4.28, show that they cannot cause non-monotonicity. The argument for b_s is the same as in Case II.1. □

Notice that condition (2) from the description above is guaranteed implicitly: The cops U^{i-1} from earlier histories guarantee that O^{i-1} is, indeed, a trap for the robbers. If we removed cops from $W_i \setminus \tilde{W}_i$ that are still in U^{i-1} from the graph as well then this would not necessarily be the case and non-monotonicity could occur. In this sense, the cops from U^{i-1} guard the critical vertices.

Now we turn to the invariants. We first give a separate lemma for (Robs), (Lin), (Cons) and (Ext) which we prove rather briefly as they can be obtained easily from the induction hypothesis, using the definition of the cops' move.

Lemma 4.31. *(Robs), (Lin), (Cons) and (Ext) are preserved by the cops' move.*

Proof. (Robs) follows by induction hypothesis. Moreover, (Lin) is obviously preserved in Case I, Case II.1 (a) and (c) and in Case II.2. In Case II.1 (b) we have to show that $\tilde{\rho}_i \sqsubset \rho_{i+1}$. First notice that $\rho_i \cdot (W_i, \tilde{b}_i) \sqsubset \rho_{i+1}$ as $\rho_{i+1} = \rho_i \cdot (W_i, \tilde{b}_i)\eta$. Furthermore, $\text{first}(\eta) = (W_i, \tilde{W}_i, \tilde{b}_i)$ as ρ_{i+1} is consistent with σ according to (Cons) and $\tilde{W}_i = \sigma(W_i, \tilde{b}_i)$. Hence, $\tilde{\rho}_i \sqsubset \rho_{i+1}$.

For (Cons), first consider Case I. If $R_{s-1} = \emptyset$, then $\rho'_{s'} = \rho_{s-1} \cdot (W_{s-1}, b_s)$ and we have $\text{last}(\rho_s) \in \{(W_s^{-1}, W_s, b_s), (W_s, b_s)\}$ and $\rho_{s-1} \sqsubset \rho_s$ and due to (Cons) both are σ-histories. Since σ is monotone, b_s is reachable from b_{s-1} in $G - (W_{s-1}^{-1} \cap W_{s-1})$, so $\rho_{s-1} \cdot (W_{s-1}, b_s)$ is a σ-history as well. If $R_{s-1} \neq \emptyset$ then $\rho_{s-1} \cdot (W_{s-1}, b)$ is consistent with σ for any $b \in R_{s-1}$ due to Lemma 4.26. In Case II.1 (a) and (b), (Cons) follows by induction hypothesis. In Case II.1 (b), (Cons) follows from (Lin) as $\rho'_{s'} = \rho_s$ is consistent with σ and $\tilde{\rho}_i \sqsubset \rho_s$. Finally, in Case II.2, ρ_s is consistent with σ due to (Cons) and $W'_s = \sigma(W_s, b_s)$, so $\rho'_{s'} = \rho'_s$ is also a σ-history.

To prove (Ext) first notice that in Case I, if $R_{s-1} = \emptyset$ (Ext) follows by induction hypothesis. Moreover, if $R_{s-1} \neq \emptyset$, then we have $s' = s$ and, by Lemma 4.26, for any $b' \in R_{s-1}$ the history $\rho_{s-1}(W_{s-1}, b')$ is consistent with σ which is monotone. So $\text{Reach}_{G-W_{s-1}}(b') \subseteq \text{Reach}_{G-W_{s-1}^{-1}}(b_{s-1})$ for all $b' \in R_{s-1}$ and therefore, $O'_{s-1} = \tilde{O}_{s-1} = \text{Reach}_{G-W_{s-1}}(R_{s-1}) \subseteq \text{Reach}_{G-W_{s-1}^{-1}}(b_{s-1})$. In Case II, (Ext) follows easily from the induction hypothesis. (Notice that in Case II.1(c) we have $W_i = W_{i+1}^{-1}$ and $\tilde{b}_i = b_{i+1}$.) □

For the remaining two invariants (Omit) and (Cops) we have two separate lemmata. The most interesting cases in the proofs are Case II.1(b) and Case II.1(c). The crucial point here is the new set $\tilde{O}_i$. First, we have to show that this set is closed under reachability in the graph $G - \tilde{W}_i$ which is illustrated in Figure 4.8. Moreover, as we have mentioned in the description above, we have to show that all gaps that can be filled in *later* histories ρ_j, $j > i$, according to the new set $\tilde{O}_i \subseteq O_i$ are already filled in the cops' move by which we extend the history ρ_i. (So that, when playing a history ρ_j, $j \leq s$, we will never place cops on vertices that have previously been omitted while playing ρ_j.)

Lemma 4.32. *(Omit) is preserved by the cops' move.*

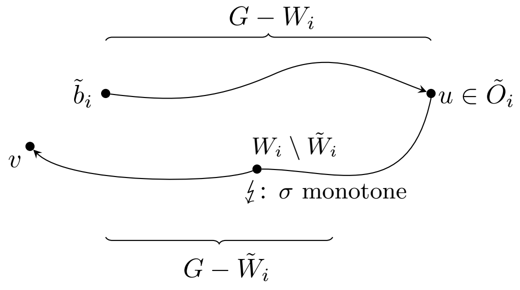


Figure 4.8: $\tilde{O}_i$ is closed under reachability in $G - \tilde{W}_i$

Proof. In Case I, if $R_{s-1} = \emptyset$, (Omit) follows by induction hypothesis, so let $R_{s-1} \neq \emptyset$. By definition, $s' = s$, $W'_{s-1} = W_{s-1}$ and $O'_{s-1} = \tilde{O}_{s-1} = \text{Reach}_{G-W_{s-1}}(R_{s-1} \setminus \{b\})$, so O'_{s-1} is closed under reachability in $G - W_{s-1}$. Moreover, by (Omit), $R_{s-1} \subseteq \text{Reach}_{G-W_{s-1}}(O_{s-1})$. In particular, $R_{s-1} \cap W_{s-1} = \emptyset$ which directly yields $R_{s-1} \setminus \{b\} \subseteq \text{Reach}_{G-W_{s-1}}(R_{s-1} \setminus \{b\}) = O'_{s-1}$.

Now consider Case II.1. In Case (a), (Omit) follows by induction hypothesis. In Case (b), $R'_i = R_i = \emptyset \subseteq O'_i$, so we have to show $O'_i = \text{Reach}_{G-W'_i}(O'_i)$. Since $O'_i = \tilde{O}_i$ and $\tilde{O}_i \cap \tilde{W}_i = \emptyset$, we clearly have $\tilde{O}_i \subseteq \text{Reach}_{G-\tilde{W}_i}(\tilde{O}_i)$. The crucial point is now to show that $\tilde{O}_i$ is closed under reachability in $G - \tilde{W}_i$ which relies on monotonicity of σ: Essentially, $\tilde{O}_i$ is the intersection of two sets which are closed under reachability in $G - W_i$ (removing vertices from $\tilde{W}_i$ does not affect reachability in $G - \tilde{W}_i$) and moving the cops from W_i to $\tilde{W}_i$ is the next move of σ against the robber $\tilde{b}_i$ which is monotone. Let us be a bit more precise.

Let $v \in \text{Reach}_{G-\tilde{W}_i}(\tilde{O}_i)$. In particular, $v \notin \tilde{W}_i$. Now let $u \in \tilde{O}_i \subseteq O_i$ such that v is reachable from u in $G - \tilde{W}_i$. By definition of $\tilde{O}_i$ we have $u \in \text{Reach}_{G-W_i}(\tilde{b}_i)$, so there is a path P from $\tilde{b}_i$ to v via u in $G - (W_i \cap \tilde{W}_i)$, cf. Figure 4.8. By (Cons), ρ_i is consistent with σ and $\tilde{W}_i = \sigma(W_i, \tilde{b}_i)$ so, by monotonicity of σ, P is also a path in $G - W_i$. (Notice that $W_i \cap P \subseteq \text{Reach}_{G-W_i \cap \tilde{W}_i}(\tilde{b}_i)$.) Therefore, $v \in \text{Reach}_{G-W_i}(u)$ which yields $v \in \text{Reach}_{G-W_i}(O_i)$ and $v \in \text{Reach}_{G-W_i}(\tilde{b}_i)$. Moreover, by (Omit), we have $\text{Reach}_{G-W_i}(O_i) = O_i$, so $v \in O_i \cap \text{Reach}_{G-W_i}(\tilde{b}_i)$ and as $v \notin \tilde{W}_i$ we obtain $v \in \tilde{O}_i$.

In Case (c), by (Omit), we have $R'_i = R_{i+1} \subseteq O_{i+1} \subseteq O'_i$. Moreover, as in Case (b), $\tilde{O}_i$ is closed under reachability in $G - \tilde{W}_i$ and as $\tilde{\rho}_i = \rho_{i+1}$ we have $\tilde{W}_i = W_{i+1} = W'_i$. By (Omit), O_{i+1} is closed under reachability in $G - W_{i+1}$, so the union $O_{i+1} \cup \tilde{O}_i$ is closed under reachability in $G - W_{i+1} = G - W'_i$ as well. Finally, in Case II.2, (Omit) follows again by induction hypothesis. □

Lemma 4.33. *(Cops) is preserved by the cops' move.*

Proof. We have to show that $U' = \bigcup_{j=1}^{s'} U'_j$. (Recall that $U'_j = W'_j \setminus (O^{j-1})'$.) In Case I, Case II.1 (a) and Case II.2, this can easily be proved using the induction hypothesis and the definition of U'. (Notice that in Case II.1 (a) we have $U_s \subseteq U_{s-1}$.) Now consider Case II (b). Then $s' = s$ and $O'_j = O_j$ for $j \neq i$ and $O'_i = \tilde{O}_i \subseteq O_i$, so $U_j \subseteq U'_j$ for $j \geq i$. Moreover, $W'_j = W_j$ for $j < i$ which implies $U'_j = U_j$ for $j < i$. Hence, $U' \subseteq \bigcup_{j=1}^{s} U'_j$ and it remains to show $U' \supseteq \bigcup_{j=1}^{s} U'_j$.

So assume that there is some $v \in (\bigcup_{j=1}^{s} U'_j) \setminus U'$. Since $U'_i \subseteq U'$ there is some $j > i$ with $v \in U'_j$ and as $v \notin U' \supseteq U_j$ we have $v \in W_j \setminus (O^{j-1})'$ but $v \in O^{j-1}$. Since $O'_l = O_l$ for $l \neq i$ this yields $v \in O_i \setminus O'_i = O_i \setminus \tilde{O}_i$ and so $v \in \tilde{W}_i$ or $v \notin \text{Reach}_{G-W_i}(\tilde{b}_i)$. Moreover, $v \notin U' \supseteq \tilde{W}_i \setminus O^{i-1}$ and as $v \notin (O^{j-1})' \supseteq O^{i-1}$ it follows that $v \notin \tilde{W}_i$, so $v \notin \text{Reach}_{G-W_i}(\tilde{b}_i)$.

Now let $\rho = \widehat{\rho}(W^{-1}, W, b)$ be the shortest prefix of ρ_j such that $v \in W$. Notice that such a prefix exists because $v \in W_j$. Due to (Cons), $\tilde{\rho}_i$ and ρ are both consistent with σ which is monotone, so since $v \notin \tilde{W}_i$ we have $\tilde{\rho}_i \sqsubset \rho$. Having this and the fact that $v \notin \mathrm{Reach}_{G-W_i}(\tilde{b}_i)$, monotonicity of σ also yields $v \notin \mathrm{Reach}_{G-W^{-1}}(b)$. But this is a contradiction to the fact that σ is thrifty.

Finally, in Case (c), we have $s' = s - 1$ as we delete the i-th element of ζ, that is, we have a shift of indices. Accounting for this, (Cops) can be proved with the same arguments as in Case (b). □

Now, before we consider the robbers' next move we make a preparation: Any robber that is still in some set O'_i after their move can be consistently associated with some history due to (Ext). On the other hand, there may also be robbers that are not in any set O'_i afterwards, and the only way to capture them is to assign them to the latest history. However, for this to be consistently possible, we need that any such robber is in $\mathrm{Reach}_{G-W_s}(b_s)$, so we have to show $\mathrm{Reach}_{G-U}(R) \setminus \mathrm{Reach}_{G-W_s}(b_s) \subseteq (O^{s'-1})' \cup U'$. (Notice that $\mathrm{Reach}_{G-U}(R) = \mathrm{Reach}_{G-U\cap U'}(R)$ as the move $U \to U'$ is monotone.)

Notice that in fact we need this property only for Cases II.1(a), II.1(b) and II.2 of the cops move: In the other cases, we don't place any additional cops on the graph so by prudence, the robbers will not move. So in the proof we consider only these cases but since in the other cases we have $\mathrm{Reach}_{G-U}(R) = \mathrm{Reach}_{G-U'}(R)$, it is easy to see that the property holds in all cases.

The reason why we prove this before we consider the next move of the robbers is that this property involves only objects that we already have at hand and is completely independent of the actual move that the robbers perform. Moreover, it helps to clear the picture of the dynamics of the possible robbers' moves.

Clearly, $\mathrm{Reach}_{G-U}(R) \setminus \mathrm{Reach}_{G-W_s}(b_s) \subseteq O^{s-1}$ can be shown with the usual arguments as in Lemma 4.28 and Lemma 4.30. The more elaborate part is to show that this inclusion also survives the update of the sets O_i that we have performed (while we consider reachability in the graph $G - U$ *before* the cops' move). So what we really have to make sure is that in Case II.1(b) and (c) of the cops' move, no robber that moves into the set O_i in the graph $G - U$ slips away, that means, any such robber is also in $\tilde{O}_i$.

Lemma 4.34. $\mathrm{Reach}_{G-U}(R) \setminus \mathrm{Reach}_{G-W_s}(b_s) \subseteq (O^{s'-1})' \cup U'$.

Proof. Let $d \in \mathrm{Reach}_{G-U}(b) \setminus \mathrm{Reach}_{G-W_s}(b_s)$ for some $b \in R$ and let $l \leq s$ with $b \in R_l$ according to (Robs). If $d \in U'$ then we're done, so assume $d \notin U'$. Now let P be a path from b to d in $G - U$. First, from Lemma 4.28 it follows that if $l < s$ then $d \in O^l$. If, on the other hand, $b = b_s$ then d is reachable from b_s in $G - U$ but not in $G - W_s$, so P goes through O^{s-1} and O^{s-1} is a trap for the robbers in $G - U$, as usual. Let $j \leq \min\{l, s-1\}$ be minimal such that $d \in O_j$.

To show that d is also in $(O^{s'-1})'$ we distinguish the moves that the cops may have made. As we have mentioned, we consider only the Cases II.1(b) and (c) and II.2. Moreover, if $O_j = O'_j$, which in particular holds in Case II.2, then $d \in O'_j \subseteq (O^{s'-1})$. Now assume that $O_j \neq O'_j$, so we are in Case II.1(b) or (c). By definition, for all $m < i$, we have $O_m = O'_m$, so $j \geq i$. Moreover, for all $m > i$, $O_m = O'_m$ (Case (b)) or $O_m \subseteq O'_{m-1}$ (Case (c)), so either $d \in (O^{s'-1})'$ or $j \leq i$. The remaining case is $j = i$. Notice that this implies $j < l$ as either $l = s$ and $j \leq s - 1$ or $l < s$ and $j \leq l$ and since $R_j = R_i = \emptyset$ and $b \in R_l \neq \emptyset$ we have $j \neq l$. We now show that $d \in \tilde{O}_j$ which implies $d \in (O^{s'-1})'$ by definition of $(O^{s'-1})'$.

By definition, $\tilde{O}_j = (O_j \cap \mathrm{Reach}_{G-W_j}(\tilde{b}_j)) \setminus \tilde{W}_j$. Now, since $d \notin U'$, if $d \in \tilde{W}_j$ then $d \in O^{j-1}$ which contradicts the minimality of j. Hence, $d \notin \tilde{W}_j$ and it remains to prove $d \in \mathrm{Reach}_{G-W_j}(\tilde{b}_j)$. Essentially, the reason is as follows. As $i = j < l$, the robber $b \in R_l$ has reacted to the move by which we have extended the *earlier* history $\rho_i \sqsubset \rho_l$ and since σ is monotone, this was possible only using vertices from O^{l-1}. However, due to the choice of j, b did not use vertices from O^{j-1}, so b can reach vertex d in the graph $G - W_j$. Therefore, $\tilde{b}_j$ can reach vertex d in the graph $G - W_j$ as well, cf. Figure 4.9. We give some more details.

First notice that since $j < l$, due to (Lin) we have $\tilde{\rho}_j \sqsubseteq \rho_l$. So as, according to (Cons), both histories are consistent with σ, which is monotone, b_l is reachable from $\tilde{b}_j$ in $G - W_j$.

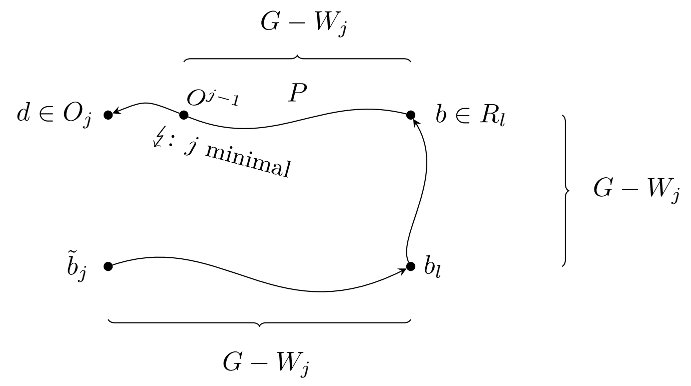


Figure 4.9: If $i = j < l$, the robber $\tilde{b}_j$ can still reach d in the graph $G - W_j$ via b.

Now, if $l < s$ then, by (Ext), $b \in R_l$ is reachable from b_l in $G - W_l^{-1}$. So, by σ-consistency of $\rho_j \sqsubseteq \rho_l$ and monotonicity of σ, b is reachable from b_l in $G - W_j$. If, on the other hand, $l = s$, then $b = b_s = b_l$, so $b \in \mathrm{Reach}_{G-W_j}(b_l)$ is trivial. Moreover, by (Cops) $U_j \subseteq U$, so vertex d is reachable from b in $G - U_j$ via P. Hence, if d is not reachable from $\tilde{b}_j$ in $G - W_j$, there is some vertex from O^{j-1} on the path P which contradicts the minimality of j. Therefore, $d \in \mathrm{Reach}_{G-W_j}(\tilde{b}_j)$. □

Notice that the reasoning in the last part of the proof works only as long as $j < l$ because this yields $\tilde{\rho}_j \sqsubseteq \rho_l$. If $j = l$, that is, $b \in R_l = R_j = R_i \neq \emptyset$ then we do not necessarily have $\tilde{\rho}_j \sqsubseteq \rho_l$, so $d \notin \mathrm{Reach}_{G-W_j}(\tilde{b}_j)$ is possible. Hence, if we continue to play histories (other than the latest one) that have still robbers assigned to them, we might let some robbers slip away!

Move of the Robbers. Now let (U^{-1}, U, R) be a position where it is the robbers' turn and let

$$\zeta = \big((\rho_1, R_1, O_1), \dots, (\rho_{s-1}, R_{s-1}, O_{s-1}), \rho_s\big)$$

be the current memory state such that all invariants are satisfied.

Notice that after the move of the cops as described above, from U to U', and the memory update, from ζ to ζ' we make a renaming $(U, U', R) =: (U^{-1}, U, R)$ and $\zeta' =: \zeta$. This helps to partially encapsulate the previous work and focus on the invariants as formulated for (U^{-1}, U, R) and ζ. Nevertheless, we will be needing the memory state *before* the move of the cops, which we denote by $\overline{\zeta}$:

$$\overline{\zeta} = ((\overline{\rho_1}, \overline{R}_1, \overline{O}_1), \dots, (\overline{\rho}_{\overline{s}-1}, \overline{R}_{\overline{s}-1}, \overline{O}_{s-1}), \overline{\rho}_{\overline{s}})$$

This is necessary, in particular, to be able to apply Lemma 4.34.

Now consider any possible response $R \to R'$ of the robbers, that is, R' is the set of vertices occupied by robbers after their move. If $R' = R$ we do not update the memory. This happens in particular in Case I and Case II.1 (a) of the cops' move: there, we do not place additional cops on the graph so, by prudence, $R' = R$. We neglect these cases. So in the following, assume that $R' \neq R$. Notice that, in particular, we have $b_s \in R$.

We assign every robber from R' to some unique history ρ_i, $i = 1, \dots, s$ which yields, for any $i \in \{1, \dots, s\}$ the new set $\tilde{R}_i$. So consider some robber $b \in R'$. If $b \in O^{s-1}$ then let

$$i = \min\{j \in \{1, \dots, s-1\} \mid b \in O_j\}$$

and assign b to ρ_i. Otherwise, assign b to ρ_s.

The reason why we assign a robber b to the *minimal* history ρ_i such that $b \in O_i$ is that otherwise the strategy $\otimes_r f$ would not necessarily be winning: If a robber $b \in O_i \cap O^{i-1}$ becomes the latest robber then we play against this robber according to the strategy σ, but we omit vertices from O^{i-1}. Hence, when σ prescribes to place a cop on b, we omit this move, so we would never catch b if he just stands idle.

The crucial point about the memory update after the move of the robbers is that our assignment of robbers to histories is meaningful in the sense that a robber which has been assigned to a certain history is also consistent with this history according to f. For the robbers assigned to histories ρ_i with $i < s$ this follows easily from the fact that $\tilde{R}_i \subseteq O_i$, similar as in Lemma 4.26. For the robbers in $\tilde{R}_s$, however, we need Lemma 4.34 which yields that each such robbers can be reached from $\overline{b_{\overline{s}}} = b_s$ in the graph $G - \overline{W_{\overline{s}}}$ so that prolonging the longest history by a move from b_s to some robber from $\tilde{R}_s$ yields again a σ-history.

Now we define the new memory state

$$\zeta' = ((\rho'_1, R'_1, O'_1), \dots, (\rho'_{s'-1}, R'_{s'-1}, O'_{s'-1}), \rho'_{s'}).$$

For the update we distinguish certain cases according to the number of robbers that have been assigned to ρ_s and according to whether the last position of ρ_s is a cops' or a robbers' position. First, to simplify the case distinction, we prove that if the cops did not play against the latest robber in their last move then at most the robber b_s (which is then identical to $\overline{b_{\overline{s}}}$) is associated with ρ_s. For this, we use prudence.

Lemma 4.35. *If ρ_s ends with a cops' position then $\tilde{R}_s \subseteq \{b_s\}$.*

Proof. Assume that ρ_s ends with a cops' position, that means, $\rho_s = \widehat{\rho}_s(W_s, b_s)$. So the last move of the cops was not as in Case II.2 and hence (as we neglect Case I and Case II.1 (a)) $W_s = \overline{W_{\overline{s}}}$. So, by Lemma 4.34, $\tilde{R}_s \subseteq \mathrm{Reach}_{G-\overline{W_{\overline{s}}}}(b_s) = \mathrm{Reach}_{G-W_s}(b_s)$ and by Lemma 4.27 we have $\mathrm{Reach}_{G-W_s}(b_s) = \mathrm{Reach}_{G-W^s}(b_s)$. Therefore, $\tilde{R}_s \subseteq \mathrm{Reach}_{G-W^s}(b_s) \subseteq \mathrm{Reach}_{G-U}(b_s)$ and since $b_s \in R$, $\tilde{R}_s \not\subseteq \{b_s\}$ contradicts the assumption that the robbers use a prudent strategy. □

Now we need to distinguish only two cases: Either, the cops have played against the latest robber b_s in their last move and $|\tilde{R}_s| \geq 1$ (so at least one robber from R' can still be consistently associated with ρ_s). Or this is not the case, that is, either the cops have not played against the latest robber or they have played against him and he has either been caught or he has returned to an earlier situation. However, these remaining cases can be treated identically.

Case 1: ρ_s ends with a robbers' position and $|\tilde{R}_s| \geq 1$.
So the cops have played against the latest robber $\overline{b_{\overline{s}}}$ in their last move and he has split into several copies. (Notice that in the proof of Lemma 4.34 we have seen that each robber $b \in R'$ that is spawned from a robber in R^{s-1} is in O^{s-1}.)
We choose some $b \in \tilde{R}_s$, define $\tilde{O}_s = \mathrm{Reach}_{G-W_s}(\tilde{R}_s \setminus \{b\})$ and set

$$\zeta' = \big((\rho_1, \tilde{R}_1, O_1), \dots, (\rho_{s-1}, \tilde{R}_{s-1}, O_{s-1}), (\rho_s, \tilde{R}_s \setminus \{b\}, \tilde{O}_s), \rho_s \cdot (W_s, b)\big).$$

Notice that this memory update is analog to Case I of the cops' move (where $R_{s-1} \neq \emptyset$): We choose one of the robbers $b \in \tilde{R}_s$ to be the new latest one which is pursued further, and we add a new history which extends ρ_s by the robbers' move from b_s to b. Moreover, with ρ_s we associate the remaining robbers and we define a new set $O'_{s'-1}$ which contains the area available for the robbers $\tilde{R}_s \setminus \{b\}$ in $G - W_s$. (Notice that if $|\tilde{R}_s| = 1$ then $\tilde{R}_s \setminus \{b\} = \emptyset$ and $O'_s = \emptyset$, so in fact, ρ_s does not have to be maintained anymore. However, this will be taken care of during the next move of the cops.)

Case 2: ρ_s ends with a position of the cop player or $|\tilde{R}_s| = 0$.

So, either the cops have not played against the latest robber in their last move or they have, and he has either been caught or he has returned to an earlier situation. In the first case, Lemma 4.35 tells us that $\tilde{R}_s \subseteq \{b_s\}$ and b_s is accounted for in ρ_s. If, in the other case, the greatest robber has returned to an earlier situation then he is accounted for there and if he is caught, then he is clearly also accounted for. Therefore, we simply define

$$\zeta' = \big((\rho_1, \tilde{R}_1, O_1), \dots, (\rho_{s-1}, \tilde{R}_{s-1}, O_{s-1}), \rho_s\big).$$

Now we prove that after the move of the robbers, all invariants that we have formulated still hold. As we have already mentioned, the crucial point about the memory update after the robbers' move is that the assignment of robbers to histories is meaningful, that means, that invariants (Ext) (in Case 1 for the $s-1$-th element of ζ', in particular) and (Cons) (in Case 1 and Case 2 for the s-th element of ζ', in particular) still hold. However, the main argument for this is given in Lemma 4.34 so we shall be quite brief.

Lemma 4.36. *All invariants are preserved by the robbers' move.*

Proof. (Robs), (Lin) and (Cops) are obvious. To prove (Omit), first notice that for $i = 1, \dots, s-1 \geq s'-2$, we have $O'_i = O_i$ and $\rho'_i = \rho_i$, so for $O'_1, \dots, O_{s'-2}$ closeness under reachability in $G - W'_i$ follows directly from (Omit) for ζ. Moreover, $R'_i = \tilde{R}_i \subseteq O_i = O'_i$ holds by definition of the sets $\tilde{R}_i$ for $i = 1, \dots, s-1$. In particular, in Case 2 there is nothing to show, so consider Case 1. There we have $s' = s+1$ and $O'_{s'-1} = O'_s = \tilde{O}_s = \mathrm{Reach}_{G-W_s}(\tilde{R}_s \setminus \{b\})$ so O'_s is obviously closed under reachability in $G - W_s = G - W'_{s'-1}$. It remains to show that $R'_{s'-1} \subseteq O'_{s'-1}$. For this, notice that $W_s \cap (\tilde{R}_s \setminus \{b\}) = \emptyset$: If $v \in W_s \cap (\tilde{R}_s \setminus \{b\})$ then $v \notin U$ so, according to (Cops), $v \notin U_s$ and hence $v \in O^{s-1}$ which contradicts $v \in \tilde{R}_s$. Therefore, $R'_{s'-1} = \tilde{R}_s \setminus \{b\} \subseteq \tilde{O}_s = O'_{s'-1}$.

Now we show (Ext). First, by (Ext) for ζ we have $O_i \subseteq \mathrm{Reach}_{G-W_i^{-1}}(b_i)$ for all $i = 1, \dots, s-1$ and as $O'_i = O_i$ and $\rho'_i = \rho_i$ for $i = 1, \dots, s-1$, (Ext) follows for ζ' for $i = 1, \dots, s-1 \geq s'-2$. In particular, in Case 2, there is nothing to show, so consider Case 1. There we have $\rho'_{s'-1} = \rho_s$ and $W_s^{-1} = \overline{W}_s$ and $b_s = \overline{b}_s$. So, since $\tilde{R}_s \subseteq \mathrm{Reach}_{G-U^{-1}}(R) \setminus (O^{s-1} \cup U)$ holds by construction of $\tilde{R}_s$, Lemma 4.34 yields

$$\tilde{R}_s \subseteq \mathrm{Reach}_{G-\overline{W}_s}(\overline{b}_s) = \mathrm{Reach}_{G-W_s^{-1}}(b_s).$$

Moreover, by definition, $O'_{s'-1} = \tilde{O}_s = \mathrm{Reach}_{G-W_s}(\tilde{R}_s \setminus \{b\})$. So, if $v \in O'_{s'-1}$, then v is reachable from some $\widehat{b} \in \tilde{R}_s \setminus \{b\}$ in $G - W_s$ and as $\tilde{R}_s \subseteq \mathrm{Reach}_{G-W_s^{-1}}(b_s)$, $\widehat{b}$ is reachable from b_s in $G - W_s^{-1}$. Thus, v is reachable from b_s in $G - (W_s^{-1} \cap W_s)$ and as $\rho_s = \widehat{\rho}(W_s^{-1}, W_s, b_s)$ is consistent with σ by (Cons) for ζ and σ is monotone, we have $v \in \mathrm{Reach}_{G-W_s^{-1}}(b_s)$.

Finally, for (Cons), Case 2 is obvious. For Case 1, as ρ_s is consistent with σ by (Cons), it suffices to show that $b \in \mathrm{Reach}_{G-W_s^{-1}}(b_s)$. As before, from $\tilde{R}_s \subseteq \mathrm{Reach}_{G-U^{-1}}(R) \setminus (O^{s-1} \cup U)$ and Lemma 4.34 we obtain $\tilde{R}_s \subseteq \mathrm{Reach}_{G-\overline{W}_s}(\overline{b}_s)$ and since $\overline{b}_s = b_s$ and $\overline{W}_s = W_s^{-1}$, this follows from $b \in \tilde{R}_s$. □

Showdown. Finally, we show that $\ominus_r\sigma$ uses in fact at most $r \cdot k$ cops and that, by playing according to $\ominus_r\sigma$, the cops finally capture all robbers. We have seen that $\ominus_r\sigma$ is a monotone strategy, so this concludes the proof of Theorem 4.22.

To show that $\ominus_r\sigma$ uses at most $r \cdot k$ cops, first notice that by (Cops), the number of cops is bounded by $|\bigcup_{i=1}^s U_i|$. Moreover, by definition of U_i we have $|\bigcup_{i=1}^s U_i| \leq |\bigcup_{i=1}^s W_i|$ and due to (Cons), all W_i have size at most k. It remains to show that there are always at most r distinct sets W_i.

Lemma 4.37. *For all (U, R) and ζ which are consistent with $\ominus_r\sigma$ we have $|\zeta| \leq r+1$ and if $|\zeta| = r+1$ then $W_{s-1} = W_s$.*

Proof. We denote by ζ the memory state before the cops' move and by $\zeta \to \zeta' \to \zeta''$ we denote the subsequent updates. We prove that if $|\zeta| = r+1$ then $|\rho_{s-1}| = |\rho_s| - 1$ which implies $W_{s-1} = W_s$. First notice that $|\zeta'| \leq |\zeta|$ and $|\zeta''| \leq |\zeta'| + 1$. Now, if $|\zeta| \leq r$ and $|\zeta''| = r+1$ then we are in Case 1 of the robbers' move and, by construction, we have $|\rho''_{s''-1}| = |\rho''_{s''}| - 1$.

Now let $|\zeta| = r+1$ and assume $|\rho_{s-1}| = |\rho_s| - 1$. In Case I of the cops' move, we either have $R_{s-1} = \emptyset$ and $|\zeta'| = r$ or $R_{s-1} \neq \emptyset$ and $|\rho'_{s'-1}| = |\rho'_{s'}| - 1$. Moreover, since we do not place any additional cops on the graph, the robbers do not move which yields $\zeta'' = \zeta'$, so $|\zeta''| = r$ as well.

Now consider Case II of the cops' move. As $|R| \leq r$, from (Robs) it follows that $R_i = \emptyset$ for some $i \in \{1, \dots, s-1\}$ which implies that we are actually in Case II.1 of the cops' move. Moreover, if $i = s-1$ then due to $|\rho_{s-1}| = |\rho_s| - 1$, extending $\rho_i = \rho_{s-1}$ by a move of the robber yields ρ_s, so we are in Case II.1(a). There we have $|\zeta'| = r$ and since we do not place any additional cops on the graph, the robbers do not move which yields $\zeta'' = \zeta'$, so $|\zeta''| = r$ as well. Finally, assume that $i < s-1$, so we are in Case II.1(b) or Case II.1(c) of the cops move. In Case II.1(c) we have $|\zeta'| = r$ and since, by Lemma 4.35, Case 1 of the robbers' move is always preceded by Case II.2 of the cops' move, we are in Case 2 of the robbers' move. Hence, $|\zeta''| = r$ as well. In Case II.1(b), we have $|\zeta'| = r+1$ and since $i < s-1$ we also have $|\rho'_{s'-1}| = |\rho'_{s'}| - 1$. Again, by Lemma 4.35, we are in Case 2 of the robbers' move which yields $|\zeta''| = r+1$ and $|\rho''_{s''-1}| = |\rho''_{s''}| - 1$. □

To prove that $\ominus_r\sigma$ is winning, we use the following additional invariant.

$$\textbf{(Progress)} \quad \text{For } i \in \{2, \dots, s-1\}, R_i \cap O^{i-1} = \emptyset \text{ and } b_s \notin O^{s-1}.$$

As we have mentioned in the description above, since σ is winning, if the robber does not move, σ will finally prescribe to place a cop on the vertex currently occupied by the robber. (Progress) tells us that we will not omit this cop-placement so, as long as the robbers use a prudent strategy, it guarantees progress of the strategy $\ominus_r\sigma$ against r robbers. Clearly, we also have to maintain this same property also for all earlier histories since, if at some point b_s is either caught or returns to an earlier situation, one of the robbers associated with a history ρ_i for some $i < s$ becomes the latest one.

We prove (Progress) separately, because it only uses the other invariants but is not further intertwined with them. Basically, (Progress) follows from the assumption that the robbers use an isolating strategy. However, as the sets O_i are defined with respect to reachability in the graphs $G - W_i$, we have to transfer this from $G - U$ to the graphs $G - W_i$ as usual.

Lemma 4.38. *(Progress) is preserved by the cops' move and the robbers' move.*

Proof. First consider the situation after the cops' move. In Case I we have $R'_j = R_j$ and $O_j = O'_j$ for $j = 1, \dots, s-2$ and hence, $R'_j \cap (O^{j-1})' = \emptyset$ follows by (Progress) for ζ. Moreover, if $R_{s-1} = \emptyset$, then $s' = s-1$, so $b'_{s'} = b_s$ and $O^{s'-1} = O^{s-2}$ we which yields $b'_{s'} \notin (O^{s'-1})'$ by (Progress) for ζ.

If , on the other hand, $R_{s-1} \neq \emptyset$ then $s' = s$ and $R'_{s-1} \subseteq R_s$, so $R'_{s-1} \cap (O^{s-2})' = \emptyset$ and $b'_s = b \notin (O^{s-2})'$ follows directly from $(O^{s-2})' = O^{s-2}$ and (Progress) for ζ. It remains to show that $b \notin O'_{s-1} = \tilde{O}_{s-1}$. Since the robbers use an isolating strategy, $b \notin \text{Reach}_{G-U}(R'_{s-1})$ and since $U_s \subseteq U_{s-1}$ we have $U = U^{s-1}$. Moreover, $U^{s-1} \subseteq W^{s-1}$ so $\text{Reach}_{G-U}(R'_{s-1}) \supseteq \text{Reach}_{G-W^{s-1}}(R'_{s-1})$. Now, Lemma 4.27 yields $\text{Reach}_{G-W^{s-1}}(R'_{s-1}) = \text{Reach}_{G-W_{s-1}}(R'_{s-1})$, so $b \notin \text{Reach}_{G-W_{s-1}}(R'_{s-1}) = \tilde{O}_{s-1}$. In Case II, (Progress) for ζ' follows easily from (Progress) for ζ using the definition of the memory update. (Recall that $\tilde{O}_i \subseteq O_i$.)

Now consider the situation after the robbers' move. In Case 2, (Progress) holds by the minimal choice in the construction of the sets $\tilde{R}_i = R'_i$ for $i = 1, \dots, s$. Moreover, in Case 1, $R'_i \cap (O^{i-1})' = \emptyset$ holds for $i = 1, \dots, s'-1$ by construction of the sets R'_i as well and $b \notin$

$(O^{s'-2})' = O^{s-1}$ holds by construction of $\tilde{R}_s$. It remains to show that $b \notin O'_{s'-1} = \tilde{O}_s$. This follows as in Case I of the robbers' move, using Lemma 4.27 and the fact that the robbers play an isolating strategy. (Notice that $\tilde{O}_s = \mathrm{Reach}_{G-W_s}(\tilde{R}_s \setminus \{b\})$ and $U = U^s$.) $\square$

Lemma 4.39. *$\ominus_r\sigma$ is winning.*

Proof. We proceed similar as in the proof of Proposition 4.24, using monotonicity of $\ominus_r\sigma$. However, we have to take into account that the cops' move is subdivided into several cases and that (Area) is not available. Instead, we have (Progress).

Consider any $\ominus_r\sigma$-play $\pi = (U_0, R_0)(U_0, U_1, R_1)(U_1, R_2)\ldots$. It suffices to show that for all $\lambda \in \mathbb{N}$ with $R_\lambda \neq \emptyset$, there is some $\gamma > \lambda$ such that $\mathrm{Reach}_{G-U_\gamma}(R_\gamma) \subsetneq \mathrm{Reach}_{G-U_\lambda}(R_\lambda)$. If this is not the case, then, since $\ominus_r\sigma$ is monotone, there is a $\lambda \in \mathbb{N}$ such that $R_\lambda \neq \emptyset$ and $\mathrm{Reach}_{G-U_\gamma}(R_\gamma) = \mathrm{Reach}_{G-U_\lambda}(R_\lambda)$ for all $\gamma > \lambda$. Since the robbers follow a prudent strategy, they do not move after round λ, that is, $R_\gamma = R_\lambda =: R$ for all $\gamma > \lambda$. By (Robs) it follows that $\zeta_\lambda \neq \emptyset$, where ζ_λ denotes the memory state at round λ.

First, there is some $\xi \geq \lambda$ such that in all rounds $\lambda \leq \nu \leq \xi$, Case I or Case II.1 of the cops' move eventuates, while after round ξ, the cops' move is always according to Case II.2. To see this, notice that, as long as $b_s \notin R$ or $R_i = \emptyset$ for some i, Case I or Case II.1 eventuates. Moreover, in Case I, either the number s of histories in ζ or $|R_{s-1}|$ decreases. and in Case II.1, histories that are shorter than ρ_s are extended (which increases their length), or deleted if they reach the next history (which decreases s). Since the robbers' don't move anymore, as long as Case I or Case II.1 eventuates, none of the values s, $|\rho_s|$ and $|R_{s-1}|$ will increase, so the existence of such a ξ follows.

So, in any round $\nu > \xi$, we are in Case II.2 of the cops' move where the cops play against the robber $b_s = b_s(\xi) \in R$ according to σ. Now consider the unique extension of $\rho_s = \rho_s(\xi)$ to the σ-play ρ where the robber just makes himself comfortable on b_s. In any round $\nu > \xi$, the strategy $\ominus_r\sigma$ does not place a cop inside $\mathrm{Reach}_{G-U(\nu)}(b_s)$ and we have $b_s \in R \subseteq \mathrm{Reach}_{G-U(\nu)}(R)$ and, by (Progress), $b_s \notin O^{s-1}(\nu) = O^{s-1}(\xi)$. So, by definition of $U(\nu+1)$, σ does not place a cop on the vertex b_s during ρ which yields that ρ is won by the robber – this contradicts the premise that σ is a winning strategy. $\square$

Chapter 5

Locally Decomposable Specifications

In [80], it has been shown that an architecture is decidable for regular specifications if, and only if, it does not contain an information fork. This gives a comprehensive yet simple criterion for decidability that is just based on the graph structure of the architecture. Moreover, it shows that the decidability of pipelines for regular specifications as proved in [171] is, essentially, the best we can do for arbitrary regular specifications: Any architecture that does not contain an information fork can be reduced to a pipeline by certain technical manipulations of the architecture (and the given specification) [80].

However, although there are interesting problems that are based on pipeline architectures, pipelines are rather simple and it would be desirable to be able to model scenarios that involve more complicated communication structures, cf. [129]. As the result from [80] shows, for this we have to restrict the specifications somehow. One possibility would be to restrict the *expressive power* beneath regular specifications. On the other hand, most practical applications involve at least safety and/or reachability specifications and, as it turns out, the undecidability result from [80] does hold for safety conditions. In [94], *external* specifications have been considered which relate only the external inputs and the external outputs of the system but do not involve the internal communication channels. Clearly, this is a natural and relevant special case of the general controller problem for distributed systems and it has been shown, that for this kind of specifications, a larger class of architectures is decidable.

Although the specifications in [94] can talk only about a subset of all channels, they can involve an arbitrary amount of information that the controllers cannot observe. Contrary to this, we are interested in restricting the extent to which specifications may involve facts that the controllers cannot observe. In Section 2.2.3, we have defined the concept of *local* specifications. Local specifications address, indeed, only facts that *all* controllers can observe. However, as we have already pointed out there, in a seriously distributed system, there are no such facts, so local specifications are trivial for distributed systems. In [137], *locally decomposable* specifications have been considered, which we have also defined in Section 2.2.3. Recall that locally decomposable specifications can be decomposed into a collection of local specifications for the individual controllers.

In [137], a complete characterization of the decidable architectures has been obtained as well, but restricted to acyclic architectures: An *acyclic* architecture is decidable for locally decomposable *regular* specifications if, and only if, any connected component is a subarchitecture of a two-flanked pipeline. We extend this characterization to architectures which may contain cycles[1]

[1]Notice that in the sense of our definition of an architecture, this means *arbitrary architectures*. However, as we have mentioned in Section 2.1.3, our architectures still have certain shortcomings like no broadcast channels and at least one output channel for each controller.

and to specifications which are regular or context-free. Similar as in [80, 137], we characterize the decidable architectures primarily along the complexity of information flow between the processes, that means, structural patterns in the architecture. We do not, however, restrict the possible architectures a priori. So, throughout this Chapter we consider distributed systems with an arbitrary number of controllers and with (up to) context-free locally decomposable specifications.

Notice that for not necessarily locally decomposable context-free specifications, Theorem 3.20 completely settles the controller problem: We have shown that even for DCR-1 specifications and any nontrivial extension of the single controller system $\mathcal{D}_1$, the problem is not even recursively enumerable. Moreover, it was already known that $\mathcal{D}_1$ is undecidable for nondeterministic context-free specifications [81] and decidable for deterministic context-free specifications [219, 128].

For locally decomposable specifications it is important to notice the following difference between the regular case and the context-free case: In Section 2.2.3 we have already noted that if a specification is locally decomposable then a decomposition into local specifications can be obtained by projection. Since regular languages are closed under projection, for a given regular specification it is decidable whether it is locally decomposable and a decomposition into a collection of local regular specifications can be computed.

For context-free specifications the situation is more involved. Although at least (a very special) one of our decidable cases is extendable to nondeterministic context-free specifications, the undecidability of $\mathcal{D}_1$ for these specifications shows that, in general, they are way too expressive. So what we are really interested in are deterministic context-free specifications which are *not* closed under projection. However, this is a language theoretic issue while our focus is on the influence of the complexity of information flow in the system on decidability. So what we do here is, we assume that a locally decomposable context-free specification L brings along a decomposition $L = (L_1, \ldots, L_n)$ into local specifications[2], one for each controller, where each L_i is *deterministic* context-free. Of course, constraining some of the L_i to be regular may have a great influence on decidability and we will incorporate this into our characterization as well.

We prove the characterization in three steps. In Section 5.1 we start with basic decidability results. We first review the solution from [137] (Section 5.1.1), in particular, the concept of communication languages and their representation as trees. Moreover, we discuss the particular problem that arises in the context of locally decomposable specifications when the architectures may have cycles (Section 5.1.2) and we show how cycles can be partially removed in this case. More precisely, we show that *feedback channels* (cf. Section 2.1.3) can be effectively removed from any given architecture by a procedure that preserves local decomposability of the specifications. After that, we have two basic decidable cases left: First, we show that pipelines are decidable if each process $p_1, \ldots, p_{n-1}$ has a regular specification (and only p_n may have a deterministic context-free one.) Moreover we show that a two flanked pipeline with backward channels is decidable for regular specifications if it has only two controllers.

Section 5.2 provides matching undecidability results. In particular, we show that any pipeline where some processes p_i with $1 \leq i \leq n-1$ has a deterministic context-free specification is undecidable and any two-flanked pipeline with more than two processes and some backward channel from the last process is already undecidable for regular specifications. Moreover, any architecture with two connected processes that have a deterministic context-free specification is undecidable. In Section 5.3 we put these results together to get a complete characterization of *all* decidable cases. In Section 5.4 we discuss the issue of synthesizing strategies, using the methods that we have developed.

This chapter is based on [86].

[2]In fact, in [137] this was assumed as well but it didn't make a real difference there.

5.1 Decidability

Before we start developing algorithms for the decidable cases we like to comment on the solution that we present here. As we have already mentioned in the introduction to this chapter, we will demonstrate how to effectively remove backward channels from pipelines and (at least partially) from two-flanked pipelines in the context of locally decomposable specifications. Using this procedure we can reduce the controller problem for (two-flanked) pipelines with feedback channels to the problem for (two-flanked) pipelines without any feedback channels. For regular specifications, we can then directly apply the methods developed in [137]. The two remaining cases will be solved in Section 5.1.3.

On the other hand, the solution presented in [86] is more complicated. In fact, it was the authors' belief then, that in the realm of locally decomposable specifications, feedback channels could not be eliminated so easily. We discuss the particular problems with feedback channels in the context of locally decomposable specifications (that have led to this belief) in Section 5.1.2. Here, we present only the easier solution that proceeds by removing feedback channels. Nevertheless, the solution presented in [86] yields a more direct and coherent method for solving the controller problem for locally decomposable specification in the presence feedback channels. In particular, this method may be interesting when it comes to actually *synthesize* controllers. We discuss this issue in Section 5.4.

Throughout this section, let

$$\mathcal{D}_{n,p} = (\mathfrak{A}, (\Sigma_c)_{c \in C})$$

be a pipeline with $n+1$ processes. W.l.o.g. we assume that from a process p_i, $i < n$, there is exactly one channel to process p_{i+1}. Additionally, each process p_i has an external output channel. There are no other channels in the architecture $\mathfrak{A}$. Notice that hidden channels from the environment do not make any difference here because the specification is locally decomposable, so we just ditch it.

We denote $\mathfrak{A} = (C, r, w)$ as follows.

- $C = \{c_0\} \cup \{c_i, c_{i,e} \mid i = 1, \dots, n-1\} \cup \{c_{n,e}\}$
- $w(c_i) = p_i$ and $r(c_i) = p_{i+1}$
- $w(c_{i,e}) = p_i$ and $r(c_{i,e}) = p_0$

We also assume that $\Sigma_c = \mathbb{B}$ for all $c \in C$ and for a controller p_i let us denote

- $\Sigma_i := \prod_{c \in O_{p_i} \cap I_{p_{i+1}}} \Sigma_c$
- $\Sigma_{\text{out}}^{\geq i} := \Sigma^{\geq p_i} (= \prod_{j=i}^{n} \Sigma_{\text{out}}^{p_j})$

Notice that $\Sigma_{\text{out}} = \Sigma_{\text{out}}^{\geq 1}$. Moreover, let $\mathcal{D}_{n,p_2}$ be the corresponding *two-flanked* pipeline that is obtained from $\mathcal{D}_{n,p}$ by adding a channel c_{0n} from p_0 to p_n.

5.1.1 Communication Languages and Trees

The abstract reason why the controller problem for locally decomposable specifications $L = (L_1, \dots, L_n)$ is easier to solve is that any joint strategy $\sigma = (\sigma_1, \dots, \sigma_n)$ that is composed of local winning strategies for the individual processes is in fact a joint winning strategy for the grand coalition. On the other hand, not any joint winning strategy is actually composed of local winning strategies which is why the problem is still non-trivial, cf. Section 2.2.3.

However, to verify that a given joint strategy σ is winning, we can proceed more locally than in the case of arbitrary specifications: Consider any controller $i \in \{1, \dots, n\}$ and assume that i receives its inputs from processes $i_1, \dots, i_k$. Then, essentially, it suffices to check that σ_i is locally winning on L_{in}^i, where $L_{\text{in}}^i \subseteq (\Sigma_{\text{in}}^i)^\omega$ represents the possible inputs that p_i might receive from $i_1, \dots, i_k$, according to their local strategies. Such a language is called a *communication language* as it represents sequences of signals which can be sent along certain communication channels in the given architecture. Of course, to determine L^i_{in} we need, in turn, the possible inputs that $i_1, \ldots, i_k$ might receive from other processes. Likewise, the possible outputs of p_i have to be propagated to other processes. In general, this makes the problem still very difficult and, as we will see, in most cases even undecidable.

Nevertheless, in [137], Madhusudan and Thiagarajan exploited this general concept to show decidability of the controller problem for two-flanked pipelines in the case of locally decomposable regular specifications, which is undecidable in general. In the following we describe their approach.[3] First, let us discuss why the approach described in Section 3.2.1 for plain pipelines fails in this case: Recall the solution for $\mathcal{D}_{n,p}$ that we have described in Section 3.2.1: The first tree automaton $\mathcal{A}_1$ that we have constructed there checks, for a given Σ_{out}-labeled Σ_0-tree $t : \Sigma_0^* \to \Sigma_{\text{out}}$, that the extended strategy $\sigma_{\geq 1}$ for p_1 which t represents is *winning*. Then we check, iteratively, whether $\sigma_{\geq 1}$ is composed of local strategies of the individual controllers. As we have seen, for pipelines this approach goes smoothly, but for two-flanked pipeline it fails: Since p_n has an additional external input, an extended strategy for p_1 cannot include the decisions of p_n. But then we have no assembly point where we could check that the joint strategy for the grand coalition meets the whole global specification.

For local specifications, on the other hand, we don't need such an assembly point. Instead, the concept described above suggests the following approach: We compute, iteratively, sets of output languages $L^i_{\text{out}} \subseteq \Sigma_i^\omega$ for the controllers p_i such that each such L^i_{out} is possible according to an input language $L^i_{\text{in}} = L^{i-1}_{\text{out}}$ (that we already have) and some local *winning* strategy σ_i of process i. Recall that Σ_i is the alphabet that labels the channel from p_i to p_{i+1}. In particular, $\Sigma_n = \emptyset$. The tricky point arises again in the last step, where we have to deal with controller p_n: Due to the additional input channel c_{0n}, the language L^n_{in} does not contain the whole input of process p_n. However, we also know the language of possible inputs via c_{0n} which is simply $\Sigma^\omega_{c_{0n}}$, so we can proceed as follows: We check that there exists a language L^n_{in} with $L^n_{\text{in}} = L^{n-1}_{\text{out}}$ for some L^{n-1}_{out} and such that there is a strategy for p_n that is locally winning on $L^n_{\text{in}} \times \Sigma^\omega_{c_{0n}}$.

The first important step towards a technical realization of this idea is now to represent the communication languages $L^i_{\text{in}} = L^{i-1}_{\text{out}}$ adequately. Notice that we need to compute, for each controller i, a set of such languages L^i_{out} and we need to check, for each such language, that there exists a local winning strategy for p_i that generates it over some given language L^{i-1}_{out}. So, a representation by ω-automata is inept. Instead, in [137], the representation of a communication language as a tree has been used: Given an alphabet Σ, a $\mathbb{B}$-labeled Σ-tree t represents the ω-language

$$L^\omega(t) = \{\alpha \in \Sigma^\omega \mid t(\alpha < k) = \top \text{ for all } k \in \mathbb{N}\}.$$

The corresponding language of finite words is $L^*(t) = \{u \in \Sigma^* \mid t(u) = \top\}$.

Now if such a tree t represents in fact a *communication* language, then the $\top$-labeled nodes of t form a nonempty subtree of t, containing the root of t, that is, t has the following properties:

(C1) $t(\varepsilon) = \top$

(C2) if $t(u) = \bot$, then $t(ua) = \bot$ for all $a \in \Sigma$

(C3) if $t(u) = \top$, then $t(ua) = \top$ for some $a \in \Sigma$

We call $\mathbb{B}$-labeled Σ-trees which have the properties (C1) - (C3) communication trees over Σ and we denote the set of all such trees by $\mathbb{T}_c(\Sigma)$. Notice that for any $t \in \mathbb{T}_c(\Sigma)$ we have

$$L^*(t) = \{u \in \Sigma^* \mid u \sqsubseteq \alpha \text{ for some } \alpha \in L^\omega(t)\}.$$

That means, the corresponding language $L^*(t)$ of finite words as defined above is precisely the set of all finite prefixes of elements from $L^\omega(t)$ which is not true in general if $t \notin \mathbb{T}_c(\Sigma)$.

Proposition 5.1. *For any alphabet Σ, $\mathbb{T}_c(\Sigma)$ is a regular tree language which can be recognized by a deterministic safety tree automaton.*

[3] A full technical exposition can be found in [135].

Therefore, we can assume that all tree languages that we deal with are relativized to $\mathbb{T}_c(\Sigma)$ so we consider as input trees only communication trees.

Now, using this concept of communication trees, the controller problem for $\mathcal{D}_{n,p_2}$ with locally decomposable regular specifications can be solved as follows. We start with the controller p_1. Clearly, the input language of p_1 is Σ_{01}^ω, so we don't need to actually represent possible input languages for p_1 as communication trees. To represent the possible output languages we build an alternating parity tree automaton $\mathcal{A}_1$ which runs over trees $t \in \mathbb{T}_c(\Sigma_1)$ and checks that there is some strategy σ_1 for p_1 that is locally winning on Σ_{01}^ω such that $\sigma_1^\omega(\Sigma_{01}^\omega) \subseteq L^\omega(t)$. Recall that, for $L_{\text{in}} \subseteq \Sigma_{01}^\omega$,

$$\sigma_1^\omega(L_{\text{in}}) = \{\sigma_1^\omega(\alpha) \mid \alpha \in L_{\text{in}}\}.$$

Notice that we have stipulated merely $\sigma^\omega(\Sigma_{01}^\omega) \subseteq L^\omega(t)$ instead of equality. In fact, it seems hard to design an automata solution that would facilitate equality at this point, cf. [137]. This will become more apparent in the more technical description of the construction given below. However, inclusion is enough: If the remaining processes $p_2, \ldots, p_n$ can handle $L^\omega(t)$ as input language then clearly they can handle any subset of $L^\omega(t)$ as well.[4] In particular, they can handle $\sigma^\omega(L_{\text{in}})$, so it is no problem if we propagate any superset of $\sigma^\omega(L_{\text{in}})$ as well.

To see how $\mathcal{A}_1$ works, let us start at the root of t. The automaton *nondeterministically* guesses an element $(a, b) \in \Sigma_{\text{out}}^{p_1} = \Sigma_1 \times \Sigma_{1,e}$ which is the action chosen by p_1 in the first step according to the strategy σ that $\mathcal{A}_1$ is guessing. Then the automaton *universally* branches over all elements $c_1, \ldots, c_r \in \Sigma_{01}$ ($r = |\Sigma_{01}|$), which are the possible signals sent by p_0 to p_1 in the first step, and proceeds to copies $(\downarrow_a, c_1), \ldots, (\downarrow_a, c_r)$. Now, being at any node $(u, w) \in \Sigma_1^* \times \Sigma_{01}^*$ of the run, $\mathcal{A}_1$ first checks whether $t(u) = \bot$. If this is the case then $u \notin L^*(t)$, to u should not have been chosen as output of σ_1 according to t. Therefore, $\mathcal{A}_1$ immediately rejects. Otherwise, the automaton proceeds as before: It guesses an element $(a, b) \in \Sigma_{\text{out}}^{p_1}$, branches universally over all c_j, and proceeds to copies $(\downarrow_a, c_1), \ldots, (\downarrow_a, c_r)$. So it guesses, for each possible input c_j that p_1 might have received in this step, an answer of σ.

Additionally, $\mathcal{A}_1$ has to check the local specification $L_1 \subseteq \Sigma_{01} \times \Sigma_{\text{out}}^{p_1}$. This is done by simulating a *deterministic* parity automaton $\mathcal{S}_1$ with $L(\mathcal{S}_1) = L_1$ during the run as usual. It is not hard to see that $L(\mathcal{A}_1)$ is as described above.

Now, in the next step it gets more involved because there, we also have to incorporate the possible output languages of p_1 that $L(\mathcal{A}_1)$ represents. So we have to construct a tree automaton $\mathcal{A}_2$ that runs over trees $t \in \mathbb{T}_c(\Sigma_2)$ and checks that there is some strategy σ_2 for p_2 that is locally winning on *some* language $L^\omega(s)$ with $s \in L(\mathcal{A}_1)$ such that $\sigma_2^\omega(L^\omega(s)) \subseteq L^\omega(t)$. So, as before, the automaton $\mathcal{A}_2$ has to simulate a deterministic parity automaton $\mathcal{S}_2$ with $L(\mathcal{S}_2) = L_2$ and, additionally, $\mathcal{A}_2$ has to *guess* a tree $s \in \mathbb{T}_c(\Sigma_1)$ and simulate the automaton $\mathcal{A}_1$ on s.

This is done as follows: $\mathcal{A}_2$ keeps track of the current state q of $\mathcal{A}_1$ and of the current label $\zeta \in \{\bot, \top\}$ of the tree s that $\mathcal{A}_2$ guesses. We start with the initial state q_{in} of $\mathcal{A}_1$ and the label $\zeta = \top$. Now, in any node $(u, w) \in \Sigma_2^* \times \Sigma_1^*$ of the run, $\mathcal{A}_2$ checks whether $t(u) = \bot$ as before and if this is the case, it immediately rejects. If not then $\mathcal{A}_2$ guesses some $(a, b) \in \Sigma_2 \times \Sigma_{2,e}$ as before and, *additionally*, $\mathcal{A}_2$ guesses a subset $X \subseteq \Sigma_1$ of possible signals from p_1 *and* a transition φ of $\mathcal{A}_1$ according to q and ζ. Moreover, if $\zeta = \top$, we have to require $X \neq \emptyset$.

Then, as before, $\mathcal{A}_2$ branches over all $c_1, \ldots, c_r \in \Sigma_1$ and proceeds to copies $(\downarrow_a, c_1), \ldots, (\downarrow_a, c_r)$. Any such copy $(\downarrow_a, c_j)$ is equipped with a pair (p, ξ) where $(\downarrow_{c_j}, p) \in \varphi$ and $\xi = \top$ if $c_j \in X$ and $\xi = \bot$ if $c_j \notin X$. So, the next state of $\mathcal{A}_1$ is chosen according to (q, ζ) and then the choice of successors of the current node in s that are labeled with $\top$ according to X is propagated to the successor nodes in the run. Clearly, once we are in a situation where $\zeta = \bot$ we must restrain the choice of X to $X = \emptyset$. Additionally, we have to take care of the fact that the simulation of the *specification* automaton $\mathcal{S}_2$ must not proceed to copies $(\downarrow_a, c_j)$ with $c_j \notin X$ because these inputs are not possible according to s.

[4]Notice, however, that this kind of reasoning strongly relies on the fact that we consider *linear* time specifications.

Notice that before we construct $\mathcal{A}_2$, we turn $\mathcal{A}_1$ into a *nondeterministic* parity tree automaton. The reason is similar as in Section 3.2.1: A *run* of $\mathcal{A}_1$ does not only contain information about the signals from p_1 but also about the particular signals from p_0 to which they correspond. Now, a strategy for p_2 must not depend on these particular signals and turning $\mathcal{A}_1$ into a nondeterministic automaton hides this information from p_2.

The main argument for the correctness of this construction is that, in any run ρ of $\mathcal{A}_2$ on some tree $t \in \mathbb{T}_c(\Sigma_2)$ and for any $w \in \Sigma_1^*$, there is exactly one node (u, w) in ρ that corresponds to w (which, in fact, follows easily from the definition of $\mathcal{A}_2$). Moreover, any such node carries a label $\zeta \in \{\bot, \top\}$ so from ρ we can indeed extract a $\mathbb{B}$-labeled Σ_1-tree s and it is not hard to see that s satisfies (C1) - (C3), so $s \in \mathbb{T}_c(\Sigma_1)$. Moreover, since we have simulated $\mathcal{A}_1$ on s it follows that $s \in L(\mathcal{A}_1)$. So, from the Σ_2-components of the nodes of ρ and the $\Sigma_{2,e}$-actions that $\mathcal{A}_2$ has guessed during the run, we obtain a strategy σ_2 for p_2 that is locally winning on $L^\omega(s)$ such that $\sigma_2^\omega(L^\omega(s)) \subseteq L^\omega(t)$.

Notice that $\mathcal{A}_2$ has to verify two separate parity conditions at the same time, so we use a Muller condition for $\mathcal{A}_2$ and afterwards, turn $\mathcal{A}_2$ into an equivalent nondeterministic parity tree automata.

Now, up to process p_{n-1} we proceed by induction. To see what we've got after this, we have to define the *delay-composition* $\sigma_i|\ldots|\sigma_j$ of a collection $(\sigma_i, \ldots, \sigma_j)$ of local strategies for controllers $p_i \preceq \ldots \preceq p_j$ with $i < j$ in some distributed system $\mathcal{D}$. We do this by induction on $j - i$. First, we set $\sigma_i|\ldots|\sigma_i = \sigma_i$. Now let $j - i \geq 2$ and let $u \in (\Sigma_{\text{sig}}^{p_i})^*$. We define

- $\sigma_i|\ldots|\sigma_j(\varepsilon) = \sigma_j(\varepsilon)$
- $\sigma_i|\ldots|\sigma_j(u) = \sigma_j(\mathrm{Pr}_{\Sigma_{\text{sig}}^{p_j}}((\sigma_i|\ldots|\sigma_{j-1})^*(u^{-1})))$ for $u \neq \varepsilon$.

So, intuitively, given an input $u \in (\Sigma_{\text{sig}}^{p_i})^*$, the word $(\sigma_i|\ldots|\sigma_{j-1})^*(u)$ is what process j actually gets to see of u, but starting only after $i - 1$ steps of routing.

Proposition 5.2. *If $\alpha \in (\Sigma^{\mathcal{D}})^\omega$ is a global system run which is consistent with $\sigma_i, \ldots, \sigma_j$ then $(\sigma_i|\ldots|\sigma_j)(Pr_{\Sigma_{sig}^{p_i}}(\alpha_{<k})) = \sigma_j(Pr_{\Sigma_{sig}^{p_j}}(\alpha_{<k}))$ for all $k \in \mathbb{N}$.*

Using the delay-composition we can formulate the properties of the alternating parity tree automaton $\mathcal{A}_{n-1}$ over trees $t \in \mathbb{T}_c(\Sigma_{n-1})$ that we have obtained in the last induction step: $\mathcal{A}_{n-1}$ accepts such a tree t if, and only if, there exist local strategies $\sigma_1, \ldots, \sigma_{n-1}$ such that:

(1) $\sigma_1|\ldots|\sigma_{n-1}$ generates a language $L_{\text{out}}^{n-1} \subseteq L^\omega(t)$ over Σ_{01}^ω

(2) Each strategy σ_i is locally winning on the language $L_{\text{in}}^i \subseteq \Sigma_{i-1}^\omega$ generated by $\sigma_1|\ldots|\sigma_{i-1}$ over Σ_{01}^ω

However, due to the additional inputs that p_n receives from p_0, in the last step, we cannot go on like this: Consider an automaton which runs over trees $t \in \mathbb{T}_c(\Sigma_n)$ that represent output languages of p_n (which now contain only external outputs). The automaton would have to check that there exists a strategy σ_n for p_n that is locally winning on $L^\omega(s) \times \Sigma_{0n}^\omega$ for *some* language $L^\omega(s)$ with $s \in L(\mathcal{A}_{n-1})$ such that $\sigma_{n-1}^\omega(L^\omega(s)) \subseteq L^\omega(t)$. To do so, the automaton needs to guess the tree s and *simultaneously* incorporate the possible (in this case all) signals from Σ_{0n}. But this may cause interferences: $\mathcal{A}_n$ could guess the sets $X \subseteq \Sigma_{n-1}$ of possible signals from p_{n-1} in dependence of the signals that p_n has received from p_0 which is, of course, fatal.

Instead, we proceed as follows: We construct an automaton $\mathcal{A}_n$ over trees $t \in \mathbb{T}_c(\Sigma_{n-1})$ that represent possible *input* languages for p_n. Then we have to incorporate only the inputs from p_{0n} that p_n receives via c_{0n}. More precisely, given an input tree $t \in \mathbb{T}_c(\Sigma_{n-1})$, $\mathcal{A}_n$ works as follows. At the root of t, $\mathcal{A}_n$ nondeterministically guesses an action $b \in \Sigma_{\text{out}}^{p_n} = \Sigma_{n,e}$. Then, $\mathcal{A}_n$ universally branches over all $(a_1, c_1), \ldots, (a_r, c_s) \in \Sigma_{n-1} \times \Sigma_{0,n}$ ($r = |\Sigma_{n-1}|$, $s = |\Sigma_{0,n}|$) and proceeds to copies $(\downarrow_{a_1}, c_1), \ldots, (\downarrow_{a_r}, c_s)$. At the same time, $\mathcal{A}_n$ simulates a deterministic parity automaton $\mathcal{S}_n$ with $L(\mathcal{S}_n) = L_n$ on the branches of the run. Now, being in any node $(u, w) \in \Sigma_{n-1}^* \times \Sigma_{0,n}^*$ of the run, if $t(u) = \bot$ then $\mathcal{A}_n$ stops simulating $\mathcal{S}_n$ on this branch and goes to some accepting state. The reason is that $t(u) = \bot$ means that $u \notin L^*(t)$, so input u

will not be generated by any collection of strategies $\sigma_1, \ldots, \sigma_{n-1}$ for $p_1, \ldots, p_n$ that correspond to t according to (1) and (2) as above. The important point is that now these possible inputs are not guessed by $\mathcal{A}_n$ but determined by the tree t so they will not depend on the additional inputs that p_n receives from p_0.

So we have $t \in L(\mathcal{A}_n)$ if, and only if, there is a strategy σ_n for p_n that is locally winning on $L^\omega(t) \times \Sigma_{0,n}^\omega$. Therefore, $t \in L(\mathcal{A}_{n-1}) \cap L(\mathcal{A}_n)$ if, and only if, there are strategies $\sigma_1, \ldots, \sigma_{n-1}$ for $p_1, \ldots, p_{n-1}$ such that (1) and (2) hold *and* there is a strategy σ_n for p_n that is locally winning on $L_{\text{out}}^{n-1} \times \Sigma_{0,n}^\omega$. Since emptiness of $L(\mathcal{A}_{n-1}) \cap L(\mathcal{A}_n)$ is decidable, the desired result follows.

Theorem 5.3. *[137] The controller problem for $\mathcal{D}_{n,p_2}$ is decidable for locally decomposable regular specifications.*

To characterize the decidable *acyclic* architectures for locally decomposable regular specifications this result is the only serious extension of the known decidability results for arbitrary regular specifications that one needs. Having this and the matching undecidability results [137], one can show the following:

Theorem 5.4. *[137] An acyclic architecture is decidable for locally decomposable regular specifications if, and only if, any connected component is a subarchitecture of a pipeline or a two-flanked pipeline.*

We extend this result to cyclic architectures and to locally decomposable specifications that are composed of a collection of deterministic context-free specifications. For this, we need two extensions of Theorem 5.3. Consider a locally decomposable specification $L = (L_1, \ldots, L_n)$ where each L_j is deterministic context-free. First, we show that any pipeline with backward channels is decidable if each L_j with $1 \leq j \leq n-1$ is regular and L_n is deterministic context-free. Second, we show that any two-flanked pipeline with backward channels is decidable if each L_j is regular and additionally, $n = 2$ or there are no backward channels from the last process p_n. In Section 5.2 we will see that, indeed, any nontrivial extension of any of these two cases is again undecidable. In the next section we first demonstrate how to effectively remove feedback channels from pipelines and two-flanked pipelines with backward channels using a procedure that preserved local decomposability of the specification. This leaves us with two cases: Pipelines where each L_j with $1 \leq j \leq n-1$ is regular and L_n is deterministic context-free and two-flanked pipelines where each L_j is regular and additionally $n = 2$. These cases will be taken care of in Section 5.1.3.

5.1.2 Removing Feedback Channels

The important point about feedback channels in the case of locally decomposable specifications is that they may increase the access of the (global) specification to the global system behavior: If, for example, we have a pipeline with $n \geq 5$ processes and there is a backward channel from process p_5 to process p_3 then this allows us to relate the input channel c_2 of process p_3 to the output channel c_6 of process p_5. Clearly, this is not possible without backward channels. Due to this increased access of the local specifications to the global system behavior, the approach described above cannot be readily applied here: It is now not sufficient to check any individual specification locally for each process but we would have to have an assembly point again as described above for arbitrary specifications.

On the other hand, in Section 2.1.3 we have seen that the grand coalition has a winning strategy if, and only if, it has a focused one (Proposition 2.4), that means, each individual strategy only reads the significant input. In other words, given a joint strategy for the grand coalition, the individual strategies for the processes p_i need not to depend on the inputs that are received via feedback channels because they can be deduced them from their own actions and the strategies of the other processes. This allows us to eliminate all feedback channels from a given architecture by turning them into external output channels, so any architecture without an information fork can be turned into a normalform that is acyclic [80]. In particular, we can eliminate backward channels from pipelines.

However, since the *specification* may talk about these feedback channels, this transformation affects the specification as well and it is obvious that, in general, it does *not* preserve local decomposability. Nevertheless, it is not very difficult to extend this construction so that it works for locally decomposable specifications as well. To describe the construction, consider a pipeline $\mathcal{D}_{n,pb}$ with backward channels. So $\mathcal{D}_{n,pb}$ is obtained from the pipeline $\mathcal{D}_{n,p}$ by adding some set B of backward channels, that is, each channel $c \in B$ satisfies $w(c) = i$ and $r(c) = j$ for some $1 \leq j < i \leq n$. We use notation as before where now, additionally, $B \subseteq C$. Notice that since we consider a one-flanked pipeline, all backward channels are feedback channels. Moreover, consider a locally decomposable specification $L = (L_1, \dots, L_n)$ where each L_i is regular or deterministic context-free.

Now consider some $c \in B$ and let $j < i$ such that $w(c) = i$ and $r(c) = j$. Let $\tilde{\mathcal{D}}_{n,pb}$ be obtained from $\mathcal{D}_{n,pb}$ as follows. First, we re-define $\tilde{r}(c) = p_0$, that is, c is now an external output channel of p_i. Moreover, we add new channels $\tilde{c}_l$ for $l = j, \dots, i-1$ with $\tilde{w}(\tilde{c}_l) = p_l$ and $\tilde{r}(\tilde{c}_l) = l+1$ and we denote $\tilde{c}_i := c$.

The specification L is modified to the new specification $\tilde{L} = (\tilde{L}_1, \dots, \tilde{L}_n)$ as follows. First, for $l \in \{1, \dots, n\} \setminus \{j, \dots, i\}$ we have $\tilde{L}_l = L_l$. Moreover, for $\beta^\frown\alpha \in (\tilde{\Sigma}_{\text{in}}^{p_j})^\omega \times (\tilde{\Sigma}_{\text{out}}^{p_j})^\omega$ we have

$$\beta^\frown\alpha \in \tilde{L}_j \quad :\Longleftrightarrow \quad \beta^\frown\text{Pr}_{\Sigma_{\tilde{c}_l}}(\alpha)^\frown\text{Pr}_{\Sigma_{\text{out}}^{p_j}}(\alpha) \in L_j.$$

So the specification of p_j in $\tilde{\mathcal{D}}_{n,pb}$ is the same as in $\mathcal{D}_{n,pb}$ with the new *output* channel $\tilde{c}_j$ of p_j in $\tilde{\mathcal{D}}_{n,pb}$ corresponding to the *input* channel c of p_j in $\mathcal{D}_{n,pb}$. Finally, for $j+1 \leq l \leq i$ we define

$$\tilde{L}_l := \{\alpha \in (\tilde{\Sigma}^{p_l})^\omega \mid \text{Pr}_{\Sigma^{p_l}}(\alpha) \in L_l\} \cap \{\alpha \in (\tilde{\Sigma}^{p_l})^\omega \mid \text{Pr}_{\Sigma_{\tilde{c}_{l-1}}}(\alpha) = \text{Pr}_{\Sigma_{\tilde{c}_l}}(\alpha)\}.$$

So on the channels from $\mathcal{D}_{n,pb}$, the specification $\tilde{L}_l$ coincides with the original specification L_l and, additionally, we require that, in each step, p_l writes the same signal to $\tilde{c}_l$ that it receives via $\tilde{c}_{l-1}$. Notice that all processes $p_j, \dots, p_i$ choose these symbols *simultaneously*, so p_l does not see the signal from $\tilde{c}_{l-1}$ before it writes it to $\tilde{c}_l$. Instead, each process p_l with $j+1 \leq l \leq i$ has to *predict* the symbol that it will receive via $\tilde{c}_{l-1}$. This, however, is possible (if and) only if all processes $p_j, \dots, p_i$ use a strategy for these new channels $\tilde{c}_l$ that p_i could also use in the original system $\mathcal{D}_{n,pb}$ for the channel c. In this way, the channel c is simulated on all the channels $\tilde{c}_l$ simultaneously. In particular, it is simulated on c and on $\tilde{c}_j$ simultaneously which guarantees that for all system runs that are won by the grand coalition, the specifications L and $\tilde{L}$ coincide (up to some technical overhead).

Proposition 5.5. *The grand coalition has a winning strategy for $(\mathcal{D}_{n,pb}, L)$ if, and only if, the grand coalition has a winning strategy for $(\tilde{\mathcal{D}}_{n,pb}, \tilde{L})$.*

Proof. First, assume that there is a joint winning strategy $\sigma = (\sigma_1, \dots, \sigma_n)$ for the grand coalition for $(\mathcal{D}_{n,pb}, L)$. According to Proposition 2.4 we can assume that σ is focused, that means, each individual strategy is a function $\sigma_l : \Sigma_{l-1}^* \to \Sigma_{\text{out}}^{p_l}$. We define a joint strategy $\tilde{\sigma} = (\tilde{\sigma}_1, \dots, \tilde{\sigma}_n)$ for the grand coalition for $\tilde{\mathcal{D}}_{n,pb}$ that consists of functions $\tilde{\sigma}_l : \Sigma_{l-1}^* \to \tilde{\Sigma}_{\text{out}}^{p_l}$. So any individual strategy is independent of the inputs that it receives via any of the new channels $\tilde{c}_l$. (The actual strategy is then defined in the obvious way as $\text{Pr}_{\Sigma_{l-1}} \circ \tilde{\sigma}_l$.)

For each $l \in \{1, \dots, n\} \setminus \{j, \dots, i-1\}$ we simply set $\tilde{\sigma}_l = \sigma_l$. For $l \in \{j, \dots, i\}$, we define

$$\tilde{\sigma}_l(u) = \sigma_l(u)^\frown\text{Pr}_{\Sigma_c}(\sigma_l| \dots |\sigma_i(u)).$$

So, on the original channels from $\mathcal{D}_{n,pb}$, p_l plays according to σ_l and on the channel $\tilde{c}_l$ p_l plays according to σ_i.

To show that $\tilde{\sigma}$ is winning, consider some global system run $\alpha \in (\Sigma^{\tilde{\mathcal{D}}_{n,pb}})^\omega$ that is consistent with $\tilde{\sigma}$ and let $\beta = \Pr_{\Sigma^{\mathcal{D}_{n,pb}}}(\alpha)$ be the corresponding run of $\mathcal{D}_{n,pb}$ where we simply ignore all the additional channels. First, for all $1 \leq l \leq n$ and any $\lambda \in \mathbb{N}$ we have $\Pr_{\Sigma^{p_l}_{\text{out}}}(\beta(\lambda)) = \Pr_{\Sigma^{p_l}_{\text{out}}}(\Pr_{\tilde{\Sigma}^{p_l}_{\text{out}}}(\alpha(\lambda))$ and since α is consistent with $\tilde{\sigma}$ it follows that

$$\Pr_{\Sigma^{p_l}_{\text{out}}}(\beta(\lambda)) = \Pr_{\Sigma^{p_l}_{\text{out}}}(\tilde{\sigma}_l(\Pr_{\Sigma_l}(\alpha_{<\lambda}))) = \sigma_l(\Pr_{\Sigma_l}(\beta_{<\lambda})).$$

Therefore, β is consistent with σ and since σ is winning, $\beta \in L$ which implies $\Pr_{\tilde{\Sigma}^{p_l}}(\alpha) \in \{\gamma \in \tilde{\Sigma}^{p_l} \mid \Pr_{\Sigma^{p_l}}(\gamma) \in L_l\}$ for all $l \in \{1, \ldots, n\} \setminus \{j\}$.

Moreover, since α is consistent with $\tilde{\sigma}$, for all $l \in \{j, \ldots, i\}$ and all $\lambda \in \mathbb{N}$ we have

$$\Pr_{\Sigma_{\tilde{c}_l}}(\alpha_\lambda) = \Pr_{\Sigma_{\tilde{c}_l}}(\tilde{\sigma}_l(\Pr_{\Sigma_{l-1}}(\alpha_{<\lambda})) = \Pr_{\Sigma_{\tilde{c}_l}}(\sigma_l | \ldots | \sigma_i(\Pr_{\Sigma_{l-1}}(\alpha_{<\lambda}))).$$

Therefore, using Proposition 5.2, we get

$$\Pr_{\Sigma_{\tilde{c}_l}}(\alpha_\lambda) = \Pr_{\Sigma_{\tilde{c}_l}}(\sigma_i(\Pr_{\Sigma_{i-1}}(\alpha_{<\lambda}))) = \Pr_{\Sigma_c}(\sigma_i(\Pr_{\Sigma_{i-1}}(\alpha_{<\lambda}))) = \Pr_{\Sigma_c}(\alpha_\lambda).$$

In particular, we have $\Pr_{\Sigma_{\tilde{c}_j}}(\alpha) = \Pr_{\Sigma_c}(\alpha)$ which implies that also $\Pr_{\tilde{\Sigma}^{p_j}}(\alpha) \in \{\gamma \in \tilde{\Sigma}^{p_j} \mid \Pr_{\Sigma^{p_j}}(\gamma) \in L_j\}$ holds, and we have

$$\Pr_{\tilde{\Sigma}^{p_l}}(\alpha) \in \{\gamma \in (\tilde{\Sigma}^{p_l})^\omega \mid \Pr_{\Sigma_{\tilde{c}_{l-1}}}(\gamma) = \Pr_{\Sigma_{\tilde{c}_l}}(\gamma)\}$$

for all $l \in \{j, \ldots, i\}$. Hence, $\alpha \in \tilde{L}$.

Now let $\tilde{\sigma} = (\tilde{\sigma}_1, \ldots, \tilde{\sigma}_n)$ be a joint winning for the grand coalition for $(\tilde{\mathcal{D}}_{n,pb}, \tilde{L})$. As before, we assume that $\tilde{\sigma}$ is focused.[5] Notice that here this means that any individual strategy $\tilde{\sigma}_l$ is a function $\tilde{\sigma}_l : \Sigma_l^* \to \tilde{\Sigma}^{p_l}_{\text{out}}$ if $l \notin \{j, \ldots, i\}$ but if $l \in \{j, \ldots, i\}$ then it is a function $\tilde{\sigma}_l : (\Sigma_l \times \Sigma_{\tilde{c}_{l-1}})^* \to \tilde{\Sigma}^{p_l}_{\text{out}}$. We construct a focused strategy $\sigma = (\sigma_1, \ldots, \sigma_n)$ for the grand coalition for $\mathcal{D}_{n,pb}$ as follows.

For all $l \in \{1, \ldots, n\} \setminus \{j, \ldots, i\}$ we simply set $\sigma_l = \tilde{\sigma}_l$ and we set $\sigma_j = \tilde{\sigma}_j \circ \Pr_{\Sigma^{p_j}_{\text{out}}}$, that is, we project the additional channel $\tilde{c}_j$ out. For $l \in \{j+1, \ldots, i\}$ we define σ_l inductively on the length of the inputs $u \in \Sigma_{l-1}^*$, and with each such input we associate a word $w_l(u) \in \Sigma_{\tilde{c}_{l-1}}^{|u|+1}$. For $u = \varepsilon$ we set $\sigma_l(\varepsilon) = \Pr_{\Sigma^{p_l}_{\text{out}}}(\tilde{\sigma}_l(\varepsilon))$ and $w_l(\varepsilon) = \Pr_{\Sigma_{\tilde{c}_l}}(\tilde{\sigma}_l(\varepsilon))$. Now let $u \neq \varepsilon$. Then

$$\sigma_l(u) = \Pr_{\Sigma^{p_l}_{\text{out}}}(\tilde{\sigma}_l(u^\frown w_l(u^{-1}))) \quad \text{and} \quad w_l(u) = \Pr_{\Sigma_{\tilde{c}_l}}(\tilde{\sigma}_l(u^\frown w_l(u^{-1}))).$$

The point here is that the input which p_l receives via $\tilde{c}_{l-1}$ in $\tilde{\mathcal{D}}_{n,pb}$ is not available for the strategy σ_l in $\mathcal{D}_{n,pb}$. However, since we know that all processes p_k with $k \in \{j, \ldots, i\}$ write the same signals into the channels $\tilde{c}_k$ according to $\tilde{\sigma}$ (otherwise, $\tilde{\sigma}$ could not be winning) we can simply plug in the output that p_l itself would have produced on channel $\tilde{c}_l$ so far in the system $\tilde{\mathcal{D}}_{n,pb}$.[6]

Now consider any global system run $\alpha \in (\Sigma^{\mathcal{D}_{n,pb}})^\omega$ of $\mathcal{D}_{n,pb}$ that is consistent with σ and let $\beta \in (\Sigma^{\tilde{\mathcal{D}}_{n,pb}})^\omega$ be the unique system run of $\tilde{\mathcal{D}}_{n,pb}$ with $\Pr_{\Sigma_0}(\beta) = \Pr_{\Sigma_0}(\alpha)$ that is consistent with $\tilde{\sigma}$. Since $\tilde{\sigma}$ is winning, β is won by the grand coalition. By definition of $\tilde{L}$ this implies $\Pr_{\Sigma_{\tilde{c}_{l-1}}}(\beta) = \Pr_{\Sigma_{\tilde{c}_l}}(\beta)$ for all $l \in \{j+1, \ldots, i\}$. In particular, we have $\Pr_{\Sigma_{\tilde{c}_j}}(\beta) = \Pr_{\Sigma_c}(\beta)$ so by definition of $\tilde{L}_j$ it only remains to show that $\Pr_{\Sigma^{\mathcal{D}_{n,pb}}}(\beta) = \Pr_{\Sigma^{\mathcal{D}_{n,pb}}}(\alpha)$. For $l \in \{1, \ldots, n\} \setminus \{j+1, \ldots, i\}$ it follows directly from the definitions, using the fact that α and β are consistent with σ and $\tilde{\sigma}$, respectively, that $\Pr_{\Sigma^{p_l}_{\text{out}}}(\beta) = \Pr_{\Sigma^{p_l}_{\text{out}}}(\alpha)$. For $l \in \{j+1, \ldots, i\}$ we show, by induction on λ, that the following holds:

[5]For this direction, however, this is just a technical simplification while for the other direction it is essential since we have deleted the feedback channel from p_i to p_j.

[6]Which is the more precise formulation of the fact that all processes $p_j, \ldots, p_i$ use a strategy for the new channels $\tilde{c}_l$ that p_i could also use in the original system $\mathcal{D}_{n,pb}$ for the channel c.

(1) $\mathrm{Pr}_{\Sigma_{\mathrm{out}}^{p_l}}(\beta_\lambda) = \mathrm{Pr}_{\Sigma_{\mathrm{out}}^{p_l}}(\alpha_\lambda)$

(2) $w_l(\mathrm{Pr}_{\Sigma_{l-1}}(\beta_{<\lambda})) = \mathrm{Pr}_{\Sigma_{\tilde{c}_{l-1}}}(\beta_{<\lambda+1})$

The case $\lambda = 0$ follows directly from the definitions, so let $\lambda > 0$. First we show condition (2). By definition we have

$$w_l(\mathrm{Pr}_{\Sigma_{l-1}}(\beta_{<\lambda})) = \mathrm{Pr}_{\Sigma_{\tilde{c}_l}}(\tilde{\sigma}_l(\mathrm{Pr}_{\Sigma_{l-1}}(\beta_{<\lambda})^\frown w_l(\mathrm{Pr}_{\Sigma_{l-1}}(\beta_{<\lambda-1}))))$$

so, by induction hypothesis for (2),

$$w_l(\mathrm{Pr}_{\Sigma_{l-1}}(\beta_{<\lambda})) = \mathrm{Pr}_{\Sigma_{\tilde{c}_l}}(\tilde{\sigma}_l(\mathrm{Pr}_{\Sigma_{l-1}\times\Sigma_{\tilde{c}_{l-1}}}(\beta_{<\lambda}))).$$

Therefore, since β is consistent with $\tilde{\sigma}$ and $\mathrm{Pr}_{\Sigma_{\tilde{c}_{l-1}}}(\beta) = \mathrm{Pr}_{\Sigma_{\tilde{c}_l}}(\beta)$, we obtain

$$w_l(\mathrm{Pr}_{\Sigma_{l-1}}(\beta_{<\lambda})) = \mathrm{Pr}_{\Sigma_{\tilde{c}_l}}(\mathrm{Pr}_{\Sigma_{\mathrm{out}}^{p_l}}(\beta_{<\lambda+1})) = \mathrm{Pr}_{\Sigma_{\tilde{c}_{l-1}}}(\beta_{<\lambda+1}).$$

Now, using (2) and the fact that β is consistent with $\tilde{\sigma}$, we obtain

$$\mathrm{Pr}_{\Sigma_{\mathrm{out}}^{p_l}}(\beta_\lambda) = \mathrm{Pr}_{\Sigma_{\mathrm{out}}^{p_l}}(\tilde{\sigma}_l(\mathrm{Pr}_{\Sigma_{l-1}}(\beta_{<\lambda})^\frown w_l(\mathrm{Pr}_{\Sigma_{l-1}}(\beta_{<\lambda-1}))))$$

and by induction hypothesis for (1), this yields

$$\mathrm{Pr}_{\Sigma_{\mathrm{out}}^{p_l}}(\beta_\lambda) = \mathrm{Pr}_{\Sigma_{\mathrm{out}}^{p_l}}(\tilde{\sigma}_l(\mathrm{Pr}_{\Sigma_{l-1}}(\alpha_{<\lambda})^\frown w_l(\mathrm{Pr}_{\Sigma_{l-1}}(\alpha_{<\lambda-1})))) = \sigma_l(\mathrm{Pr}_{\Sigma_{l-1}}(\alpha_{<\lambda})).$$

Therefore, since α is consistent with σ, (1) holds as well. □

Now, given a pipeline $\mathcal{D}_{n,pb}$ with backward channels and a locally decomposable specification $L = (L_1, \dots, L_n)$ we can eliminate all backward channels from $\mathcal{D}_{n,pb}$ by iterating the procedure described above. In the end, this yields a straight pipeline $\mathcal{D}_{b,pb}$ without any backward channels and a new specification $\tilde{L}$ which is again locally decomposable. Moreover, the decomposition $\tilde{L} = (\tilde{L}_1, \dots, \tilde{L}_n)$ can be effectively constructed from $(L_1, \dots, L_n)$ and, for each $i \in \{1, \dots, n\}$, if L_i is regular (deterministic context-free) then $\tilde{L}_i$ is regular (deterministic context-free).

In the same way, given a two-flanked pipeline $\mathcal{D}_{n,p_2b}$ with backward channels and a locally decomposable specification $L = (L_1, \dots, L_n)$, we can eliminate all backward channels from $\mathcal{D}_{n,p_2b}$ that do *not* originate in the last process p_n. Clearly, the construction described above relies on the fact that p_j is better informed than p_i and, indeed, backward channels from the last process p_n (which are not feedback channels!) cannot be removed in this way. In fact, any two-flanked pipeline with more than two processes and at least one backward channel from the last process is undecidable even if each L_i is regular, cf. Section 5.2. However, all backward channels from $p_2, \dots, p_{n-1}$ can be removed from $\mathcal{D}_{n,p_2n}$ using the construction described above.

5.1.3 The Remaining Cases

As we have mentioned before, after we have removed all feedback channels using the procedure described in the previous section, there are two cases left that we still have to prove decidable:

- Pipelines with specifications $L = (L_1, \dots, L_n)$ where each L_i with $i < n$ is regular and L_n is regular or deterministic context-free
- Two-flanked pipelines with backward channels and *two* controllers p_1 and p_2 each of which has a regular local specification L_i

Pipelines. We start with the first case. Up to process p_{n-1} we use the method from [137] that we have described in Section 5.1.1. As we have explained there, this yields an alternating parity tree automaton $\mathcal{A}_{n-1}$ over trees $t \in \mathbb{T}_c(\Sigma_{n-1})$ which accepts such a tree t if, and only if, there exist local strategies $\sigma_1, \dots, \sigma_{n-1}$ such that:

(1) $\sigma_1|\dots|\sigma_{n-1}$ generates a language $L^{n-1}_{\text{out}} \subseteq L^\omega(t)$ over Σ^ω_{01}
(2) Each strategy σ_i is locally winning on the language $L^i_{\text{in}} \subseteq \Sigma^\omega_{i-1}$ generated by $\sigma_1|\dots|\sigma_{i-1}$ over Σ^ω_{01}

Recall that to solve the strategy problem for two-flanked pipelines with completely regular specifications, we had to account for the additional input from the environment. For this, we have constructed an *alternating* parity tree automaton which essentially spanned the possible inputs from the environment over the given $\mathbb{B}$-labeled Σ_{n-1}-tree.

Here, on the other hand, we have a *deterministic context-free* specification L_n, so we cannot solve the problem by constructing an alternating tree automaton: In order to check the local specification L_n the tree automaton must be a parity pushdown tree automaton itself, but as we already know from Section 3.1.3, the emptiness problem for alternating parity pushdown tree automata is undecidable.[7] However, since we consider only a straight pipeline, we do not have any additional input channel from the environment, so there is no need to build an alternating tree automaton here: We only have to *guess* the external outputs of p_n which can be done by a nondeterministic tree automaton. We formulate this step in the following lemma and then we obtain the solution by turning the alternating parity tree automaton $\mathcal{A}_{n-1}$ that we have already have into a nondeterministic one and intersecting it with the nondeterministic parity pushdown tree automaton $\mathcal{A}_n$ that we get out of the lemma.

Lemma 5.6. *There is a nondeterministic parity pushdown tree automaton $\mathcal{A}_n$ over communication trees $t \in \mathbb{T}_c(\Sigma_{n-1})$ that accepts such a tree if, and only if, there is a strategy σ_n for p_n that is locally winning on $L^\omega(t)$.*

Proof. Let $\mathcal{P} = (\Sigma_{n-1} \times \Sigma_{n,e}, \Gamma, Q^{\mathcal{P}}, q^{\mathcal{P}}_{\text{in}}, \delta^{\mathcal{P}}, \text{col}^{\mathcal{P}})$ be a deterministic parity pushdown automaton with $L(\mathcal{P}) = L_n$. Moreover, let $\Sigma_{n-1} = \{a_0, \dots, a_{r-1}\}$. We construct the automaton $\mathcal{A}_n = (\mathbb{B}, \Gamma, Q, q_{\text{in}}, \delta, \text{col})$ as follows. The idea is straightforward: In each step, $\mathcal{A}_n$ guesses an action $b \in \Sigma_{n,e}$ and proceeds to all directions $a_0, \dots, a_{r-1} \in \Sigma_{n-1}$ and at the same time, the automaton simulates $\mathcal{P}$ on all paths. However, as soon as $\mathcal{A}_n$ encounters a label $\perp$ at some node $u \in \Sigma^*_{n-1}$ then it stops simulating $\mathcal{A}_n$ and instead goes to some accepting state because the input u will not be send by p_{n-1} according to t. Moreover, to simulate $\mathcal{P}$, at any node $u \in \Sigma^*_{n-1}$, $\mathcal{A}_n$ has to process all the ε-transitions of $\mathcal{P}$ first and then proceed to the successors of u. Let us define the automaton formally:

- $Q = Q^{\mathcal{P}} \times \Sigma_{n,e} \cup \{q_{\text{acc}}\}$
- $q_{\text{in}} = (q^{\mathcal{P}}_{\text{in}}, b)$ for some $b \in \Sigma_{n,e}$
- $\text{col}(q, b) = \text{col}^{\mathcal{P}}(q)$ and $\text{col}(q_{\text{acc}}) = 0$
- $\delta(q, A, \perp) = \bigwedge_{[a \in \Sigma_{n-1}]} (\downarrow_a, q_{\text{acc}}, A)$
- $\delta(q_{\text{acc}}, A, \zeta) = \bigwedge_{[a \in \Sigma_{n-1}]} (\downarrow_a, q_{\text{acc}}, A)$
- for $q = (q^{\mathcal{P}}, b)$ and $\delta(q^{\mathcal{P}}, A, \varepsilon) \neq \emptyset$,
$$\delta_{\circlearrowleft}(q, A) = (\delta^{\mathcal{P}}(q^{\mathcal{P}}, A, \varepsilon), b)$$
- for $q = (q^{\mathcal{P}}, b)$ and $\delta(q^{\mathcal{P}}, A, \varepsilon) \neq \emptyset$,
$$\delta_{\downarrow}(q, A, \zeta) = \bigvee_{[b' \in \Sigma_{n,e}]} \bigwedge_{a \in \Sigma_{n-1}} (\downarrow_a, (\delta^{\mathcal{P}}(q^{\mathcal{P}}, A, (a, b')), b'))$$

Recall that since $\mathcal{P}$ is deterministic, $|\delta^{\mathcal{P}}(q^{\mathcal{P}}, A, (a, b))| + |\delta^{\mathcal{P}}(q^{\mathcal{P}}, A, \varepsilon)| \leq 1$ for all transitions $(q^{\mathcal{P}}, A, (a, b)) \in Q^{\mathcal{P}} \times \Gamma \times (\Sigma_{n-1} \times \Sigma_{n,e})$. In particular, for any $(q, A) \in Q \times \Gamma$, if $\delta_{\circlearrowleft}(q, A)$ contains some transition then $\delta(q, A, \zeta)$ is empty for all $\zeta \in \mathbb{B}$.

[7]We will see in Section 5.2 that, indeed, any two-flanked pipeline with more than one controllers where at least one controller has a deterministic context-free local specification is undecidable.

Notice that the transition function δ does actually not use the $\Sigma_{n,e}$-component of Q. However, incorporating these nondeterministic choices of $\mathcal{A}_n$ into the state space explicitly helps to extract a strategy for p_n from a given run of $\mathcal{A}_n$:

Let $\rho : T \to \Sigma_{n-1}^* \times Q \times \Gamma^*\bot$ be a run of $\mathcal{A}_n$ on some given input tree $t \in \mathbb{T}_c(\Sigma_{n-1})$. By construction of $\mathcal{A}_n$, for all $u \in \Sigma_{n-1}^*$ with $t(u) = \top$, there is some $b \in \Sigma_{n,e}$ such that for all $a \in \Sigma_{n-1}$ and all $x \in T$ with $\mathrm{Pr}_{\Sigma_{n-1}^*}(\rho(x)) = ua$ we have $\mathrm{Pr}_{\Sigma_{n,e}}(\rho(x)) = b$.[8] (Notice that any two nodes $x, x' \in T$ with $\mathrm{Pr}_{\Sigma_{n-1}^*}(\rho(x)) = \mathrm{Pr}_{\Sigma_{n-1}^*}(\rho(x'))$ are connected via some ε-path in a run of $\mathcal{P}$.) So we can define a strategy for p_n on $L^*(t)$ by $\sigma_n(u) = b$ for this particular b. (On $\Sigma_{n-1}^* \setminus L^*(t)$ we can define σ_n arbitrarily.) Now let $\alpha \in L^\omega(t) \times (\Sigma_{n,e})^\omega$ be any local run of p_n that is consistent with σ_n and consider the set of nodes $x \in T$ with $\mathrm{Pr}_{\Sigma_{n-1}^*}(\rho(x)) \sqsubseteq \mathrm{Pr}_{\Sigma_{n-1}}(\alpha)$. It is easy to see that all these nodes form an infinite path π through ρ and since α is consistent with σ_n, the definition of σ_n yields $\mathrm{Pr}_{\Sigma_{n,e}}(\pi) = \mathrm{Pr}_{\Sigma_{n,e}}(\alpha)$. Therefore, $\mathrm{Pr}_{\Sigma_{n-1}\times\Sigma_{n,e}}(\pi) = \alpha$. Moreover, since $\mathrm{Pr}_{\Sigma_{n-1}}(\alpha) \in L^\omega(t)$, any node x on π satisfies $t(\mathrm{Pr}_{\Sigma_{n-1}^*}(\rho(x))) = \top$. So, by definition of $\mathcal{A}_n$, the sequence $\kappa = \mathrm{Pr}_{Q^{\mathcal{P}}\times(\Gamma_\bot)^*}(\pi)$ is a run of $\mathcal{P}$ on $\mathrm{Pr}_{\Sigma_{n-1}\times\Sigma_{n,e}}(\pi) = \alpha$ and since ρ is accepting, so is κ. Hence, $\alpha \in L(\mathcal{S}) = L_n$ which shows that σ_n is locally winning on $L^\omega(t)$.

Now let $\sigma_n : \Sigma_{n-1}^* \to \Sigma_{n,e}$ be a strategy for p_n that is locally winning on $L^\omega(t)$. We construct a run ρ of $\mathcal{A}_n$ on t inductively. We start with $\rho(\varepsilon) = (\varepsilon, q_{\mathrm{in}}, \bot)$. Now let $x \in \mathbb{N}^*$ be any node of the run that has already been constructed and let $\rho(x) = (u, q, \gamma A)$. We construct successors $x \cdot j$ of x and we denote $\rho(x \cdot j) = (u_j, q_j, \gamma_j)$. We distinguish the same cases as in the definition of the transition function δ. If $t(u) = \bot$ or $q = q_{\mathrm{acc}}$ then the successors of x are $x \cdot j$ for $j = 0, \ldots, r-1$ and we define $u_j = u \cdot a_j$, $q_j = q_{\mathrm{acc}}$ and $\gamma_j = \gamma A$ for all j. Now assume that $t(u) = \top$ and $q = (q^{\mathcal{P}}, b) \in Q^{\mathcal{P}} \times \Sigma_{n,e}$. If $\delta^{\mathcal{P}}(q^{\mathcal{P}}, A, \varepsilon) \neq \emptyset$ then, since $\mathcal{P}$ is deterministic, there is exactly one $(p^{\mathcal{P}}, \gamma') \in \delta^{\mathcal{P}}(q^{\mathcal{P}}, A, \varepsilon)$ and the only successor of x is $x \cdot 0$ with $u_0 = u$, $q_0 = (p^{\mathcal{P}}, b)$ and $\gamma_0 = \gamma\gamma'$. Finally, if $\delta^{\mathcal{P}}(q^{\mathcal{P}}, A, \varepsilon) = \emptyset$ then the successors of x are $x \cdot j$ for $j = 0, \ldots, r-1$ and we define $\rho(x)$ as follows. First, $u_j = u \cdot a_j$ for all j. Moreover, let $b' = \sigma_n(u)$ and let $j \in \{0, \ldots, r-1\}$. Then there is exactly one $(p_j^{\mathcal{P}}, \gamma_j') \in \delta^{\mathcal{P}}(q^{\mathcal{P}}, A, (a, b'))$ and we define $q_j = (p_j^{\mathcal{P}}, b')$ and $\gamma_j = \gamma\gamma_j'$. By definition of δ it is clear that ρ is, indeed, a run of $\mathcal{A}_n$ on t. Now similar as before, from a path π through ρ we obtain a local run $\alpha \in (\Sigma_{n-1} \times \Sigma_{n,e})^\omega$ of p_n. Moreover, if $t(\mathrm{Pr}_{\Sigma_{n-1}^*}(\rho(x))) = \top$ for all nodes x on π, then $\mathrm{Pr}_{\Sigma_{n-1}}(\alpha) = \mathrm{Pr}_{\Sigma_{n-1}}(\pi) \in L^\omega(t)$ and, by construction of ρ, α is consistent with σ_n. Hence, $\alpha \in L_n = L(\mathcal{S})$ which demonstrates that $\pi \in \mathrm{acc}$. Since all other paths through ρ are accepting by definition, ρ is accepting. $\square$

So, in total, we have that a tree $t \in \mathbb{T}_c(\Sigma_{n-1})$ is in $L(\mathcal{A}_{n-1}) \cap L(\mathcal{A}_n)$ if, and only if, there are local strategies $\sigma_1, \ldots, \sigma_n$ such that $\sigma_1, \ldots, \sigma_{n-1}$ satisfy (1) and (2) and σ_n is locally winning on $L_{\mathrm{out}}^{n-1} \times \Sigma_{0,n}^\omega$. As we have seen in Section 3.1.3, we can effectively construct a nondeterministic parity pushdown tree automaton $\mathcal{A}$ recognizing $L(\mathcal{A}_{n-1}) \cap L(\mathcal{A}_n)$. Moreover, by Theorem 3.9, the emptiness problem for nondeterministic parity pushdown tree automata is decidable, which gives us the desired result.

Theorem 5.7. *The controller problem for pipelines with backward channels is decidable for locally decomposable regular specifications $L = (L_1, \ldots, L_n)$ where $L_1, \ldots, L_{n-1}$ are regular and L_n is deterministic context-free.*

Two-Flanked Pipelines. The remaining case for pipelines with backward channels was rather easy to solve, given the construction from [137]: We just had to modify the last step in order to deal with a deterministic context-free specification instead of an additional input channel from the environment. The case that is left for two-flanked pipelines with backward channels is that of two mutually connected controllers, each of which has its own input channel from the environment. We denote this system as

$$\mathcal{D}_{\mathrm{wed}} = (\{p_0, p_1, p_2\}, \mathfrak{A}_{\mathrm{wed}})$$

[8]Where by $\mathrm{Pr}_{\Sigma_{n,e}}(\rho(x))$ we actually mean $\mathrm{Pr}_{\Sigma_{n,e}}(\mathrm{Pr}_Q(\rho(x)))$, of course.

with $\mathfrak{A}_{\text{wed}} = (\{c_{01}, c_{02}, c_{12}, c_{21}\}, r, w)$ where $w(c_{ij}) = i$ and $r(c_{ij}) = j$. For better readability we denote $\Gamma_i = \Sigma_{c_{0i}}$ and $\Sigma_i = \Sigma_{c_{ij}}$ for $i \neq 0$. We omit the external output channels of p_1 and p_2 just for convenience. It is straightforward how to account for these channels in the proof given below, if they are present.

Due to the mutual dependency of the inputs and outputs of the two controllers and the fact that they have incomparable information, this case is more involved. In particular, the solution from [137] cannot be readily applied because it deals only with one-way dependency of information flow. To deal with mutual dependency of information flow, we extend the concept of communication trees in order to represent ω-languages over $\Sigma_1 \times \Sigma_2$ in such a way that access to the components is maintained.

A $\mathbb{B}^2$-labeled $\Sigma_1 \times \Sigma_2$-tree t represents the language $L^\omega(t) \subseteq (\Sigma_1 \times \Sigma_2)^\omega$ with $\alpha \in L^\omega(t)$ if, and only if, for each finite prefix w of α we have $t(w) = (\top, \top)$. However, more information is stored in such a tree: the components $\text{Pr}_{\Sigma_i}(\alpha)$ may depend on each other which is expressed in the individual components of the $\mathbb{B}$-tuples. If $t(u^\frown v) = (\top, \bot)$, then this tells us that v^{-1} may be answered by u but u^{-1} may not be answered by v, analogous for $t(u^\frown v) = (\bot, \top)$. Clearly this is different from just saying that $u^\frown v$ will not occur.

Of course, any such tree t which in fact represents a joint output language of p_1 and p_2, has properties analogous to the properties (C1) - (C3) of the communication trees $\mathbb{T}_c(\Sigma)$. However, as we don't need those properties in the decidability proof, we do not require them explicitly.

Theorem 5.8. *The controller problem for $\mathcal{D}_{wed}$ is decidable for locally decomposable regular specifications.*

Proof. We construct *two* alternating parity tree automata $\mathcal{A}_1$ and $\mathcal{A}_2$ over $\mathbb{B}^2$-labeled Σ-trees which, roughly, work as follows: When running on a tree t, at each step, $\mathcal{A}_1$ *nondeterministically* guesses some output signal $b \in \Sigma_1$ which is the action chosen by the intended strategy σ_1 for p_1 at this point. Then, $\mathcal{A}_1$ *universally* branches over all possible input signal $(x, y) \in \Gamma_1 \times \Sigma_2$ and proceeds to copies $(\downarrow_{(b,y)}, x)$, keeping the corresponding Γ_1-signal in mind. If $\mathcal{A}_1$ encounters a $\bot$ in the Σ_1-component of the next node then this means that output b should not be chosen in this situation according to t, so $\mathcal{A}_1$ rejects immediately. If, on the other hand, $\mathcal{A}_1$ encounters a $\bot$ in the Σ_2-component then this means that input y will not occur in this situation according to t, so $\mathcal{A}_1$ goes into a special accepting state.

In this way, $\mathcal{A}_1$ guesses a strategy for p_1, on all inputs from Γ_1 and those inputs from Σ_2 that it may receive according to t, and which produces outputs according to t on c_{12}. Moreover, on all paths that are consistent with σ_1 and $L^\omega(t)$, $\mathcal{A}_1$ simulates a deterministic parity automaton, recognizing L_1. The automaton $\mathcal{A}_2$ works analogously. We will see that $L(\mathcal{A}_1) \cap L(\mathcal{A}_2) \neq \emptyset$ if, and only if, there is a joint winning strategy for p_1 and p_2.

To define $\mathcal{A}_1$ and $\mathcal{A}_2$ formally, let $\mathcal{S}_i = (\Sigma^{p_i}, Q^{\mathcal{S}_i}, q_{\text{in}}^{\mathcal{S}_i}, \delta^{\mathcal{S}_i}, \text{col}^{\mathcal{S}_i})$ be deterministic parity automata with $L(\mathcal{S}_i) = L_{p_i}$ for $i = 1, 2$. We construct the alternating parity tree automata $\mathcal{A}_i = (\mathbb{B}^2, Q^i, q_{\text{in}}^i, \delta^i, \text{col}^i)$ $i = 1, 2$ over $\mathbb{B}^2$-labeled $\Sigma_1 \times \Sigma_2$-trees as follows. We only define $\mathcal{A}_1 = (\mathbb{B}^2, Q, q_{\text{in}}, \delta, \text{col})$ using $\mathcal{S}_1 = (\Sigma_{p_1}, Q^{\mathcal{S}}, q_{\text{in}}^{\mathcal{S}}, \delta^{\mathcal{S}}, \text{col}^{\mathcal{S}})$, $\mathcal{A}_2$ is defined completely analogously.

- $Q = (Q^{\mathcal{S}} \uplus \{q_{\text{acc}}, q_{\text{rej}}\}) \times (\Gamma_1 \cup \{\diamond\})$ and $q_0 = (q_{\text{in}}^{\mathcal{S}}, \diamond)$
- $\text{col}(q, \tilde{a}_1) = \text{col}^{\mathcal{S}}(q)$, $\text{col}(q_{\text{acc}}, \tilde{a}_1) = 0$ and $\text{col}(q_{\text{rej}}, \tilde{a}_1) = 1$
- $\delta((q, \tilde{a}_1), (\bot, \zeta)) = \bigvee_{[b_1 \in \Sigma_1]} \bigwedge_{[(a_1,b_2) \in \Gamma_1 \times \Sigma_2]} (\downarrow_{(b_1,b_2)}, (q_{\text{rej}}, a_1))$
- $\delta((q_{\text{rej}}, \tilde{a}_1), (\zeta_1, \zeta_2)) = \bigvee_{[b_1 \in \Sigma_1]} \bigwedge_{[(a_1,b_2) \in \Gamma_1 \times \Sigma_2]} (\downarrow_{(b_1,b_2)}, (q_{\text{rej}}, a_1))$
- $\delta((q, \tilde{a}_1), (\top, \bot)) = \bigvee_{[b_1 \in \Sigma_1]} \bigwedge_{[(a_1,b_2) \in \Gamma_1 \times \Sigma_2]} (\downarrow_{(b_1,b_2)}, (q_{\text{acc}}, a_1))$
- $\delta((q_{\text{acc}}, \tilde{a}_1), (\zeta_1, \zeta_2)) = \bigvee_{[b_1 \in \Sigma_1]} \bigwedge_{[(a_1,b_2) \in \Gamma_1 \times \Sigma_2]} (\downarrow_{(b_1,b_2)}, (q_{\text{acc}}, a_1))$

Notice that any of the four cases for which we have defined δ so far is one of the special cases where $\mathcal{A}_1$ either immediately reject or goes into some special accepting state which is then propagated to the whole subtree. In particular, the nondeterministic and universal branching of δ in these cases is just implemented to make a run of $\mathcal{A}_1$ actually look like a strategy for p_1. The most interesting case in the definition of δ is the one where both labels are $\top$:

$$\delta((q,\tilde{a}_1),(\top,\top)) = \bigvee_{[b_1\in\Sigma_1]} \bigwedge_{[(a_1,b_2)\in\Gamma_1\times\Sigma_2]} (\downarrow_{(b_1,b_2)}, (\delta^{\mathcal{S}}(q,(a_1,b_2,b_1)),a_1))$$

Now we claim that $L(\mathcal{A}_1)\cap L(\mathcal{A}_2)\neq\emptyset$ if, and only if, there is a joint winning strategy $\sigma=(\sigma_1,\sigma_2)$ for p_1 and p_2. To prove this, first let $t\in L(\mathcal{A}_1)\cap L(\mathcal{A}_2)$. Moreover, for $i=1,2$, let $\rho_i : T^i\to(\Sigma_1\times\Sigma_2)^*\times Q^i$ with $T^i\subseteq\mathbb{N}^*$ be a run of $\mathcal{A}_i$ on t. We define the strategy σ_1 according to ρ_1 as follows. Using the definition of $\mathcal{A}_1$ it is easy to see that, for any $u_1{}^\frown v_2\in(\Gamma_1\times\Sigma_2)^*$, there is exactly one node x in ρ_1 such that the unique path from the root of ρ_1 to x corresponds to $u_1{}^\frown v_2$ (that means, is labeled with $u_1{}^\frown v_2$ in the $\Gamma_1\times\Sigma_2$-components). Moreover, there is some $b_1\in\Sigma_1$ such that any successor of x in ρ_1 is labeled with b_1 in the Σ_1-component. Now we define $\sigma_1(u_1{}^\frown v_2)=b_1$ for this particular b_1. The strategy σ_2 is defined completely analogously, using the run ρ_2.

Now let $\alpha=\alpha_1{}^\frown\alpha_2{}^\frown\beta_1{}^\frown\beta_2\in(\Gamma_1\times\Gamma_2\times\Sigma_1\times\Sigma_2)^\omega$ be a global system run which is consistent with $\sigma=(\sigma_1,\sigma_2)$. First, as above, there is exactly one path π_1 in ρ_1 which corresponds to $\alpha_1{}^\frown\beta_2$ and there is exactly one path π_2 in ρ_2 which corresponds to $\alpha_2{}^\frown\beta_1$. Since α is consistent with σ, $\gamma_1:=\alpha_1{}^\frown\beta_2{}^\frown\beta_1$ is consistent with σ_1 and $\gamma_2:=\alpha_2{}^\frown\beta_1{}^\frown\beta_2$ is consistent with σ_2. Therefore, by definition of σ_1 and σ_2, the path π_1 is labeled with β_1 in the Σ_1-component and the path π_2 is labeled with β_2 in the Σ_2-component. So the path which corresponds to π_1 in the tree t is $\pi=\beta_1{}^\frown\beta_2$ and the path which corresponds to π_2 in the tree t is π as well. As both, ρ_1 and ρ_2, are accepting, each node on π is labeled with $(\top,\top)$. Now by construction of $\mathcal{A}_1$, this yields, that only states from $Q^{\mathcal{S}_1}$ occur in the Q^1-component of π_1 and the infinite sequence $\rho_{\mathcal{S}}=(q_j)_{j\in\mathbb{N}}\in(Q^{\mathcal{S}_1})^\omega$ of states of $\mathcal{S}_1$ constitutes a run of $\mathcal{S}_1$ on the ω-word from $(\Sigma^{p_1})^\omega$ which is obtained from the corresponding components of the labels of π_1. Now this ω-word is precisely γ_1 and since ρ_1 is accepting, so is $\rho_{\mathcal{S}}$. Hence, $\gamma_1\in L_1$ and in the same way we obtain $\gamma_2\in L_2$.

Now let conversely $\sigma=(\sigma_1,\sigma_2)$ be a joint strategy for p_1 and p_2. We define the tree $t:(\Sigma_1\times\Sigma_2)\to\mathbb{B}^2$ as follows. For $v_1{}^\frown v_2\in(\Sigma_1\times\Sigma_2)^*$ let $t(v_1{}^\frown v_2)=(\top,\zeta)$ if there is some $u_1\in\Gamma_1^{|v_1|}$ such that $u_1{}^\frown v_2{}^\frown v_1$ is consistent with σ_1. Analogously, let $t(v_1{}^\frown v_2)=(\zeta,\top)$ if there is some $u_2\in\Gamma_2^{|v_2|}$ such that $u_2{}^\frown v_1{}^\frown v_2$ is consistent with σ_2. Now we define the run $\rho_1:T^1\to(\Sigma_1\times\Sigma_2)^*\times Q^1$ inductively such that, for all $x\in T^1$, the unique path π from the root of ρ_1 to x is labeled with states $q\neq q_{\text{rej}}$ in the Q^1-components. Moreover, $u_1{}^\frown v_2{}^\frown v_1\in(\Gamma_1\times\Sigma_2\times\Sigma_1)^*$ is consistent with σ_1, where $(v_1,v_2)=\mathrm{Pr}_{(\Sigma_1\times\Sigma_2)^*}(\rho_1(x))$ and $u_1\in\Gamma_1^{|v_1|}$ is the labeling of π in the Γ_1-components (of the Q^1-components). The run ρ_2 is constructed completely analogously.

First, $\rho_1(\varepsilon)=(\varepsilon,q_0^1)$. Now let $x\in T^1$ be any node of the run that has already been constructed, let $\rho_1(x)=(v_1{}^\frown v_2,(q,\tilde{a}_1))$ and let $t(v_1{}^\frown v_2)=(\zeta_1,\zeta_2)$. Moreover, let $\sigma_1(u_1{}^\frown v_2)=b_1\in\Sigma_1$, where $u_1\in\Gamma_1^{|x|}$ is the Γ_1-labeling of the unique path π from the root of ρ_1 to x. By definition of $\mathcal{A}_1$, $\delta(((q,\tilde{a}_1),(\zeta_1,\zeta_2))$ contains a conjunct $\bigwedge_{[(a_1,b_2)\in\Gamma_1\times\Sigma_2]}(\downarrow_{(b_1,b_2)},(q^{(a_1,b_2)},b_1))$ and we define the successors of x as $x\cdot j$, $j=0,\dots,r-1$, where $\Gamma_1\times\Sigma_2=\{b^0,\dots,b^{r-1}\}$. Moreover, for $j\in\{0,\dots,r-1\}$, $\rho_1(x\cdot j)=((v_1\cdot b_1)^\frown(v_2\cdot b_2),(q^{b_1,b_2},a_1))$ where $(b_1,b_2)=b^j$.

Now we prove that any node $x\cdot j$ fulfills the conditions formulated above. So let $j\in\{0,\dots,r-1\}$ and $b^j=(b_1,b_2)$. As $u_1{}^\frown v_2{}^\frown v_1$ is consistent with σ_1, by definition of b_1, $(u_1\cdot a_1)^\frown(v_2\cdot b_2)^\frown(v_1\cdot b_1)$ is also consistent with σ_1. For the proof that $q^{(b_1,b_2)}\neq q_{reject}$, first notice that $q\neq q_{reject}$. Therefore, if $\zeta_2=\bot$ or $q=q_{\text{acc}}$ we have $q^{(b_1,b_2)}=q_{\text{acc}}$. Moreover, since $u_1{}^\frown v_2{}^\frown v_1$ is consistent with σ_1, by definition of t we have $t(v_1{}^\frown v_2)=(\top,\zeta)$ so $\zeta_1\neq\bot$. So assume that $q\in Q^{\mathcal{S}_1}$ and $\zeta_2=\top$. Then $q^{(b_1,b_2)}=\delta^{\mathcal{S}_1}(q,(a_1,b_2,b_1))\in Q^{\mathcal{S}_1}$.

Clearly, ρ_1 is a run of $\mathcal{A}_1$ on t. To prove that ρ_1 is accepting, let π_1 be some infinite path through ρ_1 and let $\beta=\alpha_1{}^\frown\beta_2{}^\frown\beta_1\in(\Gamma_1\times\Sigma_2\times\Sigma_1)^\omega$ be the Σ^{p_1}-labeling of π_1. By construction of ρ_1, $\alpha_1(<k)^\frown\beta_2(<k)^\frown\beta_1(<k)$ is consistent with σ_1 for all $k\in\mathbb{N}$, so the local run β of p_1 is

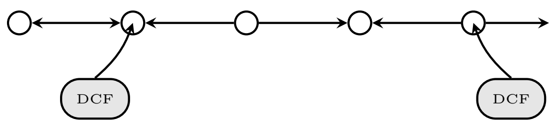


Figure 5.1: Architecture with two connected local DCF specifications

consistent with σ_1. Moreover, $\mathrm{Pr}_{Q^1}(\pi_1(<k)) = (q^k, \tilde{a}_1^k) \in (Q^{\mathcal{S}_1} \cup \{q_{\mathrm{acc}}\}) \times (\Gamma_1 \cup \{\diamond\})$ for all $k \in \mathbb{N}$ and, if $q^k = q_{\mathrm{acc}}$ for some k then π_1 is accepting. So assume that $q^k \in Q^{\mathcal{S}_1}$ for all k.

Then $t(\beta_1(<k)^\frown\beta_2(<k)) = (\top, \top)$ for all $k \in \mathbb{N}$ so, by construction of t, for all k, there is some $u_2 \in \Gamma_2^k$ such that $u_2{}^\frown\beta_1(<k)^\frown\beta_2(<k)$ is consistent with σ_2. Clearly, the set of all $u_2 \in \Gamma_2^*$ such that $u_2{}^\frown\beta_1(<|u_2|)^\frown\beta_2(<|u_2|)$ is consistent with σ_2 is prefix-closed, so by König's Lemma, there is some $\alpha_2 \in \Gamma_2^\omega$ such that $\alpha_2{}^\frown\beta_1{}^\frown\beta_2$ is consistent with σ_2. Since $\alpha_1{}^\frown\beta_2{}^\frown\beta_1$ is consistent with σ_1, it follows that $\alpha = \alpha_1{}^\frown\alpha_2{}^\frown\beta_1{}^\frown\beta_2$ is consistent with σ. Since σ is winning, $\alpha_1{}^\frown\beta_2{}^\frown\beta_1 \in L_1$, so the unique run of $\mathcal{S}_1$ on $\alpha_1{}^\frown\beta_2{}^\frown\beta_1$ is accepting. As, by the definition of $\mathcal{A}_1$, this run coincides with the $Q^{\mathcal{S}_1}$-labeling of π_1, the path π_1 is accepting. Therefore, ρ_1 is accepting. Completely analogously, one shows that ρ_2 is an accepting run of $\mathcal{A}_2$ on t. Hence, $t \in L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$. □

5.2 Undecidability

Now that we have all the additional basic decidable cases at hand, we have to prove matching undecidability results. In particular, for deterministic context-free specifications, we have shown decidability only for a rather particular case, so there have to be far-reaching undecidability results, and we structure our exposition primarily along the number of processes which (are allowed to) have a deterministic context-free local specification.

Two DCF specifications. First we demonstrate how two connected processes, each of which has a deterministic realtime 1-counter specification, can directly simulate a 2-register machine. As the non-halting problem for such machines is undecidable, any architecture which contains two such processes is undecidable. Even more, the controller problem for such architectures is not even recursively enumerable.

Theorem 5.9. *The controller problem for a connected architecture with at least two local DR1-C specifications is not recursively enumerable.*

Proof. We proceed by a reduction from the non-halting problem for 2-register machines. For those machines we use the notation from Section 3.4: Let $\mathcal{R}$ be a 2-register machine consisting of a sequence $I_0, \dots, I_{k-1}, I_k = \mathtt{stop}$ of instructions $I_j \in \{\mathtt{inc}(R_i), \mathtt{dec}(R_i), \mathtt{if}\ \ R_i = 0\ \ \mathtt{goto}\ \ l \,|\, i \in \{1,2\}, l \in [k]\}$.

Let $p_{i_0}, \dots, p_{i_m}$ be processes such that $p = p_{i_0}$ and $p' = p_{i_m}$ are connected and have DR1-C specifications, cf. Figure 5.1. Moreover, let $C_m = \{c_i | i \in [m]\}$ be a set of channels such that, $c_i \in C_{p_i}$ and $r(c_i) = p_{i+1}$ or vice versa. That is, C_m induces a (not necessarily directed) path between p and p'. We define $\Sigma_c = [k]$ for all $c \in C_m$ and we silence all other channels $c \notin C_m$ by defining $\Sigma_c = \{\sharp\}$.

The idea is now as follows. The processes p and p' both write a sequence $j_0, j_1, \dots$ of instruction numbers of $\mathcal{R}$ which is supposed to induce the (unique) run of $\mathcal{R}$ when started on empty registers. However, since both processes have a deterministic *one*-counter specification, only one of the registers can be actually checked for each of the sequences which is not sufficient to guarantee that it actually corresponds to the run of $\mathcal{R}$. To overcome this problem, we *synchronize* the processes by requiring that, in each step, the same number j is written to all channels from C_m. With the two sequences of instruction numbers equalized in this way, it is then sufficient that each of the two deterministic 1-counter specifications is responsible for only one of the two registers: The pushdown automaton $\mathcal{P}_1$ recognizing L_p simulates register R_1 and uses states q_j for $j \in [k]$, q_{nomb} and q_{rej}. Whenever an instruction number j is written such that I_j deals with register R_1 then $\mathcal{P}_1$ manipulates its counter accordingly, determines the next instruction number j and goes to q_j. When $\mathcal{P}_1$ is in state q_j and receives letter $l \in [k]$ then it checks whether $l = j$, that is, whether the instruction number that is written to c_0 is correct. If not, $\mathcal{P}_1$ goes to q_{rej}. If, on the other hand, I_j deals with register R_2, then $\mathcal{P}_1$ goes to state q_{nomb}, indicating that it is not responsible for the next instruction. (Or to state q_{rej}, if it was previously in state q_j and $l \neq j$.) If, at some point, $I_j = \texttt{stop}$ then $\mathcal{P}_1$ goes to q_{rej}.

Technically, for any process p_i with $0 < i < m$ we define

$$L_{p_i} = \{\alpha \in \Sigma_{\text{in}}^{p_i} \times \Sigma_{\text{out}}^{p_i} | \Pr_{\Sigma_{c_i}}(\alpha) = \Pr_{\Sigma_{c_{i+1}}}(\alpha)\}.$$

So, if the processes have a joint winning strategy, the information sent via c_0 and c_m has to be identical in each step. The specifications L_p and $L_{p'}$ are given by deterministic realtime 1-counter automata $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively.

We define $\mathcal{P}_1 = (\Sigma, \{A\}, Q, q_{\text{in}}, \delta, \text{col})$ as follows, where we consider only the component from Σ_{c_0}. $Q = \{q_j \mid j \in [k]\} \cup \{q_{\text{nomb}}, q_{\text{rej}}\}$, $q_{\text{in}} = q_0$, $\Sigma = \Sigma_{c_0} = [k]$ and $\text{col}(q) = 0$ for all $q \in Q \setminus \{q_{\text{rej}}\}$ and $\text{col}(q_{\text{rej}}) = 1$. Moreover, δ is defined as follows:

- if $I_j \in \{\texttt{inc}(R_2), \texttt{dec}(R_2)\}$, $\delta(q_j, j, Z) = \delta(q_{\text{nomb}}, j, Z) = (q_{\text{nomb}}, Z)$
- if $I_j = \texttt{if}\ R_2 = 0\ \texttt{goto}\ l$, $\delta(q_j, j, Z) = \delta(q_{\text{nomb}}, j, Z) = (q_{\text{nomb}}, Z)$
- if $I_j = \texttt{inc}(R_1)$, $\delta(q_j, j, Z) = \delta(q_{\text{nomb}}, j, Z) = (q_{j+1}, AZ)$
- if $I_j = \texttt{dec}(R_1)$, $\delta(q_j, j, Z) = \delta(q_{\text{nomb}}, j, Z) = (q_{j+1}, \varepsilon)$
- if $I_j = \texttt{if}\ R_1 = 0\ \texttt{goto}\ l$,
  $\delta(q_j, j, \bot) = \delta(q_{\text{nomb}}, j, \bot) = (q_l, \bot)$ and
  $\delta(q_j, j, A) = \delta(q_{\text{nomb}}, j, A) = (q_{j+1}, A)$
- if $I_j = \texttt{stop}$, $\delta(q_j, j, Z) = \delta(q_{\text{nomb}}, j, Z) = (q_{\text{rej}}, Z)$
- $\delta(q_j, l, Z) = (q_{\text{rej}}, Z)$ if $l \neq j$
- $\delta(q_{\text{rej}}, j, Z) = (q_{\text{rej}}, Z)$

The automaton $\mathcal{P}_2$ is defined in a completely analogous way with registers R_1 and R_2 swapped. It is easy to see that a sequence $\gamma \in Q^\omega$ is in $L(\mathcal{P}_0) \cap L(\mathcal{P}_1)$ if, and only if, the corresponding sequence of instructions is an *infinite* run of $\mathcal{R}$ with initially empty registers. Since any such sequence can be generated by a strategy $\sigma_{p_{i_j}}$ for process p_{i_j} (for each $j \in \{0, \ldots, m\}$), it follows that there *exists* a joint winning strategy for $p = p_{i_0}, p_{i_1}, \ldots, p_{i_m} = p'$ if, and only if, the (unique) run of $\mathcal{R}$ on initially empty registers is infinite. Notice that such a strategy needs as input only ticks | so in particular, none of the processes p_{i_l} needs to be reachable from the environment.[9] $\square$

One DCF specification. The previous result shows that we can allow at most one deterministic context-free specification. We have already seen that in this case, the controller problem is decidable if the architecture is a pipeline and the controller p with a deterministic context-free specification is the *last* one in the pipeline. In particular, any other controller in the system is *better* informed than p. The next result shows that, indeed, as soon as we have some controller p' that is *not* better informed than p, then the controller problem is again undecidable (not even recursively enumerable), already for deterministic realtime 1-counter specifications, cf. Figure 5.2. To show this, essentially, we use p and p' to simulate the proof of Theorem 3.20: Process p' has the task to construct a run of the given 2-register machine while the local specification of p is

[9]On the other hand, to actually *implement* a joint winning strategy for $p_{i_0}, \ldots, p_{i_m}$ (if it exists), in general, any of the processes p_{i_j} needs a *two*-counter memory structure!

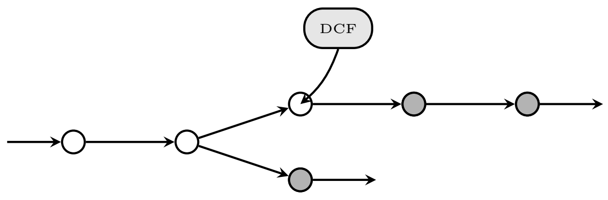


Figure 5.2: Undecidable architecture with one deterministic context-free specification

used to check the construction. Recall that we can check only one register at a time, so that p' must not notice when the construction is checked, cf. Section 3.4. This can be accomplished since p' is not better informed than p.

Theorem 5.10. *Let $\mathfrak{A}$ be an architecture with two connected controllers $p \neq p'$ such that p is reachable and has a DR1-C specification and p' is not better informed than p. Then the controller problem for $\mathfrak{A}$ is not recursively enumerable.*

Proof. Again, we proceed by a reduction from the non-halting problem for 2-register machines. For this, let $\mathcal{R}$ be a 2-register machine as in the proof of Theorem 5.9. First, we show the result for the special case where $\mathfrak{A}$ consists only of the two controllers p, p' and where p_{env} sends information to p via some channel $c_0 \in C_{p_{env}}$ and p and p' communicate via some channel c_1. Moreover, p' has an output channel $c_2 \in O_{p'}$ (which may be c_1 if $c_1 \in O_p$ already).

We define

- $\Sigma_{c_0} = \{0, 1, 2\}$ and
- $\Sigma_{c_1} = \Sigma_{c_2} = [k] \cup \{A_1, A_2\}$
- $L_{p'} = \{\alpha \in (\Sigma^{p'})^\omega | \mathrm{Pr}_{\Sigma_{c_1}}(\alpha) = \mathrm{Pr}_{\Sigma_{c_2}}(\alpha)\}$
- $L_p = \{\alpha \in (\Sigma^p)^\omega | \mathrm{Pr}_{(\Sigma_{c_0} \times \Sigma_{c_1})}(\alpha) \in \tilde{L}_p\}$

where $\tilde{L}_p \subseteq \Sigma_{c_0} \times \Sigma_{c_1}$ is the deterministic realtime 1-counter language from the proof of Theorem 3.20, that is, $\alpha_0 {}^\frown \alpha_1 \in \tilde{L}_p$ if the following conditions hold:

(1) α_1 has the form $C_0 C_1 \ldots$ where $C_j \in [k] \cdot \{A_1\}^* \cdot \{A_2\}^*$ for each j

(2) $C_0 = 0\varepsilon\varepsilon$

(3) if there exists a $r \in \mathbb{N}$ with $\alpha_0(r) \in \{1, 2\}$ then consider the smallest such r and the smallest $s \geq r$ with $\alpha_1(s) \in [k]$, and let $\alpha_1 = \alpha_1({<}s) C C' \gamma$.
Then $C \overset{\alpha_0(r)}{\vdash} C'$.

So the local specification $L_{p'}$ requires merely that $\mathrm{Pr}_{\Sigma_{c_1}} = \mathrm{Pr}_{\Sigma_{c_2}}$, that is, the same symbol has to be written to both channels in each step. If $c_1 = c_2$ this is of course trivial but if c_1 is a channel from p to p' then this means, that p and p' have to choose the same action in each step simultaneously. In particular, p' will not be able to infer anything from the input that it receives from p.

Now, if $\mathcal{R}$ does not halt if started with initially empty registers then p and p' have the following joint winning strategy: They both write the unique run of $\mathcal{R}$ when started on empty registers to the channels c_1 and c_2 simultaneously. If, on the other hand, $\mathcal{R}$ halts if started with empty registers, then let σ be any joint strategy of p and p'. To see how the environment can spoil σ, let $\alpha_1 {}^\frown \alpha_2 \in (\Sigma_{c_1} \times \Sigma_{c_2})^\omega$ be the word produced by σ when p receives input $\alpha_0 = 0^\omega$ from the environment. If $\alpha_0 {}^\frown \alpha_1 \notin \tilde{L}_p$ or $\alpha_1 \neq \alpha_2$ then σ is not a winning strategy, so assume that $\alpha_0 {}^\frown \alpha_1 \in \tilde{L}_p$ and $\alpha_1 = \alpha_2$. In particular, α_1 satisfies conditions (1) and (2), so α_1 consists of a sequence $C_0 C_1 C_2 \ldots$ of configurations of $\mathcal{R}$ starting with the initial configuration $C_0 = 0\varepsilon\varepsilon$. However, since $\mathcal{R}$ halts if started with empty registers, α_1 cannot be a run of $\mathcal{R}$.

Let j be minimal, such that $C_j \not\vdash C_{j+1}$. Then we have $C_j \overset{1}{\not\vdash} C_{j+1}$ or $C_j \overset{2}{\not\vdash} C_{j+1}$. In the first case, set $\mathbf{i} = 1$ and in the second case set $\mathbf{i} = 2$. Now let $\overline{\alpha}_1 {}^\frown \overline{\alpha}_2 \in (\Sigma_{c_1} \times \Sigma_{c_2})^\omega$ be the word produced by σ when p receives input $\overline{\alpha}_0 = 0^{|C_0 \cdots C_{j-1}|}\,\mathbf{i}\,0^\omega$ from the environment. Obviously, $\overline{\alpha}_2(0) = \alpha_2(0)$ which implies $\overline{\alpha}_1(0) = \alpha_1(0)$, so $\overline{\alpha}_2(1) = \alpha_2(1)$, thus $\overline{\alpha}_1(1) = \alpha_1(1)$ and so on, hence $\overline{\alpha}_2 = \alpha_2$ and $\overline{\alpha}_1 = \alpha_1$. Since $\overline{\alpha}_0 {}^\frown \overline{\alpha}_1$ is consistent with σ and $\overline{\alpha}_0 {}^\frown \overline{\alpha}_1 = \overline{\alpha}_0 {}^\frown \alpha_1 \notin L$, σ is not a winning strategy.

To extend the proof to the general case, we simulate the communication between p and p' as above using a sequence of channels connecting p and p'. Furthermore, the signal $\mathbf{i}$ from the environment can be transmitted to p via the (directed) path from p_{env} to p. Notice that the signal $\mathbf{i}$ will arrive at p with a delay (which depends only on the given architecture). We have to incorporate this into the specifications, that is, we have to adapt the specifications in order to postpone the starting point of the production of a sequence of configurations by p and p'. Moreover, although the set of channels used for this transmission is, in general, not disjoint from the set of channels connecting p and p', by a simple adaption of the alphabets and specifications, the different information can be sent along the same channels. Finally, since p' is not better informed than p, the signal $\mathbf{i}$ can still be kept away from p'. $\square$

Regular Specifications. Since any architecture that is undecidable for locally decomposable regular specifications is also undecidable for arbitrary regular specifications, any architecture that will be shown to be undecidable here has also been shown undecidable in [80]. In particular, any such architecture contains an information fork.

For locally decomposable specifications, however, proving undecidability may be more involved. In particular, not any information fork can be used to show undecidability as the decidability of two-flanked pipelines for locally decomposable regular specifications demonstrates. In order to be able to connect the inputs and outputs of two incomparably informed processes p and p' in the specification sufficiently, we need at least one additional process p'' that is appropriately connected with p and p'. We distinguish cases according to how the p and p' are connected with p''.

Of course, any acyclic architecture that is not decidable for locally decomposable regular specifications has been shown undecidable in [137]. The reason why *cycles* may make it even harder to prove undecidability is that, although each process has at least one output channel, there may arise cases where two processes that are incomparably informed send all their outputs to each other. So we cannot let these processes work completely independently. On the other hand, for arbitrary regular specifications this problem has also been solved in [80] using encryption: The environment sends encryption function to the processes which they have to use to encode their outputs. Since the processes do not know the encryption function that the other processes uses, they cannot deduce anything from the output of the other process that they get. We will see that for the cases that we have to deal with here, it is easy to simulate this method using locally decomposable specifications.

First we consider $\mathfrak{A}_{\text{prefork}} = (C, r, w)$ with controllers p_1,p_2 and p_3 and with

- $C = \{c_{01}, c_{12}, c_{13}, c_{20}, c_{30}\}$,
- $w(c_{ij}) = p_i$ and $r(c_{ij}) = p_j$.

So here, $p_2 = p$ and $p_3 = p'$ form an information fork and both processes are reachable via a directed path (of length one, in this case) from $p_1 = p''$.

Since $\mathfrak{A}_0$ does not contain cycles, the controller problem for this architecture is undecidable according to [137]. However, we give a different proof of this result, for which it is easier to incorporate encryption functions in order to cover the case where c_2 and c_3 are not necessarily external output channels but may be read by other controllers. Our proof uses an idea from [30], yielding a variant of the proof of the corresponding result from [171] for arbitrary regular specifications. Similarly as in the proof of Theorem 5.10, processes p_2 and p_3 will have the task of writing configurations (of a Turing machine, in this case) while the specification L_1 is used to check the correctness of their constructions.

Theorem 5.11. *The controller problem for $\mathfrak{A}_{prefork}$ for locally decomposable regular specifications is not recursively enumerable.*

Proof. We proceed by a reduction from the non-halting problem for Turing machines, so let

$$M = (\Sigma, Q, q_{\text{in}}, \delta, q_{\text{acc}})$$

be a deterministic Turing machine and set

- $\Sigma_{\text{conf}} = \Sigma \cup Q \cup \{\sharp\}$
- $\Sigma_{\text{sign}} = \{\rightsquigarrow, \curvearrowright\}$.

We use the following labeling for the channels of $\mathfrak{A}_{\text{prefork}}$:

- $\Sigma_{c_{01}} = \Sigma_{\text{sign}} \times \Sigma_{\text{sign}}$
- $\Sigma_{c_{12}} = \Sigma_{c_{13}} = \Sigma_{\text{sign}} \times \Sigma_{\text{conf}}$
- $\Sigma_{c_{20}} = \Sigma_{c_{30}} = \Sigma_{\text{conf}}$

Now we define the specifications. For some local behavior

$$\alpha = (\beta_1{}^\frown\beta_2)^\frown\alpha_{12}{}^\frown\alpha_{13} \in (\Sigma_{01} \times \Sigma_{12} \times \Sigma_{13})^\omega$$

of process p_1, we distinguish two cases: If $\{\lambda \in \mathbb{N} \mid \beta_i(\lambda) = \curvearrowright\} = \emptyset$ then $\alpha \in L_1$. If, on the other hand, $\{\lambda \in \mathbb{N} \mid \beta_i(\lambda) = \curvearrowright\} \neq \emptyset$, then let

$$\text{go}_i(\alpha) = \min\{\lambda \in \mathbb{N} \mid \beta_i(\lambda) = \curvearrowright\}{+}1.$$

and let

$$\alpha \in L_1 \quad :\Longleftrightarrow \quad |\text{go}_1(\alpha) - \text{go}_2(\alpha)| > 1 \quad \textbf{or} \quad (1) \text{ - } (3).$$

(1) $\Pr_{\Sigma_{\text{conf}}}(\alpha_{1i}) = \sharp^{\text{go}_i(\alpha)+1}\,\mathbf{C}\,\sharp^\omega$ such that
   (a) C is a non-terminal configuration of M
   (b) if $\text{go}_i(\alpha) = 1$ then $C = C_{\text{in}}$ is the initial configuration of M when started on the empty word

(2) $\Pr_{\Sigma_{\text{sign}}}(\alpha_{1i}(0))) = \rightsquigarrow$ and $\Pr_{\Sigma_{\text{sign}}}(\alpha_{1i}(\lambda))) = \Pr_{\Sigma_{\text{sign}}}(\beta_i(\lambda - 1))$ for $\lambda > 0$

(3) Let $\Pr_{\Sigma_{\text{conf}}}(\alpha_{1i}) = \sharp^{\text{go}_i(\alpha)+1} C_i \sharp^\omega$.
   (a) if $\text{go}_2(\alpha) = \text{go}_3(\alpha)$ then $C_2 = C_3$
   (b) if $\text{go}_2(\alpha) = \text{go}_3(\alpha){+}1$ then $C_3 \vdash C_2$
   (c) if $\text{go}_2(\alpha) = \text{go}_3(\alpha){-}1$ then $C_2 \vdash C_3$

Moreover, for some local behavior

$$\alpha = \alpha_{1i}{}^\frown\alpha_i \in (\Sigma^{p_i})^\omega$$

of process p_i with $i \in \{2, 3\}$, we define

$$\alpha \in L_i \quad :\Longleftrightarrow \quad \Pr_{\Sigma_{conf}}(\alpha_{1i}) = \Pr_{\Sigma_{conf}}(\alpha_i).$$

Obviously, L_1, L_2 and L_3 are all regular. Now we claim that M does not halt on the empty tape if, and only if, p_1, p_2 and p_3 have a joint winning strategy.

First, let us explain the intended meaning of a system run. Controllers p_2 and p_3 have the task of writing exactly one configuration of M, as soon as they receive symbol $\curvearrowright$ via c_{12} and c_{13}, respectively. This symbol is sent by the environment, thereby determining in which round each of the processes p_2 and p_3 start writing their configuration. However, the respective rounds of p_2 and p_3 may differ by at most one. Notice that these signals from p_{env} must be routed through p_1, so we have a delay there which is accounted for in the specification.

Now, besides propagating the signals from p_{env} to p_2 and p_3, p_1 has the task of synchronizing the Σ_{conf}-components of the channels c_{12} and c_{13} with the channels c_{20} and c_{30}, respectively, so that the local specification of p_1 can be used to relate the configurations C_2 and C_3 that p_2 and p_3 produce. More precisely, if p_2 and p_3 receive $\curvearrowright$ for the first time in the same round then C_2 and C_3 must be equal, and if p_i receives $\curvearrowright$ for the first time one round earlier than p_j then C_i must be the predecessor configuration of C_j. Moreover, if prompted by $\curvearrowright$ right at the beginning, then the processes must produce the initial configuration of M and they may also never produce any terminal configuration.

Now if M does not halt on the empty tape, then these conditions can be easily fulfilled by the controllers: For $i \in \{2, 3\}$, on receiving an input $\beta \in \{\rightsquigarrow\}^k \curvearrowright \{\rightsquigarrow, \curvearrowright\}^\omega$ with $k \geq 1$, controller p_i produces output $\sharp^{k+1}\,\mathbf{C(k)}\,\sharp^\omega$, where $C(k)$ is configuration number k in the run of M when started on the empty tape. Moreover, on receiving $\rightsquigarrow^\omega$, p_i produces $\sharp^\omega$. (Clearly, this function can be implemented by a strategy for p_i.) It is now easy to see that there is a strategy for p_1 such that the corresponding joint strategy of p_1, p_2 and p_3 is, indeed, winning: Forward the Σ_{sign}-components from c_0 to p_i in the next step, respectively, and synchronize the Σ_{conf} components of c_{1i} with c_i by anticipating the actions of p_i.

Now let, conversely, $\sigma = (\sigma_1, \sigma_2, \sigma_3)$ be a joint winning strategy for the controllers. First notice that, due to the specification L_2, the following holds: If σ is any strategy for p_2 that is locally winning on some language $L_{\text{in}} \subseteq (\Sigma_{\text{sign}} \times \Sigma_{\text{conf}})^\omega$ then for any $\beta \in \{\rightsquigarrow\}\Sigma_{\text{sign}}^\omega$, there at most one $\gamma \in \Sigma_{\text{conf}}^\omega$ such that $\beta^\frown\gamma \in L_{\text{in}}$. Since sequences $\beta \in \{\rightsquigarrow\}\Sigma_{\text{sign}}$ correspond to natural numbers $k = \min\{\lambda \in \mathbb{N} \,|\, \beta(\lambda) = \curvearrowright\} \geq 1$ or $k = \omega$, if $\beta(\lambda) = \rightsquigarrow$ for all $\lambda \in \mathbb{N}$, the strategy σ_2 can be viewed as function $\sigma_2 : \mathbb{N}_{\geq 1} \cup \{\omega\} \to \Sigma_{\text{conf}}^\omega$.

Moreover, due to L_2 and condition (2) of L_1, condition (1) of L_1 also applies to any local run of process p_2 which is part of a global system run that is consistent with σ. Therefore, σ_2 is in fact a function

$$\sigma_2 : \mathbb{N}_{\geq 1} \cup \{\omega\} \to \mathbb{N}_{\geq 1} \cup \{\omega\}$$

where $\sigma_2(l) = k$ means that upon receiving $k \in \mathbb{N} \cup \{\omega\}$, σ_2 produces $\sharp^{k+1}\,\mathbf{C(k)}\,\sharp^\omega$ or $\sharp^\omega$, if $k = \omega$. Now, the very same applies to σ_3 as well and, by specification L_1, the strategies must satisfy the following conditions:

- $\sigma_2(1) = \sigma_3(1) = 1$
- $\sigma_2(l + 1) = \sigma_3(l) + 1$
- $\sigma_3(l + 1) = \sigma_2(l) + 1$.

It is easy to see that the only such functions are $\sigma_2 = \sigma_3 = \text{id}_{\mathbb{N}}$. Therefore, the environment can request any configuration in the run of M when started on the empty tape from p_1. However, according to condition (1),(a), L_1 is violated as soon as p_1 writes a terminal configuration. Hence, in the unique run of M when started on the empty tape, there can be no terminal configuration. □

The next step is now to incorporate encryption functions into the construction that we have performed in the proof, in order to adapt the proof to the architecture $\widetilde{\mathfrak{A}}_{\text{prefork}}$ which is obtained from $\mathfrak{A}_{\text{prefork}}$ by re-defining $r(c_{20}) = 3$ and $r(c_{30}) = 2$ (for at least one of the channels c_{20} and c_{30}). The main idea is as follows. We provide the environment with the possibility to send two different encryption functions in each step and we demand of process p_1 that it distributes them to the processes p_2 and p_3. Then, processes p_2 and p_3 have to encode their output using the function which they have received in the last step. The set of functions available to the environment can, for example, be chosen as the set of all permutations on Σ_{conf}. Using these encryption functions, the environment can guarantee that neither p_2 nor p_3 can derive any information from the input that it receives from the other process, respectively.

This idea is taken from [80] where is has been applied to the architecture $\mathfrak{A}_{\text{wed}}$ from Section 5.1.3 for arbitrary regular specifications. It is obvious that the reasoning from [80] (see also [191]) can be directly applied to our scenario as well: The process p_1 just propagates the encryption functions to p_2 and p_3, respectively, and the whole encryption processing takes place at p_2 and p_3, using their respective specifications. So we obtain that the controller problem for $\tilde{\mathfrak{A}}_{\text{prefork}}$ for locally decomposable regular specifications is undecidable.

What remains to do is to extend the undecidability results to more complex architectures which consist of more than three controllers but where we can find patterns that look like $\mathfrak{A}_{\text{prefork}}$ or $\tilde{\mathfrak{A}}_{\text{prefork}}$. This is rather straightforward, similar as in the proof of Theorem 5.10: Clearly, it is not necessary that the environment has a direct channel to the process p_1 but there merely needs to be a directed path from the environment to p_1 which neither contains p_2 nor p_3. So it suffices to assume that p_1 is reachable and p_2 and p_3 are both not better informed than p_1. Moreover, if one or both of the output channels of p_2 and p_3 are neither external output channels nor read by the other process, then the proof also works just as before: For an output channel of p_2 or p_3 that is read by the other controller as in $\tilde{\mathfrak{A}}_{\text{prefork}}$, we use encryption as described above. Output channels that are read by any other process can be treated as external output channels. Hence, we obtain the following result. Notice that, in particular, p_2 and p_3 form an information fork.

Theorem 5.12. *Let $\mathfrak{A}$ be an architecture with a reachable process p_1 such that p_1 sends information to processes $p_2 \neq p_3$ which are not better informed than p_1. Then the controller problem for $\mathfrak{A}$ for locally decomposable regular specifications is not recursively enumerable.*

Now we consider the case where we have an information fork $\{p, p'\}$ and a process p'' which is reachable from both p and p'. Again, we start with a special case: Consider $\mathfrak{A}_{\text{postfork1}} = (C_1, r_1, w_1)$ with controllers p_1, p_2, p_3 and:

- $C_1 = \{c_{01}, c_{02}, c_{13}, c_{23}\}$
- $w_1(c_{ij}) = p_i$ and $r_1(c_{ij}) = p_j$

In $\mathfrak{A}_{\text{postfork}}$, it is very easy to connect p_1 and p_2: We don't have to synchronize p_1 and p_2 with p_3 like in the proof of Theorem 5.11 but we can directly merge the outputs of p_1 and p_2 at p_3. Therefore, we can simulate the original undecidability proof of [171] (or our proof of Theorem 5.11, accordingly) using the local specification straightforwardly, cf. [137]. Moreover, it is not important which processes read the output channels of p_3: The process is only needed to have a single local specification that can talk about c_{13} and c_{23} at the same time.

The processes p_1 and p_2 also do not need to have direct channels to p_3. It suffices that p_3 is reachable from p_1 via some path $\mathcal{P}_1$ that does not contain p_2, and from p_2 via a path $\mathcal{P}_2$ that does not contain p_1. Then we can propagate the outputs of p_1 and p_2 to p_3 where they are merged. Clearly, there is also no harm done in sending these outputs along the same channels, so that $\mathcal{P}_1$ and $\mathcal{P}_2$ do not have to be disjoint. Of course, these paths may have a different length which creates a delay between the outputs. However, this delay is a fixed number which depends only on the architecture and can be easily accounted for in the specification of p_3. Finally, the proof can also be easily adapted to the case where p_1 and p_2 do not receive direct inputs from the environment, but are merely reachable from the environment via directed paths.

However, if, for example, p_3 is reachable from p_2 via a path $\mathcal{P}_2$ that does not contain p_1 but *any* path from p_1 to p_3 contains p_2 then things become a little more complicated again. For this case, we consider the architecture $\mathfrak{A}_{\text{postfork2}} = (C_2, r_2, w_2)$ with controllers p_1, p_2, p_3 and

- $C_2 = \{c_{01}, c_{02}, c_{12}, c_{23}, c_{30}\}$
- $w_2(c_{ij}) = p_i$ and $r_2(c_{ij}) = p_j$.

According to [137], this architecture is undecidable and it is rather easy to extend this result to more general scenarios as described above for $\mathfrak{A}_{\text{postfork1}}$. However, in this case, it is important again that p_3 does have some output channel so we have to take of the possibility that all output channels of p_3 may be read by other processes (which, in particular, includes the possibility that they are all read by p_1). This can be done similar as described above using encryption and so we obtain the following result.

Theorem 5.13. *Let $\mathfrak{A}$ be an architecture with at least two incomparably informed processes p_1 and p_2 which are both reachable such that there is a process $p_3 \notin \{p_1, p_2\}$ which is reachable from both p_1 and p_2. Then the controller problem for $\mathfrak{A}$ for locally decomposable regular specifications is not recursively enumerable.*

5.3 Characterization

Now we characterize the exact classes of architectures which are decidable for locally decomposable regular specifications and, for each decidable architecture, we determine the exact set of processes which may have a deterministic context-free specification such that decidability still holds. Moreover, we also determine a particular case where we can allow one *nondeterministic* context-free specification.

First, we simplify the setting a little. Consider a distributed system with a locally decomposable specification

$$\mathcal{D} = (\mathfrak{A}, (\Sigma_c)_{c\in C}) \quad \text{with } \mathfrak{A} = (C, r, w) \quad \text{and } L = (L_p)_{p\in P_{\text{con}}}.$$

As we have already mentioned, hidden input channels from the environment do not make any difference if the specification is locally decomposable: None of the individual local specifications L_p can incorporate these channels, so the global system specification is independent of the outputs that the environment sends along its hidden channels. Hence, these channels can be totally neglected.

Moreover, for an *induced subarchitecture* $\mathfrak{A}' = (C', r', w')$ of $\mathfrak{A}$, by $\mathcal{D}'$ we denote the subsystem $(\mathfrak{A}', (\Sigma_c)_{c\in C'})$ and by L' we denote the locally decomposable specification $(L_p)_{p\in P'}$ of $\mathcal{D}'$. Then, the grand coalition has a joint winning strategy for $(\mathcal{D}, L)$ if, and only if, it has a joint winning strategy for every subsystem $(\mathcal{D}', L')$. Clearly, any joint winning strategy for the grand coalition contains a joint winning strategy for each such subsystem. On the other hand, if the grand coalition has a joint winning strategy for every subsystem $(\mathcal{D}', L')$, then the collection of joint winning strategies for all subsystems $(\mathcal{D}', L')$, where $\mathfrak{A}'$ is a *connected component* of $\mathfrak{A}$, also yields a joint winning strategy for $(\mathcal{D}, L)$: The specification $(L_p)_{p\in P'}$ embraces everything that L can express about the subsystem $\mathcal{D}'$ within the system $\mathcal{D}$. That means, in particular, L cannot relate the individual connected components of $\mathfrak{A}$ among each other. So, the controller problem for $\mathfrak{A}$ is decidable if, and only if, it is decidable for every connected subarchitecture of $\mathfrak{A}$. Therefore, w.l.o.g., we assume that $\mathfrak{A}$ is connected.

On the other hand, we can not assume, of course, that $\mathfrak{A}$ is strongly connected and, in particular, there may be non-reachable processes. This issue does not occur in [137]: There, any process has at least one input channel and since only acyclic architectures are considered, every process is reachable. In our setting, processes do not need to have input channels and, even if they do, cycles still make it possible to have non-reachable processes. Now, in the context of global specifications, such processes can be incorporated into the specification as their strategies are only words, cf. [80]. (Notice that all non-reachable processes are equally informed.) For locally decomposable specifications, however, this cannot be done so easily since a non-reachable process may send information to several other processes which are thereby linked to each other. In general, this link cannot be directly established via any of the local specifications. We will see, however, that, also for locally decomposable specifications, all non-reachable processes can be assembled into one large monster process.

In the following, we denote the class of all pipelines with backward-channels by $\mathcal{K}_1$ and the class of all two-flanked pipelines with backward-channels which have either only two controllers or which do not have a backward-channel from the last process by $\mathcal{K}_2$. Moreover, for an architecture $\mathfrak{A}$, $M(\mathfrak{A})$ denotes the set of all controllers which are *not reachable.* Notice that $P_{\text{con}} \setminus M(\mathfrak{A})$ induces a subarchitecture of $\mathfrak{A}$.

Theorem 5.14. *The controller problem for $\mathfrak{A}$ is decidable for locally decomposable regular specifications, if, and only if, any connected component of $\mathfrak{A}(P_{con} \setminus M(\mathfrak{A}))$ is in $\mathcal{K}_1 \cup \mathcal{K}_2$. The problem remains decidable for deterministic context-free specifications if, and only if, one of the following conditions hold.*

(1) $\mathfrak{A} \in \mathcal{K}_1$ and L_p is regular for all controllers p except the last one.

(2) There is a $p \in M(\mathfrak{A})$ such that $L_{p'}$ if regular for all $p' \in P_{con} \setminus \{p\}$.

Moreover, in case (2), L_p may even be nondeterministic context-free.

Proof. First assume that the controller problem for $\mathfrak{A}$ is decidable for locally decomposable regular specifications and let $\mathfrak{B} = (C^{\mathfrak{B}}, r^{\mathfrak{B}}, w^{\mathfrak{B}})$ be a connected component of $\mathfrak{A}(P_{\text{con}} \setminus M(\mathfrak{A}))$. If $|P^{\mathfrak{B}}| = 1$ then obviously $\mathfrak{B} \in \mathcal{K}$, so let $|P^{\mathfrak{B}}| = 1$. Since $\mathfrak{B}$ is a connected component and disjoint from $M(\mathfrak{A})$, there is at least one controller $p \in P^{\mathfrak{B}}_{\text{con}}$ that reads some channel from the environment and sends information to some controller q in $\mathfrak{B}$.

For $k \in \mathbb{N}$, let $\text{Reach}_k(p)$ be the set of all $q \in P^{\mathfrak{B}}_{\text{con}}$ with $\text{dist}(p, q) = k$, that means, there is a directed path from p to q of length k in $\mathfrak{B}$ but there is no such path of length $< k$. First we have $|\text{Reach}_k(p)| \leq 1$ for all $k \in \mathbb{N}$, because if $|\text{Reach}_k(p)| > 1$ for some $k \in \mathbb{N}$ with $q \neq q' \in \text{Reach}_k(p)$ then q and q' are incomparably informed. This puts us in the situation of Theorem 5.12. Now let

$$m = \max\{k \in \mathbb{N} \mid \text{Reach}_k(p) \neq \emptyset\}$$

and for $k \leq m$, let q_k be the unique process with $q_k \in \text{Reach}_k(p)$. (Notice that $\text{Reach}_l(p) \neq \emptyset$ implies $\text{Reach}_k(p) \neq \emptyset$ for all $k \leq l$.) Since p sends information to at least one $q \in P^{\mathfrak{B}}_{\text{con}}$ we have $m \geq 1$.

Now we show that

$$\text{Reach}(p) = \{q_0, q_1, \ldots, q_m\} = P^{\mathfrak{B}}_{\text{con}}.$$

Towards a contradiction, assume that there is some controllers $q \in P^{\mathfrak{B}}_{\text{con}} \setminus \text{Reach}(p)$. Since $\mathfrak{B}$ is connected, there is some (not necessarily directed) path from q to some q_i which implies that there is some $q' \in P^{\mathfrak{B}} \setminus \text{Reach}(p)$ that sends information to q_i or receives information from q_i. However, in the latter case we would have $q' \in \text{Reach}(p)$, so in fact q' sends information to q_i. Again, since $\mathfrak{B}$ is a connected component and disjoint from $M(\mathfrak{A})$, q' is reachable from q_{env} in $\mathfrak{B}$ and as it is not reachable from q, the controllers q' and q are incomparably informed. Moreover, if $i = 0$ then $q_1 \notin \{q, q'\}$ is reachable from both q' and q and if $i > 0$ then $q_i \notin \{q, q'\}$ is reachable from both q' and q. Therefore, we are in the situation of Theorem 5.13 and hence, $\text{Reach}(p) = P^{\text{con}}$.

Moreover, none of the processes q_i with $0 < i < m$ reads a channel from the environment because otherwise, $q_0 = p$ and q_i would be incomparably informed and q_{i+1} is reachable from q_0 and from q_i, so Theorem 5.13 would apply. So, if q_m also reads no channel from the environment, then $\mathfrak{B}$ is a pipeline with backward channels, that means, $\mathfrak{B} \in \mathcal{K}_1$. If, on the other hand, q_m does read a channel from the environment, then $\mathfrak{B}$ is a two-flanked pipeline with backward channels. Now, if $m \geq 2$ and p_m sends information to some process p_i with $i < m$, then Theorem 5.13 could be applied. So we have $m = 1$ or there are no backward channels from p_m which yields $\mathfrak{B} \in \mathcal{K}_2$.

As to context-free specifications, since $\mathfrak{A}$ is connected, Theorem 5.9 yields that the controller problem for $\mathfrak{A}$ becomes undecidable if there are at least two processes that have a deterministic context-free specification. Therefore, assume that exactly one process p has a deterministic context-free specification. If $p \in M(\mathfrak{A})$, then for all $p' \in M(\mathfrak{A}) \setminus \{p\}$, the local specification $L_{p'}$ of p' is regular, so condition (2) holds. Now consider the case $p \notin M(\mathfrak{A})$, that means, p is reachable, and let $p_{\text{env}} \to q_0 \to \ldots q_k = p$ be some path from the environment to p. We claim that $P_{\text{con}} = \{q_i \mid i = 0, \dots, k\}$: If there is some $q \in P_{\text{con}} \setminus \{q_i \mid i = 0, \dots, k\}$ then q is not better informed than p because $p_{\text{env}} \to q_0 \to \dots \to p$ is a path from p_{env} to p that does not contain q. This puts us in the situation of Theorem 5.10. Therefore, $P_{\text{con}} = \{q_i \mid i = 0, \dots, k\}$ and since $L_{q_0}, \dots, L_{q_{k-1}}$ are regular, condition (1) holds.

Now assume, conversely, that any connected component of $\mathfrak{A}(P_{\text{con}} \setminus M(\mathfrak{A}))$ is in $\mathcal{K}_1 \cup \mathcal{K}_2$ and let

$$\mathfrak{B}_1, \dots, \mathfrak{B}_m$$

be the connected components of $\mathfrak{A}(P_{\text{con}} \setminus M(\mathfrak{A}))$. Let $L \subseteq (\Sigma^{\mathfrak{A}})^\omega$ be a locally decomposable specification $L = (L_p)_{p \in P_{\text{con}}}$ for $\mathfrak{A}$ and let $(\Sigma_c)_{c \in C}$ be the underlying labeling of $\mathfrak{A}$. If condition (1) holds then Theorem 5.7 tells us that the controller problem for $\mathfrak{A}$ is decidable, so assume that condition (2) holds and, moreover, L_p may be nondeterministic context-free. So, any $\mathfrak{B}_j$ is in $\mathcal{K}_1 \cup \mathcal{K}_2$ and there is a $p \in M(\mathfrak{A})$ such that $L_{p'}$ is regular for all $p' \in P_{\text{con}} \setminus \{p\}$, cf. Figure 5.3. We abbreviate $M = M(\mathfrak{A})$ and $C_M = \bigcup_{p \in M} w^{-1}(p)$, that is, the channels in C_M are precisely the output channels of the non-reachable processes, and we set $\Sigma_M = \prod_{c \in C_M} \Sigma_c$.

First, for $p \in M$, let $\overline{L}_p = \{\alpha \in \Sigma_M^\omega \mid \text{Pr}_{\Sigma^p}(\alpha) \in L_p\}$ and let

$$L_M = \bigcap_{p \in M} \overline{L}_p.$$

That is, L_M is a global specification for the non-reachable processes of $\mathfrak{A}$. Now, a strategy for a non-reachable process $p \in M$ is simply a word from $(\Sigma^p)^\omega$. Therefore, $\alpha_M \in L_M$ if, and only if, there is a joint strategy $\sigma_M = (\sigma_p)_{p \in M}$ for the non-reachable processes such that any global system behavior α of $\mathfrak{A}$ which is consistent with σ_M fulfills all local specifications of the processes in M and satisfies $\text{Pr}_{\Sigma_M}(\alpha) = \alpha_M$. Moreover, since at most one specification L_p for $p \in M$ is context-free and all others are regular, L_M is context-free.

Now consider some connected component

$$\mathfrak{B}_i = \mathfrak{B} = (C^{\mathfrak{B}}, r^{\mathfrak{B}}, w^{\mathfrak{B}})$$

of $\mathfrak{A}(P_{\text{con}} \setminus M)$ and let

$$p_1 \preceq \dots \preceq p_n$$

be the controllers of $\mathfrak{B}$. We consider here only the case where $\mathfrak{B}$ is a two-flanked pipeline with backward-channels but no backward channels from the last process. The other two cases – pipelines with backward channels and two-flanked pipeline with backward-channels that consist of only two controllers – can be treated very similar. Our construction proceeds in four steps. First, we simulate the channels $C_{M \to \mathfrak{B}} = C_M \cap r^{-1}(P)$ from processes in M to processes in $\mathfrak{B}$ by channels from p_{n-1} to the other processes in $\mathfrak{B}$. Notice that all backward channels introduced in this way are feedback channels. In the next step, we remove all feedback channels from $\mathfrak{B}$ using the construction described in Section 5.1.2. Then we construct the nondeterministic parity tree automaton $\mathcal{N}^{\mathfrak{B}}$ from Theorem 5.3 recognizing $L(\mathcal{A}_n) \cap L(\mathcal{A}_{n-1})$, as described there. In the final step, we construct an alternating parity automaton $\mathcal{A}$ that runs over words from $(\Sigma_{C_{M \to \mathfrak{B}}})^\omega$ and checks that there is a tree in $L(\mathcal{N}^{\mathfrak{B}})$ which represents a language (a possible output language of p_{n-1}) that is constant on the channels $C_{M \to \mathfrak{B}}$) (which we have simulated by output channels of p_{n-1}).

Step 1. We define $\widehat{\mathfrak{B}} = (\widehat{C}, \widehat{r}, \widehat{w})$ and $\widehat{L} = (\widehat{L}_1, \dots, \widehat{L}_n)$ as follows. First, we set

$$\widehat{C} = C^{\mathfrak{B}} \cup C_{M \to \mathfrak{B}} \cup C^d_{M \to \mathfrak{B}}.$$

The channels from M to $P^{\mathfrak{B}}_{\text{con}}$ are simulated by the channels $C_{M \to \mathfrak{B}}$ with

- $w(C_{M \to \mathfrak{B}}) = \{p_{n-1}\}$

- $\widehat{r}(c) = r(c)$ for all $c \in C_{M\to\mathfrak{B}} \setminus r^{-1}(p_{n-1})$
- $\widehat{r}(c) = p_{\text{env}}$ for all $c \in C_{M\to\mathfrak{B}} \cap r^{-1}(p_{n-1})$.

That means, via the channels $C_{M\to\mathfrak{B}}$ in $\widehat{\mathfrak{B}}$, p_{n-1} sends information to the respective recipients of the original channels, or to the environment p_{env}, in case p_{n-1} is the recipient of the channel. Notice that, since p_{n-1} is *better informed* than any process in M, it can deduce the outputs of all processes in M from its own inputs. Moreover, the channels $C^d_{M\to\mathfrak{B}}$ are duplicate channels which are read by process p_n and the modified specification $\widehat{L}_{p_{n-1}}$ of process p_{n-1} requires that the information sent along the channels $C_{M\to\mathfrak{B}}$ is the same as the information sent along the channels $C^d_{M\to\mathfrak{B}}$ in each step:

- $w(C_{M\to\mathfrak{B}}) = \{p_{n-1}\}$ and $r(C_{M\to\mathfrak{B}}) = \{p_n\}$
- $\widehat{L}_{p_{n-1}}$ is the intersection of
  - $\{\beta \in (\widehat{\Sigma}^{p_{n-1}}) \mid \mathrm{Pr}_{\Sigma^{p_{n-1}}}(\beta) \in L_{p_{n-1}}\}$
  - $\{\beta \in (\widehat{\Sigma}^{p_{n-1}}) \mid \mathrm{Pr}_{\Sigma_{M\to\mathfrak{B}}}(\beta) = \mathrm{Pr}_{\Sigma^d_{M\to\mathfrak{B}}}(\beta)\}$.

Clearly, the fact that the channels from $C_{M\to\mathfrak{B}}$ that a process p_i reads are now channels from p_{n-1} instead of channels from processes in M does not require any modification of the specification L_{p_i}. So, apart from the equality constraint on the channels from $C_{M\to\mathfrak{B}}$ and their respective duplicate channels, the specifications of the processes are just as before. That means, $\widehat{L}_{p_i} = L_{p_i}$ for $i = 1, \dots, n-2$ and the specification $\widehat{L}_{p_n}$ is L_{p_n}, adapted to the new channels (on which $\widehat{L}_{p_n}$ does not impose any conditions).

Steps 2 & 3. First, we apply the construction described in Section 5.1.2 to remove all feedback channels from $\widehat{\mathfrak{B}}$ which yields a two-flanked pipeline $\tilde{\mathfrak{B}}$ and a new specification $\tilde{L} = (\tilde{L}_1, \dots, \tilde{L}_n)$ for $\tilde{\mathfrak{B}}$. Then we use the construction from [137] as described in Section 5.1.1 to obtain a nondeterministic parity tree automaton $\mathcal{N}^{\mathfrak{B}} = (\mathbb{B}, Q^{\mathcal{N}}, q^{\mathcal{N}}_{\text{in}}, \delta^{\mathcal{N}}, \text{col}^{\mathcal{N}})$ over communication trees $t \in \mathbb{T}_c(\tilde{\Sigma}_{n-1})$ which accepts such a tree if, and only if:

(1) There are strategies $\tilde{\sigma}_1, \dots, \tilde{\sigma}_{n-1}$ for $p_1, \dots, p_{n-1}$ such that
   (a) $\tilde{\sigma}_1 | \dots | \tilde{\sigma}_{n-1}$ generates a language $L^{n-1}_{\text{out}} \subseteq L^\omega(t)$ over Σ^ω_{01}
   (b) Each strategy $\tilde{\sigma}_i$ is locally winning on the language $L^i_{\text{in}} \subseteq \tilde{\Sigma}^\omega_{i-1}$ generated by $\tilde{\sigma}_1 | \dots | \tilde{\sigma}_{i-1}$ over Σ^ω_{01}

(2) There is a strategy $\tilde{\sigma}_n$ for p_n that is locally winning on $L^\omega(t) \times \Sigma^\omega_{0,n}$.

However, this solution is still not sufficient for our concerns because process p_{n-1} is now in charge of the channels $C_{M\to\mathfrak{B}}$ but since it better informed than the processes in M, it may create outputs on these channels that could not be produces by the processes in M. In fact, as we have mentioned above, in order to be sure that p_{n-1} creates outputs on the channels $C_{M\to\mathfrak{B}}$ that are consistent with some strategy for the processes in M, p_{n-1} may not use any information. To ensure this, we use the duplicate channels: We construct a solution of the controller problem for $\tilde{\mathfrak{B}}$ as described above, where the language $L^\omega(t)$ is *constant* on the channels $C^d_{M\to\mathfrak{B}}$. That means, there is some word $\mu \in (\Sigma^d_{M\to\mathfrak{B}})^\omega$ such that $\mathrm{Pr}_{\Sigma^d_{M\to\mathfrak{B}}}(L^\omega(t)) = \{\mu\}$. This is done in the final step.

Step 4. We construct an alternating parity automaton $\mathcal{A}^{\mathfrak{B}}$ over ω-words $\mu \in (\Sigma^d_{M\to\mathfrak{B}})^\omega$ such that $\mu \in L(\mathcal{A}^{\mathfrak{B}})$ if, and only if, there is a $t \in \mathbb{T}_c(\tilde{\Sigma}_{n-1})$ with

$$\mathrm{Pr}_{\Sigma^d_{M\to\mathfrak{B}}}(L^\omega(t)) = \{\mu\} \quad \text{and} \quad t \in L(\mathcal{N}^{\mathfrak{B}}).$$

The idea for the automaton is as follows. It runs over the word μ, *universally* spans a $\tilde{\Sigma}_{n-1}$-tree over μ and *nondeterministically* guesses a $\mathbb{B}$-labeling for it, such that the resulting $\mathbb{B}$-labeled $\tilde{\Sigma}_{n-1}$-tree t is in $\mathbb{T}_c(\tilde{\Sigma}_{n-1})$. Moreover, we have the additional constraint that $\mathcal{A}^{\mathfrak{B}}$ only guesses the label $\top$ for a node $u \in \tilde{\Sigma}_{n-1}^*$ if $u = \mu(<|u|)$. This can be achieved by ensuring, in each step, that u is only $\top$-labeled if $\mathrm{last}(u) = \mu(|u| - 1)$. At the same time, $\mathcal{A}^{\mathfrak{B}}$ simulates $\mathcal{N}^{\mathfrak{B}}$ on t. Formally,

$$\mathcal{A}^{\mathfrak{B}} = (\Sigma^d_{M\to\mathfrak{B}}, Q, q_{\mathrm{in}}, \delta, \mathrm{col})$$

with

- $Q = Q^{\mathcal{N}} \times \mathbb{B} \times (\tilde{\Sigma}_{n-1} \cup \{\varepsilon\})$
- $q_{\mathrm{in}} = (q_{\mathrm{in}}^{\mathcal{N}}, \top, \varepsilon)$
- $\mathrm{col}(q, \zeta, b) = \mathrm{col}^{\mathcal{N}}(q)$
- $\delta((q, \perp, \tilde{b}), a) = \bigvee_{[\varphi\in\delta^{\mathcal{N}}(q,\top)]} \bigwedge_{[b\in\tilde{\Sigma}_{n-1}]} (\downarrow, (p, \perp, b))$ where $(\downarrow_b, p) \in \varphi$

The case in the definition of δ that we have already done – where $\zeta = \perp$, that is, the label that has been chosen in the previous step is $\perp$ – is the obvious one: We just propagate $\perp$ to all directions from $\tilde{\Sigma_{n-1}}$ and continue to simulate $\mathcal{N}^{\mathfrak{B}}$. The more involved case is $\zeta = \top$:

$$\delta((q, \top, \tilde{b}), \tilde{a}) = \bigvee_{[\emptyset\neq X\subseteq\tilde{\Sigma}_{n-1}]} \bigvee_{[\varphi\in\delta^{\mathcal{N}}(q,\top)]} \bigwedge_{[b\in\tilde{\Sigma}_{n-1}]} (\downarrow, q^{(X,\varphi,b,\tilde{a})})$$

where

$$q^{(X,\varphi,b,\tilde{a})} = \begin{cases} (p, \top, b) & \text{if } b \in X \text{ and } \mathrm{Pr}_{\Sigma^d_{M\to\mathfrak{B}}}(b) = \tilde{a} \text{ and } (\downarrow_b, p) \in \varphi \\ (p, \perp, b) & \text{if } (b \notin X \text{ or } \mathrm{Pr}_{\Sigma^d_{M\to\mathfrak{B}}}(b) \neq \tilde{a}) \text{ and } (\downarrow_b, p) \in \varphi \end{cases}$$

Putting it All Together. Consider some $\mu \in L(\mathcal{A}^{\mathfrak{B}})$, that means, there is a tree $t \in \mathbb{T}_c(\tilde{\Sigma}_{n-1})$ such that $\mathrm{Pr}_{\Sigma^d_{M\to\mathfrak{B}}}(L^\omega(t)) = \{\mu\}$ and $t \in L(\mathcal{N}^{\mathfrak{B}})$. As $t \in L(\mathcal{N}^{\mathfrak{B}})$, there is a joint winning strategy $\tilde{\sigma} = (\tilde{\sigma}_1, \ldots, \tilde{\sigma}_n)$ for the grand coalition for $\tilde{\mathfrak{B}}$ and $\tilde{L}$ such that the language generated by $\tilde{\sigma}_1|\ldots|\tilde{\sigma}_{n-1}$ over Σ_{01}^ω is a subset of $L^\omega(t)$. According to Proposition 5.5 the grand coalition has a winning strategy $\widehat{\sigma} = (\widehat{\sigma}_1, \ldots, \widehat{\sigma}_n)$ for $\widehat{\mathfrak{B}}$ and $\widehat{L}$ and it is easy to see from the construction of $\widehat{\sigma}$ that the language generated by $\widehat{\sigma}_1|\ldots|\widehat{\sigma}_{n-1} \circ \mathrm{Pr}_{\widehat{\Sigma}_{n-1}}$ over Σ_{01}^ω is also a subset of $L^\omega(t)$. (Notice that, by construction of $\tilde{\mathfrak{B}}$, $\tilde{\Sigma}_{n-1} = \widehat{\Sigma}_{n-1}$.) So any global system behavior $\widehat{\beta}$ of $\widehat{\mathfrak{B}}$ which is consistent with $\widehat{\sigma}$ satisfies $\mathrm{Pr}_{\Sigma^d_{M\to\mathfrak{B}}}(\widehat{\beta}) = \mu$.

We define the joint strategy $\sigma^{\mathfrak{B}} = (\sigma_1, \ldots, \sigma_n)$ for the processes $p_1, \ldots, p_n$ in the architecture $\mathfrak{A}$ as follows. First, $\sigma_i = \widehat{\sigma_i}$ for $i = 1, \ldots, n-2$ and $\sigma_{n-1} = \widehat{\sigma}_{n-1} \circ \mathrm{Pr}_{\Sigma^{p_{n-1}}_{\mathrm{out}}}$. Moreover, $\sigma_n(u) = \widehat{\sigma}_n(u^\frown\mu_{<|u|})$ for $u \in (\Sigma^{p_n}_{\mathrm{in}})^*$. Now consider any global system behavior α of $\mathfrak{A}$ which is consistent with $\sigma^{\mathfrak{B}}$ and fulfills $\mathrm{Pr}_{\Sigma_{M\to\mathfrak{B}}}(\alpha) = \mu$. We define $\widehat{\beta} = \mathrm{Pr}_{\Sigma^{\mathfrak{B}}}(\alpha)^\frown\mu$.

Then $\mathrm{Pr}_{\Sigma^{p_i}}(\widehat{\beta}) = \mathrm{Pr}_{\Sigma^{p_i}}(\alpha)$ for $i = 1, \ldots, n-2$, so $\widehat{\beta}$ of $\widehat{\mathfrak{B}}$ is consistent with $\tilde{\sigma}_1, \ldots, \tilde{\sigma}_{n-2}$ by definition of $\sigma_1, \ldots, \sigma_{n-2}$. Moreover, $\mathrm{Pr}_{\Sigma^d_{M\to\mathfrak{B}}}(\widehat{\beta}) = \mathrm{Pr}_{\Sigma_{M\to\mathfrak{B}}}(\widehat{\beta})$, so as $\widehat{\sigma}$ is winning, $\widehat{\beta}$ is consistent with $\widehat{\sigma}_{n-1}$. Finally, by definition of σ_n, $\widehat{\beta}$ is consistent with $\widehat{\sigma}_n$, so altogether we have that $\widehat{\beta}$ is consistent with $\widehat{\sigma}$. As $\widehat{\sigma}$ is winning, $\mathrm{Pr}_{\tilde{\Sigma}^{p_i}}(\widehat{\beta}) \in \tilde{L}_{p_i}$ and thusly $\mathrm{Pr}_{\Sigma^{p_i}}(\alpha) \in L_{p_i}$ for $i = 1, \ldots, n$.

The converse from a strategy $\sigma^{\mathfrak{B}}$ for $p_1, \ldots, p_n$ to a word $\mu \in L(\mathcal{A}^{\mathfrak{B}})$ can be shown similarly. Therefore, we have $\mu \in L(\mathcal{A}^{\mathfrak{B}})$ if, and only if, there is a strategy $\sigma^{\mathfrak{B}} = (\sigma_1, \ldots, \sigma_n)$ for the processes $p_1, \ldots, p_n$ in the architecture $\mathfrak{A}$ such that any global system behavior α of $\mathfrak{A}$ which is consistent with σ and satisfies $\mathrm{Pr}_{\Sigma_{M\to\mathfrak{B}}}(\alpha) = \mu$, fulfills all local specifications of $p_1, \ldots, p_n$.

Now let, for any connected component $\mathfrak{B}_j$ of $\mathfrak{A}(P_{\mathrm{con}} \setminus M)$ let

$$\overline{L}(\mathcal{A}^{\mathfrak{B}_j}) = \{\alpha \in \Sigma_M^\omega \mid \mathrm{Pr}_{\Sigma_{M\to\mathfrak{B}_j}}(\alpha) \in L(\mathcal{A}^{\mathfrak{B}_j})\}$$

and let

$$\overline{L} = \bigcap_{j=1}^{m} \overline{L}(\mathcal{A}^{\mathfrak{B}_j}) \cap L_M.$$

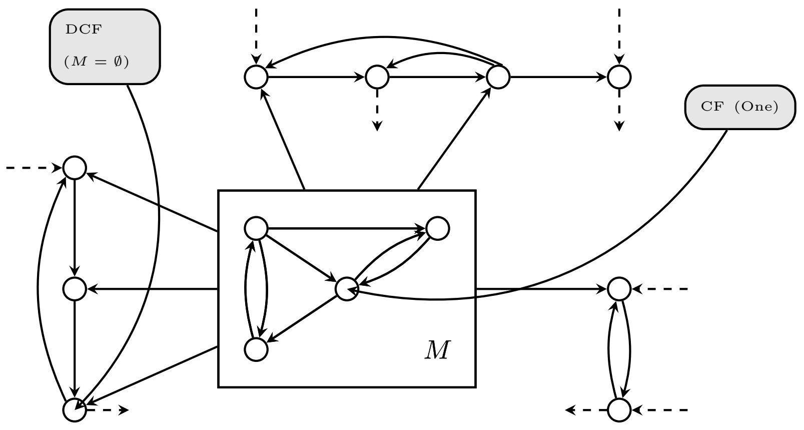


Figure 5.3: Generic decidable architecture.

Then we have $\overline{L} \neq \emptyset$ if, and only if, there is a word $\mu \in L_M$ such that $\mathrm{Pr}_{\Sigma_{M \to \mathfrak{B}_j}}(\mu) \in L(\mathcal{A}^{\mathfrak{B}_j})$ for each j, that means, for each j:

- there is a joint strategy $\sigma^{\mathfrak{B}_j}$ for the controllers in $\mathfrak{B}_j$
- each system run α of $\mathfrak{A}$ that is consistent with $\sigma^{\mathfrak{B}_j}$ and satisfies $\mathrm{Pr}_{\Sigma_{M \to \mathfrak{B}_j}}(\alpha) = \mathrm{Pr}_{\Sigma_{M \to \mathfrak{B}_j}}(\mu)$ fulfills all local specifications of the controller in $\mathfrak{B}_j$.

As we have mentioned, the words in L_M correspond directly to strategies for the non-reachable processes, so we conclude that $\overline{L} \neq \emptyset$ if, and only if, the grand coalition a joint winning strategy for $\mathfrak{A}$ and L. Now, any of the languages $\overline{L}(\mathcal{A}^{\mathfrak{B}_j})$ is regular because alternating parity ω-automata and deterministic parity ω-automata are equally expressive (Theorem 3.8 and Theorem 2.8). Therefore, $\overline{L}$ is context-free, so emptiness of $\overline{L}$ can be decided. □

Remarks. From our proofs and constructions in Section 5.1 and Section 5.2, we obtain some amendments to Theorem 5.14.

Weaker Specifications: By Theorem 5.9 and Theorem 5.10, all undecidability results for deterministic context-free specifications hold even for deterministic realtime 1-counter specifications. It is also not very hard to see that all undecidability results go through for safety conditions: In the proofs of Theorem 5.9 and Theorem 5.10 it is sufficient to have DR1-C automata that use a safety acceptance condition and in the proof of Theorem 5.11, all specifications can easily be seen to be recognizable by an ω-automaton that uses safety acceptance. For Theorem 5.12 and Theorem 5.13 we did not demonstrate the technical details but for these results, safety conditions suffice as well, cf. [137, 135, 80, 190].

Enumerability and Finite State Strategies: Any of the controller problems that is not decidable, is not even recursively enumerable as has been stated in our undecidability results. On the other hand, these proofs do not directly establish undecidability of the finite state winning strategy problem, that means, does the grand coalition have a joint winning strategy $\sigma = (\sigma_1, \ldots, \sigma_n)$ such that each σ_i can be implemented by a finite memory structure? The reason is that our reduction from *non*-halting problems, while showing non-enumerability, does require the controllers to use infinite state strategies because they have to cover the infinite number of steps that the machine runs.[10]

[10]Notice that obviously we cannot have both at the same time because the finite state controller problem for (not necessarily locally decomposable) deterministic context-free specifications *is* recursively enumerable for *any* architecture: We can successively enumerate all finite state controllers using one state, then those using two

Nevertheless, the proofs of all theorems from Section 5.2 can be adapted so that they show undecidability of the finite state controller problem. The key step is to perform reductions from *halting* problems instead of non-halting problems. However, this also involves different kinds of winning conditions. For Theorem 5.9 and Theorem 5.10, for example, we need DR1-C automata with *reachability* acceptance condition. Moreover, for Theorem 5.12 and Theorem 5.13 we need specifications that can express at least disjunctions of safety and reachability conditions. The original proof (for not necessarily locally decomposable specifications) can be found in [171], for adaptions to locally decomposable specifications see [137, 86].

Nondeterministic Contextfree Specifications: The addendum concerning nondeterministic context-free specifications establishes, in fact, the only case where such a specification can be allowed: The controller problem for $\mathfrak{A}$ is undecidable as soon as there is some reachable process that has a nondeterministic context-free specification. The reason is that already the controller problem for the single controller architecture $\mathfrak{A}_1$ is undecidable for nondeterministic context-free specifications, cf. Section 2.2.2 and Section 3.1.3 and if we have a process p that reads some channel from the environment and has a nondeterministic context-free specification, then the controller problem for $\mathfrak{A}$ comprises the controller problem for $\mathfrak{A}$. If, on the other hand, p is merely reachable from p_{env} via some directed path $p_0 \to p_1 \to \ldots \to p_k = p$, then the controller problem for $\mathfrak{A}_1$ can be easily reduced to the controller problem for $\mathfrak{A}$: Given a specification L for $\mathfrak{A}_1$, we label the channel from p_i to p_{i+1} for $i = 0, \ldots, k-1$ with $\Sigma^{\mathfrak{A}_1}_{c\in}$, where c_{in} is the channel from the environment to the single controller in $\mathfrak{A}_1$. Moreover, we label some output channel of p with $\Sigma^{\mathfrak{A}_1}_{c_{\text{out}}}$, where c_{out} is the external output channel of the controller in $\mathfrak{A}_1$. Then we require that all the processes $p_1, \ldots, p_{k-1}$ just propagate the signals that they receive from p_{env} in any step while $p_k = p$ has the same specification as the controller in $\mathfrak{A}_1$ (up to the delay that is created by propagating the signals from p_0 to p.)

Processes without Output Channels. So far, we have assumed that any controller has at least one output channel. We have already mentioned in Section 2.1.3 that, for locally decomposable specifications, this assumption cannot be neglected a priori: Processes that do not have any output channels may still increase the expressiveness of the individual local specifications. This is very similar to the way in which non-reachable processes can connect certain channels and increase the expressiveness of local specifications. As we have seen in Theorem 5.14, non-reachable processes may be connected arbitrarily among each other and may send information to other controllers in an arbitrary fashion. Processes without output channels, on the other hand, do not send information to any process, so in particular, they are not connected among each other at all, but they may receive information from other processes.

However, as we have seen in Theorem 5.13, as soon as a controller p receives information from two controllers $p_1 \neq p_2$ which are both reachable, then the controller problem is undecidable, even if p does not have any output channels. Now assume that p receives information from processes $p_1, \ldots, p_k$ but at most p_1 is reachable. First, if p has a regular local specification, then p can be eliminated as follows: Let c_i be the channel from p_i to p. We re-define $r(c_1) = p_{\text{env}}$ and $r(c_i) = p_1$ for all $i = 2, \ldots, k$. Then the local specification of p can be checked at p_1 and since none of the processes $p_2, \ldots, p_k$ is reachable, this construction does not harm decidability of $\mathfrak{A}$. If p has a deterministic context-free specification then, since p is reachable, Theorem 5.10 yields that as soon as there is some process p' that is connected to p and not better informed than p, then $\mathfrak{A}$ is undecidable. So, if $\mathfrak{A}$ is in fact decidable then we have $k = 1$ and the only process that is connected to p_1 and is not better informed than p_1 is p. So in this case, $\mathfrak{A}$ is a pipeline with last process p, so p can also be eliminated by incorporating its local specification into the one of p_1. Since $\mathfrak{A}$ is still a pipeline with last process p_1, $\mathfrak{A}$ remains decidable.

Therefore, although Theorem 5.14 does not hold in the stated form if we allow controllers

states and so on. Moreover, once we have constructed a joint finite state strategy σ, we can check whether it is winning: The set of all system runs that are consistent with σ is then regular and $X \subseteq Y$ is decidable if X is regular and Y is deterministic context-free.

without output channels, we can extend the characterization to this case: We inspect each such controller and check, according to the cases constituted above, whether this controller makes $\mathfrak{A}$ undecidable. If we don't find any such controller then we know that we can characterize $\mathfrak{A}$ along the patterns from Theorem 5.14, ignoring the controllers without output channels. To solve the controller problem for a given specification, we then eliminate all processes without output channels in advance, as described above.

Complexity. Recall that the complexity of the controller problem for pipeline architectures is nonelementary, cf. 3.3: Given a pipeline $\mathcal{D}_{n,p}$ with n controllers and an LTL specification φ, solving the controller problem for $(\mathcal{D}_{n,p}, \varphi)$, asymptotically, takes time at least $\exp_n(|\varphi|)$. The reason is the iterative application of a construction which consists of removing alternation from a parity tree automaton and simulating the resulting nondeterministic parity tree automaton by the next alternating parity tree automaton. Since this nondeterministic parity tree automaton is exponentially bigger than the original alternating one, we get an exponential blow up in each step. The same kind of construction has been applied to solve the controller problem for two-flanked pipelines with locally decomposable regular specifications. In fact, it is not hard to see that also for locally decomposable regular specifications, the controller problem has a nonelementary complexity, even for straight pipelines.

The reason is that we can simulate an arbitrary regular specification L for a pipeline $\mathfrak{A}$ with controllers $p_1, \dots, p_n$ by a collection of local regular specifications $L_1, \dots, L_n$, see also [135]: Add backward channels c_{i1} for $i = 2, \dots, n$ which are labeled with $\Sigma_{c_{i1}} = \Sigma_{\text{out}}^{p_i}$ and modify the local specification of p_i to require $\mathrm{Pr}_{\Sigma_{c_{i1}}}(\beta) = \mathrm{Pr}_{\Sigma_{\text{out}}^{p_i}}(\beta)$ for each local run of p_i. That means, p_i must write the same signal to c_{i1} that it writes into its original output channels in each step. Then, the global specification L can be checked using just L_1. Moreover, using the construction from Section 5.1.2, we can transform the modified system back into a system with a pipeline architecture and a locally decomposable regular specification. Clearly, this whole construction takes only time polynomial in the size of the system and the specification.

Corollary 5.15. *The controller problem for $\mathcal{D}_{n,p}$ is n-*EXPTIME*-hard for locally decomposable regular specifications.*

Now, in view of practical applicability of the decision procedures that we have developed (and those that were already known previously for distributed systems [171, 129, 137]) this may seem very discouraging. On the other hand, in [129], it has been mentioned that the high complexity of synthesis algorithms is somewhat misleading because it is measured only in terms of the size of the specification while the complexity of verification algorithms is measured in terms of the size of the system as well. Although the system about which we're talking is yet to be build in the case of synthesis, it might seem more appropriate to take its size into account as well. Since those systems have nonelementary size as well (cf. Section 5.4), the complexity then appears in a different light.

5.4 Controller Synthesis

Theorem 5.14 provides us with a comprehensive yet simple and, in particular, decidable criterion for the decidability of a given architecture for locally decomposable regular and context-free specifications and there is an algorithm that solves the controller problem for any decidable architecture. Now, as usual, once we know that a joint winning strategy for the grand coalition exists, we also want to implement such a strategy by some kind of computing device. We have already discussed in Section 3.2.1 how we can obtain such implementations by iteratively applying Rabin's regularity theorem (Theorem 3.7) and plugging in the solution from the last step. Here, we will also obtain implementations of winning strategies from our automata theoretic solution of the controller problem for the decidable architectures for locally decomposable specifications. However, since the trees over which the automata run do not directly represent strategies for the controllers, the construction is more involved. We consider only the regular case and we will see that finite state strategies are sufficient. We will make some remarks on the context-free case at the end of this section.

In the following, consider a distributed system with a locally decomposable specification

$$\mathcal{D} = (\mathfrak{A}, (\Sigma_c)_{c \in C}) \quad \text{with } \mathfrak{A} = (C, r, w) \quad \text{and } L = (L_p)_{p \in P_{\text{con}}}.$$

such that L_p is regular for all $p \in P_{\text{con}}$ and $\mathfrak{A}$ is decidable for locally decomposable regular specifications, that means, each connected component of $\mathfrak{A}(P_{\text{con}} \setminus M(\mathfrak{A}))$ is in $\mathcal{K}_1 \cup \mathcal{K}_2$ as defined in Section 5.3. Let $\mathfrak{B}_1, \dots, \mathfrak{B}_m$ be the connected components of $\mathfrak{A}(P_{\text{con}} \setminus M(\mathfrak{A}))$ and, for each $\mathfrak{B}_i$ let $\mathcal{A}^{\mathfrak{B}_i}$ be an alternating parity automaton over words $\mu \in (\Sigma_{M \to \mathfrak{B}_i})^\omega$ as in the proof of Theorem 5.14.

We have to start at the end. If the grand coalition does have a winning strategy for $(\mathcal{D}, L)$ then $\overline{L} \neq \emptyset$, so there is a word $\nu \in \overline{L}$. According to Rabin's regularity theorem[11] there is also a regular word $\nu \in \overline{L}$, that means, ν can be generated by a finite automaton

$$\mathcal{C} = (\{|\}, Q^{\mathcal{C}}, \delta^{\mathcal{C}}, q_{\text{in}}^{\mathcal{C}}, \tau^{\mathcal{C}})$$

with output function $\tau^{\mathcal{C}} : Q^{\mathcal{C}} \to \Sigma_M$ as $\nu(k) = \tau^{\mathcal{C}}((\delta^{\mathcal{C}})^*(u))$ for $u = |^k$. This automaton immediately yields an implementation of a joint strategy σ_ν^M for the non-reachable processes. Moreover, this strategy can be easily decomposed into individual strategies σ_p for $p \in M$ by simply projecting to the respective output of p. (Recall that a strategy for p just depends on the ticks of the system.)

Now we want to synthesize a joint strategy for all the reachable processes that agrees with ν. For this, we can consider each component $\mathfrak{B}_i$ individually as in the proof of Theorem 5.14. Again, we consider only the case where $\mathfrak{B}$ is a two-flanked pipeline with backward-channels but no backward channels from the last process. So, consider some connected component

$$\mathfrak{B}_i = \mathfrak{B} = (C^{\mathfrak{B}}, r^{\mathfrak{B}}, w^{\mathfrak{B}})$$

of $\mathfrak{A}(P_{\text{con}} \setminus M)$ and let

$$p_1 \preceq \dots \preceq p_n$$

be the controllers of $\mathfrak{B}$. Moreover, let $\widehat{\mathfrak{B}}$ and $\tilde{\mathfrak{B}}$ be as in the proof of Theorem 5.14, that is, $\widehat{\mathfrak{B}}$ simulates the channels from M to $\mathfrak{B}$ by feedback channels and $\tilde{\mathfrak{B}}$ simulates the feedback channels by forward channels. Now consider the following tree automata:

- $\mathcal{A}_{n-1} = (\mathbb{B}, Q^{n-1}, q_{\text{in}}^{n-1}, \delta^{n-1}, \text{col}^{n-1})$
- $\mathcal{A}_n = (\mathbb{B}, Q^n, q_{\text{in}}^n, \delta^n, \text{col}^n)$
- $\mathcal{N} = (\mathbb{B}, Q^{\mathcal{N}}, q_{\text{in}}^{\mathcal{N}}, \delta^{\mathcal{N}}, \text{col}^{\mathcal{N}})$

$\mathcal{A}_{n-1}$ and $\mathcal{A}_n$ are the alternating parity tree automata from Theorem 5.3, applied to $\tilde{\mathfrak{B}}$ and $\mathcal{N}$ is a nondeterministic parity tree automaton with $L(\mathcal{N}) = L(\mathcal{A}_{n-1}) \cap L(\mathcal{A}_n)$. Since $\nu \in \overline{L}$ we have

$$\mu = \text{Pr}_{M \to \mathfrak{B}}(\nu) \in L(\mathcal{A}^{\mathfrak{B}}),$$

so there is some tree $t \in L(\mathcal{N}) \subseteq \mathbb{T}_c(\tilde{\Sigma}_{n-1})$ such that $\text{Pr}_{\Sigma^d_{M \to \mathfrak{B}}}(L^\omega(t)) = \{\mu\}$. By construction of $\mathcal{A}_{n-1}$ and $\mathcal{A}_n$ we know that:

(1) There are strategies $\tilde{\sigma}_1, \dots, \tilde{\sigma}_{n-1}$ for $p_1, \dots, p_{n-1}$ such that
  (a) $\tilde{\sigma}_1 | \dots | \tilde{\sigma}_{n-1}$ generates a language $L_{\text{out}}^{n-1} \subseteq L^\omega(t)$ over Σ_{01}^ω
  (b) Each strategy $\tilde{\sigma}_i$ is locally winning on the language $L_{\text{in}}^i \subseteq \tilde{\Sigma}_{i-1}^\omega$ generated by $\tilde{\sigma}_1 | \dots | \tilde{\sigma}_{i-1}$ over Σ_{01}^ω

[11]Notice that here, Rabin's regularity theorem is actually overkill as $\overline{L}$ is ω-regular. It suffices to use Büchi's result that any ω-regular language contains an ultimately periodic word.

(2) There is a strategy $\tilde{\sigma}_n$ for p_n that is locally winning on $L^\omega(t) \times \Sigma_{0,n}^\omega$.

Now we would like to synthesize such strategies, starting with $\tilde{\sigma}_n$. As we have mentioned, the problem is that the *input* trees of $\mathcal{A}_n$ do not represent strategies of process p_n but just possible *inputs* that p_n might receive from p_{n-1}. On the other hand, from the proof of Theorem 5.3 as described in Section 5.1.1 we know that we can obtain an appropriate strategy for p_n from an accepting *run* of $\mathcal{A}_n$: A run of $\mathcal{A}_n$ on some given input tree $t \in \mathbb{T}_c(\tilde{\Sigma}_{n-1})$ is a $\tilde{\Sigma}_{n-1}^* \times Q^n$-labeled $\mathbb{N}$-tree ρ. Now, Q^n has components (q, b, c) where $q \in Q^{\mathcal{S}_n}$ is a state of the deterministic parity automaton $\mathcal{S}_n$ recognizing $\tilde{L}_n$, $b \in \Sigma_{n,e}$ is some output signal of p_n and $c \in \Sigma_{0,n}$ is some external input signal of p_n.[12] Moreover, any run ρ has the property that for all $(u, w) \in \tilde{\Sigma}_{n-1}^* \times \Sigma_{0,n}^*$ there is exactly one node x in ρ that corresponds to (u, w), that means, $\mathrm{Pr}_{\tilde{\Sigma}_{n-1}^*}(\rho(x)) = u$ and the unique path from the root of ρ to x is labeled with w in the Σ_{0n}-components.

Therefore, a run of $\mathcal{A}_n$ on t can be represented as a $\mathbb{B} \times Q^{\mathcal{S}_n} \times \Sigma_{n,e}$-labeled $\tilde{\Sigma}_{n-1} \times \Sigma_{0,n}$-tree

$$\rho : (\tilde{\Sigma}_{n-1} \times \Sigma_{0,n})^* \to \mathbb{B} \times Q^{\mathcal{S}_n} \times \Sigma_{n,e}$$

where the $\mathbb{B}$-components are the corresponding labels of t. Now, if such a tree ρ represents, in fact, an accepting run of $\mathcal{A}_n$ on some input tree $t \in \mathbb{T}_c(\tilde{\Sigma}_{n-1})$ then any path through ρ which is labeled with $\top^\omega$ in the $\mathbb{B}$-component is labeled with an accepting run of $\mathcal{S}_n$ in the $Q^{\mathcal{S}_n}$-component. So, indeed, ρ represents a strategy $\tilde{\sigma}_n$ for p_n that is locally winning on $L^\omega(t) \times \Sigma_{0,n}^\omega$.

However, Rabin's regularity theorem only guarantees that there is some regular *input* tree, not a regular run. Even worse, the set of all (accepting) runs of a given *alternating* tree automaton is *not* a regular tree language in general. The problem is that for a tree $\rho : (\tilde{\Sigma}_{n-1} \times \Sigma_{0,n})^* \to \mathbb{B} \times Q^{\mathcal{S}_n} \times \Sigma_{n,e}$ to be actually a run of $\mathcal{A}_n$ on some input tree t, it must satisfy the consistency constraint that

$$\mathrm{Pr}_{\mathbb{B}}(u, w) = \mathrm{Pr}_{\mathbb{B}}(u, w')$$

that is, the $\mathbb{B}$-components that represent the labels of the input tree t may, of course, depend only on the nodes of t and not on the nodes $w \in \Sigma_{0,n}^*$. However, we already know that such consistency constraints cannot be checked by tree automata, cf. Section 3.2.

On the other hand, if we consider runs of $\mathcal{A}_n$ on some *fixed* tree $t \in \mathbb{T}_c(\tilde{\Sigma}_{n-1})$ then these $\mathbb{B}$-components are determined by the labels of t – and if t is *regular*, then these labels can be computed while running over ρ. Therefore, the set of all (accepting) runs of $\mathcal{A}_n$ on some regular tree t is regular and therefore, by Rabin's regularity theorem, we can find again a *regular* tree

$$t' : (\tilde{\Sigma}_{n-1} \times \Sigma_{0,n})^* \to \mathbb{B} \times Q^{\mathcal{S}_n} \times \Sigma_{n,e}$$

which represents an accepting run of $\mathcal{A}_n$ on t, so it contains an implementation of a strategy $\tilde{\sigma}_n$ for p_n that is locally winning on $L^\omega(t) \times \Sigma_{0,n}^\omega$.

Now, since we need the *same* tree t also for synthesizing a winning strategy for p_{n-1}, we start by construction a regular tree $t \in L(\mathcal{N})$. However, we cannot take any such tree but we have to construct one where $\mathrm{Pr}_{\Sigma^d_{M \to \mathfrak{B}}}(L^\omega(t)) = \{\mu\}$. Since μ can be generated by $\mathcal{C}$ (using projection to $\Sigma_{M \to \mathfrak{B}}$) we can construct such a tree by plugging μ into the emptiness game for $\mathcal{N}$.

Constructing $t \in L(\mathcal{N})$ with $\mathrm{Pr}_{\Sigma^d_{M \to \mathfrak{B}}}(L^\omega(t)) = \{\mu\}$. To guarantee that $\mathrm{Pr}_{\Sigma^d_{M \to \mathfrak{B}}}(L^\omega(t)) = \{\mu\}$, we relativize the emptiness game of $\mathcal{N}$ to μ similar as described in Section 3.2.1. However, here we don't just take the product of the game graph $\mathcal{G}(\mathcal{A}_n)$ and the automaton $\mathcal{C}$ that generates μ, because now, μ is not a sequence of labels but of (partial) *directions* of the trees $t \in \mathbb{T}_c(\tilde{\Sigma}_{n-1})$.

In the game graph $\mathcal{G}(\mathcal{A}_n, \mu)$, positions of player 1 are of the form (q, p, ζ) where $q \in Q^{\mathcal{N}}$, $p \in Q^{\mathcal{C}}$ and $\zeta \in \mathbb{B}$. Positions of player 0 are of the form $(q, \zeta, q_1, \ldots, q_s, p)$ where $\tilde{\Sigma}_{n-1} = \{a_1, \ldots, a_s\}$,

[12] Notice that we did not mention the fact that Q^n contains the $\Sigma_{0,n}$-component explicitly in Section 5.1.1. However, we always assume that an alternating tree automaton carries its universal branching in its state space, cf. Section 3.2.1 and Theorem 5.8. Here, this makes it easier to write down the strategy that we obtain from such a run. However, in the end this will make no difference.

$(q, \zeta, q_1, \ldots, q_s)$ is a transition of $\mathcal{A}_n$ and $p \in Q^{\mathcal{C}}$. The initial position is $(q^{\mathcal{N}}_{\text{in}}, q^{\mathcal{C}}_{\text{in}}, \top)$. From a position (q, p, ζ), player 1 can move to any position $(q, \zeta', q_1, \ldots, q_s, \delta^{\mathcal{C}}(p, |))$ such that

$$\zeta = \bot \implies \zeta' = \bot.$$

From a position $(q, \zeta, q_1, \ldots, q_s, p)$ can move to any position (q_j, p, ζ') such that

$$\zeta' = \top \iff \mathrm{Pr}_{\Sigma_{M\to\mathfrak{B}}}(\tau^{\mathcal{C}}(p)) = \mathrm{Pr}_{\Sigma^d_{M\to\mathfrak{B}}}(a_j).$$

So, being in a position $(q, \zeta, q_1, \ldots, q_s, p)$ of player 0 and in some node $u \in \tilde{\Sigma}^*_{n-1}$ of the input tree, we look at all the successors $u \cdot a_j$ of u and for each such successor with $\mathrm{Pr}_{\Sigma^d_{M\to\mathfrak{B}}}(a_j) \neq \mu(|u|)$ we predetermine the label $\bot$ for $u \cdot a_j$. In this way, player 1 constructs a tree $t \in L(\mathcal{N})$ such that, for each node $u \in \tilde{\Sigma}^*_{n-1}$,

$$\mathrm{Pr}_{\Sigma^d_{M\to\mathfrak{B}}}(u) \neq \mathrm{Pr}_{\Sigma_{M\to\mathfrak{B}}}(\mu(<|u|)) \implies t(u) = \bot,$$

that means, $\mathrm{Pr}_{\Sigma^d_{M\to\mathfrak{B}}}(L^\omega(t)) = \{\mu\}$.

Constructing $\tilde{\sigma}_n$. Now we can use t to construct a strategy $\tilde{\sigma}_n$ for p_n as described above: We construct a nondeterministic parity tree automaton $\mathcal{A}'_n$ that runs over a $\mathbb{B} \times Q^{\mathcal{S}_n} \times \Sigma_{n,e}$-labeled $\tilde{\Sigma}_{n-1} \times \Sigma_{0,n}$-tree t' and checks that t' is an accepting run of $\mathcal{A}_n$ on t. For this, $\mathcal{A}'_n$ uses the finite automaton $\mathcal{C}_t$ that generates t to compare the $\mathbb{B}$-labels of t' with those of t. Moreover, $\mathcal{A}'_n$ simulates $\mathcal{A}_n$ which can now be done by a nondeterministic tree automaton: Nondeterminism is needed because $\mathcal{A}'_n$ still has to *guess* the transitions of $\mathcal{A}_n$ that have been used to build the alleged run t'. On the other hand, alternation is not needed because the branching of t' comprises the universal branching of $\mathcal{A}_n$. Now, by Rabin's theorem we can construct a regular tree $t' \in L(\mathcal{A}'_n)$, from which we immediately obtain an implementation of a strategy $\tilde{\sigma}_n$ for p_n that is locally winning on $L^\omega(t) \times \Sigma^\omega_{0,n}$.

Constructing $\tilde{\sigma}_{n-1}, \ldots, \tilde{\sigma}_1$. When trying to synthesize a strategy $\tilde{\sigma}_{n-1}$ for the process p_{n-1}, we run into the same kind of problem as before: A tree $t \in \mathbb{T}_c(\tilde{\Sigma}_{n-1})$ does not represent a strategy for p_{n-1} but merely possible *outputs* that p_{n-1} might send to p_n. Again, we have to construct a regular *run* of $\mathcal{A}_{n-1}$: In any run ρ of $\mathcal{A}_{n-1}$ on some tree $t \in \mathbb{T}_c(\tilde{\Sigma}_{n-1})$ and for any $w \in \tilde{\Sigma}^*_{n-2}$, there is exactly one node x in ρ that corresponds to w, that means, the unique path from the root of ρ to x is labeled with w in the $\tilde{\Sigma}_{n-2}$-components. Now, the states of $\mathcal{A}_{n-1}$ have components (q, b, c, ζ) where $q \in Q^{\mathcal{S}_{n-1}}$ is a state of the deterministic parity automaton $\mathcal{S}_{n-1}$ recognizing $\tilde{L}_{n-1}$, $b \in \Sigma_{n-1,e}$ is some external output signal of p_{n-1} and $c \in \tilde{\Sigma}_{n-2}$ is some input signal of p_{n-1}. Moreover, ζ is the label of the current node $w \in \tilde{\Sigma}^*_{n-2}$ of the tree $s \in \mathbb{T}_c(\tilde{\Sigma}_{n-2})$ that $\mathcal{A}_{n-1}$ guesses. Therefore, a run of $\mathcal{A}_{n-1}$ can represented as a $\mathbb{B}_1 \times \mathbb{B}_2 \times Q^{\mathcal{S}_{n-1}} \times \Sigma_{n-1,e}$-labeled $\tilde{\Sigma}_{n-2}$-tree

$$\rho : (\tilde{\Sigma}_{n-2})^* \to \mathbb{B}_1 \times \mathbb{B}_2 \times Q^{\mathcal{S}_{n-1}} \times \Sigma_{n-1,e}$$

where $\mathbb{B}_1 = \mathbb{B}_2 = \mathbb{B}$ and the first $\mathbb{B}$-component is the corresponding label of the input tree t.

As before, we construct a nondeterministic parity tree automaton $\mathcal{A}'_{n-1}$ that runs over such trees $t' = \rho$ and checks whether t' is an accepting run of $\mathcal{A}_{n-1}$ on t, using the finite automaton that generates t. Then we construct a regular such tree, which yields a strategy $\tilde{\sigma}_{n-1}$ for p_{n-1} and a regular tree $s \in \mathbb{T}_c(\tilde{\Sigma}_{n-2})$ such that, by construction of $\mathcal{A}_{n-1}$, the following holds:

(1) There are strategies $\tilde{\sigma}_1, \ldots, \tilde{\sigma}_{n-2}$ for $p_1, \ldots, p_{n-2}$ such that
   (a) $\tilde{\sigma}_1 | \ldots | \tilde{\sigma}_{n-2}$ generates a language $L^{n-2}_{\text{out}} \subseteq L^\omega(s)$ over Σ^ω_{01}
   (b) Each strategy $\tilde{\sigma}_i$ is locally winning on the language $L^i_{\text{in}} \subseteq \tilde{\Sigma}^\omega_{i-1}$ generated by $\tilde{\sigma}_1 | \ldots | \tilde{\sigma}_{i-1}$ over Σ^ω_{01}

(2) $\tilde{\sigma}_{n-1} \circ \mathrm{Pr}_{\tilde{\Sigma}_{n-1}}$ generates a language $L^{n-1}_{\mathrm{out}} \subseteq L^\omega(t)$ over $L^\omega(s)$

In the next step, we proceed completely analogously, using the tree s as a regular input tree for $\mathcal{A}_{n-2}$. In the end, we obtain a joint winning strategy

$$\tilde{\sigma} = (\tilde{\sigma}_1, \ldots, \tilde{\sigma}_n)$$

for the processes $p_1, \ldots, p_n$ in $\tilde{\mathfrak{B}}$ such that each $\tilde{\sigma}_i$ is implemented by a finite memory structure. Moreover, $\tilde{\sigma}_1 | \ldots | \tilde{\sigma}_{n-1}$ generates a language L^{n-1}_{out} over Σ^ω_{01} that satisfies $\mathrm{Pr}_{\Sigma^d_{M\to\mathfrak{B}}}(L^{n-1}_{\mathrm{out}}) = \{\mu\}$.

Constructing $\sigma^{\mathfrak{B}}$. The strategy $\tilde{\sigma}$ is a joint winning strategy for the grand coalition for $\tilde{\mathfrak{B}}$ that is constant on the channels $C_{M\to\mathfrak{B}}$. We have seen in the proof of Theorem 5.14 how to construct from $\tilde{\sigma}$ a joint strategy $\sigma^{\mathfrak{B}} = (\sigma_1, \ldots, \sigma_n)$ for the processes $p_1, \ldots, p_n$ in the architecture $\mathfrak{A}$ such that any global system behavior α of $\mathfrak{A}$ which is consistent with σ and satisfies $\mathrm{Pr}_{\Sigma_{M\to\mathfrak{B}}}(\alpha) = \mu$, fulfills all local specifications of $p_1, \ldots, p_n$. This involves, in particular the construction described in the proof of Proposition 5.5. It is easy to see how to obtain an implementation of $\sigma^{\mathfrak{B}}$ from an implementation of $\tilde{\sigma}$: We re-define the transition function of each automata so that it uses, for the input that it expects on the additional channels (which have been introduced to simulate the feedback channels), its own corresponding output in the last step. Once we have got an implementation of such a winning strategy $\sigma^{\mathfrak{B}_i}$ for each component $\mathfrak{B}_i$ we can easily combine these strategies with σ^M_ν to obtain a joint strategy for the grand coalition for $\mathfrak{A}$ and L.

Theorem 5.16. *Let $\mathfrak{A}$ be an architecture that is decidable for locally decomposable regular specifications and let L be a locally decomposable regular specification for $\mathfrak{A}$ such that the grand coalition has a joint winning strategy for $(\mathfrak{A}, L)$. Then one can effectively construct finite state implementations of individual strategies σ_i such that $\sigma = (\sigma_1, \ldots, \sigma_n)$ is a joint winning strategy for the grand coalition for $(\mathfrak{A}, L)$. The size of each individual strategy σ_i is at most $\exp_n(|L|)$.*

Alternative Solution. Our construction for removing feedback channels in the context of locally decomposable specifications is rather simple. So is the construction of an implementation of $\sigma^{\mathfrak{B}}$ from a given implementation of $\tilde{\sigma}$ as obtained from the proof of Proposition 5.5. Nevertheless, we have to modify the given system in advance which, in particular, involves the specification of the system, that is, we have to manipulate the parity automata recognizing the individual local specifications. Moreover, once we have an implementation of $\tilde{\sigma}$ we have to manipulate the strategy automata in order to obtain an implementation of $\sigma^{\mathfrak{B}}$.

In [86] an alternative construction has been described that does not proceed via removal of feedback channels but yields a more direct approach. Although the solution is more complicated than just removing feedback channels and applying the construction from [137], it also offers a more coherent method for solving the controller problem for locally decomposable specification in the presence feedback channels because it does not involve any explicit manipulations of the given system. So throughout the whole construction, we consider only the actual system for which we want to synthesize a winning strategy. In particular, we do not have to modify the strategy automata a posteriori. In the following, we give a rough description of the construction. In particular, we explain the main concept, so-called *strategy products* of communication trees.[13]

As we have mentioned, feedback channels increase the expressive power of the individual local specifications. So, if we don't remove feedback channels a priori, the construction from [137] cannot be directly applied. Instead, we rather need (partial) assembly points where we can check the more global specifications. On the other hand, as we have already discussed, the construction from Section 3.2.1 does neither work for two-flanked pipeline with regular local specifications nor for pipelines with a deterministic context-free specification for the last process.

[13] A complete exposition of the construction can be found in the full version of [86], currently available at www.logic.rwth-aachen.de. (Notice that the formulation of Lemma 5 as stated there is incorrect: The function that generates a language $L \subseteq L^\omega(t_{\mathrm{in}})$ over Σ^ω_0 should be the *delay-composition* of $\sigma_1, \ldots, \sigma_{i-1}$ as in Section 5.1.)

The idea is to somehow combine the two approaches: We let the first controller p_1 use an *extended* strategy that anticipates the strategies of all other controllers (except for the last one, if we deal with two-flanked pipelines) but we *encode* these strategies in the output language of p_1.

This encoding of strategies in output languages is formalized by the notion of *strategy product* of communication trees, introduced in [86]: Given a tree t_{in} which represents input sequences that a process receives and a tree t_{out} which represents output sequences that the process may write, we define the strategy product $t_{\text{in}} \hookrightarrow t_{\text{out}}$ of t_{in} and t_{out} as a set of trees t, each of which defines an assignment of input sequences from $L^\omega(t_{\text{in}})$ to output sequences from $L^\omega(t_{\text{out}})$, so it yields a strategy $\sigma(t)$ for the process.

Formally, for a tree $t_{\text{in}} \in \mathbb{T}_c(\Sigma_{\text{in}})$ and a tree $t_{\text{out}} \in \mathbb{T}_c(\Sigma_{\text{out}})$, the strategy product $t_{\text{in}} \hookrightarrow t_{\text{out}}$ is defined as the set of all $\mathbb{B}$-labeled $\Sigma \times \Sigma'$-trees t such that the following conditions hold:

(S1) if $t_{\text{in}}(u) = \bot$ or $t_{\text{out}}(v) = \bot$ then $t(u^\frown v) = \bot$

(S2) if $t_{\text{in}}(u) = \top$ then there is exactly one $v \in (\Sigma_{\text{out}})^{|u|}$ such that $t(u^\frown v) = \top$

(S3) if $t(u^\frown v) = \top$ then there is some $b \in \Sigma_{\text{out}}$ such that for all $a \in \Sigma_{\text{in}}$ with $t_{\text{in}}(ua) = \top$ we have $t(ua^\frown vb) = \top$ and $t(ua^\frown vc) = \bot$ for $c \in \Sigma_{\text{out}} \setminus \{b\}$

Now, given some $t \in t_{\text{in}} \hookrightarrow t_{\text{out}}$, the strategy $\sigma(t)$ represented by t is defined as follows. For $u \in (\Sigma_{\text{in}})^*$ with $t_{\text{in}}(u) = \top$, let v be the unique element from $(\Sigma_{\text{out}})^*$ with $t(u, v) = \top$ and let a be the unique element from Σ_{out} with $t(ub, va) = \top$ for any $b \in \Sigma_{\text{in}}$ with $t_{\text{in}}(ub) = \top$.

We set

$$\sigma(t)(u) = a.$$

Moreover, if $t_{\text{in}}(u) = \bot$ then $\sigma(t)(u) = a$ for some $a \in \Sigma_{\text{out}}$. Notice that, in fact, for any such t, we have $\text{Pr}_{\Sigma_{\text{in}}}(L^\omega(t)) = L^\omega(t_{\text{in}})$ and $\text{Pr}_{\Sigma_{\text{out}}}(L^\omega(t)) \subseteq L^\omega(t_{\text{out}})$. Moreover, notice that $t_{\text{in}} \hookrightarrow t_{\text{out}} \subseteq \mathbb{T}_c(\Sigma_{\text{in}} \times \Sigma_{\text{out}})$.

Now using trees $t \in t_{\text{in}} \hookrightarrow t_{\text{out}}$ that encode extended strategies for the controllers instead of usual communication trees that just represent the output of one controller, we can proceed similarly as in [137]. Of course, we have to show that we can handle such trees with tree automata. As it turns out, strategy products have good regularity properties:

Proposition 5.17. *Let $\mathcal{A}$ be a nondeterministic parity tree automaton over $\mathbb{B}$-labeled $\Sigma_{in} \times \Sigma_{out}$-trees. The following tree languages are regular.*

(1) The set of all $\mathbb{B}$-labeled Σ_{out}-trees t_{out} such that there is a $\mathbb{B}$-labeled Σ_{in}-tree with $t_{in} \hookrightarrow t_{out} \cap L(\mathcal{A}) \neq \emptyset$.

(2) The set of all $\mathbb{B}$-labeled $\Sigma_{in} \times \Sigma_{out}$-trees t such that there is a $\mathbb{B}$-labeled Σ_{in}-tree t_{in} and a $\mathbb{B}$-labeled Σ_{out}-tree t_{out} with $t \in t_{in} \hookrightarrow t_{out}$.

Property (1) can be used in the inductive construction for the processes p_i with $i \in \{2, \dots, n-1\}$ where $\Sigma_{\text{in}} = \Sigma_{i-1}$ and $\Sigma_{\text{out}} = \Sigma_{\text{out}}^{\geq p_i}$. A noteworthy detail of the construction is the application of the widening operator $\text{wide}_{\Sigma_i^b}$ in each step, where Σ_i^b is the alphabet labeling the backward channels of p_i. That means, we project the backward channels of p_i out: The input trees $t \in \mathbb{T}_c(\Sigma_{\text{out}}^{\geq p_i})$ of $\mathcal{A}_i$ also contain the outputs that p_i writes into its backward channels but the strategy for p_{i+1} must not depend on these inputs. The important observation here is that the application of the widening operator does *not* increase the complexity by another exponential:[14] We can apply Proposition 3.4 directly to the alternating parity tree automaton $\mathcal{A}_i$ which yields an alternating parity tree automaton $\tilde{\mathcal{A}}_i$ of size polynomial in $\mathcal{A}_i$. Only then, we remove alternation and proceed to the next step in the iteration.

Moreover, notice that applying the widening operator at this point of the construction means that the tree $t' = \text{wide}_{\Sigma_i^b}(t) \in \mathbb{T}_c(\Sigma_{\text{out}}^{\geq p_i})$ (where $t \in t_{\text{in}} \hookrightarrow t_{\text{out}} \subseteq \mathbb{T}_c(\Sigma_i \times \Sigma_{\text{out}}^{\geq p_{i+1}})$ is the tree that

[14]If it did, then the increased complexity would obliterate any possible merits of this solution.

the automaton $\mathcal{A}_{i+1}$ guesses) allows process p_i to produce any possible output on its backward channels: If process p_i may produce $u \in \Sigma_i^*$ on c_i according to t then p_i may produce any output $u^\frown v \in (\Sigma_i \times \Sigma_i^b)^*$ on $\Sigma_{\mathrm{out}}^{p_i}$ because $t'(u, v) = \top$ if, and only if, $t(u) = \top$. To see why this is appropriate notice that, while process p_{i+1} must take into account *all* outputs that p_i may produce on c_i according to t, the processes p_j with $j < i$ do *not* have to take all the outputs into account that p_i may produce on its backward channels according to t': The strategy that p_i uses is determined by the extended strategies of these processes.[15] Therefore, a process p_j must take into account only those $v \in (\Sigma_i^b)^*$ with $t'(u, v) = \top$ (for some u) that it has actually chosen itself to be a response to some of its own inputs! Thusly, while t must be chosen carefully (with as few $\top$'s as possible), the extension of t to t' can be chosen maximally (as $\mathrm{wide}_{\Sigma_i^b}(t)$).

For the case of pipelines, property (2) gives us the main argument how to treat the last process p_n. For two-flanked pipelines, the last process is even easier because there are no backward channels from it. In any case, the induction basis is the easiest step because the inputs that p_1 may receice from p_0 are not restricted. Once we know that there is some joint winning strategy for the grand coalition, we can synthesize such a strategy along the same lines as above. Notice that, although the trees in $L(\mathcal{A}_i)$ encode strategies, a regular tree $t \in L(\mathcal{A}_i)$ does *not* directly yield a strategy for p_i: Such a tree is in $\mathbb{T}_c(\Sigma_{\mathrm{out}}^{\geq p_i})$, so it contains strategies for the processes p_j with $j > i$ but not for process p_i. Also notice that we have to apply the widening operator in each step. Clearly, if t is regular then $\mathrm{wide}_{\Sigma_i^b}(t)$ is regular as well.

Size of Implementations. We have already mentioned in Section 5.3 that the undecidability proofs from Section 5.2 demonstrate that there are distributed systems with two controllers and locally decomposable *safety* specifications where the following holds: The grand coalition has a joint winning strategy but each such strategy requires infinite memory (for both individual strategies). Moreover, the corresponding reductions from the halting problem (rather than the non-halting problem) demonstrate that there are such systems where the following holds:[16] The grand coalition has a winning strategy σ where σ_1 and σ_2 can be implemented by finite state machines but the size of these machines in the size of the specification is not computable. (If it was, the controller problem could be decided by successively trying the finitely many possible implementations of joint winning strategies using that many states, cf. Section 5.3. See also [30].) However, these lower bounds on the memory apply to *undecidable* architectures.

For architectures which are *decidable* for locally decomposable regular specifications, the constructions that we have presented yield that, whenever the grand coalition has a joint winning strategy $\sigma = (\sigma_1, \ldots, \sigma_n)$, then there is such a strategy where each σ_i is implemented by a finite state machine. Furthermore, the size of the implementations measured in the size of the specification is *computable*, but *nonelementary.* More precisely, given a pipeline $\mathcal{D}_{n,p}$ and a locally decomposable LTL specification $\varphi = \varphi_1 \wedge \ldots \wedge \varphi_n$, for any joint winning strategy $\sigma = (\sigma_1, \ldots, \sigma_n)$, the size of each strategy σ_i is at least $\exp_n(|\varphi|)$.

Moreover, from the nonelementary lower bound on the computational complexity in Corollary 5.15 we obtain a nonelementary lower bound on the overall size of the strategy σ as well, using the same argumentation as above: If the size of each strategy σ_i could be bounded by $\exp_k(|\varphi|)$ for some *fixed* $k \in \mathbb{N}$ then we could solve the controller problem for $(\mathcal{D}_{n,p}, \varphi)$ by successively trying the finitely many possible implementations of joint winning strategies, using at most $\exp_k(|\varphi|)$ many states. Once we have constructed such an implementation of $\sigma = (\sigma_1, \ldots, \sigma_n)$, checking whether σ is a joint winning strategy for the grand coalition simply amounts to constructing an ω-automaton $\mathcal{C}$ that recognized all global system runs that are consistent with σ and checking whether $L(\mathcal{C}) \subseteq L(\varphi)$.

The point is that now, not only are there finitely many implementations using at most

[15]Recall that, by construction, $\mathcal{A}_i$ simulates all automata $\mathcal{A}_j$ with $j < i$, so it is guaranteed that a tree which is accepted by $\mathcal{A}_i$ is actually part of such an extended strategy.

[16]Notice that in this case we need locally decomposable specifications that can express at least disjunctions of safety and reachability conditions.

$\exp_k(|\varphi|)$ states, but the number is elementary in the size of the specification. So trying all these possible implementations successively yields a decision procedure that is also elementary in the size of the specification which contradicts the lower bound on the complexity, cf. Section 5.3.[17]

Therefore, we cannot hope to obtain implementations of lower size than we get from the methods that we have presented here. The best we can do is to choose the lesser evil – removing feedback channels or constructing larger tree automata. Of course, there may be cases where it is possible (or even desirable) to remove feedback channels from the actual system instead of adapting the implementation to the original system with feedback channels. However, if the communication structure of the system is already implemented by some sort of expensive wiring or something, then this might not be the best thing to do.

Contextfree Specifications. So far, we have considered the implementation of strategies for the controllers only for architectures where each process has a regular local specification. If there is some controller in the architecture that may have a context-free local specification then, according to Theorem 5.14, this controller is either not reachable or the whole architecture is a pipeline. Let us consider the first case. As in the regular case, we start with some $\nu \in L_M$ but this time, ν cannot be generated by a finite state automaton but we have to use a pushdown automaton with output to represent ν finitely.[18]

In the next step, we play the emptiness game of the tree automaton $\mathcal{N}$ relativized to $\mu = \mathrm{Pr}_{\Sigma_M \to \mathfrak{B}}(\nu)$ just as before only that now, this game is not played on a finite graph but on an infinite graph generated by a pushdown automaton. By Walukiewicz's result [219], we can synthesize a pushdown strategy for player 1 which yields a tree $t \in L(\mathcal{N})$ that is generated by a pushdown automaton and agrees with μ. We continue this procedure just as in the regular case but now, in each step, we have a tree at hand that is generated by a pushdown automaton and, accordingly, we have to solve a parity game on a pushdown graph to construct the next tree. In the case where we have a reachable process with a deterministic context-free specification and the whole architecture is a pipeline, we proceed analogously, starting from the emptiness game for the nondeterministic parity pushdown tree automaton $\mathcal{A}_n$.

A noteworthy point about this construction is that, in general, it yields a joint strategy $\sigma = (\sigma_1, \ldots, \sigma_n)$ for the grand coalition where each σ_i is implemented by a pushdown automaton, even if the only process with a context-free specification is not reachable. It is also not hard to see that this is unavoidable, that means, in general, any controller p_i needs a pushdown memory as soon as there is some local specification that is context-free.

[17]Notice that this reasoning does not necessarily result in a precise lower bound of $\exp_n(|\varphi|)$. It is mentioned in [129], however, that such a lower bound does hold.

[18]Notice that such an automaton is guaranteed to exist, for example, by [219].

Chapter 6

Knowledge in Multiplayer Games

Our perception of the uncertainties and communication of the players in a game was rather pragmatic throughout the previous chapters and driven by the mere need to model the limited information of individual processes. In particular, in a distributed system the information that a process receives is determined by its communication channels and constitutes its strategic powers. These strategic powers, in turn, can be captured by tree automata so that for deciding the controller problem, we never have to retrace the epistemic dynamics of the process' state of mind. We conclude our presentation with a more explicit analysis of knowledge in multiplayer games and we incorporate our models into a more epistemological context.

In fact, while the construction in Section 3.5 can be straightforwardly derived from the usual powerset construction for nondeterministic reachability ω-automata, the knowledge tracking construction that we have advertised as a generalization of the powerset construction from Section 3.5 and as a method to obtain an explicit representation of the knowledge of the players in an arbitrary multiplayer game on a (finite) graph. calls for some basic knowledge about epistemic models and, in particular, possible worlds semantics. We present the required notions in Section 6.1.1. We shall, however, not be too precise and formal at this point but merely present some basic concepts and intuitions that are helpful to comprehend the construction in Section 6.2 and to deepen the understanding of epistemic reasoning in multiagent systems in general. In particular, we introduce the so-called S5-axioms and briefly discuss their consequences on the properties of knowledge in the models that we consider.

In Section 6.1.2 we have a look at the epistemic temporal logic ETL and we show that winning conditions which can be defined in ETL using both, synchronous and asynchronous, knowledge operators are, in fact, ω-regular. This yields a simplified proof of the decidability of ETL model checking on synchronous systems and it also extends this result to our particular case of asynchronous observability. Moreover, it has significant consequences on the decidability of synthesis from ETL specifications which we will discuss in Section 6.1.4. There, we also have a closer look at the strategic dependencies in multiplayer games, in particular, the fact that the question whether the grand coalition of cooperating players has a winning strategy includes the possibility of the players to *coordinate* on the joint strategy. We have already discussed this issue in Section 2.1.3 as well as in Chapter 5 and we return to it in this more epistemic context.

We present the knowledge tracking construction in Section 6.2. The main idea is to represent each possible epistemic state of the grand coalition during a play of the game as an individual position in the new game graph. Moreover, the strategic dependencies are captured as follows: Each player of the grand coalition chooses, in each *possible world* of the current epistemic state, an action and the successor states of the current state are determined by all these possible *macro* joint actions (Section 6.2.1). Notice that, by definition, this construction yields an infinite game graph. The main feature of the construction is now that, under the assumption of an *observable* winning condition, it allows for a more succinct representation by taking the quotient under *homomorphic equivalence*. That means, we can identify all homomorphically equivalent epistemic models and still obtain a game that is equivalent to the original game with imperfect information in the sense of (uniformly translatable) winning strategies (Section 6.2.2.)[1]

In general, the resulting game graph will still be infinite. In fact, we cannot represent all the possible mental states of the grand coalition in a finite way in general, even for games with winning conditions that allow positions strategies in the full information case: The reason is the *higher order knowledge* of the players, that is, knowledge about knowledge about ... We discuss this issue in Section 6.1.4. However, we will see that for hierarchical games, abstraction by homomorphic equivalence is, indeed, sufficient to yield a finite result for all game graphs. So we can infer that the strategy problem for hierarchical games with observable deterministic context-free winning conditions is decidable.

The method is also interesting beyond that result. First, as we have already mentioned in the introduction, since the construction yields an explicit representation of the possible states of knowledge of the players and the dynamics of this knowledge, analyzing the possibilities and limitations of knowledge tracking enhances the understanding of epistemic reasoning in multiplayer games. Moreover, the method may serve as a platform to approach decidability results for classes of games where we do not only limit the patterns of information flow between the players but also the structure of the game graph as well as amount of uncertainties that a player may have, cf. Chapter 4.

And finally, the method yields a semi-decision procedure for arbitrary games with observable omega-regular winning conditions already in its current form. That means, the procedure is not guaranteed to terminate but, if it does, it yields the correct answer.[2] In particular, while all known automata-based methods rely on a fixed structure of the communication graph (architecture) [171, 129, 137, 80], cf. Section 3.2.1 and Chapter 5, this algorithm does *not* require any a priori restriction of the information flow between the players. So, while we know that for (in)appropriately chosen game graphs, the method is bound to fail, even for safety conditions (cf. Theorem 5.11), the method can take advantage of the structure of the given game graph and may solve instances for which there is no way to get even started with automata-based methods.

Except when mentioned otherwise, throughout this chapter, we use a somewhat different version of game graphs with partial information which has the form

$$\mathcal{G} = (V, \Delta, (\sim_i^V)_{i\in\underline{n}}) \quad \text{with} \quad \Delta \subseteq V \times (A_1 \times \ldots \times A_n) \times V,$$

that means, we don't have actions for the environment explicitly but it just resolves the nondeterminism that is left over once all the players $1, \ldots, n$ have chosen their actions. We denote $A = A_1 \times \ldots A_n$. We view a play as a sequence of *positions* $\pi = v_0 v_1 \ldots$ such that for all $l \in \mathbb{N}$ there is some $a_l \in A$ with $(v_l, a_l, v_{l+1}) \in \Delta$. In particular, winning conditions and strategies are position based.[3] So, we implicitly assume that all actions are indistinguishable for all players. In particular, a player cannot distinguish his own actions. We will discuss the implications of this assumption in Section 6.1.4. We also have an equivalence relation for the environment player 0 which is meaningful in this more epistemic context where we refer to the knowledge of the players explicitly.

This chapter is based on [172, 173, 31]. The figures in Section 6.2 are reproduced from [31].

[1]Notice that, since multiplayer games with observable safety conditions are undecidable in general, this result is still nontrivial.

[2]Notice, however, that this does *not* yield recursive enumerability of the strategy problem for such games – this would require the algorithm to terminate on all positive instances which is impossible because the problem is *not* recursively enumerable, cf Chapter 5.

[3]It should be mentioned, that it is not difficult to incorporate actions into the constructions and results in this chapter. Technically, however, it would be rather costly and result in a significant blow up of notation.

6.1 Knowledge and Cooperation

We start with some background on knowledge and cooperation in infinite games and, more generally, knowledge in any kinds of multiagent systems. We just give a broad overview over some of the main ideas and concepts. For an in-depth treatment of epistemic reasoning in multiagent systems we recommend the book [78] and the original fundamental work of Halpern and Moses [105].[4] The articles [16, 17] by van Benthem and the paper [180] discuss some interesting notions concerning knowledge that are more related to games.

In Section 6.1.2 we consider the epistemic temporal logic ETL and we show how to translate ETL formulas into S1S formulas. In Section 6.1.4 we discuss the implications of this translation on synthesis from ETL specifications, in particular for hierarchical games. We also give a detailed discussion of epistemic reasoning when strategic dependencies and winning conditions come into play. This aims at deepening the understanding of knowledge in multiplayer games and serves as a preparation for the construction in Section 6.2.

6.1.1 Reasoning About Knowledge

Reasoning about knowledge originates in philosophy where it establishes the field of epistemology. Epistemology is concerned with the very nature of knowledge, that means, with its inherent properties and tries to answer questions like

- How is knowledge acquired?
- How can we know what we know?
- How can we distinguish knowledge from belief?

The idea of using logical languages and methods to put the analysis of those questions on a formal basis has been brought to full effect by the groundbreaking work of Jaakko Hintikka [109] in 1962. Hintikka formulated several rules of consistency for the notion of knowledge which (should) form the formal basis of reasoning about knowledge. Hintikka used a modal style logic and he developed a possible worlds semantics for this language.

The modal logic has operators K_a where a refers to a person and $K_a p$ for a certain sentence p means *a knows that p*. Semantics is given in terms of model systems which are sets of model sets, each of which is a set of modal sentences that satisfy the rules of consistency. On a model system, for a person referred to by a, an alternative relation is defined and the sentence $K_a p$ is in a model set μ if, and only if, p belongs to any alternative μ^* to μ with respect to a. In other words, in a *world* μ, the person referred to by a knows the fact p if p holds in any world μ^*, that a considers possible.

Possible worlds semantics were also investigated by other philosophers like Rudolph Carnap and Saul Kripke and are now widely used on so-called Kripke structures which also encompass Hintikka's model systems in a more abstract setting. A *Kripke structure* has the form

$$\mathcal{K} = (K, \mathrm{Prop}, (E_i)_{i \in I})$$

- K is a (finite) set (of possible worlds)
- Prop is a set of unary relations (atomic propositions)
- each E_i is a binary relation (alternative relation)

So, a Kripke-structure is a directed graphs with labeled edges and, for any vertex k, a set $\Phi_k = \{P \in \mathrm{Prop} \mid k \in P\}$ of atomic propositions assigned to k, where Prop contains all the relevant *basic* facts about the possible worlds. In a game for example one could have atomic propositions like *it is player i's turn* and *action a is available to player i* and so on. Epistemic logic is defined by

$$\varphi ::= P | \varphi \wedge \varphi | \neg\varphi | K_i \varphi$$

[4]It is also worth mentioning that Hintikka's groundbreaking essay [109] has been reprinted 2005 by King's College Publications.

- $\mathcal{K}, k \models P$ if $k \in P$
- $\mathcal{K}, k \models K_i\varphi$ if $\mathcal{K}, \tilde{k} \models \varphi$ for all $\tilde{k}$ with $(k, \tilde{k}) \in E_i$

So epistemic logic is just basic modal logic with $[i]\varphi$ written as $K_i\varphi$ and read as *agent i knows φ*. We also define the dual operator P_i as $P_i := \neg K_i \neg\varphi$ which might be read as *agent i considers it possible that φ*.

This formal treatment of knowledge has many applications. For example, in artificial intelligence, a robot should not only complete his task but he should also know when his task is completed which he can only guarantee using his sensors, to acquire basic data and using logical inference rules, to obtain more complex knowledge. In economy, a bargaining situation may require knowledge over certain facts which may be obtained using facts that are already known but may slip the bargainers attention. Moreover, in our context of synthesis under partial information, this is particularly important. For example, a safety critical action should only be performed in a system, if the controller knows that the current state of the system ensures a safe execution. Moreover, security protocols often involve requirements like *no component of the system will ever know the value of any internal variable of some other component.*

Notice that the formalization of knowledge by means of Kripke structures entails that the agents can do perfect reasoning, which is also called *logical omniscience*, that is, they know all tautologies and they know all logical consequences of their knowledge, cf. [78]. More precisely, if $\mathcal{M}_n$ is the class of all Kripke-structures with n agents, then the following propositions holds:

(1) $\mathcal{M}_n \models$ (A1) $(K_i\varphi \wedge K_i(\varphi \to \psi)) \to K_i\psi$
(Distribution Axiom)

(2) For all $\mathcal{K} \in \mathcal{M}_n$: (A2) if $\mathcal{K} \models \varphi$ then $\mathcal{K} \models K_i\varphi$
(Knowledge Generalization Rule)

Moreover, in these applications, in particular in the context of synthesis under partial information, it is often assumed that the alternative relations E_i are equivalence relations. First, let us discuss the, quite far-reaching, consequences of this assumption on the logical system. Afterwards we broach the issue of appropriateness. Let $\mathcal{M}_n^\chi$ for $\chi \in \{s, r, t\}$ be the class of all Kripke structures with n agents where all alternative relations E_i are symmetric, reflexive and transitive, respectively. We say that a formula φ is valid in $\mathcal{K} \in \mathcal{M}_n$, written $\mathcal{K} \models \varphi$, if φ holds at each vertex of $\mathcal{K}$. The formula is valid in a class $\mathcal{M} \subseteq \mathcal{M}_n$, written $\mathcal{M} \models \varphi$, if it is valid in each structure from $\mathcal{M}$. The following propositions hold:

(3) $\mathcal{M}_n^r \models$ (A3) $K_i\varphi \to \varphi$
(Knowledge Axiom)

(4) $\mathcal{M}_n^t \models$ (A4) $K_i\varphi \to K_iK_i\varphi$
(Positive Introspection Axiom)

(5) $\mathcal{M}_n^s \cap \mathcal{M}_n^t \models$ (A5) $\neg K_i\varphi \to K_i\neg K_i\varphi$
(Negative Introspection Axiom)

These five axioms are also called the S5-axioms.[5] The S5-axioms together with all tautologies of propositional calculus and modus ponens as inference rule, form a sound and complete axiom system for modal logic with respect to $\mathcal{M}_n^s \cap \mathcal{M}_n^t \cap \mathcal{M}_n^r$, cf. [78]. However, the positive and, even more, the negative introspection axiom are highly controversial among philosophers. In fact, while (A1)-(A4) are also amongst Hintikka's rules of consistency, he rejected the negative introspection axiom (A5). Notice that by proposition (5) this means that the alternative relations E_i will be not symmetric in general.

In our context (and in applications in computer science in general), the alternative relations will usually be equivalence relations intrinsically and therefore, the S5-axioms are indisputable in the relevant models. The reason why we have equivalence relations is that the alternative

[5]Notice that one can easily check that the above classes are *not* the largest classes in which the respective axioms are valid.

relation of an agent i is defined by the *information* that this agent has: A controller may have some predefined information that is stored in some sort of memory (which is included in the state space of the computing device implementing the controller) and any other information that the controller has, is received via sensors, communication channels linking it to other components and so on. This information concerns only a fixed amount of facts that we include in the model a priori and it can be encoded by bit strings. The controller does not have any form of uncontrollable perception, cognition or consciousness that would have to be taken into account. So we can represent the knowledge of the agents (controllers) in such a system by a Kripke structure

$$\mathcal{K} = (K, (\sim_i)_{i \in \underline{n}}, \text{Prop})$$

where K is the set of all possible states of the system, Prop is the set of relevant atomic facts about the system that we have decided to include and we define

$$k \ \sim_i \ \tilde{k} \quad \Longleftrightarrow \quad \Phi^i_k \ = \ \Phi^i_{\tilde{k}}$$

where $\Phi^i_k \subseteq \Phi_k = \{P \in \text{Prop} \mid k \in P\}$ is the information that agent i has about the state k. So in the world k, agent i considers the world $\tilde{k}$ possible, if its information is insufficient to distinguish whether the actual world is k or $\tilde{k}$. In a somewhat less precise sense we usually say that the worlds k and $\tilde{k}$ are indistinguishable for agent i. Clearly, any relation $\sim_i$ is an equivalence relation.

This demonstrates that alternative relations which are defined via some fixed amount of facts that the agents have are equivalence relations. Clearly we can model any equivalence relation $\sim_i$ on K in this way by using atomic propositions $\text{Prop} = 2^K$ with $\Phi^i_k = \{[k]_{\sim_i}\}$. However, usually we write epistemic formulas over some fixed set Prop which is independent of K, so we do not assume that the information that the agents may have about the system can be completely described using Prop.

Notice that we have also included an alternative relation $\sim_0$ for agent 0 which, in our models, represents the environment/nature. In fact, throughout this section we assume that the environment has partial information about the system as well. As we have mentioned repeatedly, so far, the information of the environment about the system was completely irrelevant. However, now that we address the knowledge of the components of the system explicitly, this is no longer true. It might be, for example, desirable to be able to express that the environment does not know the internal states of the controllers because knowing the internal states of the controllers of the system might add up to breaking the systems security or something like that.[6]

Although the representation of knowledge as described above is very natural in the context of the systems that we consider, we have to appreciate the properties of knowledge that this particular model entails, that means, we have to be aware of the S5-axioms. As we have mentioned, throughout the previous chapters, this issue was completely dispensable since we never had to care about the actual states of mind of the agents but we were concerned only with their strategic powers and handling and representing strategies appropriately. So we didn't have to concern ourself with the fact that from $K_i P$ and $K_i(P \to Q)$ it follows that $K_i Q$ holds as well. However, now that we address the knowledge of the components of the system explicitly, it is absolutely crucial to be aware of the fact that if, for example, a system invariably uses RSA-encryption with a *private* key x and a public key y, and agent i knows the value of y, then agent i also knows the value of x.[7]

Of course, if the controllers have a joint winning strategy, once we do have a finite state implementation, their reasoning powers are fixed by the particular computing device. However,

[6]Notice that the argumentation why $\sim_i$ will be an equivalence relation in our setting does not apply to $\sim_0$ if we include the possibility that the environment may be a human user. However, we don't discuss this issue any further but stick with the assumption that any $\sim_i$ is, indeed, an equivalence relation.

[7]Notice that we had to deal with some sort of encryption functions already in the proof of Theorem 5.12. However, it was not necessary to have an explicit epistemic representation of the knowledge of the players there but it was sufficient to reason about their strategic abilities.

if it is necessary for the controllers to break some encryption in the system in order to win, then the computing device will have the capacity to do so. This is somehow a limitation of our model because in many applications we would take the view that a task which requires to break RSA-encryption is intractable: The computing devices would have to be too large and it would take them too long to compute the necessary information. We do not discuss this issue or other consequences of the S5-axioms here further. We do suggest, however, to mind their presence and we refer to [78] and the literature given there for further discussions of this matter.

6.1.2 Epistemic Temporal Logic.

At this stage, the representation of the knowledge in a system as defined above does not directly incorporate the dynamic character of reactive systems. It applies to static systems or it may represent a snapshot of a dynamic system where, however, the agents' state of the mind is completely determined by their current observations and does not include their memory about the past. This, of course, is crucial because the knowledge of the agents *evolves* during a system run. In fact, as we have seen, even for winning conditions that allow positional strategies in the full information case, infinite memory may be needed in systems with some information fork. So, in general, there will even be *infinitely* many possible mental states that the agents may have during a system run.[8]

To incorporate the dynamics of a system over time into the above representation of knowledge, we take K to be the set of all *histories* π of events in the system instead of just current states. Moreover, the atomic propositions are inherited from the *states* of the system, that is, $\pi \in P$ if $\text{last}(\pi) \in P$. The alternative relations $\sim_i$ can, in principle, be any equivalence relations on $K \times K$. In the cases that we have treated in Chapters 4 – 5 the relations will be defined by synchronous or asynchronous observability, given the equivalence relations on the possible states/events of the system. In particular, the resulting Kripke structure is finitely presentable.[9]

On the other hand, in any formula of epistemic logic, we can only address a finite number of particular points in time but we can not express eventuality or necessity which is crucial for specifying nonterminating reactive systems. Therefore, we consider stronger epistemic logics that also incorporate time in LTL-style. Many incarnations of this concept have been investigated, for an overview and discussion we refer to [78, 104]. More recent approaches can be found for example in [3, 68]. They also deal with model checking knowledge and information flow, especially in interactive situations like games, but are significantly different from epistemic temporal logic as we consider it here. We won't discuss these approaches any further.

First, we subsume the above way of capturing the knowledge dynamics of a (nonterminating, reactive) system by a Kripke structure with the abstract notion of a multiagent system, cf. [78]. A *multiagent system* has the form

$$\mathcal{E} = (R, (\text{Prop}, L), (\sim_i)_{i \in \underline{n}})$$

- R is a set of *system runs*
- $L : R \times \mathbb{N} \to \text{Prop}$ is the propositional labeling
- each $\sim_i$ is an equivalence relation on $R \times \mathbb{N}$

A *finitely generated* multiagent system is one, which is defined in the above way from some given description of a finite state system

$$\mathcal{S} = (V, \text{Prop}, \Delta, (\sim_i^V))$$

[8]As we have mentioned, the reason is the *higher order* knowledge of the players, cf. Section 6.1.4.

[9]Recall that our construction in Section 6.2 follows a different approach: There, we encapsulate each possible state of mind of the agents, relative to *strategic dependencies*, as an individual position of the new (game) structure.

where V is a finite set of states and $\Delta \subseteq V \times A \times V$ is the move relation, assuming a finite set A of actions. We denote the generated system by $\mathcal{E}(\mathcal{S})$. Each alternative relation $\sim_i$ is assumed to be either $\sim_i^*$ or $\leftrightarrows_i^*$. The system is called *synchronous* if each $\sim_i$ is $\sim_i^*$.

Moreover, for a state $v \in V$ of $\mathcal{S}$, by $\mathcal{E}(\mathcal{S}, v)$ we denote the multiagent system that is defined in the same way but consists only of those system runs that start in the initial position v. We call such systems also *initialized* systems.

There are many other interesting cases of indistinguishability relations to be studied in the context of epistemic logics. We have already discussed general methods to define equivalence relations by means of logical formulas or automata. Let us just mention two more particular cases which are especially interesting:

- **Clock:** $(\pi, n) \sim_i (\rho, m) \iff n = m$ and $\text{last}(\pi) \sim_i^V \text{last}(\rho)$
- **Observational (Forgetful):** $(\pi, n) \sim_i (\rho, m) \iff \text{last}(\pi) \sim_i^V \text{last}(\rho)$

Now, *epistemic temporal logic* ETL is defined as follows:[10]

$$\varphi ::= P \mid \varphi \wedge \varphi \mid \neg\varphi \mid X\varphi \mid \varphi U \varphi \mid K_i\varphi \mid$$

- $\mathcal{E}, (\pi, t) \models P$ if $P \in L(\pi, t)$
- $\mathcal{E}, (\pi, t) \models X\varphi$ if $\mathcal{E}, (\pi, t+1) \models \varphi$
- $\mathcal{E}, (\pi, t) \models \varphi U \psi$ if there is some $s \geq t$ such that $\mathcal{E}, (\pi, s) \models \psi$ and $\mathcal{E}, (\pi, r) \models \varphi$ for all $t \leq r < s$
- $\mathcal{E}, (\pi, t) \models K_i\varphi$ if $\mathcal{E}, (\rho, s) \models \varphi$ for all $(\rho, s) \sim_i (\pi, t)$

We write $\mathcal{E} \models \varphi$ if $\mathcal{E}, (\pi, 0) \models \varphi$ for all $\pi \in R$. Moreover, for a finite state system $\mathcal{S}$ and some state $v \in V$, we write

$$\mathcal{S}, v \models \varphi \quad :\iff \quad \mathcal{R}(\mathcal{S}, v) \models \varphi.$$

Although we will not be concerned with common knowledge very much throughout our studies, due to its central importance in the theory of epistemic reasoning, we define this notion here as well. Epistemic Temporal Logic ETL $+C$ with *common knowledge* is defined as ETL with the additional rule

$$\varphi ::= C_B\varphi$$

where $B \subseteq \{1, \ldots, n\}$. The semantics is defined as

$$\mathcal{E}, (\pi, t) \models C_B\varphi \quad \text{if} \quad \text{for all } (\rho, s) \sim_B (\pi, t) \text{ we have } \mathcal{E}, (\rho, s) \models \varphi$$

where $\sim_B$ is the *transitive closure* of the *union* of all $\sim_i$ for $i \in B$. That means,

$$\mathcal{E}, (\pi, t) \models C_B\varphi \iff \text{for all } i_1, \ldots, i_k \in B : \mathcal{E}, (\pi, t) \models K_{i_1} \ldots K_{i_k}\varphi.$$

Notice that if we write these knowledge statements for all $i_1, \ldots, i_k$ down explicitly, it reads as follows:

- $\mathcal{E}, (\pi, t) \models K_i\varphi$: Everyone in B knows φ
- $\mathcal{E}, (\pi, t) \models K_iK_j\varphi$: Everyone in B knows that everyone in B knows φ
- ...

Model Checking. Using this formalism, we want to check systems for epistemic properties: Given a finite state system $\mathcal{S}$, a state v and a formula $\varphi \in$ ETL, does

$$\mathcal{S}, v \models \varphi$$

[10]The abbreviation ETL is used in [18], but it is not quite standard and used for other logics as well. Here, however, we shall use it only for epistemic temporal logic so that no confusion can arise.

hold? This is called model checking knowledge and time. We first collect some of the very fundamental results on model checking knowledge and time which help to appraise the expressiveness of the logical systems and the computational characteristics of formal reasoning about knowledge (and time).

First, model checking knowledge without any temporal dynamics means to check a formula of propositional modal logic on a finite Kripke structure. More precisely, the model checking problem for epistemic logic is stated as follows: Given a finite Kripke structure $\mathcal{K}$, a state k and an epistemic formula φ, does $\mathcal{K}, k \models \varphi$ hold? This problem can be solved using the usual labeling technique, cf. [78], which gives a time complexity that is linear in the size of the system and the size of the formula. Where the size of the system is defined as

$$||\mathcal{K}|| = |K| + |{\sim_0}| + \ldots + |{\sim_n}|.$$

Moreover, the size of the formula is its length $|\varphi|$ as a word. Notice that the complexity also depends on the number of atomic propositions. However, since only those propositions are relevant that actually occur in φ, this is already accounted for in $|\varphi|$.

Proposition 6.1. *See for example [78]*
The model checking problem for epistemic logic can be solved in time $O(||\mathcal{K}|| \cdot |\varphi|)$. If we add common knowledge, the problem can still be solved in polynomial time.

On the other hand, model checking time without knowledge means to evaluate LTL-formulas on finitely generated systems. This problem is well known to be PSPACE-complete.

Theorem 6.2. *See for example [215]*
LTL *model checking is* PSPACE*-complete.*

Now we consider both, knowledge and time. The results that we mention here are from the fundamental work of van der Meyden and Shilov [208] where they considered synchronous systems.[11] We start with the most expressive language.

Theorem 6.3. *[208]* ETL $+C$ *model checking is undecidable for synchronous two-agent systems.*

Van der Meyden and Shilov explain this undecidability intuitively as a result of the arbitrary reach through two orthogonal dimensions of the semantic structures, which the operators *until* and *common knowledge* facilitate. This explanation seems very plausible, in particular in view of the decidability of model checking for the logics that we get if we neglect at least one of the two operators.

Theorem 6.4. *[208]* ETL $-U+C$ *model checking is* PSPACE*-complete for synchronous systems.*

Without the until operator, we have just basic epistemic logic with common knowledge which is simple and, additionally, we have the next operator. This operator, however, can only occur finitely many times in any given formula and so, a formula φ of this restricted logic is only capable of looking at most $|\varphi|$ steps into the system.

Theorem 6.5. *[208]* ETL *model checking is decidable for synchronous systems but has non-elementary time complexity.*

Van der Meyden and Shilov provide two different proofs for this result. The first proceeds via a reduction of to chain logic with equal level predicate. This logic is obtained from the monadic logic of n successors SnS by constraining the set quantifiers to range only over chains, that is, sets that are totally ordered by the prefix relation, and adding the equal level predicate. Thomas has shown [201] that chain logic with equal level predicate is decidable on regular trees by a translation into S1S. The other proof is more direct and incorporates the knowledge-depth

[11]Notice that their notion of synchronous perfect recall coincides with our notion of synchronous observability.

of ETL-formulas, that is, the maximum nesting $d(\varphi)$ of knowledge operators that occurs in such a formula φ. The proof leads to a space complexity which is polynomial in $|\varphi| \cdot \exp_{d(\varphi)}(O(||\mathcal{S}||))$.

We give yet another proof of Theorem 6.5 that proceeds by a more direct translation of ETL-formulas into S1S-formulas and which we also adapt to not necessarily synchronous systems. This shows that, for any given ETL-formula φ and finite state system $\mathcal{S}$, the set $L(\varphi) \subseteq V^\omega$ of all runs π of $\mathcal{S}$ that satisfy $\mathcal{E}(\mathcal{S}), (\pi, 0) \models \varphi$ is ω-regular. In particular, Theorem 6.5 holds for not necessarily synchronous systems as well. However, the fact that $L(\varphi)$ is regular turns out to have also significant consequences on the strategy problem for games with ETL winning conditions, in particular, hierarchical games, cf. Section 6.1.4.

Now given a finite state system $\mathcal{S} = (V, \mathrm{Prop}, \Delta, (\sim_i^V))$, a formula $\varphi \in \mathrm{ETL}$ defines a set

$$L_{\mathcal{S}}(\varphi) = \{\pi \in V^\omega \mid \mathcal{S}, (\pi, 0) \models \varphi\}.$$

To prove that $L_{\mathcal{S}}(\varphi)$ is ω-regular, we construct from φ an equivalent S1S formula.

Let us recall the definition of S1S: For an alphabet Σ we consider the signature $\mathcal{T}_\Sigma = \{S, \min, <, (P_a)_{a \in \Sigma}\}$ and for an ω-word $\alpha \in \Sigma^\omega$ we consider the $\mathcal{T}_\Sigma$ structure

$$\underline{\alpha} = (\omega, S, \min, <, (P_a^\alpha)_{a \in \Sigma})$$

- S is the successor function on ω
- min and $<$ are as usual
- $P_a^\alpha = \{i \in \mathbb{N} \mid \alpha(i) = a\}$ for all $a \in \Sigma$

Now an $\mathrm{MSO}(\mathcal{T}_\Sigma) = \mathrm{S1S}$ sentence defines an ω-language

$$L(\varphi) = \{\alpha \in \Sigma^\omega \mid \underline{\alpha} \models \varphi\}.$$

We have already mentioned the result of Büchi from 1962, that an ω-language is regular if and only if it is S1S definable. Moreover, the translations from formulas into automata and vice versa are effective, cf. Section 2.2.

Before we demonstrate the translation from ETL formulas to S1S formulas, we state a simple technical lemma on S1S that we use in the proof.

Lemma 6.6. *Let $\varphi(\overline{x})$ be an* S1S *formula, $\alpha \in \Sigma^\omega$ and let X_a for $a \in \Sigma$ be set variables. Moreover, let $\mathcal{I} : \{X_a \mid a \in \Sigma\} \to \mathbb{N}$ be an interpretation for these variables and let $\beta \in \Sigma^\omega$ be defined as $\beta(\lambda) = a$ iff $\lambda \in \mathcal{I}(X_a)$. For any $\overline{u} \in \mathbb{N}^{|\overline{x}|}$:*

$$(\underline{\alpha}, \mathcal{I}, \overline{u}) \models \varphi[P_a/X_a](\overline{x}) \iff (\underline{\beta}, \overline{u}) \models \varphi(\overline{x}),$$

where $\varphi[P_a/X_a]$ denotes the formula that is obtained by replacing each occurrence of P_a in φ with X_a, for any $a \in \Sigma$.

Theorem 6.7. *For any finite state system $\mathcal{S}$ and formula $\varphi \in$* ETL*, we can effectively construct an* S1S*-sentence $\hat{\varphi}$ such that $L(\hat{\varphi}) = L_{\mathcal{S}}(\varphi)$.*

Proof. We construct, inductively over the structure of φ, an S1S formula $\hat{\varphi}(x)$ over the signature $\mathcal{T}_V$ such that for each system run π and all $t \in \mathbb{N}$ we have

$$\mathcal{S}, (\pi, t) \models \varphi \iff \underline{\pi} \models \hat{\varphi}(t).$$

For the LTL-part this translation is well known. We repeat the definition of $\hat{\varphi}$ for the base case and temporal operators as preparation for the more complicated case where knowledge operators are involved. Boolean combinations are trivial. For a formula $\theta(x)$ and a term t, the formula $\theta[x/t]$ is obtained from $\theta(x)$ by replacing (simultaneously) each free occurrence of x with t.

- $\varphi = P \quad \rightsquigarrow \quad \hat{\varphi}(x) = \bigvee_{v \in P} P_v x$
- $\varphi = X\psi \quad \rightsquigarrow \quad \hat{\varphi}(x) = \hat{\psi}[x/Sx]$
- $\varphi = \varphi_1 U \varphi_2 \quad \rightsquigarrow \quad \hat{\varphi}(x) = \exists y(x \leq y \wedge \hat{\varphi}_2[x/y] \wedge \forall z(x \leq z < y \rightarrow \hat{\varphi}_1[x/z]))$

It remains to construct $\hat{\varphi}(x)$ for $\varphi = K_i\psi$. First, assume that $\sim_i = \sim_i^*$ and let $\hat{\psi}(x)$ be the corresponding formula for ψ according to induction hypothesis. Then we have $\mathcal{S}, (\pi, t) \models K_i\psi$ if, and only if, for each run ρ with $\rho(\leq t) \sim_i^* \pi(\leq t)$ we have $\mathcal{S}, (\rho, t) \models \neg\psi$, so by induction hypothesis, $\mathcal{S}, (\pi, t) \models K_i\psi$ if, and only if, for each run ρ with $\rho(\leq t) \sim_i^* \pi(\leq t)$ we have $\underline{\rho} \models \neg\hat{\psi}(t)$.

The idea how to express this in S1S is to quantify over runs ρ by quantifying over *partitions* of $\mathbb{N}$ into sets X_v, where v ranges over V, such that the following holds. Whenever this partition constitutes a run ρ with $\pi(\leq t) \sim_i^* \rho(\leq t)$, then $\underline{\rho} \models \hat{\psi}(t)$. To check that $\pi(\leq t) \sim^* \rho(\leq t)$, we only have to compare the labels at corresponding positions within the two ω-words. Moreover, $\underline{\rho} \models \hat{\psi}(t)$ can be checked, according to Lemma 6.6, by replacing the atomic propositions P_v for $v \in V$ in $\hat{\psi}$ with the set variables X_v. So, we define $\hat{\varphi}(x)$ as follows, where v ranges over V, that is, we quantify over set variables $\{X_v \mid v \in V\}$.

$$\begin{array}{lll} \forall X_v \;\; [& & \forall y(\bigvee_{v \in V} (X_v y \wedge \bigwedge_{w \neq v} \neg X_w y)) \\ & \wedge & \forall y \forall z (Sy = z \rightarrow \bigvee_{(v,a,w) \in \Delta} X_v y \wedge X_w z) \\ & \wedge & \forall (y \leq x)(\bigwedge_{v \in V} (P_v y \rightarrow \bigvee_{w \sim_i v} X_w y)) \\ & \rightarrow & \hat{\psi}(P_v / X_v) \qquad]. \end{array}$$

Given the explanations above, it is easy to see that for all runs π and all $t \in \mathbb{N}$ we have $\pi, t \models \varphi$ if and only if $\underline{\pi} \models \hat{\varphi}(t)$.

On the other hand, in the asynchronous case, that means $\sim_i = \overleftarrow{\sim}_i^*$, we have $\mathcal{S}, (\pi, t) \models K_i\psi$ if, and only if, for each run ρ *and each* s with $\rho(\leq s) \overleftarrow{\sim}^* \pi(\leq t)$ we have $\underline{\rho} \models \hat{\psi}(s)$, where $s \neq t$ is possible: Between each two observations of agent i, there may be a priori unboundedly long sequences of states which are not observed by agent i. Due to this, it is not clear how to apply the above approach for defining the formula $\hat{\varphi}$ directly to the asynchronous case. Instead, we make a detour through automata. We also apply negation, in order to be able to use *nondeterministic* automata for the construction, which makes it somehow easier to deal with the unboundedly long sequences of nonobservable events. We describe the construction here only informally.[12]

Using the usual translation of S1S-formulas into automata, we obtain a nondeterministic Büchi-automaton $\mathcal{A}$ over the alphabet $V \times \{0, 1\}$ such that

$$L(\mathcal{A}) = \{\pi^\frown \alpha_t \in (V \times \{0,1\})^\omega \mid \underline{\pi} \models \neg\hat{\psi}(t)\},$$

where for $s \in \mathbb{N}$, $\alpha_t(s) = 1$, if $s = t$ and $\alpha_t(s) = 0$, if $s \neq t$. Notice that, by induction hypothesis, $L(\mathcal{A}) = \{\pi^\frown \alpha_t \in (V \times \{0,1\})^\omega \mid \mathcal{S}, (\pi, t) \models \neg\psi\}$.

The nondeterministic Büchi automaton $\mathcal{B}$, while reading $\pi^\frown \alpha_t$, nondeterministically guesses a run ρ and some s such that $\pi(\leq t) \overleftarrow{\sim}^* \rho(\leq s)$ and simulates $\mathcal{A}$ on $\rho^\wedge \alpha_s$. To handle the fact that $s \neq t$ is possible, reading $\pi^\frown \alpha_t$ and guessing $\rho^\frown \alpha_s$ must be asynchronous for the first t steps. More precisely, while *not* reading a letter $(v, 1)$, the automaton $\mathcal{B}$ skips all the nonobservable moves, that means $u \rightarrow v$ with $u \sim_i^V v$, which occur in π: it does not change the current state but only keeps track of the current equivalence class of states.

As soon as $\mathcal{B}$ reads some observable move in π, it extends ρ by a sequence of nonobservable moves (of possibly different length than the sequence which has occurred in π!) and inputs the previous observable move together with this sequence of private moves into the automaton $\mathcal{A}$.

[12]The rather lengthy technical description of the automaton can be found in the full version of [173], currently available at www.logic.rwth-aachen.de. Notice that the notation there is slightly different because the construction is tailored to game graphs with two players.

$$\begin{tikzcd}
v_1 \arrow[r, "a"] & \overset{P}{w_1} \arrow[loop right, "a"] \\
v_2 \arrow[r, "a"] \arrow[d, dotted, no head] & \overset{P}{w_2} \arrow[loop right, "a"] \\
v_3 \arrow[r, "a"] & w_3 \arrow[loop right, "a"]
\end{tikzcd}$$

Figure 6.1: A finite state system with two isomorphic runs

Finally, when $\mathcal{B}$ reads some letter $(v, 1)$, then it once more extends ρ by two sequences of private moves (before the position corresponding to v as well as after that position) and then inputs a corresponding letter $(w, 1)$ into $\mathcal{A}$. Afterwards, $\mathcal{B}$ does not have to take care of equivalence of prefixes of π and ρ anymore.

Extending the run ρ which $\mathcal{B}$ *guesses* by sequences of nonobservable moves is done similarly as in the construction in Section 3.5.4: We have transitions which imply finite sequences of nonobservable moves such that a corresponding infix of a run of $\mathcal{A}$ on ρ is guaranteed to exist and which is also guaranteed to visit F if some flag is set accordingly. A similar reasoning as in the proof of Proposition 3.32 shows, that $\mathcal{B}$ can be constructed effectively from $\mathcal{A}$ and $\mathcal{S}$: All conditions determining the membership of a transition to $\Delta^{\mathcal{B}}$ are regular properties of finite histories in $\mathcal{S}$ and can thus be transformed into finite automata.

It is not hard to see, that $L(\mathcal{B}) = \{\pi^\frown\alpha_t \,|\, \mathcal{S}, (\pi, t) \models \neg K_i\psi\}$. Finally, using the usual converse translation of Büchi automata into S1S formulas, we obtain an S1S-formula $\theta(x)$ such that for all runs $\pi \in V^\omega$ and all $t \in \mathbb{N}$ we have $\underline{\pi} \models \theta(t)$ if, and only if, $\pi^\frown\alpha_t \in L(\mathcal{B})$. That means, $\underline{\pi} \models \neg\theta(t)$ if, and only if, $\mathcal{S}, (\pi, t) \models K_i\psi$, giving us the desired formula $\hat{\varphi}(x) = \neg\theta(x)$. $\square$

Uniform definability. Notice that this result does *not* yield *uniform* S1S-definability of ETL-specifications: If φ is an ETL-formula over unary predicates $(P_a)_{a\in\Sigma}$ for some finite set Σ, then Theorem 6.7 does not give us a monadic second order sentence $\hat{\varphi}$ over the signature $\mathcal{T}_\Sigma = \{S, \min, <, (P_a)_{a\in\Sigma}\}$, such that for *any* finite state system $\mathcal{S}$ we have $L_{\mathcal{S}}(\varphi) = L(\hat{\varphi})$. Even more, it is obvious, that such a formula $\hat{\varphi}$ does not exist in general, since the semantics of S1S is completely determined by the word models $\underline{\pi}$ and does not additionally access the underlying system $\mathcal{S}$. So if the alphabet Σ over which the word models are defined must be independent of $\mathcal{G}$, there is no way to define knowledge operators:

Consider the ETL-formula $\varphi = K_0FP$ over Prop $= \{P\}$ which expresses that agent 0 knows that at some point in the future (including now), P will hold and consider the system $\mathcal{S}$ depicted in Figure 6.1. Moreover, for $\lambda = 1, 2, 3$ let $\pi_\lambda = v_\lambda w_\lambda^\omega$, Obviously, $\mathcal{S}, (\pi_1, 0) \models \varphi$ but $\mathcal{S}, (\pi_2, 0) \not\models \varphi$, since $\pi_3(\leq 0) \sim_0^* \pi_2(\leq 0)$ and $\mathcal{S}, (\pi_3, 0) \not\models FP$. But for any S1S sentence φ over $\tau = \{S, \min, <, P\}$ we have $\underline{\pi_1} \models \varphi$ if, and only if $\underline{\pi_2} \models \varphi$: since $\underline{\pi_1}$ and $\underline{\pi_2}$ are isomorphic τ-structures, they cannot be distinguished by any S1S(τ)-sentence.

From Theorem 6.7 we can now easily derive that model checking for ETL is also decidable for not necessarily completely synchronous systems.

Corollary 6.8. ETL *model checking is nonelementary decidable.*

Remark. The translation from Theorem 6.7 also works if we consider a mixed ETL-language, where we have two knowledge operators K_i and $\overleftarrow{K}_i$ for each agent, where K_i has synchronous semantics and $\overleftarrow{K}_i$ has asynchronous semantics. That means, ETL definable specifications are still ω-regular if we can refer to the synchronous and the asynchronous view of each agent within the same formula.

6.1.3 Digression: Dynamic Epistemic Logic

Another approach to describing epistemic dynamics in a formal logical language is *dynamic epistemic logic* DEL. In contrast to epistemic temporal logic which is evaluated on the unfolding of a system, formulas of dynamic epistemic are evaluated on Kripke structures that represent a static epistemic situation and the dynamic aspect is captured by the operators of the logic which incorporate so called *epistemic actions.*[13] In the following, an *epistemic structure* is a Kripke structure where each alternative relation is an equivalence relation. Intuitively, an epistemic action can be any kind of action that may happen in a given epistemic situation which only affects the *knowledge* of the players, for example, revealing a card in a card game, privately answering a question or making a public announcement about certain facts (atomic propositions) that are included in the formal description of the situation.

Formally, dynamic epistemic logic is obtained from basic epistemic logic by allowing formulas of the form

$$[\alpha]\varphi,$$

where α is an epistemic action and φ is a formula. The meaning of the formulas is: After the execution of the epistemic action α, the epistemic formula φ holds. For example, after the public announcement that player 0 holds the blue card, the fact that player 0 holds the blue card is common knowledge among all players. Formally, the semantics of $[\alpha]\varphi$ is defined as

$$\mathcal{K}, v \models [\alpha]\varphi \iff (\mathcal{K}, v)[\alpha] \models \varphi,$$

where $(\mathcal{K}, v)[\alpha]$ is the epistemic structure that is obtained from $\mathcal{K}, v$ by applying the epistemic action α. We do not introduce epistemic actions and the epistemic update $(\mathcal{K}, v)[\alpha]$ formally. Essentially, α is an epistemic model as well and $(\mathcal{K}, v)[\alpha]$ is the product of $(\mathcal{K}, v)$ and α. So, roughly speaking, the knowledge of the agents in $(\mathcal{K}, v)[\alpha]$ is the knowledge in $(\mathcal{K}, v)$, *relativized* to the knowledge in $[\alpha]$.

The development of dynamic epistemic logic originates in logics for analyzing *public announcements*, a special form of epistemic actions, and was partly inspired by dynamic modal logics like PDL and the modal μ-Calculus, as well as epistemic temporal logics. Early fundamental work includes the papers by Plaza [166] as well as Gerbrandy and Groeneveld [97]. The development of dynamic epistemic languages which incorporate more general epistemic updates started mainly in the late 90's with works of Baltag, Moss and Solecki [10], Gerbrandy [95], van Ditmarsch [69] and others. We refer to [209] for an excellent presentation and comprehensive overview over the different incarnations and aspects of dynamic epistemic logic.

Here we just mention some fundamental properties of dynamic epistemic languages that help to appraise the expressiveness of the languages. First, basic epistemic logic with epistemic actions[14] is no more expressive than basic epistemic logic [209]. That means, for basic epistemic logic, updates with epistemic actions are just syntactic sugar. This changes, as soon as we add *common knowledge*: Dynamic epistemic logic with common knowledge is more expressive than epistemic logic with common knowledge but can still be translated into propositional dynamic logic PDL and, hence, into the (two-variable fragment of the) modal μ-Calculus [121, 210, 19], see also [209]. On the other hand, once we allow to *iterate* relativization (by action models) then the satisfiability problem for the resulting logic becomes undecidable [146] and therefore, the logic is not a fragment of the modal μ-Calculus anymore.

It is important to observe that dynamic epistemic logic, as we have considered it here, really talks about *epistemic* changes only. That means, after an update with some epistemic action, the *knowledge* of the players may have changed, but the factual situation remains unchanged. In

[13]In fact, we mix up the terminology of dynamic epistemic logic a little: What we call an epistemic action is usually referred to as action model while epistemic actions establish a different language, cf. [209]. However, we do not consider these kind of epistemic actions so, since action models are intuitively epistemic actions we well, we like to refer to them as such.

[14]Defined by *finite* action models.

contrast, the temporal operators of ETL allow to quantify over possible time points in the future, at which factual change may have occured very well. In [19], a logic has been investigated that is capable of expressing epistemic dynamics but also incorporates factual change. In [18], an extensive study of the relationship between epistemic temporal logic and dynamic epistemic logic has been carried out.

6.1.4 Reasoning About Knowledge in Games

In Section 6.1.1 we have studied formal methods for reasoning about knowledge in systems. There, it didn't matter whether the system was actually a game or a simple message passing system or maybe a formal model for some sort of bargaining situation: We simply took the set of all possible runs of the system and we let epistemic temporal formulas talk about the agents state of minds throughout these runs. However, when it comes to games, we are particularly interested in *strategies*. We will see that as soon as strategic dependencies are incorporated, reasoning about knowledge becomes much more cumbersome. Of course, we cannot cover all facets of this intriguing subject. We just give some main ideas and intuitions that help to appreciate the complexity of knowledge in games and which serve as a preparation for our construction in Section 6.2.

Strategic Dependencies. Let us start with the task to synthesize a controller for a system that is specified by an ETL formula. We might proceed as follows: We take a game graph

$$\mathcal{G} = (V, \Delta, (\sim_i^V))$$

with partial information that models the system and we have a set Prop of atomic propositions where we assume that $\mathcal{G}$ brings along an interpretation of the atomic propositions, explicitly represented as subsets of V. Our approach would be to synthesize a joint winning strategy for the grand coalition for the game played on $\mathcal{G}$ where the winning condition is $W = L_{\mathcal{G}}(\varphi)$. This, however, ignores one crucial point: $\mathcal{E}(\mathcal{G}), (\pi, t) \models K_i\psi$ holds if $\mathcal{E}(\mathcal{G}), (\rho, s) \models \psi$ for *all* $(\rho, s) \sim_i (\pi, t)$. So we do not take into account the fact that once the grand coalition follows some joint winning strategy σ, the whole system will be relativized by σ, that means, only plays occur which are consistent with σ.

But does this actually mean that the semantics of the knowledge operators should be relativized to consistent plays as well? Intuitively, we would clearly assume that any player knows his own strategy. Moreover, throughout the previous chapters we have repeatedly insinuated that each player *knows* the strategies of all other players. Now, for the construction of a joint winning strategy this assumption is perfectly appropriate because the formulation of the strategy problem includes the possibility of the players to *coordinate* on the joint strategy: The value of a strategy σ_i for player i is only relevant on histories which are already consistent with the joint winning strategy, so in this sense, player i can rely on the other players playing the strategies that they all have agreed on. This entails that a player i which is better informed than a player j, can predict the actions of player j. We have used this fact excessively for the construction of winning strategies, cf. Section 2.1.3, Chapter 3 and Chapter 5.

On the other hand, if we are concerned with an explicit representation of the players states of mind, then we should revise the expression of the players *knowing* each others and, in particular, their own strategies. In fact, to *rely* on a strategy being played is not the same as *knowing* that a strategy is actually played, cf. Figure 6.2. There we use a turn based notation: As usual, circle positions belong to player 1, while square positions belong to player 2. The dotted lines indicate the indistinguishabilities of player 1, while player 2 has full information. Position goal is the goal. It is obvious that player 1 and player 2 can coordinate on a joint winning strategy: Player 2 chooses action l and, despite the fact that player 1 does not observe this action, relying on this choice of player 2, player 1 chooses action a. On the other hand, given the history $\pi = 0\,l\,1$, the question whether player 1 *knows* that he is in position left depends on how we model the knowledge of player 1.

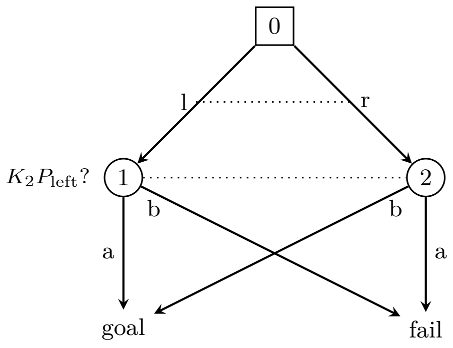


Figure 6.2: Knowing strategies vs. relying on strategies

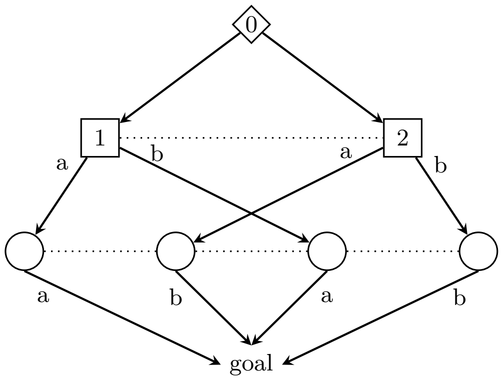


Figure 6.3: Observing actions is not sufficient for deducing strategies

For example, a controller in a reactive system cannot be actually said to realize what it is doing but it just sends and receives signals. Recall that we have not postulated that a player can distinguish any two of its own actions in general. (In fact, in this chapter we use a model where this is *not* the case.) We have merely mentioned that for the strategy problem this will make no difference because for constructing a strategy we can *rely* on the fact, that the actions which are carried out are those that the strategy prescribes. So, if we hide its own actions from a player then it is not clear what we mean by saying that the player *knows* its own strategy, even less the strategies of the other controllers. Here it also interesting to mention that *observing* the actions of the other players in the game is *not* sufficient for knowing their strategies, cf. Figure 6.3. We use the same notation as in Figure 6.2 and, additionally, diamond positions belong to the environment player 0. Again, player 1 and player 2 can easily coordinate on a joint winning strategy: At position 1, player 2 chooses action a and at position 2, player 2 chooses action b. Player 1 then just needs to copy the action of player 2. On the other hand, player 1 cannot deduce the strategy that player 2 uses from merely observing his actions: If he observes action a then, since he does not know whether the game was in 1 or 2 in the previous step, he does not know whether player 2 chooses action a at 1 or at 2 (or at both).[15]

So rather than assuming that the controllers actually *know* each others strategies, we might define the knowledge of a controller as exactly that what can be deduced from the information which it has received explicitly. Basically, this is a design decision that depends on the situation

[15]On the other hand, observing his own actions is sufficient for a player to know his *own* strategy: If $\pi \sim_i^* \rho$ and player i is able to observe all his actions, then π is consistent with σ_i if, and only if, ρ is.

at hand. However, as we will see, it has significant consequences.

First, let us see how we should model the strategy problem for ETL-specifications if we assume that each player actually *knows* all the strategies that are being used. We would then have to ask, given a game graph $\mathcal{G}$ with partial information and an ETL-formula φ, is there a joint winning strategy σ for the grand coalition such that

$$\mathcal{E}(\mathcal{G}, \sigma) \models \varphi?$$

The multiagent system $\mathcal{E}(\mathcal{G}, \sigma)$ is now defined with respect to σ, that means, it contains only histories which are consistent with σ and therefore, the knowledge operators are relativized to the joint winning strategy. Let us call this the realizability problem for φ, following the terminology in [143] and distinguishing it clearly from the strategy problem with winning condition $L_{\mathcal{G}}(\varphi)$, where the knowledge operators are not relativized to strategies.

In [143] it has been shown that for two-player games and ETL-specifications that contain only the synchronous knowledge operator K_1 for player 1, this problem is decidable.

Theorem 6.9. *[143] The realizability problem for* ETL *specifications using only the synchronous knowledge operator for player* 1 *is decidable for games with two players and can be solved in doubly exponential time.*

Notice that if we assume that player 1 can distinguish any two of his own actions, then the strategy problem for two-player games with ETL winning conditions and the realizability problem for ETL specifications *coincide* because then, for any strategy σ of player 1 and all histories π, ρ with $\pi \sim_1^* \rho$, we have that π is consistent with σ if, and only if, ρ is consistent with σ. Therefore, relativization of the knowledge operators to σ is without effect.

If we do not make this assumption, decidability can still be easily obtained from the methods that we have developed so far. Theorem 6.7 yields a (nonelementary) translation into an ω-regular winning condition and for two-player games with ω-regular winning conditions we have plenty of possibilities to solve the strategy problem.[16] The doubly exponential time complexity (which is actually singly exponential in the size of the system and only doubly exponential in the length of the formula) cannot be obtained that directly but on closer inspection one finds that it holds as well. We do not discuss any details here but just draw the conclusion that, in the two-player case, the particular choice of knowledge semantics is not important for the computational properties of the strategy problem. In the multiplayer case, the situation is completely different: Theorem 6.7 still yields that any ETL-definable winning condition is ω-regular and we already know that the strategy problem for hierarchical games with ω-regular winning conditions is decidable.

Theorem 6.10. *The strategy problem games on finite graphs with partial information and* ETL*-definable winning conditions is decidable.*

On the other hand, van der Meyden and Wilke have shown that the following result holds:

Theorem 6.11. *[144] The realizability problem for* ETL *specifications using only the synchronous knowledge operator for player* 1 *is undecidable for games with three players.*

This also remains true with even stronger restrictions. In particular, player 2 can be assumed to have full information while player 1 can be assumed to be blind. Moreover, the strategy of player 1 may be fixed so that only the strategy of player 2 has to be actually synthesized. So, as soon as we have three players and we assume that player 1 *knows* the strategy of player 2, the problem becomes undecidable. This reveals a fundamental difference between the fact that each player of the grand coalition can *rely* on the strategies which his companions are using and making this explicit by assuming that each player of the grand coalition *knows* these strategies.

[16]Notice that once we have an ω-regular winning condition at hand, we can assume w.l.o.g. that player 1 can distinguish any two of his own actions.

Winning Conditions. We have seen that reasoning about knowledge becomes tricky if strategic dependencies come into play because those dependencies generate a sort of implicit knowledge on the possible histories and the question whether or not we make this knowledge explicit has huge impact on the computational properties of the strategy problem for ETL specifications.

As soon as we incorporate winning conditions, reasoning about knowledge becomes even more intricate, because winning conditions generate a similar form of implicit knowledge: Consider a history π such that either each extension of π to a play in $\mathcal{G}$ is won by the grand coalition or each such extension is lost by the grand coalition. Then any player i of the grand coalition can completely ignore π, even if player i consider π possible, that is, $\pi \sim_i^* \rho$, where ρ is the history that has been played so far. The reason is just as for histories which are not consistent with the joint strategy of the grand coalition: Player i can define his strategy on π arbitrarily as long as it is still constant over equivalent histories (which is just a matter of sensible choice here).

Now in this case, one would certainly tend to keep this knowledge of the players implicit, because histories where the winner is already determined are still very much possible. So it seems not reasonable to say that player i *knows* that an atomic proposition P holds if $\text{last}(\rho) \in P$ but $\text{last}(\pi) \notin P$. On the other hand, the construction that turns a reachability condition $R \subseteq V$ on some game graph $\mathcal{G}$ with partial information into an *observable* reachability condition does exactly this: It turns the implicit knowledge of the players about *relevant* histories into explicit knowledge about *possible* histories. So while this construction is certainly useful in view of the strategy problem, it should be considered carefully when we are interested in the actual knowledge of the players. Notice that, while observably decomposable winning conditions may still generate implicit knowledge in general, observable winning conditions, of course, don't.

Higher Order Knowledge. We already know that in general, infinitely many mental states of the grand coalition may arise in a multiplayer game. This can be seen, for example, from the fact that finite memory is not sufficient to implement winning strategies for the grand coalition in safety games with two players which are incomparably informed, cf. Chapter 5. The reason why we need unboundedly large epistemic structures to represent the mental states of the grand coalition is the higher order knowledge, that means, knowledge about knowledge about knowledge .... Let us start with a small example where the strategic dependencies on second order knowledge become evident. Notice that player 1 has *full* information in the example.

Consider the game graph depicted in Figure 6.4. We use notation as in Figure 6.3. Although player 2 has full information and the game has a reachability winning condition, player 2 needs to know something beyond the current position: The positions that player 1 considers possible! If the game is in y then player 2 needs to know whether the game was previously in position 1 or 2. If the game was in 2, then player 1 knows that x is *not* the actual position *and*, remembering that the previous position was 2, player 2 does know that. Analogously, if the game was in 1. Then, player 2 can *signal* player 1 the actual position by choosing action b if the game was in 1 and choosing action a if the game was in 2.

It is also easy to see that the higher order knowledge of the players can evolve unboundedly in games with at least two incomparably informed players. However, to see that an unbounded amount of higher order knowledge has to be taken into account in the *strategic* reasoning, we have to have a closer look at such a situation. In particular, we have to take into account the winning condition! The following epistemic analysis was inspired by an example from [30]. The relevant part of the corresponding extensive game with imperfect information is depicted in Figure 6.5. The relations $\sim_1^*$ and $\sim_2^*$ on the histories of the game tree are indicated by dotted and dashed lines, respectively.

Consider the distributed system $\mathcal{D}_2$ with two incomparably informed controllers, cf. Section 3.2.1. The undecidability proof of Theorem 5.11 can be directly adapted to $\mathcal{D}_2$. In fact, it gets even easier: The two controllers p_1 and p_2 directly receive the signals $\rightsquigarrow$ and $\curvearrowright$ from the environment and they do not observe which signals the other controller receives.

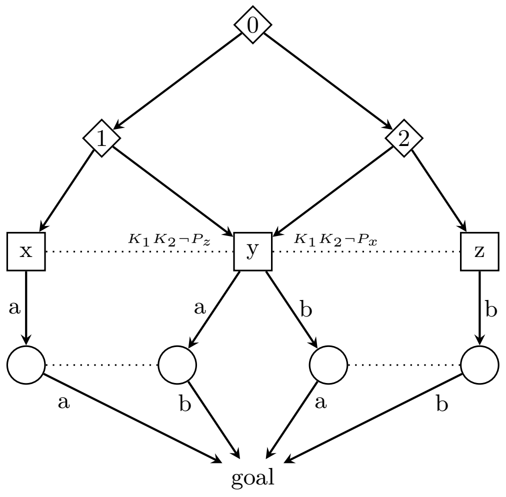


Figure 6.4: Higher order knowledge in a simple interactive situation

Assume that neither p_1 nor p_2 have received $\curvearrowright$ in the first step. Clearly, p_1 considers it possible that p_2 has received $\curvearrowright$ in the first step. (Also notice that p_1 *knows* that p_2 considers it possible that p_1 has received $\curvearrowright$ in the first step.) Now assume that neither p_1 nor p_2 have received $\curvearrowright$ in the second step as well. Then p_1 considers it possible that p_2 has received $\curvearrowright$ in the first or in the second step, but the possibility that p_2 has received $\curvearrowright$ in the first step is no longer relevant, because there, the controllers have already won. In the following, let us treat this as if p_1 actually *knows* that p_2 did not receive $\curvearrowright$ in the first step.

However, and now it gets tricky, although p_2 itself knows very well that it has not received $\curvearrowright$ in the first step, p_2 does *not* know that p_1 knows this. The reason is that p_2 considers it possible that p_1 has received $\curvearrowright$ in the *second* step and in this situation, p_1 still considers it possible that p_2 has received $\curvearrowright$ in the *first* step. In fact, if some oracle (like the father in the muddy girls puzzle, cf. [78]) told p_2 that p_1 knows that p_2 has not received $\curvearrowright$ in the first step, then p_2 would *know* that p_1 has received $\rightsquigarrow$ in the first two steps. But given the objective of the players as specified by the winning condition, this information must clearly be concealed from p_2. Therefore, although both players know that p_2 has not received $\curvearrowright$ in the first step, this possibility can still not be dismissed from a full representation of the mental state of the controllers![17]

This effect is now propagated from stage to stage. In the next step, the reasoning has to be taken to the next level of the knowledge hierarchy: p_1 considers it possible that p_2 considers it possible that p_1 considers it possible that p_2 has received $\curvearrowright$ in the first step. Therefore, although both players know that p_2 has not received $\curvearrowright$ in the first step *and* both players know that both players know this, this possibility can still not be dismissed from a full representation of the mental state of the controllers: If p_1 knew that p_2 knows that p_1 knows that p_2 as not received $\curvearrowright$ in the first step, then p_1 also knew that p_2 has received $\rightsquigarrow$ in the first three steps. Therefore, we have to use unboundedly large (and hence infinitely many) epistemic structures for a full representation of all possible mental states of the grand coalition.

We have already mentioned that if infinite memory is needed to implement winning strategies in some game with a safety winning condition then the higher order knowledge of the players

[17]Notice that what we have described here seems like higher order *uncertainties* rather than higher order *knowledge*. However, from a logical point of view, these notions are just dual to each other ($P_i\varphi :\equiv \neg K_i\neg\varphi$), so we stick with the term of higher order knowledge.

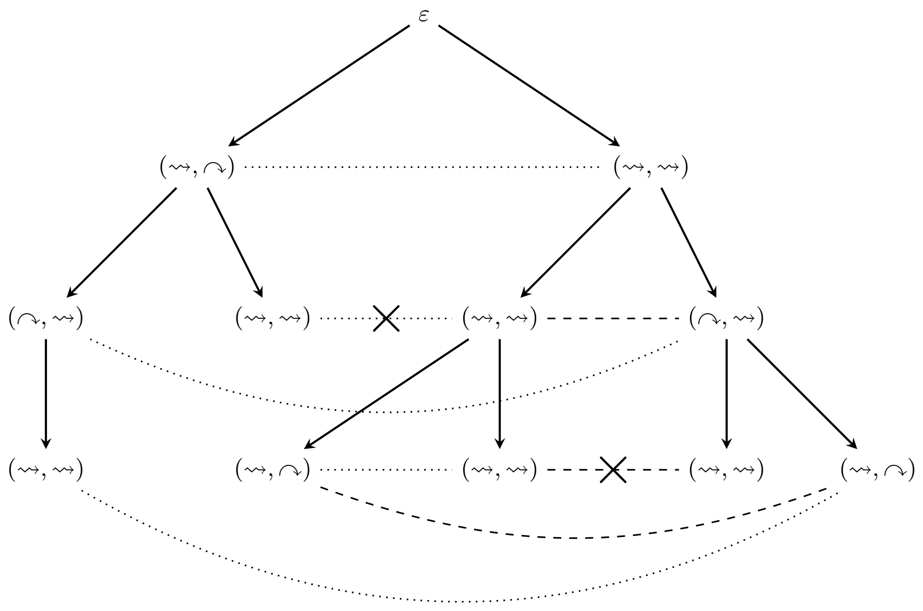


Figure 6.5: Unbounded Evolution of Higher Order Knowledge

necessarily evolves unboundedly. The reason is, roughly speaking, as follows: If this was not the case then there would be only finitely many possible mental states of the grand coalition which we could represent explicitly using our knowledge tracking construction from Section 6.2. Then we could synthesize a positional winning strategy in the resulting game with full information which yields a finite state winning strategy for the original game with imperfect information which is not possible. It is important to realize, however, that the converse is *not* true in general: If we consider a trivial winning condition for the system $\mathcal{D}_2$ like $W = (\Sigma^{\mathcal{D}_2})^\omega$ then no memory is needed to implement winning strategies for the resulting game with imperfect information. However, the higher order knowledge of the players still evolves in the very same way as before. The reason is that the knowledge of the players about the relevant plays according to the winning condition (in this case: none) is *implicit*, as discussed above. In order to downscale the evolution of knowledge in this case, we would have to make this implicit knowledge explicit.

Finally, we would like to comment on the notion of *common knowledge* in games. Due to the absence of broadcast channels, common knowledge can be achieved in *distributed systems* only about more or less trivial facts (like for example that the last event in the system has been some action from $\Sigma^{\mathcal{D}}$). On the other hand, in game graphs with partial information we can have public variables: Assume that the positions have the form $v = (v_0, v_1, \ldots, v_n, w)$ where v_i is private to player i and w is public. Then the value of w is always common knowledge! Now, in view of our translation of game graphs with partial information into distributed systems one might wonder what kind of black hole absorbs this common knowledge if we apply the translation. Well, it is the winning condition: The translation requires to encode the structure of the game graph into the specification and the environment distributes the visible portions of the current positions to the respective individual processes. However, the processes have no way of *knowing* whether the environment does this faithfully. In fact, none of the processes *knows* the value of w at any time. On the other hand, if the value of w which a process p_i has received is incorrect then the current history is *irrelevant.* Hence, taking into account the implicit knowledge that the winning condition generates, in any situation where the environment has not lost yet, the value of w is common knowledge again. So in view of the *knowledge* of the players, the translation of game graphs into distributed systems is also critical.

6.2 Knowledge Tracking for Multiplayer Games

We now tackle the task of developing a procedure that turns a game graph with partial information into one with full information and allows for a direct and uniform translation of strategies. So the new game graph must somehow comprise all the possible states of minds that the grand coalition may have during some play of the game as well as the dynamics of these mental states. We already know that there infinitely many such states in general, so we cannot hope to find a construction that guarantees a finite representation.

In Section 6.1.2 we have used the system $\mathcal{E}(\mathcal{G})$ to evaluate epistemic temporal properties of the game, so this system should be a sincere representation of all the knowledge that may be important throughout the game. However, this representation is still not explicit enough: The game that is played on $\mathcal{E}(\mathcal{G})$ is the corresponding *extensive* game of *imperfect* information, cf. Section 2.1. The positions in the game tree are histories in $\mathcal{G}$ and the possible states of minds of the grand coalition can only be carved out by considering unboundedly many positions in the game tree at the same time.

Instead, each *single* position in the new game graph should comprise an entire possible mental state of the grand coalition, that means, a snapshot of what players know at some stage of the game. Then, the limited strategic powers of the members of the grand coalition are also encapsulated at each such position. The main idea of our construction is to start with an extensive representation where we gather *all* histories, which *some* player of the grand coalition considers possible after some finite history has been played, in Kripke structures called *epistemic models*. Moreover, we incorporate the strategic dependencies in the game *explicitly* into these epistemic models, that means, the representation of the mental states of the grand coalition indeed makes the implicit knowledge of the players about each others strategies *explicit*!

Clearly, the new game graph is one of full information, but is infinite by definition. The next step is to find a more succinct representation. For this, we define an abstraction based on homomorphic equivalence which we prove to be sound for games with *observable* winning conditions. Moreover, we show that for *hierarchical* games, this abstraction is strong enough.

6.2.1 General Knowledge Tracking

Before we get started, let us introduce a little example that we will use to illustrate our construction and to discuss the problems that arise when we try to apply the quotient modulo homomorphic equivalence to games with nonobservable winning condition. Essentially, the game graph $\mathcal{G}_{\|}$ is obtained by translating the distributed system $\mathcal{D}_2$ into a game graph with partial information using the construction described in Section 2.3. In particular, the strategy problem is undecidable for $\mathcal{G}_{\|}$, even for safety winning conditions, cf. Chapter 5.

The game graph is depicted in Figure 6.6. Here, we use concurrent notation, that means, the players make their moves simultaneously. The game starts at position x where the players of the grand coalition have only trivial moves $\perp$ and the environment chooses a letter from $\{a, b\}$ and a digit from $\{0, 1\}$. The label of the successor position reflects this choice. Player 1 only observes whether the environment has chosen a or b, whereas player 2 observes whether it was 0 or 1. (The relations $\sim_1^V$ and $\sim_2^V$ are represented by dotted and dashed lines, respectively.) Next, player 1 chooses a letter from $\{a, b\}$ and player 2 a digit from $\{0, 1\}$, again reflected by the label of the successor position. After that, the game returns to x.

Epistemic Models. To describe the mental states of the grand coalition, that is, the knowledge acquired by its members during a play, we use epistemic models. An *epistemic model* over a game graph $\mathcal{G}$ is a Kripke structure

$$\mathcal{K} = (K, (P_v)_{v \in V}, (\sim_i)_{i=1}^{n})$$

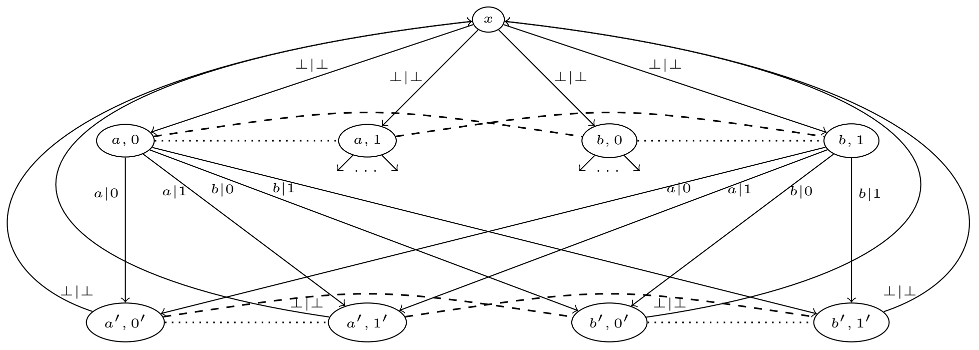


Figure 6.6: The game graph with partial information $\mathcal{G}_{\|}$.

where $(P_v)_{v\in V}$ is a partition of K and each $\sim_i$ is an equivalence relation on K such that, for all $k, k' \in K$:

$$k \sim_i k' \quad \Longrightarrow \quad v_k \sim_i v_{k'}$$

with v_k denoting the unique element from V such that $k \in P_{v_k}$.

We assume that $\mathcal{K}$ is connected by $\sim_\cup^n = \bigcup_{i=1}^n \sim_i$, except when indicated otherwise. Notice that $\sim_\cup^n$ is not necessarily an equivalence relation.

Epistemic Unfolding. When unfolding a game $\mathcal{G}$, we will keep track of the knowledge of the grand coalition in epistemic models. Thus, the states of the unfolding are epistemic models over $\mathcal{G}$. Since we assume that all players know the initial position, the initial epistemic model is just a one-element structure:

$$\mathcal{K}_0 = (\{v_0\}, (P_v)_{v\in V}, (\sim_i)),$$

where $P_{v_0} = \{v_0\}$, $P_w = \emptyset$ for $w \neq v_0$, and $\sim_i = \{(v_0, v_0)\}$.

Now, given a state of the unfolding represented by an epistemic model $\mathcal{K} = (K, (P_v)_{v\in V}, (\sim_i))$, what are the possible actions of the players of the grand coalition and what kind of knowledge dynamics will the execution of such an action imply? As we have mentioned, we *relativize* the knowledge of the players by their *possible* strategies, that is, we let the players choose a joint action in *each* possible world $k \in K$ which results in a tuple $(a_k)_{k\in K}$ of joint actions $a_k \in A$. Each possible next epistemic model will be defined with respect to such a macro action. Of course, we have to respect the limited strategic abilities of the members of the grand coalition in the original game. To capture this, we allow only tuples $(a_k)_{k\in K}$ which are consistent with the players' knowledge, that means, for all $i < n$ and all $k, k' \in K$:

$$k \sim_i k' \quad \Longrightarrow \quad (a_k)_i = (a_{k'})_i.$$

We define the, possibly disconnected, epistemic model

$$\text{Update}(\mathcal{K}, (a_k)_{k\in K}) := (\tilde{K}, (\tilde{P}_v)_{v\in V}, (\tilde{\sim}_i)_{i=1}^n)$$

- $\tilde{K} = \{kv \,|\, k \in K, k \in P_w \text{ and } (w, a_k, v) \in \Delta\}$,
- $\tilde{P}_v = \{kv \,|\, kv \in \tilde{K}\}$,
- $kv \;\tilde{\sim}_i\; k'v' \;\;:\Longleftrightarrow\;\; k \sim_i k' \text{ and } v \sim_i^V v'$.

The set of epistemic successor models $\text{Next}(\mathcal{K}, (a_k)_{k\in K})$ consists of the $\sim_\cup^n$-connected components of $\text{Update}(\mathcal{K}, (a_k)_{k\in K})$. The moves of the environment player 0 in the epistemic unfolding will consist of *choosing* one of the epistemic successors in $\text{Next}(\mathcal{K}, (a_k)_{k\in K})$.

To unfold a game $\mathcal{G}$ and track the knowledge with epistemic models, we start with the initial structure $\mathcal{K}_0$ as above and consider all possible joint actions a the players can take. We get the epistemic models $\mathrm{Next}(\mathcal{K}, a)$ as next states, and continue the unfolding from there, considering all possible macro actions. With this dynamic process in mind, we give the following declarative definition.

The *epistemic unfolding* of a game graph $\mathcal{G} = (V, \Delta, (\sim_i^V))$ is a game graph with *full* information

$$\mathrm{Tr}(\mathcal{G}) = (V^t, \Delta^t)$$

- V^t is the set of all epistemic models $\mathcal{K}$ over $\mathcal{G}$ with $K \subseteq V^*$
- $\Delta^t = \{(\mathcal{K}, (a_k)_{k\in K}, \mathcal{K}') \mid (a_k)_{k\in K} \in A^{|K|} \text{ and } \mathcal{K}' \in \mathrm{Next}(\mathcal{K}, (a_k)_{k\in K})\}$

If $W \subseteq V^\omega$ is a winning condition for $\mathcal{G}$, similar as for the generalized powerset construction in Section 3.5.1, the winning condition of $\mathrm{Tr}(\mathcal{G})$ requires that all paths through the sequence of Kripke structures are winning in the original game: For a play $\pi^t = \mathcal{K}_0\mathcal{K}_1\ldots$ in $\mathrm{Tr}(\mathcal{G})$, a *path through* π^t is a sequence $\rho = k_0k_1\ldots$ such that

- $k_l \in \mathcal{K}_l$
- $k_{l+1} = k_l v$ for some v with $(v_{k_l}, a, v) \in \Delta$ for some a

For such a path ρ, the corresponding play $\pi_\rho = v_0v_1\ldots$ in $\mathcal{G}$ is defined by

$$v_j = v_{k_j} \quad \text{for all } j \in \mathbb{N}.$$

Notice that the definition of a *path* through π^t implies that π_ρ is indeed a play in $\mathcal{G}$. The set $\Pi(\pi^t) \subseteq V^\omega$ is the set of all plays in $\mathcal{G}$ that correspond to some path through π^t. Now,

$$\pi^t \in W^t \iff \Pi(\pi^t) \subseteq W.$$

Given that the epistemic models in $\mathrm{Tr}(\mathcal{G})$ actually consist of histories in $\mathcal{G}$, the definition of the winning condition W^t may seem a little complicated. Indeed, $\Pi(\pi^t)$ is the set of all plays in $\mathcal{G}$ such that $\pi(\leq l) \in \mathcal{K}_l$ for all $l \in \mathbb{N}$. However, the more abstract definition that uses only the worlds $k_l \in \mathcal{K}_l$ and the atomic propositions P_v is useful when we consider the more succinct representation of epistemic unfolding modulo homomorphic equivalence.

Notice that W^t is now position based. Of course, prerequisite for this is the fact that W is position based. Moreover, since the players of the grand coalition have no knowledge about the actions in the game whatsoever, the winning condition is also independent of the actions of the environment (which are only implicit in this model anyway).[18]

The reason why W^t is also independent of the (joint) actions of the grand coalition, on the other hand, is that the histories that are contained in the epistemic models are *relativized* to strategies of the grand coalition. So, for any play π^t in $\mathrm{Tr}(\mathcal{G})$ there is a joint winning strategy for the grand coalition such that any play $\pi \in \Pi(\pi^t)$ is consistent with this strategy, see also the proof of Proposition 6.12.[19]

As in the case of the generalized powerset construction, observable winning conditions W can be described more explicitly. For simplicity, we assume that W is given by a pair (col, W_o) where $\mathrm{col} : V \to C$ is an observable coloring of V for some finite set $C \subseteq \mathbb{N}$ and $W_o \subseteq C^\omega$ is a set of good sequences of colors. Since epistemic models are $\sim_\cup^n$-connected, the coloring col is constant over all worlds of a position $\mathcal{K} \in \mathrm{Tr}(\mathcal{G})$ and we write $\mathrm{col}(\mathcal{K})$ for this color. Then,

$$\mathcal{K}_0\mathcal{K}_1\ldots \in W^t \iff \mathrm{col}(\mathcal{K}_1)\mathrm{col}(\mathcal{K}_2)\ldots \in W_o.$$

Just as in the case of the generalized powerset construction, this property follows by König's Lemma for finite game graphs but does *not* hold in general for infinite game graphs as there may be infinite plays in $\mathrm{Tr}(\mathcal{G})$ for which there is no corresponding infinite play in $\mathcal{G}$.

[18]Recall that the winning condition for the generalized powerset construction from Section 3.5.1 was also independent of the actions of player 0 if player 1 didn't have any information about these actions.

[19]We will see that for the epistemic unfolding modulo homomorphic equivalence where we obtain a more succinct representation by identifying homomorphic equivalent epistemic models, the situation is different.

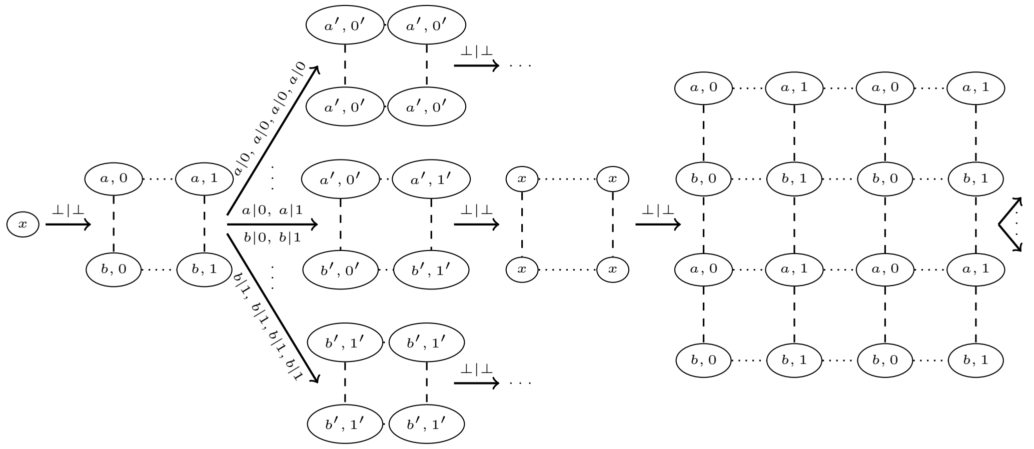


Figure 6.7: Epistemic unfolding $\mathrm{Tr}(\mathcal{G}_{\|})$ of the game graph $\mathcal{G}_{\|}$.

Observe that, since $\mathrm{Tr}(\mathcal{G})$ is a game graph with full information, in particular all players of the grand coalition have the same information. Thus, the grand coalition can be regarded as a single super-player who chooses actions on behalf of every member, and the game can be solved as if it was a two-player game between this super-player and the environment, cf. Section 2.1.

In Figure 6.7, we present a few first steps of the epistemic unfolding $\mathrm{Tr}(\mathcal{G}_{\|})$ of $\mathcal{G}_{\|}$. Note that the structures get larger as more and more knowledge of the players has to be accounted for. Also observe that, in contrast to the standard unfolding into the corresponding extensive game with imperfect information, the branching degree in $\mathrm{Tr}(\mathcal{G}_{\|})$ may grow with increasing level.

Proposition 6.12. *The grand coalition has a winning strategy for* $(\mathcal{G}, W, v_0)$ *if, and only if the grand coalition has a winning strategy for* $(\mathrm{Tr}(\mathcal{G}), W^t, \mathcal{K}_0)$.

Proof. First, let $\sigma = (\sigma_1, \dots, \sigma_n)$ be a winning strategy for the grand coalition for $(\mathcal{G}, W, v_0)$. We define the strategy $\sigma^t = (\sigma_1^t, \dots, \sigma_n^t)$ for the grand coalition for $\mathrm{Tr}(\mathcal{G})$ by induction over the length of histories in $\mathrm{Tr}(\mathcal{G})$ from initial position $\mathcal{K}_0$ such that, for each history $\pi = \mathcal{K}_0 \dots \mathcal{K}_r$ that is consistent with σ^t, any $\pi \in K_r$ is consistent with σ. For $r = 0$ there is nothing to do.

Let now $\pi^t = \mathcal{K}_0 \dots \mathcal{K}_r$ be a history in $\mathrm{Tr}(\mathcal{G})$ consistent with σ^t. We define $\sigma^t(\pi^t) = (a_k)_{k \in K_r}$ by

$$a_k = \sigma(k) \quad \text{for} \quad k \in K.$$

Notice that each $k \in K_r$ is a σ-history in $\mathcal{G}$ from v_0. We observe that $(a_k)_{k \in K_r} \in \mathrm{act}(\mathcal{K}_r)$: If $k \sim_i k'$ (in the epistemic model), then $k \sim_i^* k'$ (in $\mathcal{G}$), and since σ_i is a strategy for player i for $\mathcal{G}$, we have $(a_k)_i = \sigma_i(k) = \sigma_i(k') = (a_{k'})_i$. Now consider an epistemic successor $\mathcal{K}_{r+1} \in \mathrm{Next}(\mathcal{K}_r, (a_k)_{k \in K_r})$ of $\mathcal{K}_r$ with respect to $(a_k)_{k \in K_r}$. By definition, $\pi^t \mathcal{K}_{r+1}$ is consistent with σ^t and every $\pi \in K_{r+1}$ is consistent with σ. This concludes the induction.

Now consider any play $\pi^t = \mathcal{K}_0 \mathcal{K}_1 \dots$ in $\mathrm{Tr}(\mathcal{G})$ from $\mathcal{K}_0$ that is consistent with σ^t and let $\rho = k_0 k_1 \dots$ be any path through π^t. Since $k_0 = v_0$ and, by construction, each k_i has the form $k_i = v_0 \dots v_i$ such that $v_0 \dots v_i$ is a σ-history, we get that $v_0 v_1 \dots \in W$, and thus $\pi^t \in W^t$, by definition. Hence, σ^t is a winning strategy.

Now let, conversely, $\sigma^t = (\sigma_1^t, \dots, \sigma_n^t)$ be a winning strategy for the grand coalition for $(\mathrm{Tr}(\mathcal{G}), W^t, \mathcal{K}_0)$. We define the strategy $\sigma = (\sigma_1, \dots, \sigma_n)$ for the grand coalition for $(\mathcal{G}, W, v_0)$ by induction over the length of histories of $\mathcal{G}$ from v_0 and, simultaneously, with each such history $\pi = v_0 \dots v_r$ that is consistent with σ, we associate a history

$$\zeta(\pi) = \mathcal{K}_0 \dots \mathcal{K}_r$$

in $\mathrm{Tr}(\mathcal{G})$ from $\mathcal{K}_0$, such that the following conditions hold.

(1) $\pi \in K_r$

(2) if $\rho \sim_i^* \pi$ for some σ-history ρ in $\mathcal{G}$ from v_0 then $\zeta(\rho) = \zeta(\pi)$

(3) $\zeta(\pi)$ is consistent with σ^t

(4) $\zeta(v_0 \dots v_l) = \mathcal{K}_0 \dots \mathcal{K}_l$ for any $l \leq r$

For $\pi = v_0$ we define $\zeta(\pi) = \mathcal{K}_0$. Now let $\pi = v_0 \dots v_r$ be any history in $\mathcal{G}$ from v_0 that is consistent with σ and let $\zeta(\pi) = \mathcal{K}_0 \dots \mathcal{K}_r$. We define $\sigma_i(\pi) = (a_\pi)_i$, where

$$(a_k)_{k \in K_r} = \sigma^t(\zeta(\pi)),$$

that means, $\sigma_i(\pi)$ is the projection to the i-th component of the action, chosen by player i at $\zeta(\pi)$ for the position $\pi \in K_r$ according to σ^t. First, we observe that σ_i is constant over $\sim_i^*$-equivalence classes: if $\rho \sim_i^* \pi$ for some σ-history ρ in $\mathcal{G}$ from v_0, then by (1) and (2), $\rho \in \zeta(\rho) = \zeta(\pi)$, so $\sigma_i(\rho) = (a_\rho)_i$. Moreover, as $\pi \sim_i^* \rho$ and $(a_k)_{k \in K_r} \in \mathrm{act}(\mathcal{K}_r)$, we have $(a_\pi)_i = (a_\rho)_i$.

Now let $v_{r+1} \in V$ such that $(v_r, \sigma(\pi), v_{r+1}) \in \Delta$ (that means, πv_{r+1} is a σ-history) and let $\mathcal{K}_{r+1} \in \mathrm{Next}(\mathcal{K}_r, (a_k)_{k \in K_r})$ such that $\pi v_{r+1} \in K_{r+1}$, that means, $\mathcal{K}_{r+1}$ is the unique $\sim_{\cup}^n$-connected epistemic successor from $\mathrm{Next}(\mathcal{K}, (a_k)_{k \in K_r})$ that contains πv_{r+1}. Observe that, since $\mathrm{last}(\pi) = v_r$ and $(v_r, a_\sigma, v_{r+1}) \in \Delta$, the history πv_{r+1} is contained in $\mathrm{Update}(\mathcal{K}, (a_k)_{k \in K_r})$, ensuring (1) and, by induction, (4). By definition, $\zeta(\pi v_{r+1}) = \zeta(\pi)\mathcal{K}_{r+1}$ is consistent with σ^t, ensuring (3), so it remains to show (2), that means, if $\rho v \sim_i^* \pi v_{r+1}$ for some σ-history ρv of $\mathcal{G}$ from v_0, then $\zeta(\rho v) = \zeta(\pi v_{r+1})$.

First, notice that $\rho v \sim_i^* \pi v_{r+1}$ implies $\rho \sim_i^* \pi$, so $\zeta(\rho) = \zeta(\pi)$. Moreover, the construction of $\zeta(\pi v_{r+1})$ from $\zeta(\pi) = \zeta(\rho)$ is independent of πv_{r+1}, except for the choice of the $\sim_{\cup}^n$-connected component $\mathcal{K}_{r+1} \in \mathrm{Next}(\mathcal{K}_r, a)$ of $\mathrm{Update}(\mathcal{K}, (a_k)_{k \in K_r})$. As ρv is a σ-history with $\rho \in \mathcal{K}_r$, by definition of $\sigma(\rho)$, we have $\rho v \in \mathrm{Update}(\mathcal{K}_r, (a_k)_{k \in K_r})$, since $\rho v \sim_i^* \pi v_{r+1}$, ρv and πv_{r+1} lie in the same $\sim_{\cup}^n$-connected component of $\mathrm{Update}(\mathcal{K}_r, (a_k)_{k \in K_r},)$.

Finally, consider any play $\pi = v_0 v_1 \dots$ in $\mathcal{G}$ from v_0 that is consistent with σ and let $\pi^t = \zeta(\pi) = \mathcal{K}_0 \mathcal{K}_1 \dots$ be the play in $\mathrm{Tr}(\mathcal{G})$ from $\mathcal{K}_0$ associated with π By construction, any finite prefix $v_0 v_1 \dots v_l$ of π is a path through π^t of the form $k_0 k_1 \dots k_l$, and this extends to the whole play π. Since π^t is consistent with σ^t and thus won by the grand coalition, by definition of the winning condition W^t, we get that $\pi \in W$. □

6.2.2 Knowledge Tracking Modulo Homomorphic Equivalence

As we have mentioned, the game graph $\mathrm{Tr}(\mathcal{G})$ is infinite, by definition, for any game graph $\mathcal{G}$ which allows at least one infinite play. So, in order to be able to use the construction for solving the strategy problem, it is crucial to represent the game graph $\mathrm{Tr}(\mathcal{G})$ more succinctly, that is, we have apply some sort of compression that identifies certain epistemic models in $\mathrm{Tr}(\mathcal{G})$ in an appropriate way. So the question we have to answer is: What would be a good notion of similarity between epistemic models that is

(A) strong enough to yield a sufficient degree of succinctness

(B) guarantees that two epistemic models that are identified approximately represent the same mental state of the grand coalition?

Of course, appropriate degree of succinctness and approximately the same mental state are vague expressions. In fact, there is room for variation. We will see that the notion of similarity we come up with yields a finite result for *any* hierarchical game graph but, in general, it retains the relevant information carried by the epistemic models only in the case of observable winning conditions. There may be other notions that still guarantee finite results for all hierarchical game graphs but are sound for arbitrary winning conditions.

We start with a brief discussion of similarity of epistemic models. In particular, we introduce the notion of homomorphic equivalence and we show that the epistemic unfolding modulo this notion of similarity preserves winning strategies for the grand coalition in the case of *observable* winning conditions. This yields a semi-decision procedure (the procedure does not necessarily terminate, even on positive instances) for the strategy problem for games with imperfect information played on finite graphs with observable ω-regular and deterministic context-free specifications. At the end of this section we discuss the case of nonobservable winning conditions, especially ω-regular ones, and we we will see that for this case, the construction at least yields a *partial* solution method for the strategy problem (the procedure does not necessarily terminate and may produce wrong answers, but the positive answers are *always* correct).

Similarity of Epistemic Models. One simple approach to define a similarity on epistemic models would be to identify isomorphic models. Clearly, two isomorphic epistemic models represent the same mental state of the grand coalition. On the other hand, isomorphic epistemic models have, in particular, the same number of worlds, so for hierarchical games, the quotient modulo isomorphic equivalence would not be finite in general. Therefore, this does not yield a sufficient degree of succinctness.

Since epistemic models are, in particular, Kripke structures, another natural approach would be to define similarity of epistemic models as *bisimilarity* of Kripke structures. When used as system models, bisimilarity is the most popular notion of equivalence between Kripke structures. In particular, modal logic (and hence standard epistemic logic) cannot distinguish bisimilar Kripke structures. However, as it turns out, bisimulation is somewhat difficult to handle when it comes to epistemic unfolding. Of course, it may be that we have just fumbled with bisimulation[20] but there are also certain independent objections against bisimulation as a notion of similarity for Kripke structures that represent epistemic states. For a detailed discussion of epistemic equivalence and bisimulation we refer to [96] and the references given there.

Here we just mention the following aspect, picking up an example from [96]. Consider the Kripke structure $\mathcal{K} = (\{A, B, C, D\}, \{P\}, \sim_1, \sim_2)$ with $P = \{A, D\}$, $\sim_1$ is the closure of $\{(A, B), (C, D)\}$ and $\sim_2$ is the closure of $\{(A, C), (B, D)\}$. With respect to usual epistemic logic over $\text{Prop} = \{P\}$, agent 1 and agent 2 have the same knowledge in $\mathcal{K}$. On the other hand, $\mathcal{K}, A \models D_{1,2}P$ where $D_{1,2}$ is the operator for distributed knowledge: $\mathcal{K}, A \models D_{1,2}\varphi$ if $\mathcal{K}, X \models \varphi$ for all X with $X \sim_1 A$ *and* $X \sim_2 A$. So φ is distributed knowledge among agent 1 and agent 2 if they know φ if the pool their information about the possible worlds together.

But then, how is $\mathcal{K}, A \models D_{1,2}P$ possible if agent 1 and agent 2 have the same knowledge in $\mathcal{K}$? The reason is that the knowledge of the agents in terms of the propositions as expressible in epistemic logic is not the same as the knowledge that the agents have about the possible worlds. In the structure $\mathcal{K}$, both agents can distinguish the worlds A and D as well as the worlds B and C, although they satisfy the same atomic propositions. However, if we quotient the structure by bisimulation, those worlds a merged. In fact, the full epistemic situation in the bisimulation quotient is not reflected in $\mathcal{K}$. In particular, there is no homomorphism from the bisimulation quotient to the original structure $\mathcal{K}$. Recall that we have explicitly mentioned in Section 6.1 that we do *not* assume that the information of the agents in a multiagent system can be described completely by the atomic propositions. In particular, this is not the case if the possible worlds are histories and the atomic propositions just refer to the last position.

Technically, the problem is, indeed, the missing homomorphism from the bisimulation quotient to the original structure. Of course, we want to translate strategies both ways, from the extensive unfolding $\text{Tr}(\mathcal{G})$ to the succinct unfolding $\text{Tr}^{\text{succ}}(\mathcal{G})$ and vice versa. So, whenever we consider a position $\mathcal{L}$ in $\text{Tr}^{\text{succ}}(\mathcal{G})$ and the original position $\mathcal{K}$ in $\text{Tr}(\mathcal{G})$ then the possible worlds in $\mathcal{L}$ must have appropriate counterparts in $\mathcal{K}$. Now, we won't let this discussion escalate but just use this observation to lead over to the notion of *homomorphic equivalence* where, indeed, we have homomorphisms both ways by definition.

Homomorphic Equivalence. We recall the notion of graph homomorphism, which we apply

[20]In fact, we do not have an example that would show that bisimulation is not appropriate.

to epistemic models. Let $\mathcal{K} = (K, (P_v)_{v\in V}, (\sim_i))$ and $\mathcal{K}' = (K', (P'_v)_{v\in V}, (\sim'_i))$ be epistemic models. A function f is a *homomorphism* from $\mathcal{K}$ to $\mathcal{K}'$, if $P_v(k)$ implies $P'_v(f(k))$ and $k \sim_i k'$ implies $f(k) \sim'_i f(k')$. The models are *homomorphically equivalent*, denoted $\mathcal{K} \approx \mathcal{K}'$, if there exists a homomorphism from $\mathcal{K}$ to $\mathcal{K}'$ and a homomorphism from $\mathcal{K}'$ to $\mathcal{K}$. Notice that $\approx$ is an equivalence relation because the composition of two homomorphisms is again a homomorphism.

For a finite epistemic model $\mathcal{K}$, a *core* is an epistemic model $\mathcal{K}' \approx \mathcal{K}$ with the minimal number of elements. One crucial observation is that the core of a model is unique up to isomorphism. The proof uses the same arguments as the standard proof for graphs.

Lemma 6.13. *The core of a finite epistemic model is unique up to isomorphism.*

Proof. Let $\mathcal{K}_1 = (K_1, (P^1_v), (\sim^1_i))$ and $\mathcal{K}_2 = (K_2, (P^2_v), (\sim^2_i))$ be cores of $\mathcal{K}$. By definition of core, $|K_1| = |K_2|$ and as $\mathcal{K}_1$ and $\mathcal{K}_2$ are both homomorphically equivalent to $\mathcal{K}$, $\mathcal{K}_1$ and $\mathcal{K}_2$ are homomorphically equivalent as well. Moreover, all homomorphisms $f_{12} : \mathcal{K}_1 \to \mathcal{K}_2$ and $f_{21} : \mathcal{K}_2 \to \mathcal{K}_1$ are both bijections.

Indeed, if f_{12} was not a bijection, then the homomorphic image $f_{12}(\mathcal{K}_1)$ would be homomorphically equivalent to $\mathcal{K}_1$, as f_{12} is a homomorphism from $\mathcal{K}_1$ to $f_{12}(\mathcal{K}_1)$ and $f_{21} \restriction f_{12}(\mathcal{K}_1)$ is a homomorphism in the other direction. Hence, $f_{12}(\mathcal{K}_1)$ would be homomorphically equivalent to $\mathcal{K}$ but as f_{12} is not a bijection, $|f_{12}(K_1)| < |K_1|$ which contradicts the definition of a core. Analogous for f_{21}.

Now consider the composition $f := f_{21} \circ f_{12}$ of f_{12} and f_{21}. As f_{12} and f_{21} are both bijections, f is a bijection on K_1 and as K_1 is finite, there is a natural number $k \in \mathbb{N}$ such that f^k is the identity on K_1. Now consider the function $g := f_{12} \circ f^{k-1}$. As the composition of homomorphisms is again a homomorphism, g is indeed a bijective homomorphism from $\mathcal{K}_1$ to $\mathcal{K}_2$. Moreover, $f_{21} \circ g = f_{21} \circ f_{12} \circ f^{k-1} = f^k = \mathrm{id}_{K_1}$. So f_{21} is the unique inverse function of the bijection g. Hence, if $k, k' \in K_1$ such that $g(k) \sim^2_i g(k')$ then $k = f_{21}(g(k)) \sim^1_i f_{21}(g(k')) = k'$ and if $k \in K_1$ such that $g(k) \in P^2_v$ for some $v \in V$ then $k = f_{21}(g(k)) \in P^1_v$ since f_{21} is a homomorphism. Hence, g is a bijective strong homomorphism, that means, an isomorphism. $\square$

Epistemic Unfolding up to Homomorphic Equivalence. Essentially, epistemic unfolding up to homomorphic equivalence consists of performing the tracking construction while identifying homomorphically equivalent models. Since there may be many possible models equivalent to a model $\mathcal{K}$, we describe this unfolding with respect to a function q, defined on all epistemic models, which chooses for every epistemic model $\mathcal{K}$ a homomorphically equivalent companion model $q(\mathcal{K}) \approx \mathcal{K}$.

The epistemic unfolding of a game graph $\mathcal{G}$ with partial information *up to homomorphic equivalence*, with respect to a function q, is a game graph with *full* information

$$\mathrm{Tr}^q(\mathcal{G}) = (V^q, \Delta^q)$$

- V^q is the set $\{q(\mathcal{K}) \mid \mathcal{K} \text{ is an epistemic model over } \mathcal{G}\}$,
- $\Delta^q = \{(\mathcal{K}, (a_k)_{k\in K}, q(\mathcal{K}')) \mid (a_k)_{k\in K} \in A^{|K|} \text{ and } \mathcal{K}' \in \mathrm{Next}(\mathcal{K}, (a_k)_{k\in K})\}$,

Note that, since $\mathcal{K} \approx \mathcal{K}$, the unfolding Tr^q is a generalization of the tracking construction Tr obtained with $q(\mathcal{K}) = \mathcal{K}$. However, as we have mentioned, the construction is only sound for *observable* winning conditions. The definition of W^q for the case of an observable winning condition $W = (\mathrm{col}, W_o)$ is just as in the case of the general unfolding construction:[21]

$$W^q = \{\pi^q = \mathcal{L}_0\mathcal{L}_1 \ldots \mid \mathrm{col}(\mathcal{L}_0)\mathrm{col}(\mathcal{L}_1) \ldots \in W_o\}.$$

We now extend Theorem 6.12 to all unfoldings Tr^q for games with observable winning conditions. The key point is how to extend the homomorphisms from a model to the next one in a tracking. This is an interesting observation in itself and we formulate it as a separate lemma.

[21]We will see a more general definition that corresponds to the definition of W^t as given above at the end of this section.

Lemma 6.14. *Let $\mathcal{K}$ and $\mathcal{L}$ be epistemic models, let $h : \mathcal{K} \to \mathcal{L}$ be a homomorphism, and let $(b_l)_{l \in L}$ be a tuple of actions for $\mathcal{L}$. Then $(a_k)_{k \in K}$ with $a_k = b_{h(k)}$ is a tuple of actions for $\mathcal{K}$, and for each connected component $\mathcal{K}'$ of* $\text{Update}(\mathcal{K}, (a_k)_{k \in K})$*, there is a connected component $\mathcal{L}'$ of* $\text{Update}(\mathcal{L}, (b_l)_{l \in L})$ *such that there is a homomorphism $h' : \mathcal{K}' \to \mathcal{L}'$.*

Proof. Since h is a homomorphism, $(a_k)_{k \in K}$ is obviously a tuple of actions for $\mathcal{K}$. Let $\mathcal{K}'$ be a connected component of $\text{Update}(\mathcal{K}, (a_k)_{k \in K})$ and consider the connected component of $\text{Update}(\mathcal{L}, (b_l)_{l \in L})$ that contains all elements $h(k)v$ with $kv \in K'$. Note that since $\mathcal{K}'$ is connected by $\sim_{\cup}^{n}$ and h is a homomorphism, the elements $h(k)v$ are $\sim_{\cup}^{n}$-connected as well and thus are included in a single $\mathcal{L}'$, which we denote by $h(\mathcal{K}')$. The mapping $h' : \mathcal{K}' \to \mathcal{L}'$ with $h'(kv) = h(k)v$ is again a homomorphism, now from $\mathcal{K}'$ to $\mathcal{L}'$. $\square$

Proposition 6.15. *Let $W = (col, W_o)$ be an observable winning condition. The grand coalition has a winning strategy for* $(\text{Tr}(\mathcal{G}), W^t, \mathcal{K}_0)$ *if, and only if, the grand coalition has a winning strategy for* $(\text{Tr}^q(\mathcal{G}), W^q, q(\mathcal{K}_0))$.

Proof. First, let σ^t be a joint winning strategy for the grand coalition for $(\text{Tr}(\mathcal{G}), W^t, \mathcal{K}_0)$. We define the joint winning strategy σ^q for the grand coalition for $(\text{Tr}^q(\mathcal{G}), W^q, q(\mathcal{K}_0)$ by induction on the length of histories in $\text{Tr}^q(\mathcal{G})$ and simultaneously, with each such history $\pi^q = \mathcal{L}_0\mathcal{L}_1 \ldots \mathcal{L}_r$ that is consistent with σ^q, we associate a history

$$\zeta(\pi^q) = \mathcal{K}_0\mathcal{K}_1 \ldots \mathcal{K}_r$$

in $\text{Tr}(\mathcal{G})$, such that the following conditions hold:

(1) $\zeta(\pi^q)$ is consistent with σ^t

(2) there is a homomorphism $f : \mathcal{L}_r \to \mathcal{K}_r$

(3) $\zeta(\mathcal{L}_0\mathcal{L}_1 \ldots \mathcal{L}_s) = \mathcal{K}_0\mathcal{K}_1 \ldots \mathcal{K}_s$ for each $s \leq r$

For $r = 1$, there is only one history $\pi^q = \mathcal{L}_0 = q(\mathcal{K}_0)$. Thusly, let $\zeta(\pi^q) = \mathcal{K}_0$ and let $f : q(\mathcal{K}_0) \to \mathcal{K}_0$ be any homomorphism. Let now $\pi^q = \mathcal{L}_0 \ldots \mathcal{L}_r$ be a history consistent with σ^q, let $\zeta(\pi^q) = \mathcal{K}_0 \ldots \mathcal{K}_r$ be the associated history consistent with σ^t, and let $f : \mathcal{L}_r \to \mathcal{K}_r$ be a homomorphism according to (2). Consider the actions

$$(a_k)_{k \in K_r} = \sigma^t(\zeta(\pi^q))$$

prescribed by σ^t given the history $\zeta(\pi^q)$ in the game on $\text{Tr}(\mathcal{G})$.

We define $\sigma^q(\pi^q)$ by $\sigma^q(\pi^q)(l) = a_{f(l)} = \sigma^t(\zeta(\pi^q))(f(l))$. By Lemma 6.14, $\sigma^q(\pi^q)$ is a tuple of actions for $\mathcal{L}_r$. So, for any connected component $\mathcal{L}'$ of $\text{Update}(\mathcal{L}_r, \sigma^q(\pi^q))$, the sequence $\rho^q = \pi^q\mathcal{L}_{r+1}$ with $\mathcal{L}_{r+1} = q(\mathcal{L}')$ is a history in $\text{Tr}^q(\mathcal{G})$ which is, by definition, consistent with σ^q. Moreover, Lemma 6.14 yields a homomorphism $g : \mathcal{L}' \to \mathcal{K}'$ for some $\mathcal{K}' \in \text{Next}(\mathcal{K}_r, \sigma^t(\zeta(\pi^q)))$. So, by composing some homomorphism h from $\mathcal{L}_{r+1}$ to $q(\mathcal{L}_{r+1}) = \mathcal{L}'$ with the homomorphism g from $\mathcal{L}'$ to $\mathcal{K}'$, we obtain a homomorphism from $\mathcal{L}_{r+1}$ to $\mathcal{K}'$ and we set $\zeta(\rho^q) = \zeta(\pi^q)\mathcal{K}'$. By construction, $\zeta(\rho^q)$ is consistent with σ^t.

Now let $\pi^q = \mathcal{L}_0\mathcal{L}_1 \ldots$ be a play in $\text{Tr}^q(\mathcal{G})$ consistent with σ^q. By (3), the sequence $\zeta(\mathcal{L}_0), \zeta(\mathcal{L}_0\mathcal{L}_1), \ldots$ yields a play $\pi^t = \mathcal{K}_0\mathcal{K}_1 \ldots$ in $\text{Tr}(\mathcal{G})$ consistent with σ^t such that, for each $r \in \mathbb{N}$, there is a homomorphism $f_r : \mathcal{L}_r \to \mathcal{K}_r$. In particular we have $\text{col}(\mathcal{L}_r) = \text{col}(\mathcal{K}_r)$ for all $r \in \mathbb{N}$. Since σ^t is a winning strategy, $\pi^t \in W^t$, so $\text{col}(\mathcal{K}_0)\text{col}(\mathcal{K}_1) \ldots \in W_o$ and thusly $\pi^q \in W^q$. Hence, σ^q is a winning strategy for the grand coalition.

Now let σ^q be a joint winning strategy for the grand coalition for $(\text{Tr}^q(\mathcal{G}), W^q, q(\mathcal{K})_0)$. We define the joint winning strategy σ^t for the grand coalition for $(\text{Tr}^t(\mathcal{G}), W^t, \mathcal{K}_0$ similarly as before. We proceed by induction on the length of histories in $\text{Tr}(\mathcal{G})$ and simultaneously, with each such history $\pi^t = \mathcal{K}_0\mathcal{K}_1 \ldots \mathcal{K}_r$ that is consistent with σ^t, we associate a history

$$\zeta(\pi^t) = \mathcal{L}_0\mathcal{L}_1 \ldots \mathcal{L}_r$$

in $\text{Tr}^q(\mathcal{G})$, such that the following conditions hold:

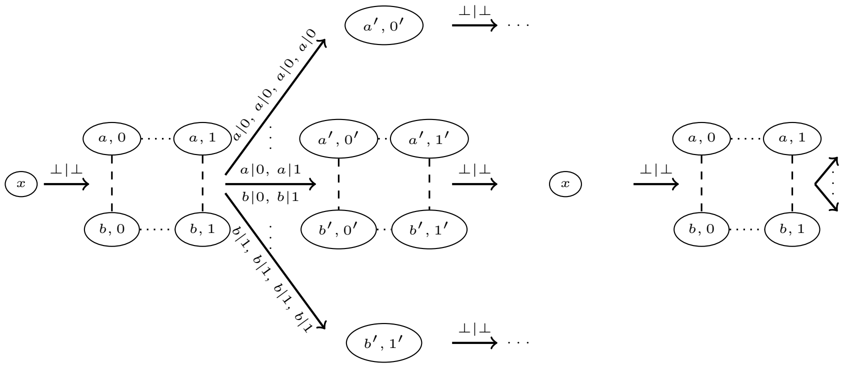


Figure 6.8: Epistemic unfolding $\mathrm{Tr}(\mathcal{G}_{||})$ quotiented by core.

(1) $\zeta(\pi^t)$ is consistent with σ^q

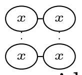
(2) there is a homomorphism $f : \mathcal{K}_r \to \mathcal{L}_r$

(3) $\zeta(\mathcal{K}_0\mathcal{K}_1 \dots \mathcal{K}_s) = \mathcal{L}_0\mathcal{L}_1 \dots \mathcal{L}_s$ for each $s \leq r$

For $r = 1$, there is only one history $\pi^t = \mathcal{K}_0$. Thusly, let $\zeta(\pi^t) = q(\mathcal{K}_0)$ and let $f : \mathcal{K}_0 \to q(\mathcal{K}_0)$ be any homomorphism. Let now $\pi^t = \mathcal{K}_0 \dots \mathcal{K}_r$ be a history consistent with σ^t, let $\zeta(\pi^t) = \mathcal{L}_0 \dots \mathcal{L}_r$ be the associated history consistent with σ^q, and let $f : \mathcal{K}_r \to \mathcal{L}_r$ be a homomorphism according to (2). Consider the actions $(b_l)_{l \in L_r} = \sigma^q(\zeta(\pi^t))$ prescribed by σ^q at $\zeta(\pi^t)$ in the game on $\mathrm{Tr}^q(\mathcal{G})$.

We define $\sigma^t(\pi^t)$ by $\sigma^t(\pi^t)(k) = b_{f(k)} = \sigma^q(\zeta(\pi^t))(f(k))$. By Lemma 6.14, $\sigma^t(\pi^t)$ is a tuple of actions for $\mathcal{K}_r$. So, for any connected component $\mathcal{K}'$ of $\mathrm{Update}(\mathcal{K}_r, \sigma^t(\pi^t))$, the sequence $\rho^t = \pi^t\mathcal{K}_{r+1}$ with $\mathcal{K}_{r+1} = \mathcal{K}'$ is a history in $\mathrm{Tr}(\mathcal{G})$ which is, by definition, consistent with σ^t. Moreover, Lemma 6.14 yields a homomorphism $g : \mathcal{K}' \to \mathcal{L}'$ for some $\mathcal{L}' \in \mathrm{Next}(\mathcal{L}_r, \sigma^q(\zeta(\pi^t)))$. So, by composing g with some homomorphism h from $\mathcal{L}'$ to $q(\mathcal{L}')$, we obtain a homomorphism from $\mathcal{K}_{r+1}$ to $q(\mathcal{L}')$ and we set $\zeta(\rho^t) = \zeta(\pi^t)q(\mathcal{L}')$. By construction, $\zeta(\rho^t)$ is consistent with σ^q.

Now let $\pi^t = \mathcal{K}_0\mathcal{K}_1 \dots$ be a play in $\mathrm{Tr}(\mathcal{G})$ consistent with σ^t. By (3), the sequence $\zeta(\mathcal{K}_0), \zeta(\mathcal{K}_0\mathcal{K}_1), \dots$ yields a play $\pi^q = \mathcal{L}_0\mathcal{L}_1 \dots$ in $\mathrm{Tr}^q(\mathcal{G})$ consistent with σ^q such that, for each $r \in \mathbb{N}$, there is a homomorphism $f_r : \mathcal{K}_r \to \mathcal{L}_r$. In particular we have $\mathrm{col}(\mathcal{K}_r) = \mathrm{col}(\mathcal{L}_r)$ for all $r \in \mathbb{N}$. Since σ^q is a winning strategy, $\pi^q \in W^q$, so $\mathrm{col}(\mathcal{L}_0)\mathrm{col}(\mathcal{L}_1) \dots \in W_o$ and thusly $\pi^t \in W^t$. Hence, σ^t is a winning strategy for the grand coalition. $\square$

While the proposition above can be applied to an arbitrary tracking Tr^q such that $q(\mathcal{K}) \approx \mathcal{K}$, we will concentrate on a specific one, namely $\mathrm{Tr}^{\mathrm{core}}(\mathcal{G})$, obtained with the function that maps every structure $\mathcal{K}$ to its core. In Figure 6.7, we have presented a few positions from the epistemic unfolding $\mathrm{Tr}(\mathcal{G}_{||})$. In Figure 6.8 we present the same situation, but these structures are now replaced by their cores. Note that, for example, 

gets quotiented to

and thus, from the fourth stage, the structures are repeated. Since we identify isomorphic Kripke structures, the game $\mathrm{Tr}^{\mathrm{core}}(\mathcal{G}_{||})$ is a finite game with perfect information.

The uniqueness of a core allows us to prove the following property.

Proposition 6.16. *There exists a finite tracking* $\mathrm{Tr}^q(\mathcal{G})$ *of* $\mathcal{G}$ *if, and only if, the tracking* $\mathrm{Tr}^{\mathrm{core}}(\mathcal{G})$ *is finite.*

Proof. Of course if $\mathrm{Tr}^{\mathrm{core}}(\mathcal{G})$ is finite then we have a finite tracking, so one direction is trivial. For the other one let q be a function such that $\mathrm{Tr}^q(\mathcal{G})$ is finite, that means, the set V^q. Then clearly $(V^q)^{\text{core}} = \{\text{core}(\mathcal{K}) \mid \mathcal{K} \in V^q\}$ is finite as well and we show that $(V^q)^{\text{core}} \supseteq V^{\text{core}}$.[22]

Consider some $\mathcal{K} \in V^{\text{core}}$, i.e., there is some epistemic model $\mathcal{L}$ over $\mathcal{G}$ with $L \subseteq V^*$ such that $\text{core}(\mathcal{L}) = \mathcal{K}$, and set $\mathcal{L}' = q(\mathcal{L})$. Then $\mathcal{L}'$ and $\mathcal{L}$ are homomorphically equivalent, and we claim that $\text{core}(\mathcal{L}')$ is also a core of $\mathcal{L}$. From this, by Lemma 6.13, it follows that $\text{core}(\mathcal{L}')$ is isomorphic to $\text{core}(\mathcal{L}) = \mathcal{K}$.

To see that $\text{core}(\mathcal{L}')$ is a core of $\mathcal{L}$, notice that $\text{core}(\mathcal{L}')$ is homomorphically equivalent to $\mathcal{L}'$ and thus to $\mathcal{L}$. Moreover, if there exists some $\tilde{\mathcal{L}}$ that is homomorphically equivalent to $\mathcal{L}$ such that $|\tilde{\mathcal{L}}| < |\text{core}(\mathcal{L}')|$, then $\tilde{\mathcal{L}}$ is also homomorphically equivalent to $\mathcal{L}'$, which contradicts the assumption that $\text{core}(\mathcal{L}')$ has the minimal number of worlds among all epistemic models over $\mathcal{G}$ that are homomorphically equivalent to $\mathcal{L}$. □

As a consequence, we obtain a semi-decision procedure for the strategy problem for games on finite game graphs with partial information and observable ω-regular and deterministic context-free winning conditions: compute $\text{Tr}^{\text{core}}(\mathcal{G})$ and if it is finite, solve the strategy problem for $(\text{Tr}^{\text{core}}(\mathcal{G}), W^{\text{core}})$. Thus, the procedure takes arbitrary game graphs with partial information and observable (ω-regular and deterministic context-free) winning conditions as input. This is in contrast to tree-automata based methods, which require a certain information-order among the players, and hence a-priori restrict possible inputs, cf. Section 3.2 and Chapter 5.

If, indeed, there is a joint winning strategy σ^q for the grand coalition for $(\text{Tr}^{\text{core}}(\mathcal{G}), W^{\text{core}})$ then a finite state winning strategy (pushdown winning strategy, respectively) can be constructed. At the end of this section, we discuss how we can compute a winning strategy for the grand coalition for $(\mathcal{G}, W)$ from such an implementation of σ^q.

Nonobservable Winning Conditions. Let us attempt an epistemic unfolding modulo homomorphic equivalence for a position based but not necessarily observable winning condition $W \subseteq V^\omega$. We consider here only $q = \text{core}$. A general definition of a winning condition W^q that applies to W can be derived from the definition of W^t as given above but, as we have mentioned, now we have to incorporate the actions (of the grand coalition) as well, cf. Section 3.5.1.[23] Moreover, when defining the notion of a path through a play $\pi^q = \mathcal{L}_0(a_l^0)_{l\in L_0}\mathcal{L}_1(a_l^1)_{l\in L_1}\dots$ (with actions for the grand coalition) in $\text{Tr}^q(\mathcal{G})$, we have to migrate to the core model in each step.

Now, a path through a play

$$\pi^q = \mathcal{L}_0(a_l^0)_{l\in L_0}\mathcal{L}_1(a_l^1)_{l\in L_1}\dots$$

(with actions) in $\text{Tr}^q(\mathcal{G})$ is a sequence $\rho = l_0 l_1 \dots$ such that

- $l_r \in \mathcal{L}_r$
- $l_{r+1} = q(l_r v)$ for some v with $(v_{l_r}, a_{l_r}^r, v) \in \Delta$

Where $q(l_r v)$ denotes the homomorphic image of $l_r v$ in $q(\text{Update}(\mathcal{L}, (a_l)_{l\in L}))$ under some homomorphism $f : \text{Update}(\mathcal{L}, (a_l)_{l\in L}) \to q(\text{Update}(\mathcal{L}, (a_l)_{l\in L}))$. For such a path ρ, the corresponding play $\pi_\rho = v_0 v_1 \dots$ in $\mathcal{G}$ is defined as before by $v_j = v_{k_j}$ for all $j \in \mathbb{N}$ and the set of plays that correspond to some path through π^q is denoted $\Pi(\pi^q)$. Also as before,

$$\pi^q \in W^q \quad \Longleftrightarrow \quad \Pi(\pi^q) \subseteq W.$$

Notice that, although W is position based, this definition does imply that W^q depends on the actions that occur in the plays π^q in $\text{Tr}^q(\mathcal{G})$.

[22]More precisely, we show that any epistemic model from V^{core} is isomorphic to an epistemic model from $(V^q)^{\text{core}}$. As we take V^{core} to contain no duplicate isomorphic copies, V^{core} is finite.

[23]Actually, since we only want to prove the strategy translation from $\text{Tr}^q(\mathcal{G})$ to $\mathcal{G}$, we could stick with a purely position based definition of W^q. Then, however, there would be even more cases where a winning strategy for $\mathcal{G}$ does not imply one for $\text{Tr}^q(\mathcal{G})$ which causes the partial solution method to fail gratuitously.

Now let us see why, with this definition of W^q, Proposition 6.15 does not extend to the nonobservable case. For this, consider the game $\mathcal{G}_{\|}$ and its epistemic unfolding $\mathrm{Tr}^q(\mathcal{G}_{\|})$ up to homomorphic equivalence as depicted in Figure 6.8. Notice that the topmost epistemic structure in the third step gets quotiented to a single element structure, so in particular, the histories $\pi_a = x(a,0)(a',0')$ and $\pi_b = x(b,0)(a',0')$ are merged. If the winning condition is observable, this is totally appropriate of course, because such a winning condition cannot distinguish between these two histories.

On the other hand, we have $\pi_a \not\sim_1^* \pi_b$, so a strategy for player 1 *can* distinguish between the histories – and it will have to in general, if the winning condition is not observable: Consider, for example, a winning condition W for $\mathcal{G}$ which requires player 1 to copy the first signal that he receives from player 0, but only at the second stage of the game, after player 0 has chosen another signal. The task of the two players in their first turn is just to choose $(a,0)$. Of course, the grand coalition has a joint winning strategy for $(\mathcal{G}, W, x)$. However, the grand coalition does not have a winning strategy for $(\mathrm{Tr}^q(\mathcal{G}), W^q, \mathcal{L}_0)$: Consider any strategy σ^q for the grand coalition. First, they will have to choose the sequence $\perp a \perp \perp$ of actions, where $a = (a|0, a|0, a|0, a|0)$ as depicted in Figure 6.8. Let $\pi^q = \mathcal{L}_0 \perp \mathcal{L}_1 a \mathcal{L}_2 \perp \mathcal{L}_3 \perp \mathcal{L}_4$ be the unique history in $\mathrm{Tr}^q(\mathcal{G})$ that is consistent with this choice of actions.

Now, for the world $(a,0)$ in $\mathcal{L}_4$, player 1 has to choose either action a or action b. However, none of these possible choices will guarantee winning because $\pi_a x(a,0)$ and $\pi_b x(a,0)$ are both histories that correspond to π^q. So, if player 1 chooses a then $\pi_b x(a,0)a(a',0')$ corresponds to $\pi^q \sigma^q(\pi^q)\mathcal{L}_5$ but $\pi_b x(a,0)a(a',0')$ is lost by the grand coalition in the game on $\mathcal{G}$. Analogously, if player 1 chooses b. Therefore, σ cannot be winning for $(\mathrm{Tr}^q(\mathcal{G}), W^q, \mathcal{L}_0)$.

Given this situation, there might arise some confusion: The winning condition W can be recognized by a deterministic safety automaton $\mathcal{B}$ and, as we already know, safety conditions can easily be transformed into observable safety condition. However, for observable safety conditions, the epistemic unfolding modulo homomorphic equivalence is sound. The point here is that to transform the given winning condition W into an observable safety condition, we first have to take the product of $\mathcal{B}$ and $\mathcal{G}$ and apply the epistemic unfolding to this new game graph. There, the histories which player 1 has to distinguish in order to win will carry the relevant information explicitly and therefore satisfy different atomic propositions in the epistemic structures – hence, they will not be merged.

This demonstrates that a strategy translation from $\mathcal{G}$ to $\mathrm{Tr}^q(\mathcal{G})$ is not meaningful for nonobservable winning conditions in general because in proceeding from $\mathcal{G}$ to $\mathrm{Tr}^q(\mathcal{G})$, too much information is lost for the grand coalition: Certain histories which some players still need to distinguish can get merged. On the other hand, since in $\mathcal{G}$ the grand coalition has at least as much information as in $\mathrm{Tr}^q(\mathcal{G})$, it seems reasonable that a winning strategy for the grand coalition for $(\mathrm{Tr}^q(\mathcal{G}), W^q)$ induces one for $(\mathcal{G}, W)$, for arbitrary winning conditions W. This is indeed the case. Moreover, similar as in Section 3.5.1, one can show that if W is ω-regular *and* $\mathrm{Tr}^q(\mathcal{G})$ is finite, then W^q is ω-regular as well. We will show these two facts in the next two propositions. First, let us discuss the consequences.

Given a finite game graph with partial information and some ω-regular winning condition W we can proceed as follows: Compute $\mathrm{Tr}^q(\mathcal{G})$ and, if it is finite, compute W^q. Then solve the strategy problem for $(\mathrm{Tr}^q(\mathcal{G}), W^q)$. and if there exists a joint winning strategy for the grand coalition for $(\mathrm{Tr}^q(\mathcal{G}), W^g)$ then construct a finite state implementation of such a strategy from which, in turn, a finite state implementation of a winning strategy for the grand coalition for $(\mathcal{G}, W)$ can be computed.

This yields a partial solution method for the strategy problem for (arbitrary) finite game graphs with partial information and ω-regular winning condition: The procedure is not guaranteed to halt, even on positive instances and if it does not find a winning strategy for the grand coalition, this does *not* mean, that none exists. On the other hand, if it *does* terminate and yields a positive answer then this answer is correct and, as we will see at the end of this section, we can implement a finite state winning strategy for the grand coalition for $(\mathcal{G}, W)$. For practical purposes, this can be useful. Of course, sometimes we write specifications for systems, being not sure whether they are actually realizable and we would like a synthesis method to establish this positively. However, if we have carefully written a specification for a realistic task then we are usually convinced that the specification is realizable but it might be rather involved to come up with such a realization. There, our partial solution method can be useful.

The strategy translation from $\mathrm{Tr}^q(\mathcal{G})$ to $\mathrm{Tr}(\mathcal{G})$ (which implies one to $\mathcal{G}$ by Proposition 6.12) is the same as in the proof of Proposition 6.15. However, we have to use an additional property of the histories $\zeta(\pi^t)$ in $\mathrm{Tr}^q(\mathcal{G})$, that we associate with the histories in $\mathrm{Tr}(\mathcal{G})$. The point is that if we have a history $\pi \in V^*$ in $\mathcal{G}$ that corresponds to π^t, not only must π also correspond to $\zeta(\pi^t)$ (which holds for the converse translation as well), but π must correspond to a path through $\zeta(\pi^t)$ that can be obtained by applying the homomorphisms to the path through π^t! In fact, due to the definition of W^q, if $k_0k_1\ldots k_r$ is a path through π^t then $f_0(k_0)f_1(k_1)\ldots f_r(k_r)$ is a path through $\zeta(\pi^t)$, where f_s for $s \leq r$ are the homomorphisms according to (2). This property does not hold for the converse translation, cf. the definition of W^t.

Proposition 6.17. *Let $W \subseteq V^\omega$ be an arbitrary winning condition. If the grand coalition has a joint winning strategy for $(\mathrm{Tr}^q(\mathcal{G}), W^q, q(\mathcal{K}_0))$ then the grand coalition has a winning strategy for $(\mathrm{Tr}(\mathcal{G}), W^t, \mathcal{K}_0)$.*

Proof. Let σ^q be a winning strategy for the grand coalition for $(\mathrm{Tr}^q(\mathcal{G}), W^q, q(\mathcal{K}_0))$. Analogously as in the proof of Proposition 6.15, we define the strategy σ^t and the function ζ, which associates with each such history $\pi^t = \mathcal{K}_0\mathcal{K}_1\ldots\mathcal{K}_r$ that is consistent with σ^t, a history

$$\zeta(\pi^t) = \mathcal{L}_0\mathcal{L}_1\ldots\mathcal{L}_r$$

in $\mathrm{Tr}^q(\mathcal{G})$, such that the conditions (1) – (3) hold. Moreover, this time, we also incorporate the *actions* into the histories. Now let

$$\pi^t = \mathcal{K}_0(a_k^0)_{k\in K_0}\mathcal{K}_1(a_k^1)_{k\in K_1}\cdots$$

be a play in $\mathrm{Tr}(\mathcal{G})$ that is consistent with σ^t. Using condition (3) we obtain that the sequence $\zeta(\mathcal{K}_0), \zeta(\mathcal{K}_0(a_k^0)_{k\in K_0}\mathcal{K}_1), \ldots$ yields a play

$$\pi^q = \mathcal{L}_0(b_l^0)_{l\in L_0}\mathcal{L}_1(b_l^1)_{l\in L_1}\cdots$$

in $\mathrm{Tr}^q(\mathcal{G})$ that is consistent with σ^q such that, for each $r \in \mathbb{N}$, there is a homomorphism $f_r : \mathcal{K}_r \to \mathcal{L}_r$. To show that π^t is won by the grand coalition, let $\pi = v_0v_1\ldots \in \Pi(\pi^t)$ and consider any path $\rho^t = k_0k_1\ldots$ through π^t such that π corresponds to ρ^t, that means, $v_r = v_{k_r}$ for all $r \in \mathbb{N}$. Since π also corresponds to $\rho^q = f_0(k_0)f_1(k_1)\ldots$ and π^q is won by the grand coalition, it only remains to show that ρ^q is a path through π^q as well: Clearly we have $f_r(k_r) \in \mathcal{L}_r$ for any $r \in \mathbb{N}$. Moreover, for each $r \in \mathbb{N}$ we have $k_{r+1} = k_rv$ for some $v \in V$ with $(v_{k_r}, a_{k_r}^r, v) \in \Delta$. Since f_r is a homomorphism, we have $v_{k_r} = v_{f_r(k_r)}$ and, by definition, $a_{k_r}^r = b_{f_r(k_r)}^r$. Therefore, $(v_{f_r(k_r)}, b_{f_r(k_r)}^r, v) \in \Delta$. By definition, this means that ρ^q is, indeed, a path through π^q. □

Proposition 6.18. *If W is ω-regular and $\mathrm{Tr}^{\mathrm{core}}(\mathcal{G})$ is finite then W^{core} is ω-regular and can be computed from W and $\mathcal{G}$.*

Proof. Let $\mathcal{A} = (\mathrm{VA}, Q, q_{\mathrm{in}}, \delta, \mathrm{col})$ be a deterministic parity automaton recognizing W. By definition of W^{core}, we have $\pi^q = \mathcal{L}_0(a_l^0)_{l\in\mathcal{L}_0}\mathcal{L}_1(a_l^1)_{l\in\mathcal{L}_1}\ldots \in W^{\mathrm{core}}$ if, and only if, for *each* path $\rho = l_0l_1\ldots$ through π^t, that means $l_r \in \mathcal{L}_r$ and $l_{r+1} = \mathrm{core}(l_rv)$ for some v with $(v_{l_r}, a_{l_r}^r, v) \in \Delta$, we have $v_{l_0}v_{l_1}\ldots \in W$. We construct a universal parity automaton

$$\mathcal{A}^{\mathrm{core}} = (V^{\mathrm{core}}A^{\mathrm{core}}, Q^{\mathrm{core}}, q_{\mathrm{in}}^{\mathrm{core}}, \delta^{\mathrm{core}}, \mathrm{col}^{\mathrm{core}})$$

that recognizes W^{core}. The automaton works similarly as the automaton constructed in Section 3.5.1: It universally branches over all such sequences $\pi = l_0 l_1 \dots$ and simulates $\mathcal{A}$ on the corresponding plays $v_{l_0} v_{l_1} \dots$ in $\mathcal{G}$. Of course, when proceeding from an epistemic model $\mathcal{L}_r$ with current world l_r to the epistemic model $\mathcal{L}_{r+1}$, we have to compute $\text{core}(l_r v)$ (for any v with $(v_{l_r}, a^r_{l_r}, v) \in \Delta$)), to determine the particular world l_{r+1} of $\mathcal{L}_{r+1}$ with which the automaton proceeds in this branch.

Formally, the automaton $\mathcal{A}^{\text{core}}$ is defined as follows. First $Q^{\text{core}} = Q \times L^{\text{core}}$ where L^{core} is the union of all the worlds contained in some epistemic model from V^{core} and $q^{\text{core}}_{\text{in}} = (q_{\text{in}}, v_0)$. Notice that we consider only plays in V^{core} from the initial epistemic model $\text{core}(\mathcal{K}_0) \cong \mathcal{K}_0$ which consists only of world v_0. Moreover, $\text{col}^{\text{core}}(q,l) = \text{col}(q)$. The transition function δ^{core} is defined as follows. For a given state $(q, l^*) \in Q^{\text{core}}$ and a symbol $(\mathcal{L}, (a_l)_{l\in L})$ which the automaton currently reads, we distinguish two cases. If $l^* \notin L$ then $\delta^{\text{core}}(\mathcal{L}(a_l)_{l\in L}, (q, l^*))$ is not defined. Otherwise,

$$\delta^{\text{core}}(\mathcal{L}(a_l)_{l\in L}, (q, l^*)) = \bigwedge_{v\in V\colon\ (v_{l^*}, a_{l^*}, v)\in\Delta} (\delta(q, v_{l^*} a_{l^*}), \text{core}(l^* v))$$

So the automaton $\mathcal{A}^{\text{core}}$ always stores the world $l^* = \text{core}(k^* w)$ that has been chosen for the current branch. If this world is not contained in the epistemic model $\mathcal{K}$ which the automaton reads next, then clearly, this branch should not be traced further. Otherwise, $\mathcal{A}^{\text{core}}$ branches over all successor positions v of v_{l^*} according to the action $(a_l)_{l\in L}$ that it reads. Then, any next world that may have to be traced (according to the definition of W^{core}) is of the form $\text{core}(l^* v)$ for such a v. Moreover, the next epistemic model that $\mathcal{A}^{\text{core}}$ reads determines which of these worlds have to be checked in fact. In this way, precisely the relevant sequences are examined and $\mathcal{A}^{\text{core}}$ accepts if, and only if, $\mathcal{A}$ accepts all corresponding plays in $\mathcal{G}$. Hence, $\pi^q \in L(\mathcal{A}^{\text{core}})$ if, and only if, $\pi^q \in W^q$. □

Remark. In the definition of the $\Pi(\pi^q)$ for some play π^q in $\text{Tr}^q(\mathcal{G})$, we have stipulated that $q(l_r v)$ denotes the homomorphic image of $l_r v$ in $q(\text{Update}(\mathcal{L}, (a_l)_{l\in L}))$ under *some* homomorphism $f : \text{Update}(\mathcal{L}, (a_l)_{l\in L}) \to q(\text{Update}(\mathcal{L}, (a_l)_{l\in L}))$. So implicitly, we quantify over all possible such homomorphisms f. Alternatively, we could have assume that for each $\mathcal{L} \in \text{Tr}^q(\mathcal{G})$ and each possible action $(a_l)_{l\in L}$ for $\mathcal{L}$ we have a fixed homomorphism

$$f_{[\mathcal{L},(a_l)_{l\in L}]} : \text{Update}(\mathcal{L}, (a_l)_{l\in L}) \to q(\text{Update}(\mathcal{L}, (a_l)_{l\in L})).$$

For $\mathcal{L} \in \text{Tr}^q(\mathcal{G})$, some action $(a_l)_{l\in L}$ for $\mathcal{L}$ and a world $lv \in \text{Update}(\mathcal{L}, (a_l)_{l\in L})$, we could then have denoted

$$q(lv) = f_{[\mathcal{L},(a_l)_{l\in L}]}(lv).$$

It is not hard to see, however, that this yields the same set $\Pi(\pi^q)$ and therefore, the same winning condition W^q: If $\rho = k_0 k_1 \dots$ is a path through some play

$$\pi^q = \mathcal{L}_0 (a^0_l)_{l\in L_0} \mathcal{L}_1 (a^1_l)_{l\in L_1} \cdots$$

in $\text{Tr}^q(\mathcal{G})$ that is defined with respect to arbitrary homomorphisms, then we can easily construct a path $\tilde{\rho} = l_0 l_1 \dots$ through π^q that is defined with respect to the fixed homomorphisms $f_{[\mathcal{L},(a_l)_{l\in L}]}$ such that $v_{k_r} = v_{l_r}$ for all $r \in \mathbb{N}$: We define $l_0 = k_0$ and, if $l_0 \dots l_r$ is constructed, then we have $k_{r+1} = f(k_r v)$ for some $v \in V$ with $(v_{k_r}, a^r_{k_r}, v) \in \Delta$ and some homomorphism $f : \text{Update}(\mathcal{L}_r, (a^r_l)_{l\in L_r}) \to \text{core}(\text{Update}(\mathcal{L}_r, (a^r_l)_{l\in L_r}))$. Since $v_{k_r} = v_{l_r}$ we have $l_r v \in \text{Update}(\mathcal{L}_r, (a^r_l)_{l\in L_r})$ and we set

$$l_{r+1} = f_{[\mathcal{L}_r,(a^r_l)_{l\in L_r}]}(l_r v).$$

Since $f_{[\mathcal{L}_r,(a^r_l)_{l\in L_r}]}$ is a homomorphism, we have $v_{l_{r+1}} = v = v_{k_{r+1}}$.

Implementation of Strategies. Assume that there is a joint winning strategy σ^q for the grand coalition for $(\mathrm{Tr}^q(\mathcal{G}), W^q, q(\mathcal{K}_0))$. We consider only the case where W^q allows positional strategies, so that we can assume that σ^q is a function $V^q \to A$. The extension to more general winning conditions is straightforward, incorporating the computing devices which implement σ^q into the construction similar as described in Section 3.5.2. The main line of argumentation is the same as there as well: The strategy automaton reads histories in $\mathcal{G}$ and constructs, step by step, the corresponding position in $\mathrm{Tr}^q(\mathcal{G})$ according to the proofs of Proposition 6.15 and Proposition 6.12. However, there are certain particular issues in the multiplayer case that we have to take care of.

In the proof of Proposition 6.12, we have constructed a joint winning strategy σ for the grand coalition for $(\mathcal{G}, W, v_0)$ which is not decomposed into strategies for the individual players but it is a function $\sigma : V^* \to A$ such that $\mathrm{Pr}_{A_i}(\sigma)$ satisfies the consistency condition (S2) for all i. However, for an actual realization of some distributed reactive system, we need separate implementations of all individual strategies where each strategy automaton $\mathcal{A}_i$ reads only the signals that player i actually receives, that is, it reads observations $[v]_i$. Let us describe how such an automaton $\mathcal{A}_i$ works.

The states of the automaton have the form $(\mathcal{L}, L')$ where $\mathcal{L} \in V^q$ and $L' \subseteq L$ is a subset of the possible worlds of $\mathcal{L}$ with $k \sim_i l$ for all $k, l \in L'$. This subset represents those worlds in $\mathcal{L}$ that player i actually considers possible. The initial state is $(\mathcal{L}_0, L_0)$ with $\mathcal{L}_0 = q(\mathcal{K}_0)$. Now, given a state $(\mathcal{L}, L')$, the output of the automaton is $\mathrm{Pr}_{A_i}(\mathrm{Pr}_k(\sigma^q(\mathcal{L})))$ for an arbitrary $k \in L'$. That means, we look at the joint macro action $\sigma^q(\mathcal{L}) = (a_l)_{l \in L}$ that σ^q prescribes at the position $\mathcal{L}$, we choose one of the worlds $k \in L'$ that player i considers possible, and we let player i choose the action $\mathrm{Pr}_{A_i}(a_k)$ in the game on $\mathcal{G}$.

Now in the next step, reading an observation $[u]_i$ of player i, the automaton $\mathcal{A}_i$ proceeds as follows: First, it computes $\mathrm{Update}(\mathcal{L}, \sigma^q(\mathcal{L}))$ and then it chooses the connected component $\tilde{\mathcal{L}} \in \mathrm{Next}(\mathcal{L}, \sigma^q(\mathcal{L}))$ of $\mathrm{Update}(\mathcal{L}, \sigma^q(\mathcal{L})$ that corresponds to $[v]_i$ *and* L'.[24] To see how this component is determined, consider the set $\tilde{L}'$ of all possible worlds kv from $\mathrm{Update}(\mathcal{L}, \sigma^q(\mathcal{L}))$ that satisfy $k \in L'$ and $v \sim_i^V u$. Then there is exactly *one* connected component $\tilde{\mathcal{L}}$ of $\mathrm{Update}(\mathcal{L}, \sigma^q(\mathcal{L}))$ such that $\tilde{L}' \subseteq \tilde{L}$. The next state of the automaton is now

$$(\mathrm{core}(\tilde{\mathcal{L}}), \mathrm{core}(\tilde{L}')),$$

where $\mathrm{core}(\tilde{L}') = \{\mathrm{core}(l) \mid l \in \tilde{L}'\}$ and, as before, $\mathrm{core}(l)$ denotes the homomorphic image of l in $\mathrm{core}(\tilde{\mathcal{L}})$ under some homomorphism $f : \tilde{\mathcal{L}} \to \mathrm{core}(\mathcal{L})$.

Then, the joint strategy

$$(\sigma_1, \dots, \sigma_n)$$

for the grand coalition, where each individual strategy σ_i is implemented by the automaton $\mathcal{A}_i$, coincides with the strategy σ as constructed in the proofs of Proposition 6.15 and Proposition 6.12, so it is a winning strategy.

6.3 Hierarchical Games

We apply our construction to hierarchical games which are games played on finite game graphs $\mathcal{G} = (V, \Delta, (\sim_i^V))$ with hierarchical partial information, that means,

$$\sim_1^V \subseteq \sim_2^V \subseteq \ldots \subseteq \sim_n^V .$$

So the information of the players is ordered linearly: Player 1 is the best informed one, player n has the least information. Such game graphs correspond to distributed systems with pipeline

[24]Notice that without the set L', we could not choose a unique component because the observation $[v]_i$ may occur in several connected components of $\mathrm{Update}(\mathcal{L}, \sigma^q(\mathcal{L}))$.

architectures, cf. Section 2.3 so from the results that we have discussed in Section 3.2 it follows that the strategy problem is decidable for hierarchical games with ω-regular winning conditions, cf. [171, 129, 80]. On the other hand, pipeline architectures are undecidable for deterministic context-free specifications, even if we have only two controllers (or at least one hidden channel from the environment), cf. Section 3.4.

In this section we show that the strategy problem is decidable for hierarchical games with *observable* deterministic context-free winning conditions. For this, we demonstrate that for any finite game graph $\mathcal{G}$ with hierarchical partial information, $\mathrm{Tr}^{\mathrm{core}}(\mathcal{G})$ is finite. This is also an interesting observation in itself because it shows that the abstraction by homomorphic equivalence yields a reasonable degree of succinctness.

Proposition 6.19. *Let V be a finite set and let n be a natural number. Up to homomorphic equivalence, there are at most $\exp_n(|V|)$ different Kripke structures $\mathcal{K} = (K, (P_v)_{v\in V}, (\sim_i)_{i=1}^n)$ such that:*

1. *$(P_v)_{v\in V}$ is a partition of K*
2. *$\sim_1 \subseteq \ldots \subseteq \sim_n$ are equivalence relations*
3. *$\mathcal{K}$ is connected by $\sim_{\cup}^n$.*

Proof. We denote by $\Psi_n(V)$ the class of all Kripke structures $\mathcal{K}$ which satisfy 1. – 3. We prove inductively that, for each $n \in \mathbb{N}$, there is a class $\Psi_n^{\approx}(V)$ of Kripke structures from $\Psi_n(V)$ with $|\Psi_n^{\approx}(V)| = \exp_n(|V|)$ such that each structure from $\Psi_n(V)$ is homomorphically equivalent to one from $\Psi_n^{\approx}(V)$.

First, $\Psi_1^{\approx}(V)$ is the set of all Kripke structures $\mathcal{K} = (K, (P_v)_{v\in V}, \sim_1)$ with $K \subseteq V$, $P_v = \{v\}$ for $v \in V$ and $\sim_1 = K \times K$. Hence, any structure in $\Psi_1^{\approx}(V)$ can be identified with a subset of V, so $|\Psi_1^{\approx}(V)| = 2^{|V|} = \exp_1(|V|)$. Clearly, $\Psi_1^{\approx}(V) \subseteq \Psi_1(V)$. Now let

$$\mathcal{L} = (L, (P_v)_{v\in V}, \sim_1) \in \Psi_1(V).$$

We define a homomorphism f on $\mathcal{L}$ by

$$f(l) = v_l.$$

(Recall that v_l is the unique $v \in V$ such that $l \in P_v$.) The homomorphic image $f(\mathcal{L}) = (K, (P_v)_{v\in V}, \sim_1)$ is in $\Psi_1^{\approx}(V)$ and $g : f(\mathcal{L}) \to \mathcal{L}$ with $g(v) = l$ for some $l \in L \cap P_v$ is a homomorphism. Hence, $\mathcal{L}$ and $f(\mathcal{L})$ are homomorphically equivalent.

For $n > 1$, suppose $\Psi_{n-1}^{\approx}(V)$ has already been constructed. Without loss, we assume that all Kripke structures from $\Psi_{n-1}^{\approx}(V)$ are pairwise disjoint. We define $\Psi_n^{\approx}(V)$ as the set of all Kripke structures $\mathcal{K} = (K, (P_v)_{v\in V}, (\sim_i)_{i=1}^n)$ which consist of a union of epistemic models from $\Psi_{n-1}^{\approx}(V)$ and we set $\sim_n = K \times K$. Hence, any structure in $\Psi_n^{\approx}(V)$ can be identified with a subset of $\Psi_{n-1}^{\approx}(V)$, so $|\Psi_n^{\approx}(V)| = 2^{\exp_{n-1}(|V|)} = \exp_n(|V|)$. Now, let

$$\mathcal{L} = (L, (P_v)_{v\in V}, (\sim_i)_{i=1}^n)$$

be any Kripke structure from $\Psi_n(V)$. As $\mathcal{L}$ is connected by $\sim_{\cup}^n$, we have $\sim_n = L \times L$: any $l, l' \in L$ are connected in $\mathcal{L}$ via some $\sim_{\cup}^n$-path and as $\sim_{\cup}^{n-1} \subseteq \sim_n$ and $\sim_n$ is transitive, it follows that $l \sim_n l'$.

Consider the decomposition of $\mathcal{L}$ into $\sim_{\cup}^{n-1}$-connected components $\mathcal{L}_1, \ldots, \mathcal{L}_r$. Clearly, $\mathcal{L}_j \in \Psi_{n-1}(V)$ for $j = 1, \ldots, r$ and hence, each $\mathcal{L}_j$ is homomorphically equivalent to a Kripke structure from $\Psi_{n-1}^{\approx}(V)$. For $j \in \{1, \ldots, r\}$ we fix a homomorphism f_j on $\mathcal{L}_j$ such that the homomorphic image $f_j(\mathcal{L}_j)$ is in $\Psi_{n-1}^{\approx}(V)$ and a homomorphism g_j from $f_j(\mathcal{L}_j)$ to $\mathcal{L}_j$. Moreover, we define the homomorphism f on $\mathcal{L}$ by

$$f \restriction \mathcal{L}_j = f_j$$

for $j = 1, \dots, r$. As the components $\mathcal{L}_j$ are pairwise disjoint, this is well defined and it is easy to see that the homomorphic image $f(\mathcal{L})$ is in $\Psi_n^{\approx}(V)$.

Furthermore, we define $g : f(\mathcal{L}) \to \mathcal{L}$ as follows. For any $\sim_{\cup}^{n-1}$-connected component $\mathcal{M}$ of $f(\mathcal{L})$, there is some $j \in \{1, \dots, r\}$ such that $f_j(\mathcal{L}_j) = \mathcal{M}$ and we define $g{\upharpoonright}\mathcal{M} = g_j$, for an arbitrary such j. The mapping g is a homomorphism and hence, $\mathcal{L}$ and $f(\mathcal{L})$ are homomorphically equivalent. $\square$

Theorem 6.20. *The strategy problem is decidable for hierarchical games with observable deterministic context-free winning conditions.*

Proof. Let $\mathcal{G}$ be a game graph with hierarchical partial information and let $W = (\mathrm{col}, W_o)$ be an observable winning condition such that W_o is deterministic context-free.

By Proposition 6.15, the grand coalition has a joint winning strategy for $(\mathcal{G}, W)$ if, and only if, it has one for $(\mathrm{Tr}^{\mathrm{core}}(\mathcal{G}), W^{\mathrm{core}})$, and since $\mathcal{G}$ has hierarhical partial information, by Proposition 6.19, $\mathrm{Tr}^{\mathrm{core}}(\mathcal{G})$ is finite.

Moreover, since W_o is deterministic context-free, so it is easy to see that W^{core} is deterministic context-free as well. Hence, the strategy problem for $(\mathcal{G}, W)$ can be decided by solving the problem for $(\mathrm{Tr}^{\mathrm{core}}(\mathcal{G}), W^{\mathrm{core}})$ – a finite game graph with full information and deterministic context-free winning condition. $\square$

Two-Player Games. A very special case of hierarchical games are two-player games on finite game graph with partial information which we have studied extensively in Section 3.5 and in Chapter 4. It is not hard to see that the powerset construction as presented in Section 3.5 is a special case of the epistemic unfolding modulo homomorphic equivalence: For any game graph with partial information and two players we have

$$2^{\mathcal{G}} = \mathrm{Tr}^{\mathrm{core}}(\mathcal{G}).$$

Moreover, for any given winning condition W we have

$$\overline{W} = W^{\mathrm{core}}.$$

For nonobservable winning conditions, of course, this is only true if we use the general definition that incorporates actions as well but then, the strategy translation is sound both ways. The reason is that the grand coalition consists only of a single player, so whenever two histories are merged by the core, they are indistinguishable for that player and therefore, a strategy cannot make a difference between them anyway.

Chapter 7

Conclusion

Synthesis is computationally harder than verification and synthesis for systems with partial information is computationally harder than synthesis for systems with full information. This has been sort of the guiding theme for the research we have presented. Although this statement is clearly not beyond reproach,[1] it strongly motivates a close look at the fine structure of synthesis under partial information, where multiple essential parameters crucially influence the computational complexity. We have found that by carefully and elaborately tuning certain parameters, interesting and relevant cases are obtained that can be solved algorithmically and, in some more restricted cases, even by polynomial time algorithms. In Chapter 3 we have assembled and reviewed as well as slightly extended and unified fundamental solutions and hardness results. In Chapters 4, 5 and 6, we have made further contributions that substantially extend known results to more general cases and also offer some completely new perspectives.

In view of our experiences, it seems obvious that the most crucial factor for the computational complexity is the difference between two players and three players or more. We have intensively studied the special case where only two players interact on a finite game graph, particularly for parity conditions. The strategy problem for these games is EXPTIME-complete for both, observable and non-observable, parity conditions. However, for the case of observable parity conditions, the usual powerset construction is sufficient and no additional automata constructions are involved. This makes it particularly practicable to analyze this case further, especially in terms of *complexity of the game graphs* which was the subject of Chapter 4.

The starting points were the corresponding PTIME results for parity games with full information on finite game graphs of bounded complexity. We have seen that for game graphs with unbounded partial information, these results do not hold. More precisely, even for game graphs with partial information of DAG-width at most three, the strategy problem is EXPTIME-hard for (observable) reachability as well as safety conditions. Moreover, on acyclic game graphs, the problem is still PSPACE-hard. This demonstrates that the intrinsic complexity caused by (unbounded) partial information is high, even on very simple graphs.

On the other hand, for game graphs with bounded partial information, we have shown that the strategy problem can be solved in polynomial time for observable parity conditions. Since we could rely on the algorithmic solution from [24, 25], our task was merely to show that the powerset construction preserves DAG-width. For this, however, we have used heavy machinery. We have discussed the notion of *concurrent graph* searching which implies the problem of interferences between different plays. To handle this problem, after a detour via directed path-width, we have introduced a new concept of graph searching with mulitple robbers which can not only run on the graph but also have the capacity to *split*. We have shown that if

[1] In particular, in view of the fact that the computational effort of verification is measured in terms of both, the size of the specification *and* the size of the (given) system, while for synthesis, only the size of the specification is taken into account, cf. [129].

k cops monotonously capture a single robber on a directed graph G then $k \cdot r$ cops monotonously capture r jumping robbers on G. The way in which we have defined the cops' strategy for the game against r robbers from the strategy against a single robber is quite intricate and it is far from obvious whether this can be substantially simplified.

Of course, if one could show that DAG-width has bounded monotonicity cost, then the PTIME-result would follow without using graph searching with multiple robbers.[2] Notice that since we already know that DAG-width does not have monotonicity cost zero, this solution would not directly yield the same precise bound of $k \cdot r \cdot 2^{r-1}$ on the DAG-width of $2^{\mathcal{G}}$ but at least a constant factor of $\frac{4}{3}$ would have to be added, cf. the example in [123]. Moreover, although we did not pursue any further aspects of this here, our new concept of graph searching on directed graphs with multiple robbers that can also *split* seems quite interesting in itself.

Completely independently of this, a solution to the monotonicity problem would be a great contribution to the theory of complexity measures for directed graphs which is currently at a rather early stage, cf. Section 4.1.2. We have also mentioned several normal forms of strategies for the cop player as well as the robber player that would help to understand the particular properties of graph complexity measures for directed graphs and which might be a first step towards a solution of the monotonicity problem and several other important open questions. Notice that for a particular normal form that would have been most helpful for proving Theorem 4.22, we have shown that it does *not* exist.

Our analysis yields rather tight results: For the case of unbounded partial information we have shown that even on very simple graphs, the strategy problem for reachability games with imperfect information is computationally hard. One of the few interesting cases that are still open is that of game graphs of DAG-width at most two and winning conditions that allow positional winning strategies in the full information case. On the other hand, for bounded partial information, the problem can be solved in polynomial time on any class of graphs with bounded DAG-width which is much less restrictive than bounded tree-width or directed path-width. For other interesting graph complexity measures like Kelly-width, directed tree-width and clique-width, however, the question is still open.

Besides these gaps and variations that we were not able to solve, the everlasting quest for tractable subcases of the strategy problem for two-player games on finite graphs with partial information and preferably expressive winning conditions raises a lot of challenges that we did not take up here at all. For example, the question whether the strategy problem for parity games with full information can be solved in polynomial time on classes of graphs with bounded directed tree-width is still open. Moreover, as we have mentioned, no general criteria are known which would guarantee that the antichains method (cf. Section 3.5.3) terminates after polynomially many steps. Another challenging case is that of nonobservable winning conditions. So far, there seems to be no way around the automata constructions that are usually used to handle nonobservable winning conditions, but chewing upon this intensively might finally bring some manageable subcases to light.

Currently one of the most important and intriguing open problems in the theory of synthesis and verification of nonterminating reactive finite state systems is whether the strategy problem for parity games on finite game graphs with full information is in PTIME.[3] Notice that this would also directely imply that the strategy problem for two-player games on finite game graphs with bounded partial information and observable parity conditions can be solved in polynomial time, regardless of the complexity of the game graph.

[2]Unless, of course, graph searching with multiple robbers would be used for the proof of the monotonicity result. However, although certain difficulties that we have encountered in the proof of Theorem 4.22 are quite related to the monotonicity problem, it does not seem likely that our concept of graph searching with multiple robbers could be used as a tool for this.

[3]A nice side-aspect of this problem is that if one could show that the problem is actually NP-hard it would follow that NP = co-NP.

After this close analysis of the two-player case, we have turned to more general scenarios where, in particular, an arbitrary number of players is involved. We have built on the long tradition of papers establishing decidability and undecidability results for architectures along certain patterns of information flow. A distinguishing feature of these results is that, unlike results on game graphs with partial information, they do not consider any a priori restrictions of the system, like a predefined labeling for the channels or a given subset of the possible system runs. The case of arbitrary safety/regular (and CTL / CTL*) specifications was already completely settled [171, 129, 80] but of course, there are lots of possible further extensions and restrictions of the specifications that are worthwile to be considered. Here, we focus on the informational aspect of interaction, so it is particularly expedient to consider special cases of specifications that restrain the extent to which the specification can address facts that (some of) the controllers do not observe.

We have seen that *observable* specifications are too restricted for the model of distributed systems as we have used it. On the other hand, locally decomposable specifications are still expressive enough to cover many relevant and interesting cases. In [137], a complete characterization of all decidable architectures has been proved, but only for regular specifications and acyclic architectures. In particular, the restriction to acyclic architectures is a serious limitation of the scenario because in complex communication networks, cycles occur frequently. Moreover, by extending the analysis to context-free specifications we enter the area of infinite state systems which is also very appealing in view of complex computing systems.

The investigation in Chapter 5 has covered both these aspects and, ultimately, we have again provided a complete characterization of all decidable architectures for locally decomposable specifications. A particular feature of our characterization is that different processes in the architecture may have different types of local specifications: Some are restricted to regular local specifications, others may have deterministic context-free specifications and, in rare cases, we can even allow arbitrary context-free specifications without losing decidability.

Also here, our analysis yields a fairly complete picture. One of the cases that has been left open is that of architectures with broadcast channels. As we have mentioned, in principle, broadcast channels can be simulated in our setting, using the specification. However, if there are broadcast channels from the environment, it is not clear whether this can be done in such a way that local decomposability is preserved. It is also not clear whether there are specifications that are more expressive than regular ones but for which the controller problem is decidable in certain cases where deterministic context-free (even DR1-C) specifications cause undecidability. Another open problem is whether the decidability results for locally decomposable regular specifications can be transfered to branching time specifications like CTL and CTL*. As we have mentioned, the problem is that the languages which are represented by the communication trees $t \in L(\mathcal{A}_i)$ are *overapproximations* of the languages that may actually produced by the controller p_i according to some stategy. In the linear time paradigm this is ok but in the branching paradigm it fails because CTL-formulas can also require that certain branches in a strategy tree have to *exist*. In fact, in [135] it has been conjectured that two-flanked pipelines are undecidable for locally decomposable CTL-specifications.

Besides these gaps and variations that we were not able to solve, the everlasting quest for decidable subcases of the controller problem for distributed systems and preferably expressive winning conditions raises a lot of challenges that we did not take up here at all. One other approach that we have already mentioned is that of [94] where *external specifications* have been considered that only relate external input and output channels. Also there, certain decidability results have been shown for so-called well connected architectures that contain information forks.[4] Clearly, this is a relevant and interesting subcase of the general problem and it seems appealing to pursue this into further directions. Many other restricted settings and variations of the synthesis problem have been considered, see for example [8, 148, 92, 220, 195, 196, 155, 15, 136, 162, 79].

[4]In [93] it has been shown that well connectedness of architectures is also decidable.

As promised in the introduction, our decision procedures also imply techniques for synthesizing winning strategies which do not require significantly more computational effort than the solution of the decision problem. For the polynomial time algorithm that is obtained from our results in Chapter 4 this is straightforward, given the methods that we can build on: We apply the algorithm from [24, 25] to the powerset graph $2^{\mathcal{G}}$ of which we know that $\mathrm{dw}(2^{\mathcal{G}}) \leq k \cdot r \cdot 2^{r-1}$ and, if player 1 does have a winning strategy for $(2^{\mathcal{G}}, \overline{\mathrm{col}}, \{v_0\})$, the algorithm constructs a positional such strategy in polynomial time. As discussed in Section 3.5.2, from this strategy we obtain a finite memory winning strategy for $(\mathcal{G}, \mathrm{col}, v_0)$ which uses only polynomially many states.

For the algorithms that we have developed in Chapter 5, controller synthesis was not completely straightforward. First, we had to deal with the fact that the structure of the decidable architectures may be rather involved as compared to simple, straight pipelines and two-flanked pipelines. Second, the usual constructive emptiness test for nondeterministic (parity) tree automata (Rabin's Theorem) is not sufficient, because the languages of the tree automata that we construct in the first instance do not consist of strategies. However, we have seen that, using an extended constructive emptiness test and adroitly combining the partial solutions for the individual parts of the architecture, we can construct a finite memory winning strategy for the grand coalition whenever the grand coalition has a joint winning strategy for a given locally decomposable regular specification. For context-free specifications, of course, we have to use pushdown machines to implement winning strategies.

In the last chapter we have considered multiplayer games with imperfect information from a more epistemological viewpoint, focussing on knowledge rather than strategic powers. We have seen that already without strategic dependencies, analyzing the dynamics of knowledge in multiagent systems is quite involved and computationally hard. As soon as strategic dependencies have to be taken into account, the dynamics of knowledge become even much more intricate. Here, we found the reason for the undecidability of the strategy problem for games with multiple players: While this is clear for sufficiently powerful specification formalisms, it is less obvious for winning conditions that allow positional winning strategies in the full information case. Indeed, the infinite character of such games that makes the strategy problem undecidable comes from the unbounded evolution of higher order knowledge. Of course, the winning condition plays a crucial role here as well but, as illustrated in Figure 6.5, already for safety conditions, unboundedly many histories may have to be taken into account by the members of the grand coalition.

Our ambition here was to find a way to represent the possible states of mind of the players and the dynamics of this knowledge explicitly, resulting in what we have advertised as *knowledge tracking*. For this, rather than just looking at the extensive form of a game on some given finite game graph, we have encapsulated complete states of mind of the members of the grand coalition in epistemic structures and each such epistemic structure is an individual position in the new game graph. Moreover, the transitions between these states are relativ to the strategic powers of the grand coalition in the original game. This representation allows for an abstraction by homomorphic equivalence in the case of *observable* winning conditions which yields a semi-algorithm that we have shown to terminate at least for hierarchical game graphs. This yields decidability for hierarchical games with observable deterministic context-free winning conditions. Moreover, for nonobservable regular winning conditions, it yields a partial solution method that does not give false positive answers. Finally, we have shown that the powerset construction which we have presented in Section 3.5, and which we have used extensively in Chapter 4, is indeed a special case of this generalized knowledge tracking construction.

With this epistemic analysis of multiplayer games and its very special case of two-player games, we have closed the circle to the beginnings of synthesis of winning strategies for interaction under partial information, in the context of computing systems and, in particular, nonterminating reactive systems.

Besides all the open problems that we have mentioned during our subsumption, there are a lot of broader issues and challenges in the theory of design and analysis of computing systems, and in particular synthesis of open nonterminating reactive systems, that lie ahead. We have mentioned in Section 2.1.4 several more general possibilities how imperfect information can be defined in games. Especially the notions of finite memory strategies and imperfect recall are particularly important in computing systems and should be closely investigated. A lot of research has also been done on several special cases of asynchronous systems, but classifying and relating different kinds of asynchronism is still a huge project. This holds also true for several other kinds of models and problems that are explored and which often emerge from different fields of research and come in plethora of different disguises. Their precise relationship and relative merits, particularly in view of practical applications, are far from being fully understood.

One of the absolute major issues in computing system design and analysis is the balance between expressivity and computational manageability of the models. Clearly, to model practically relevant scenarios, Church's original setting is not the ultimate deal. Besides the extension to systems with partial information and multiple processes, we have also mentioned lots of other variations and extensions, especially stochastic and hybrid systems. But we have seen that we hit the undecidability barrier pretty fast as soon as we try to incorporate more complex elements like multiple players and infinite state components. The same holds true for probabilistic and continuous aspects. The only way out is to advance sophisticatedly: Analyzing the reasons why problems become complex, and elaborately containing and tuning relevant parameters, with real life scenarios at the back of our mind, has led to numerous strong and useful decidability and tractability results. In the course of the past six chapters, we have pursued this line of research and in some directions, we have pushed the limits a little further.

Bibliography

[1] M. Abadi, L. Lamport, and P. Wolper. Realizable and Unrealizable Specifications of Reactive Systems. In *16th International Colloqium on Automata, Languages and Programming, ICALP '89*, pages 1–17. Springer, 1989.

[2] I. Adler. Directed tree-width examples. *Journal of Combinatorial Theory, Series B*, 97:718–725, 2007.

[3] R. Alur, P. Cerný, and S. Chaudhuri. Model Checking on Trees with Path Equivalences. In *13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS '07*, pages 664–678. Springer, 2007.

[4] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The Algorithmic Analysis of Hybrid Systems. *Theoretical Computer Science*, 138:3–34, 1995.

[5] R. Alur, C. Courcoubetis, T. A. Henzinger, and P. H. Ho. Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. In *Hybrid Systems*, pages 209–229. Springer, 1992.

[6] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-Time Temporal Logic. *Journal of the ACM*, 49:672–713, 2002.

[7] S. Arnborg. Efficient Algorithms for Combinatorial Problems on Graphs with Bounded Decomposability - A Survey. *BIT*, 25:2–23, 1985.

[8] A. Arnold, A. Vincent, and I. Walukiewicz. Games for Synthesis of Controllers with Partial Observation. *Theoretical Computer Science*, 303:7–34, 2003.

[9] A. Arnold and I. Walukiewicz. Nondeterministic Controllers of Nondeterministic Processes. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, volume 2 of *Texts in Logic and Games*, pages 29–52. Amsterdam University Press, 2007.

[10] A. Baltag, L. S. Moss, and S. Solecki. The Logic of Public Announcements, Common Knowledge and Private Suspicions. In *7th Conference on Theoretical Aspects of Rationality and Knowledge, TARK '98*, pages 43–56. Morgan Kaufmann, 1998.

[11] J. Barát. Directed Path-Width and Monotonicity in Digraph Searching. *Graphs and Combinatorics*, 22:161–172, 2006.

[12] W. Belkhir. Undirected Graphs of Entanglement 3. *CoRR*, abs/0904.1696, 2009.

[13] W. Belkhir and L. Santocanale. Undirected Graphs of Entanglement 2. In *27th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS '07*, pages 508–519. Springer, 2007.

[14] P. Bellenbaum and R. Diestel. Two Short Proofs Concerning Tree-Decompositions. *Combinatorics, Probability & Computing*, 11:541–547, 2002.

[15] S. Bensalem, D. Peled, and J. Sifakis. Knowledge Based Scheduling of Distributed Systems. In *Time for Verification, Essays in Memory of Amir Pnueli*, pages 26–41. Springer, 2010.

[16] J. van Benthem. Games in Dynamic-Epistemic Logic. *Bulletin of Economic Research*, 53:219–248, 2001.

[17] J. van Benthem. Rational Dynamics and Epistemic Logic in Games. *International Game Theory Review*, 9:13–45, 2007.

[18] J. van Benthem, J. Gerbrandy, T. Hoshi, and E. Pacuit. Merging Frameworks for Interaction. *Journal of Philosophical Logic*, 38:491–526, 2009.

[19] J. van Benthem, J. van Eijck, and B. P. Kooi. Logics of Communication and Change. *Information and Computation*, 204:1620–1662, 2006.

[20] O. Bernholtz, M. Y. Vardi, and P. Wolper. An Automata-Theoretic Approach to Branching-Time Model Checking (Extended Abstract). In *6th International Conference on Computer-Aided Verification, CAV '94*, pages 142–155. Springer, 1994.

[21] D. Berwanger. *Games and Logical Expressiveness.* PhD thesis, RWTH Aachen, 2005.

[22] D. Berwanger, K. Chatterjee, L. Doyen, T. A. Henzinger, and S. Raje. Strategy Construction for Parity Games with Imperfect Information. In *19th International Conference on Concurrency Theory, CONCUR '08*, pages 325–339. Springer, 2008.

[23] D. Berwanger, K. Chatterjee, M. De Wulf, L. Doyen, and T. A. Henzinger. Alpaga: A Tool for Solving Parity Games with Imperfect Information. In *15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS '09*, pages 58–61. Springer, 2009.

[24] D. Berwanger, A. Dawar, P. Hunter, and S. Kreutzer. DAG-Width and Parity Games. In *23rd Annual Symposium on Theoretical Aspects of Computer Science, STACS '06*, pages 524–536. Springer, 2006.

[25] D. Berwanger, A. Dawar, P. Hunter, S. Kreutzer, and J. Obdrzálek. The DAG-Width of Directed Graphs. *Journal of Combinatorial Theory, Series B*, 102:900–923, 2012.

[26] D. Berwanger and E. Grädel. Entanglement - A Measure for the Complexity of Directed Graphs with Applications to Logic and Games. In *11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR '04*, pages 209–223. Springer, 2004.

[27] D. Berwanger, E. Grädel, Ł. Kaiser, and R. Rabinovich. Entanglement and the Complexity of Directed Graphs. *Theoretical Computer Science*, 463:2–25, 2012. Special Issue on Theory and Applications of Graph Searching Problems.

[28] D. Berwanger, E. Grädel, and G. Lenzi. On the Variable Hierarchy of the Modal mu-Calculus. In *11th EACSL Annual Conference on Computer Science Logic, CSL '02*, pages 352–366. Springer, 2002.

[29] D. Berwanger, E. Grädel, and G. Lenzi. The Variable Hierarchy of the μ-Calculus is Strict. *Theory of Computing Systems*, 40:437–466, 2007.

[30] D. Berwanger and Ł. Kaiser. Information Tracking in Games on Graphs. *Journal of Logic, Language and Information*, 19:395–412, 2010.

[31] D. Berwanger, Ł. Kaiser, and B. Puchala. A Perfect Information Construction for Coordination in Games. In *31st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS '11*, pages 387–398. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011.

[32] D. Berwanger and G. Lenzi. The variable hierarchy of the μ-calculus is strict. In *STACS 2005, Proceedings of the 22nd Symposium on Theoretical Aspects of Computer Science*, pages 97–109. Springer, 2005.

[33] G. Bhat and R. Cleaveland. Efficient Model Checking via the Equational μ-Calculus. In *11th Annual IEEE Symposium on Logic in Computer, LICS 96*, pages 304–312. IEEE Computer Society, 1996.

[34] D. Bienstock and P. D. Seymour. Monotonicity in Graph Searching. *Journal of Algorithms*, 12:239–245, 1991.

[35] H. L. Bodlaender. Treewidth: Algorithmoc Techniques and Results. In *22nd International Symposium on Mathematical Foundations of Computer Science, MFCS '97*, pages 19–36. Springer, 1997.

[36] H. L. Bodlaender. Discovering Treewidth. In *31st Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM '05*, pages 1–16. Springer, 2005.

[37] H. L. Bodlaender, A. Grigoriev, and A. M. C. A. Koster. Treewidth Lower Bounds with Brambles. In *13th Annual European Symposium on Algorithms, ESA '05*, pages 391–402. Springer, 2005.

[38] U. Boker, O. Kupferman, and A. Rosenberg. Alternation Removal in Büchi Automata. In *37th International Colloqium on Automata, Languages and Programming, ICALP '10*, pages 76–87. Springer, 2010.

[39] J. C. Bradfield. The Modal μ-Calculus Alternation Hierarchy is Strict. *Theoretical Computer Science*, 195:133–153, 1998.

[40] J. C. Bradfield. Parity of Imperfection or Fixing Independence. In *12th EACSL Annual Conference on Computer Science Logic, CSL '03*, pages 72–85, 2003.

[41] J. C. Bradfield and S. Kreutzer. The Complexity of Independence-Friendly Fixpoint Logic. In *14th EACSL Annual Conference on Computer Science Logic, CSL '05*, pages 355–368. Springer, 2005.

[42] T. Brázdil, V. Brozek, V. Forejt, and A. Kucera. Stochastic Games with Branching-Time Winning Objectives. In *21st IEEE Symposium on Logic in Computer Science, LICS '06*, pages 349–358. IEEE Computer Society, 2006.

[43] J. R. Büchi. On a Decision Method in Restricted Second Order Arithmetic. In *1960 International Congress on Logic, Methodology and Philosophy of Science*, pages 1–11. Stanford University Press, 1962.

[44] J. R. Büchi and L. H. Landweber. Solving Sequential Conditions by Finite-State Strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.

[45] N. Buhrke, H. Lescow, and J. Voege. Strategy Construction in Infinite Games with Streett and Rabin Chain Winning Conditions. In *2nd International Workshop on Tools and Algorithms for Construction and Analysis of Systems, TACAS '96*, pages 207–225. Springer, 1996.

[46] F. Canavoi, E. Grädel, and R. Rabinovich. The Discrete Strategy Improvement Algorithm for Parity Games and Complexity Measures for Directed Graphs. In *Third Symposium on Games, Automata, Logic, and Formal Verification, GANDALF '12*, EPTCS, 2012.

[47] F. Cassez, A. David, K. G. Larsen, D. Lime, and J.-F. Raskin. Timed Control with Observation Based and Stuttering Invariant Strategies. In *5th International Symposium on Automated Technology for Verification and Analysis, ATVA '07*, pages 192–206. Springer, 2007.

[48] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28:114–133, 1981.

[49] A K. Chandra and L. J. Stockmeyer. Alternation. In *17th Annual Symposium on Foundations of Computer Science, FOCS '76*, pages 98–108. IEEE Computer Society, 1976.

[50] K. Chatterjee. *Stochastic ω-Regular Games.* PhD thesis, University of California at Berkeley, 2007.

[51] K. Chatterjee and L. Doyen. The Complexity of Partial-Observation Parity Games. In *17th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR 17*, pages 1–14. Springer, 2010.

[52] K. Chatterjee, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Algorithms for Omega-Regular Games with Imperfect Information. *Logical Methods in Computer Science*, 3(3), 2007.

[53] K. Chatterjee, M. Jurdziński, and T. A. Henzinger. Simple Stochastic Parity Games. In *12th EACSL Annual Conference on Computer Science Logic, CSL '03*, pages 100–113. Springer, 2003.

[54] A. Church. Applications of Recursive Arithmetic to the Problem of Circuit Synthesis. In *Summaries of the Summer Institute of Symbolic Logic*, volume 1, pages 3–50. Cornel University, Ithaca, 1957.

[55] A. Church. Logic, Arithmetic, and Automata. In *International Congress of Mathematicians*, pages 23–35. Institut Mittag-Leffler, Djursholm, Sweden, 1963.

[56] E. M. Clarke and E. A. Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Logic of Programs, Workshop*, pages 52–71, 1981.

[57] R. S. Cohen and A. Y. Gold. Theory of Omega-Languages. I: Characterizations of ω-Context-Free Languages. *Journal of Computer and System Sciences*, 15:169–184, 1977.

[58] R. S. Cohen and A. Y. Gold. Theory of Omega-Languages. II: A Study of Various Models of ω-Type Generation and Recognition. *Journal of Computer and System Sciences*, 15:169–184, 1977.

[59] R. S. Cohen and A. Y. Gold. Omega-Computations on Deterministic Pushdown Machines. *Journal of Computer and System Sciences*, 16:275–300, 1978.

[60] B. Courcelle. Graph Rewriting: An Algebraic and Logic Approach. In *Formal Models and Semantics*, volume B of *Handbook of Theoretical Computer Science*, pages 193–242. Elsevier and MIT Press, 1990.

[61] M. Dam. CTL* and ECTL* as Fragments of the Modal μ-Calculus. In *17th Colloquium on Trees in Algebra and Programming, CAAP '92*, pages 145–164. Springer, 1992.

[62] L. de Alfaro. Stochastic Transition Systems. In *9th International Conference on Concurrency Theory, CONCUR '98*, pages 423–438. Springer, 1998.

[63] L. de Alfaro, T. A. Henzinger, and O. Kupferman. Concurrent Reachability Games. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98*. IEEE Computer Society, 1998.

[64] L. de Alfaro, T. A. Henzinger, and R. Majumdar. From Verification to Control: Dynamic Programs for Omega-Regular Objectives. In *Sixteenth Annual IEEE Symposium on Logic in Computer Science, LICS '01*, pages 279–290. IEEE Computer Society, 2001.

[65] M. de Wulf, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Antichains: A New Algorithm for Checking Universality of Finite Automata. In *18th International Conference on Computer-Aided Verification, CAV '06*, pages 17–30, 2006.

[66] M. de Wulf, L. Doyen, and J.-F. Raskin. A Lattice Theory for Solving Games of Imperfect Information. In *9th International Workshop on Hybrid Systems: Computation and Control, HSCC '06*, pages 153–168. Springer, 2006.

[67] R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, 4th edition, 2010. Corrected Reprint 2012.

[68] R. Dimitrova, B. Finkbeiner, M. Kovács, M. N. Rabe, and H. Seidl. Model Checking Information Flow in Reactive Systems. In *13th International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI '12*, pages 169–185. Springer, 2012.

[69] H. P. van Ditmarsch. *Knowledge Games.* PhD thesis, Rijksuniversiteit Groningen, 2000.

[70] B. Dixon, E. Hoogland, and P. McIntosh, editors. *Modelling Intelligent Interaction (LogICCC).* European Science Foundation (ESF), August 2011. ESF EUROCORES Programme Logics in the Humanities, Social and Computational Sciences. A Retrospective.

[71] S. Dziembowski, M. Jurdziński, and I. Walukiewicz. How Much Memory is Needed to Win Infinite Games? In *12th Annual IEEE Symposium on Logic in Computer Science, LICS '97*, pages 99–110. IEEE Computer Society, 1997.

[72] E. A. Emerson. Automata, Tableaux and Temporal Logics (Extended Abstract). In *Logic of Programs, Conference*, pages 79–88. Springer, 1985.

[73] E. A. Emerson and J. Y. Halpern. "Sometimes" and "Not Never" Revisited: On Branching Versus Linear Time. In *10th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '83*, pages 127–140. ACM, 1983.

[74] E. A. Emerson and C. S. Jutla. The Complexity of Tree Automata and Logics of Programs (Extended Abstract). In *29th Annual Symposium on Foundations of Computer Science, FOCS '88*, pages 328–337. IEEE Computer Society, 1988.

[75] E. A. Emerson and C. S. Jutla. Tree Automata, Mu-Calculus and Determinacy (Extended Abstract). In *32nd Annual Symposium on Foundations of Computer Science, FOCS '91*, pages 368–377, 1991.

[76] E. A. Emerson, C. S. Jutla, and A. P. Sistla. On Model-Checking for Fragments of μ-Calculus. In *5th International Conference on Computer Aided Verification, CAV '93*, pages 385–396. Springer, 1993.

[77] E. A. Emerson and A. P. Sistla. Deciding Full Branching Time Logic. *Information and Control*, 61:175–201, 1984.

[78] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge.* The MIT Press, 2003.

[79] J. Fearnley, D. Peled, and S. Schewe. Synthesis of Succinct Systems. In *10th International Symposium on Automated Technology for Verification and Analysis*, pages 208–222. Springer, 2012.

[80] B. Finkbeiner and S. Schewe. Uniform Distributed Synthesis. In *20th IEEE Symposium on Logic in Computer Science, LICS '05*, pages 321–330. IEEE Computer Society, 2005.

[81] O. Finkel. Topological Properties of Omega Context-Free Languages. *Theoretical Computer Science*, 262:669–697, 2001.

[82] M. J. Fischer and R. E. Ladner. Propositional Dynamic Logic of Regular Programs. *Journal of Computer and System Sciences*, 18:194–211, 1979.

[83] J. Flum, E. Grädel, and T. Wilke, editors. *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, volume 2 of *Texts in Logic and Games.* Amsterdam University Press, 2007.

[84] F. V. Fomin, P. Fraigniaud, and N. Nisse. Nondeterministic Graph Searching: From Pathwidth to Treewidth. In *30th International Symposium on Mathematical Foundations of Computer Science, MFCS '05*, pages 364–375. Springer, 2005.

[85] W. Fridman. Formats of Winning Strategies for Six Types of Pushdown Games. In *First Symposium on Games, Automata, Logic, and Formal Verification, GANDALF '10*, EPTCS, pages 132–145, 2010.

[86] W. Fridman and B. Puchala. Distributed Synthesis for Regular and Contextfree Specifications. In *36th International Symposium on Mathematical Foundations of Computer Science, MFCS '11*, pages 532–543. Springer, 2011.

[87] O. Friedmann. An Exponential Lower Bound for the Parity Game Strategy Improvement Algorithm as We Know it. In *24th IEEE Symposium on Logic in Computer Science, LICS '09*, pages 145–156. IEEE Computer Society, 2009.

[88] O. Friedmann. A Subexponential Lower Bound for Zadeh's Pivoting Rule for Solving Linear Programs and Games. In *Integer Programming and Combinatoral Optimization - 15th International Conference, IPCO '11*, pages 192–206. Springer, 2011.

[89] O. Friedmann, T. D. Hansen, and U. Zwick. Subexponential Lower Bounds for Randomized Pivoting Rules for the Simplex Algorithm. In *43rd Annual ACM Symposium on Theory of Computing, STOC '11*, pages 283–292. ACM, 2011.

[90] D. Gale and F. M. Stewart. Infinite Games with Perfect Information. In *Contributions to the Theory of Games*, volume 28 of *Annals of Mathematics Studies*, pages 245–266. Princeton University Press, 1953.

[91] R. Ganian, P. Hlinený, J. Kneis, D. Meister, J. Obdrzálek, P. Rossmanith, and S. Sikdar. Are There Any Good Digraph Width Measures? In *5th International Symposium on Parameterized and Exact Computation, IPEC '10*, pages 135–146. Springer, 2010.

[92] P. Gastin, B. Lerman, and M. Zeitoun. Distributed Games and Distributed Control for Asynchronous Systems. In *6th Latin American Symposium on Theoretical Informatics, LATIN '04*, pages 455–465. Springer, 2004.

[93] P. Gastin and N. Sznajder. Decidability of well-connectedness for distributed synthesis. *Information Processing Letters*, 2012. To appear.

[94] P. Gastin, N. Sznajder, and M. Zeitoun. Distributed Synthesis for Well-Connected Architectures. In *26th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS '06*, pages 321–332. Springer, 2006.

[95] J. Gerbrandy. Dynamic Epistemic Logic. In *Logic, Language and Computation*, volume 2, pages 67–84. CSLI Publications, Stanford, 1999.

[96] J. Gerbrandy. Epistemic Equivalence and Bisimulation, 2005. Online available at http://www.di.unito.it/~gerbrand/papers/.

[97] J. Gerbrandy and W. Groeneveld. Reasoning about Information Change. *Journal of Logic, Language and Information*, 6:147–169, 1997.

[98] E. Grädel. Back and Forth Between Logics and Games. In *Lectures in Game Theory for Computer Scientists*, pages 99–145. Springer, 2011.

[99] E. Grädel, Ł. Kaiser, and R. Rabinovich. Directed Graphs of Entanglement Two. In *17th International Symposium on Fundamentals of Computation Theory, FCT '09*, pages 169–181. Springer, 2009.

[100] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[101] E. Grädel and M. Ummels. Solution Concepts and Algorithms for Infinite Multiplayer Games. In *New Perspectives on Games and Interaction*, volume 4 of *Texts in Logic and Games*, pages 151–178. Amsterdam University Press, 2008.

[102] Y. Gurevich and L. Harrington. Trees, Automata, and Games. In *14th Annual ACM Symposium on Theory of Computing, STOC '82*, pages 60–65, 1982.

[103] R. Halin. S-functions for graphs. *Journal of Geometry*, 8:171–186, 1976.

[104] J. Y. Halpern, R. van der Meyden, and M. Y. Vardi. Complete Axiomatizations for Reasoning about Knowledge and Time. *SIAM Journal on Computing*, 33:674–703, 2004.

[105] J. Y. Halpern and Y. Moses. Knowledge and Common Knowledge in a Distributed Environment. In *Third Annual ACM Symposium on Principles of Distributed Computing, PODC '84*, pages 50–61. ACM, 1984.

[106] D. Harel and D. Raz. Deciding Emptiness for Stack Automata on Infinite Trees. *Information and Computation*, 113:278–299, 1994.

[107] L. Henkin. Some Remarks on Infinitely Long Formulas. In *Infinistic Methods*, pages 167–183. Pergamon Press, 1961.

[108] T. A. Henzinger and N. Piterman. Solving Games Without Determinization. In *15th EACSL Annual Conference on Computer Science Logic, CSL '06*, pages 395–410. Springer, 2006.

[109] J. Hintikka. *Knowledge and Belief - An Introduction to the Logic of the Two Notions.* Cornell University Press, 1962. Republished 2005 by King's College Publications.

[110] J. Hintikka and G. Sandu. A Revolution in Logic? *Nordic Journal of Philosophical Logic*, 1:169–183, 1996.

[111] F. Horn. Explicit Muller Games are PTIME. In *28th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS '08*, pages 235–243. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2008.

[112] P. Hunter. Losing the +1. Or Directed Path-Width Games are Monotone. Online available at http://www.comlab.ox.ac.uk/people/paul.hunter/papers/losing.pdf, 2006.

[113] P. Hunter. *Complexity and Infinite Games on Finite Graphs.* PhD thesis, Computer Laboratory, University of Cambridge, 2007.

[114] P. Hunter and S. Kreutzer. Digraph Measures: Kelly Decompositions, Games, and Orderings. In *18th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '07*, pages 637–644. SIAM, 2007.

[115] P. Hunter and S. Kreutzer. Digraph Measures: Kelly Decompositions, Games, and Orderings. *Theoretical Computer Science*, 399:206–219, 2008.

[116] T. Johnson, N. Robertson, P. D. Seymour, and R. Thomas. Directed Tree-Width. *Journal of Combinatorial Theory, Series B*, 82:138–154, 2001.

[117] N. D. Jones. Blindfold Games are Harder than Games with Perfect Information. *Bulletin European Association for Theoretical Computer Science*, 6:4–7, 1978.

[118] M. Jurdziński. *Games for Verification: Algorithmic Issues.* PhD thesis, University of Aarhus, 2000.

[119] M. Jurdziński, M. Paterson, and U. Zwick. A Deterministic Subexponential Algorithm for Solving Parity Games. In *7TH Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '06*, pages 117–123, 2006.

[120] L. M. Kirousis and C. H. Papadimitriou. Searching and Pebbling. *Theoretical Computer Science*, 47:205–218, 1986.

[121] B. Kooi and J. van Benthem. Reduction Axioms for Epistemic Actions. Manuscript, 2004. Online available at http://irs.ub.rug.nl/dbi/4432690db3a02.

[122] D. C. Kozen. Results on the Propositional μ-Calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[123] S. Kreutzer and S. Ordyniak. Digraph Decompositions and Monotonicity in Digraph Searching. In *34th International Workshop on Graph-Theoretic Concepts in Computer Science, WG '08*, pages 336–347. Springer, 2008.

[124] H. W. Kuhn. Extensive Games and the Problem of Information. In *Contributions to the Theory of Games II*, volume 28 of *Annals of Mathematics Studies*, pages 193–216. Princeton University Press, 1953.

[125] O. Kupferman, N. Piterman, and M. Y. Vardi. Pushdown Specifications. In *9th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR '02*, pages 262–277. Springer, 2002.

[126] O. Kupferman and M. Y. Vardi. Synthesis with Incomplete Informatio. In *2nd International Conference on Temporal Logic, ICTL '97*, pages 91–106. Kluwer Academic Publishers, 1997.

[127] O. Kupferman and M. Y. Vardi. Church's Problem Revisited. *The Bulletin of Symbolic Logic*, 5:245–263, 1999.

[128] O. Kupferman and M. Y. Vardi. An Automata-Theoretic Approach to Reasoning about Infinite-State Systems. In *12th International Conference on Computer-Aided Verification, CAV '00*, pages 36–52. Springer, 2000.

[129] O. Kupferman and M. Y. Vardi. Synthesizing Distributed Systems. In *16th Annual IEEE Symposium on Logic in Computer Science, LICS '01*, pages 389–398. IEEE Computer Society, 2001.

[130] O. Kupferman and M. Y. Vardi. Safraless Decision Procedures. In *46th IEEE Symposium on Foundations of Computer Science, FOCS '05*, pages 531–540. IEEE Computer Society, 2005.

[131] L. Lamport. "Sometime" is Sometimes "Not Never" - On the Temporal Logic of Programs. In *7th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '80*, pages 174–185. ACM, 1980.

[132] A. S. LaPaugh. Recontamination Does Not Help to Search a Graph. *Journal of the ACM*, 40:224–245, 1993.

[133] C. Löding. Methods for the Transformation of omega-Automata: Complexity and Connection to Second Order Logic. Master's thesis, Christian-Albrechts-Universität zu Kiel, 1998.

[134] C. Löding. Optimal Bounds for Transformations of omega-Automata. In *19th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS '99*, pages 97–109. Springer, 1999.

[135] P. Madhusudan. *Control and Synthesis of Open Reactive Systems.* PhD thesis, University of Madras, Chennai, India, 2001.

[136] P. Madhusudan. Synthesizing reactive programs. In *20th EACSL Annual Conference on Computer Science Logic, CSL '11*, pages 428–442. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011.

[137] P. Madhusudan and P. S. Thiagarajan. Distributed Controller Synthesis for Local Specifications. In *28th International Colloquium on Automata, Languages and Programming, ICALP '01*, pages 396–407. Springer, 2001.

[138] Z. Manna and R. J. Waldinger. A Deductive Approach to Program Synthesis. *ACM Transactions on Programming Languages and Systems*, 2(1):90–121, 1980.

[139] Z. Manna and P. Wolper. Synthesis of Communicating Processes from Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems*, 6(1):68–93, 1984.

[140] D. A. Martin. Borel Determinacy. *Annals of Mathematics*, 102:363–371, 1975.

[141] R. McNaughton. Testing and Generating Infinite Sequences by a Finite Automaton. *Information and Control*, 9:521–530, 1966.

[142] R. McNaughton. Infinite Games Played on Finite Graphs. *Annals of Pure and Applied Logic*, 65:149–184, 1993.

[143] R. van der Meyden and M. Y. Vardi. Synthesis from Knowledge-Based Specifications (Extended Abstract). In *9th International Conference on Concurrency Theory, CONCUR '98:*, pages 34–49. Springer, 1998.

[144] R. van der Meyden and T. Wilke. Synthesis of Distributed Systems from Knowledge-based Specifications. In *16th International Conference on Concurrency Theory, CONCUR '05*, pages 562–576. Springer, 2005.

[145] A. R. Meyer. Weak Monadic Second Order Theory of Successor is not Elementary-Recursive. In *Logic Colloquium*, pages 132–154. Springer, 1975.

[146] J. S. Miller and L. S. Moss. The Undecidability of Iterated Modal Relativization. *Studia Logica*, 79:373–407, 2005.

[147] S. Miyano and T. Hayashi. Alternating Finite Automata on omega-Words. *Theoretical Computer Science*, 32:321–330, 1984.

[148] S. Mohalik and I. Walukiewicz. Distributed Games. In *23rd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS '03*, pages 338–351. Springer, 2003.

[149] A. W. Mostowski. Regular Expressions for Infinite Trees and a Standard Form of Automata. In *Computation Theory - Fifth Symposium*, pages 157–168. Springer, 1984.

[150] A. W. Mostowski. Games with Forbidden Positions. Technical Report 78, Uniwersytet Gdański, 1991.

[151] D. E. Muller. Infinite Sequences and Finite Machines. In *4th Annual Symposium on Switching Circuit Theory and Logical Design, SWCT (FOCS) '63*, pages 3–16. IEEE Computer Society, 1963.

[152] D. E. Muller and P. E. Schupp. Alternating Automata on Infinite Objects, Determinacy and Rabin's Theorem. In *Automata on Infinite Words, Ecole de Printemps d'Informatique Théorique,*, pages 100–107. Springer, 1985.

[153] D. E. Muller and P. E. Schupp. Alternating Automata on Infinite Trees. *Theoretical Compututer Science*, 54:267–276, 1987.

[154] D. E. Muller and P. E. Schupp. Simulating Alternating Tree Automata by Nondeterministic Automata: New Results and new Proofs of the Theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141:69–107, 1995.

[155] A. Muscholl. Taming distributed asynchronous systems. In *21st International Conference on Concurrency Theory, CONCUR '10*, pages 40–47. Springer, 2010.

[156] J. Obdržálek. Fast mu-Calculus Model Checking when Tree-Width is Bounded. In *15th International Conference on Computer Aided Verification, CAV '03*, pages 80–92. Springer, 2003.

[157] J. Obdržálek. *Algorithmic Analysis of Parity Games.* PhD thesis, School of Informatics, University of Edinburgh, 2006.

[158] J. Obdržálek. DAG-Width: Connectivity Measure for Directed Graphs. In *17 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '06*, pages 814–821. ACM, 2006.

[159] J. Obdrzálek. Clique-width and parity games. In *16th EACSL Annual Conference on Computer Science Logic, CSL '07*, pages 54–68. Springer, 2007.

[160] M. J. Osborne and A. Rubinstein. *A Course in Game Theory.* MIT Press, 1994.

[161] C. H. Papadimitriou. *Computational Complexity.* Addison Wesley, 1994.

[162] D. Peled and S. Schewe. Practical Distributed Control Synthesis. In *13th International Workshop on Verification of Infinite-State Systems*, EPTCS, pages 2–17, 2011.

[163] G. L. Peterson and J. H. Reif. Multiple-Person Alternation. In *20th Annual Symposium on Foundations of Computer Science, FOCS '79*, pages 348–363. IEEE Computer Society, 1979.

[164] S. Pinchinat and O. Serre. Emptiness of Alternating Parity Tree Automata Using Games with Imperfect Information. Technical Report 1992, IRISA, 2012.

[165] N. Piterman. From Nondeterministic Büchi and Streett Automata to Deterministic Parity Automata. In *21st IEEE Symposium on Logic in Computer Science, LICS '06*, pages 255–264. IEEE Computer Society, 2006.

[166] J. A. Plaza. Logics of Public Communications. In *4th International Symposium on Methodologies for Intelligent Systems*, pages 201–216, 1989.

[167] A. Pnueli. The Temporal Logic of Programs. In *18th Annual IEEE Symposium on Foundations of Computer Science, FOCS '77*, pages 46–57. IEEE Computer Society, 1977.

[168] A. Pnueli. Linear and Branching Structures in the Semantics and Logics of Reactive Systems. In *12th International Colloqium on Automata, Languages and Programming, ICALP '85*, pages 15–32. Springer, 1985.

[169] A. Pnueli and R. Rosner. On the Synthesis of a Reactive Module. In *16th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'89*, pages 179–190. ACM, 1989.

[170] A. Pnueli and R. Rosner. On the Synthesis of an Asynchronous Reactive Module. In *16th International Colloquium on Automata, Languages and Programming, ICALP '89*, pages 652–671. Springer, 1989.

[171] A. Pnueli and R. Rosner. Distributed Reactive Systems are Hard to Synthesize. In *31st Annual Symposium on Foundations of Computer Science, FOCS '90*, pages 746–757. IEEE Computer Society, 1990.

[172] B. Puchala. Infinite Two Player Games with Partial Information: Logic and Algorithms. Master's thesis, RWTH Aachen, 2008.

[173] B. Puchala. Asynchronous Omega-Regular Games with Partial Information. In *35th International Symposium on Mathematical Foundations of Computer Science, MFCS '10*, pages 592–603. Springer, 2010.

[174] B. Puchala and R. Rabinovich. Parity Games with Partial Information Played on Graphs of Bounded Complexity. In *35th International Symposium on Mathematical Foundations of Computer Science, MFCS '10*, pages 604–615. Springer, 2010.

[175] B. Puchala and R. Rabinovich. Graph Searching, Parity Games and Imperfect Information. *CoRR*, abs/1110.5575, 2011.

[176] M. O. Rabin. Decidability of Second-Order Theories and Automata on Infinite Trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

[177] M. O. Rabin. *Automata on Infinite Objects and Church's Problem.* American Mathematical Society, 1972.

[178] A. Rabinovich and W. Thomas. Logical Refinements of Church's Problem. In *16th EACSL Annual Conference on Computer Science Logic, CSL '07*, pages 69–83. Springer, 2007.

[179] P. J. G. Ramadge and W. M. Wonham. The Control of Discrete Event Systems. *Proceedings of the IEEE*, 77:81–98, 1989.

[180] R. Ramanujam and S. E. Simon. A Communication Based Model for Games of Imperfect Information. In *21st International Conference on Concurrency Theory, CONCUR '10*, pages 509–523. Springer, 2010.

[181] J. H. Reif. Universal Games of Incomplete Information. In *11th Annual ACM Symposium on Theory of Computing, STOC '79*, pages 288–308. ACM, 1979.

[182] J. H. Reif. The Complexity of Two-player Games of Incomplete Information. *Journal of Computer and System Sciences*, 29:274–301, 1984.

[183] D. Richerby and D. M. Thilikos. Graph Searching in a Crime Wave. *SIAM Journal on Discrete Mathematics*, 23:349–368, 2009.

[184] N. Robertson and P. D. Seymour. Graph minors. III. Planar Tree-Width. *Journal of Combinatorial Theory, Series B*, 36:49–64, 1984.

[185] N. Robertson and P. D. Seymour. Graph Minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92:325–357, 2004.

[186] K. Rudie and W. Wonham. Think Globally, Act Locally. *IEEE Transactions on Automatic Control*, 37:1692–1708, 1992.

[187] S. Safra. On the Complexity of ω-Automata. In *29th IEEE Symposium on Foundations of Computer Science, FOCS '88*, pages 319–327. IEEE Computer Society, 1988.

[188] S. Safra. Exponential Determinization for omega-Automata with Strong-Fairness Acceptance Condition (Extended Abstract). In *24th Annual ACM Symposium on Theory of Computing, STOC '92*, pages 275–282. ACM, 1992.

[189] S. Schewe. Solving Parity Games in Big Steps. In *27th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS '07*, pages 449–460. Springer, 2007.

[190] S. Schewe. An Optimal Strategy Improvement Algorithm for Solving Parity and Payoff Games. In *17th EACSL Annual Conference on Computer Science Logic, CSL '08*, pages 369–384. Springer, 2008.

[191] S. Schewe. *Synthesis of Distributed Systems.* PhD thesis, Universität des Saarlandes, 2008.

[192] S. Schewe. Büchi Complementation Made Tight. In *26th International Symposium on Theoretical Aspects of Computer Science, STACS '09*, pages 661–672. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2009.

[193] S. Schewe. Tighter Bounds for the Determinisation of Büchi Automata. In *12th International Conference on Foundations of Software Science and Computation Structures, FoSSaCS '09*, pages 167–181. Springer, 2009.

[194] S. Schewe and B. Finkbeiner. Synthesis of Asynchronous Systems. In *16th International Symposium on Logic-Based Program Synthesis and Transformation, LOPSTR '06*, pages 127–142. Springer, 2006.

[195] S. Schewe and B. Finkbeiner. Bounded Synthesis. In *5th International Symposium on Automated Technology for Verification and Analysis, ATVA '07*, pages 474–488. Springer, 2007.

[196] S. Schewe and B. Finkbeiner. Distributed Synthesis for Alternating-Time Logics. In *5th International Symposium on Automated Technology for Verification and Analysis, ATVA '07*, pages 268–283. Springer, 2007.

[197] H. Seidl. Deciding Equivalence of Finite Tree Automata. *SIAM Jounal on Computing*, 19:424–437, 1990.

[198] P. D. Seymour and R. Thomas. Graph Searching and a Min-Max Theorem for Tree-Width. *Journal of Combinatorial Theory, Series B*, 58:22–33, 1993.

[199] C. Stirling. Games and Modal Mu-Calculus. In *Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems, TACAS '96*, pages 298–312. Springer, 1996.

[200] W. Thomas. Automata on Infinite Objects. In *Handbook of Theoretical Computer Science, Volume B : Formal Models and Semantics*, pages 133–192. Elsevier and MIT Press, 1990.

[201] W. Thomas. Infinite Trees and Automation-Definable Relations over omega-Words. *Theoretical Computer Science*, 103:143–159, 1992.

[202] W. Thomas. On the Synthesis of Strategies in Infinite Games. In *12th Annual Symposium on Theoretical Aspects of Computer Science, STACS '95*, pages 1–13. Springer, 1995.

[203] W. Thomas. Languages, Automata, and Logic. In *Handbook of Formal Languages*, volume 3, pages 389–455. Springer, 1997.

[204] W. Thomas. Solution of Church's Problem: A Tutorial. In *New Perspectives on Games and interaction*, volume 4. Amsterdam University Press, 2008.

[205] W. Thomas. Facets of Synthesis: Revisiting Church's Problem. In *Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures, FOSSACS '09*, pages 1–14. Springer, 2009.

[206] M. Ummels. *Stochastic Multiplayer Games: Theory and Algorithms.* PhD thesis, RWTH Aachen University, 2010.

[207] J. Väänänen. *Dependence Logic.* Cambridge University Press, 2007.

[208] R. van der Meyden and N. V. Shilov. Model Checking Knowledge and Time in Systems with Perfect Recall (Extended Abstract). In *19th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS '99*, pages 432–445. Springer, 1999.

[209] H. van Ditmarsch, W. van der Hoek, and B. Kooi. *Dynamic Epistemic Logic*, volume 337 of *Synthese Library.* Springer, 1 edition, 2007.

[210] J. van Eijck. Reducing Dynamic Epistemic Logic to PDL by Program Transformation. Technical Report SEN-E0423, Centrum voor Wiskunde en Informatica, 2004.

[211] M. Y. Vardi. An Automata-Theoretic Approach to Fair Realizability and Synthesis. In *7th International Conference on Computer-Aided Verification, CAV '95*, pages 267–278. Springer, 1995.

[212] M. Y. Vardi. An Automata-Theoretic Approach to Linear Temporal Logic. In *Logics for Concurrency - Structure versus Automata (8th Banff Higher Order Workshop)*, pages 238–266. Springer, 1995.

[213] M. Y. Vardi. Reasoning about The Past with Two-Way Automata. In *25th International Colloqium on Automata, Languages and Programming, ICALP '98*, pages 628–641. Springer, 1998.

[214] M. Y. Vardi. Branching vs. Linear Time: Final Showdown. In *7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS '01*, pages 1–22. Springer, 2001.

[215] M. Y. Vardi and T. Wilke. Automata: From Logics to Algorithms. In J. Flum, E- Grädel, and T. Wilke, editors, *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, volume 2 of *Texts in Logic and Games*, pages 629–736. Amsterdam University Press, 2007.

[216] M. Y. Vardi and P. Wolper. Automata-Theoretic Techniques for Modal Logics of Programs. *Journal of Computer and System Sciences*, 32:183–221, 1986.

[217] K. Wagner and G. Wechsung. *Computational Complexity.* D. Reidel Publishing, 1986.

[218] R. J. Waldinger and R. C. T. Lee. PROW: A Step Toward Automatic Program Writing. In *1st International Joint Conference on Artificial Intelligence, IJCAI '69*, pages 241–252. William Kaufmann, 1969.

[219] I. Walukiewicz. Pushdown Processes: Games and Model Checking. In *8th International Conference on Computer Aided Verification, CAV '96*, pages 62–74. Springer, 1996.

[220] I. Walukiewicz. A Landscape with Games in the Background. In *19th IEEE Symposium on Logic in Computer Science, LICS '04*, pages 356–366. IEEE Computer Society, 2004.

[221] H. Wong-Toi and D. L. Dill. Synthesizing Processes and Schedulers from Temporal Specifications. In *2nd International Workshop on Computer Aided Verification, CAV '90*, pages 272–281. Springer, 1991.

[222] W. Zielonka. Infinite Games on Finitely Coloured Graphs with Applications to Automata on Infinite Trees. *Theoretical Computer Science*, 200:135–183, 1998.