
Wouter Kuijper
Compositional
Synthesis
of Safety
Controllers

Compositional Synthesis of Safety Controllers

Wouter Kuijper

Graduation committee:

Chairman:	Prof. dr. ir. A. J. Mouthaan
Promoter:	Prof. dr. J. C. van de Pol

Members:

Prof. dr. B. R. H. M. Haverkort	University of Twente
Prof. dr. J. J. M. Hooman	Radboud University Nijmegen
Prof. dr. D. A. Peled	Bar Ilan University, Israel
Prof. dr. R. J. Wieringa	University of Twente

IPA Ph.D.-thesis Series No. 2012-16
Institute for Programming research and Algorithmics

ISBN 978-90-365-3487-1
Digital edition: `http://dx.doi.org/10.3990/1.9789036534871`

The work in this thesis is supported by the Netherlands' Organization for Scientic Research (NWO) under project no. 600.065.120.24N20

Printer: Ipskamp Drukkers
Cover design: Gabriella Sperotto

COMPOSITIONAL SYNTHESIS OF SAFETY CONTROLLERS

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
Prof. dr. H. Brinksma,
volgens besluit van het College voor Promoties,
in het openbaar te verdedigen
op vrijdag 7 december 2012 om 14.45 uur

door

Wouter Kuijper

geboren op 16 april 1980
te Katwijk aan Zee

Dit proefschrift is goedgekeurd door:
Prof. dr. J. C. van de Pol (promotor)

Abstract

In my thesis I investigate compositional techniques for synthesis of safety controllers. A safety controller, in this context, is a state machine that gives the set of safe control outputs for every possible sequence of observations from the plant under control. Compositionality, in this context, refers to the ability to compose the plant model with a safety controller that is derived in a local context, meaning we only consider a selected subsets of the full set of plant model components.

The main research question addressed in the thesis is how compositional techniques can have a beneficial effect on scalability. Here scalability applies to the way the running time and memory requirements of the synthesis algorithm increase with the number of plant model components. The working hypothesis was that compositionality should indeed have a beneficial impact on scalability. The intuition behind this is that using compositional techniques we should be able to avoid or at least partly alleviate the type of state explosion problem that is typically seen when synthesizing controllers for larger plant models that consist of the paralel composition of multiple plant model components.

The experimental results presented in the thesis are positive in the sense that they indeed support the working hypothesis. We see on a natural example the compositional algorithm exhibits linear scaling behavior whereas the monolithic (non compositional) algorithm exhibits super–exponential scaling behavior. We see this even for an example that intrinsically requires a combination of local control constraints and a global control constraint, where the local constraints are each in turn dependent on a small number of adjacent plant components, whereas the global constraint is intrinsically dependent on all plant model components simultaneously.

A first main contribution is a symbolic algorithm that works directly on a compact symbolic representation of the controller thereby avoiding explicit construction of the underlying state graph. The algorithm works by refining

the representation of the control strategy in a counterexample driven manner. Upon termination the algorithm will yield a symbolic representation of the most permissive, safe control strategy for the given plant model. The algorithm is specifically designed for models that feature partial observability, meaning that certain internal state of the plant model is not directly observable by the controller.

A second main contribution is a compositional technique that also explicitly takes partial observability into account. For this we develop a compositional algorithm that invokes the aforementioned strategy refinement algorithm repeatedly. In particular the compositional algorithm performs a two step synthesis process for each relevant subset of the plant model: (1) computation of the *local context* which effectively forms a local overapproximation of the allowable behavior, (2) computation of the *local controller* which effectively forms a local underapproximation of the denyable behavior. We prove that upon termination of the algorithm the context and the controller signatures coincide and we obtain precisely the desired *most permissive safety controller*, yet constructed in an incremental, compositional fashion.

What sets these contributions apart from other contributions in the field is the fact that I consider compositionality in combination with partial observability, and also the fact that the resulting compositional algorithm does not rely on any type of explicit, congruence based state minimization procedure. Even though the two aforementioned main contributions can be considered separately, it may be more informative to view them in combination: it is the compositional algorithm that manages to exploit to the maximal extent the symbolic strategy refinement algorithm that underlies it, or, vice versa, it is the symbolic strategy refinement algorithm that enables the compositional algorithm that relies on it to scale well on larger problem instances.

Acknowledgements

The people that contributed to this thesis are many. I can only hope that, somehow, the end–result does them all justice and repays, in some small way, the freedom that I enjoyed in preparing it. Because, no matter what I might have said at various points along the way, being able to do research is just great, and I'm grateful for having had the opportunity.

First and foremost I would like to thank my supervisor and promotor Jaco van de Pol who really helped me, through a lot of discussion, a lot of trust, a lot of feedback, a lot of coaching, and a lot of patience. I'd like to thank the members of my committee who generously provided fresh feedback on the thesis, and made many suggestions for improving it. Many thanks also to the staff, students and all my colleagues at FMT/UT, the entire MOCA team, Angelica Mader, Jelena Marincic, all the people at ULB, and, further back, UvA, and, of course, all the external people with whom I have had the pleasure of working on case studies, academic visits, workshops and conferences.

A special thank–you goes out to Gabriella Sperotto who really surprised me with her beautiful, witty, yet playful cover design.

I found out along the way that doing a PhD and then writing a thesis is actually not so easy. And I'm afraid that, dualy, a significant portion of the people around me must have come to similar conclusions about being around someone who is doing a PhD and then writing a thesis. So I'm really grateful for all the love and support that I received during these last few years from my family and friends, my brothers, my mom and dad, my wife Anna; I love you all, very, very much.

Contents

List of Figures

List of Tables

Part I

Synthesis

Chapter 1

Introduction

As of this writing, Toyota, the largest car manufacturer in the world, is forced to recall close to 150 thousand cars because of a problem in the braking system (Toyota, 2010). Under certain rare combinations of circumstances the braking system in the car may briefly fail. The problem originates in the software that controls the brakes. The models affected are hybrid cars. For a hybrid car, the brakes attempt to generate electric power at every braking action. At light braking the car is slowed down by driving a generator rather than by the traditional method of applying friction on the braking discs. But when the driver requests a hard brake, or an immediate stop, the braking discs are still to be used. This braking system is efficient when implemented correctly. However, it is not straightforward to implement correctly because several distinct modes of operation must be switched on and off correctly under all possible circumstances.

Around the same time, a group of scientists at ESA, the European Space Agency, are conducting their first successful tests with a recommissioned Heterodyne Instrument for the Far Infrared (HIFI). This device is an extremely high resolution spectrography instrument aboard the Herschel Space Observatory. HIFI was scheduled to take measurements on interstellar gas clouds (de Graauw and Helmich, 2001), however the instrument failed shortly after mission launch (de Graauw et al., 2010). The failure was reproduced using similar hardware on earth. After a consistent failure scenario was found, the scientists were able to suggest a fix to the problem that could be executed remotely. The problem was diagnosed to be ultimately caused by a series of unexpected events that was eventually misinterpreted by a subsystem supposed to protect the instru-

ment's circuitry from under–voltage due to power failure. The standby mode was engaged erroneously (because there was no real power failure) and a diode in one of the power converters was damaged as a result.

These two examples are quite different in nature, but there are also similarities that are symptomatic of a general problem faced by system engineers. So what do both examples have in common, when it comes to the ultimate reason of their system failure?

A first reason, is the fact that, in both cases, the systems have reached a level of complexity that makes the number of distinct modes of operation reach a certain threshold where straightforward enumeration of all possible combinations becomes infeasible. In the case of Toyota, the reason for this is economics: the braking system is complex because it makes the car more fuel–efficient. In the case of HIFI, the reason was scientific curiosity: the instrument is complex because it needs to look further back into the time of the early universe than ever before.

A second reason, is the fact that both systems were deployed in an environment that is hard to recreate in the laboratory or factory. In the case of Toyota, this is due to the large volumes in which the car is produced: it is not possible to test under all the circumstances in which the hundreds of thousands of drivers will take the car once it is taken into production. In the case of HIFI, there is only one operational instance of the system, but it is in space. These circumstances are also very hard and expensive to recreate and test for on earth.

The examples illustrate two growing trends in our increasingly technology driven society. First, the man–made systems our society is depending on are fast growing in complexity. Second, the environment in which these systems are to function is often partially unknown and even hostile. These two facts taken together lead to an interesting, and relevant scientific challenge. The question is how to control a complex, engineered system correctly under all circumstances in a hostile environment.

What can we hope to gain from studying the control of complex systems in a partially unknown, hostile environment? Coming back to our examples we see that the potential gain is substantial. An unknown number of people has been at great risk or worse due to the failing of the brakes in their car. Toyota suffers huge costs from the subsequent recall of their cars. Note that, once recalled, to fix the car is relatively straightforward: a firmware update will alleviate the problem. HIFI was almost damaged beyond repair, once diagnosed the fix was again simple: reprogram the device to bypass the malfunctioning subsystem. Clearly, if these kinds of problems arise in software they can also be solved in software, and, even better, they can be prevented by writing better software.

1.1 Formal Methods

A first response from the computer science community to the observations above has been the advent of formal methods. Although many different types of formal methods exist, the idea is always more–or–less the same. We formalize the system itself and the assumptions that the system makes on its environment in some suitable logic. It then becomes possible to *prove*, with mathematical certainty, that the system will behave correctly under all possible influences of the environment.

This type of *formal verification* should then lead to the highest attainable degree of confidence that we can invest in a system. Unfortunately, like the examples in the introduction, many systems in industry do not support this level of confidence. The reason being that, even with the state of the art in assisted theorem proving, the systematic construction of a correctness proof for a nontrivial system is still extremely costly in terms of the amount of highly skilled labour required. This explains why this type of formal verification is generally only applied to the most safety critical of systems for which failure would incur huge economic, social or scientific loss. And even then, the method is not a panacea as there is always the risk of inadequate specifications creating a false sense of confidence.

1.2 Model Checking

To alleviate the problem with the cost of applying formal methods, automated techniques like *model checking* have been developed. Here, the actual construction of a proof is done by means of exhaustive search. Another benefit of model checking is the automatic construction of a counterexample in case the property turns out not to hold. In fact many who have experience with this technique will confirm that it is usually the *negative* information, in the form of counterexamples, that is most useful. A negative answer from the model checker would typically give the engineer one of two types of information:

1. There is a bug in the system.

2. There is an assumption missing from the property.

The first type of information is useful for obvious reasons. The second type of information is useful because it makes explicit an assumption that will have to be fulfilled by the context in which the system will function. Consequently

the response to the first type of information will typically be to strengthen the guarantees embodied by the system model, for example by strengthening a trigger condition on a transition. The response to the second type of information will typically be to weaken the property, for example by making the property into an implication with the additional assumption in the premise and the original property in the consequent.

Although model checking represents a major improvement over theorem proving in terms of making formal methods more practical, its iterative, counter-example–driven workflow still requires a substantial investment in terms of skilled labor spend in verification.

1.3 Synthesis

One problem that is shared by both of the aforementioned approaches is the fact that, in both cases, the tool support is there only to be applied a posteriori, for *checking* our work. This is true regardless of whether we are constructing a proof of correctness using a proof assistant or verifying a model using a model checker. In both cases the consequences are that we are forced to add many detailed formal assumptions and formal guarantees that are necessary to make the proof go through in the strict logical sense. This is tedious and error prone work. Even if the required time is invested, there remains the activity of justification that is inherently external to the model. An engineer must check the model to justify its adequacy (Marincic et al., 2008). Yet, for very detailed models this justification effort becomes increasingly problematic in itself. So we see that, also in this sense, it would be nice to have a procedure that allows us to focus on the essential aspects of our model and frees us from having to consider excessive detail. Synthesis can be seen as an attempt to do just this: we focus on the essence and an automated procedure fills in the blanks.

The basic idea behind synthesis is to offer an automated procedure that takes a requirement as input and produces a program as output such that the program satisfies the requirement *by construction*. Of course much depends on what types of specifications and what types of programs we are considering. In general there is often some limit that we put on the expressivity of the specification language and the computational power of the synthesized programs. Another essential aspect to consider is whether or not the programs that are to be synthesized have a connection to the outside world, whether or not the variables that describe the outside world are *controllable* by the program, and whether or not these variables are *observable*.

One particular type of synthesis called *control synthesis* (Ramadge and Wonham, 1989) starts from the idea of automatically synthesizing a reactive controller for enforcing some desired behavior in a plant. This setting is especially natural for applying synthesis techniques. In particular there is a very natural distinction between what is controllable and what is not. This is determined by the actuators that connect the control machine to the plant allowing it to carry out useful tasks. There is also a natural distinction between what is observable and what is not. This is determined by the sensors that connect the plant to the control machine allowing it to keep an internal representation of the plant state, that can subsequently be used to make control decisions.

The main difficulty that any effective procedure for controller synthesis must face is that the uncontrolled state space generated by the plant description is typically large. This is mainly due to concurrency in the model defined by the plant description. The latter is a central issue also in model checking. However, for synthesis the problem is amplified by two additional, complicating factors. First, we typically see a higher degree of non–determinism because of the variability that is left in the model. Second, it is often the case that the state of the plant is only partially observable for the controller. This leads to the controller having only imperfect information about the actual state of the plant. This is typically resolved by considering sets of actual states (information sets) which the controller can know the plant to be in based on past observations. However, doing so naively, using the classical subset construction, incurs another exponential blowup. All considered, controller synthesis becomes a difficult combinatorial problem.

1.4 State of The Art

Synthesis of reactive systems was first considered by Church (1957) who suggested the problem of finding a set of restricted recursion equivalences mapping an input signal to an output signal satisfying a given requirement (Thomas, 2009). The classical solutions to Church's Problem by Buchi and Landweber (1969); Rabin (1972) in principle solve the synthesis problem for omega–regular specifications. Since then, much of the subsequent work has focused on extending these results to richer classes of properties and systems, and on making synthesis more scalable.

Pioneering work on synthesis of closed reactive systems was done by Manna and Wolper (1984); Clarke and Emerson (1982) who use a reduction to the satisfiability of a temporal logic formula. That it is also possible to synthesize

open reactive systems is shown by Pnueli and Rosner (1989a,b) using a reduction to the satisfiability of a CTL* formula, where path operators force alternation between the system and the environment. Around the same time, another branch of work by Ramadge and Wonham (1989); Lin and Wonham (1990) considers the synthesis problem specifically in the context of control of discrete event systems, this introduces many important control theoretical concepts, such as *observability*, *controllability*, and the notion of a *control hierarchy*.

More recently several contributions have widened the scope of the field and addressed several scalability issues. Symbolic methods, already proven successful in a verification setting, can be applied also for synthesis (Asarin et al., 1995). Symbolic techniques also enable synthesis for hybrid systems which incorporate continuous as well as discrete behaviour (Asarin et al., 2000). Controller synthesis under partial information can be solved by a reduction to the emptiness of an alternating tree automaton (Kupferman and Vardi, 2000). This method is very general and works in branching and linear settings. However, scalability issues remain as the authors note that most of the combinatorial complexity is shifted to the emptiness check for the alternating tree automaton. Kupferman et al. (2006) presents a compositional synthesis method that reduces the synthesis problem to the emptiness of a non–deterministic Büchi tree automaton. And Maler et al. (2007) consider the specific case of hard real time systems. They argue that the full expressive power of omega regular languages may not be necessary in such cases, since a bounded response requirement can be expressed as a safety property.

Even the earliest solutions to Church's problem, essentially involve solving a game between the environment and the control (Thomas, 1995). As such there is a need to study the (symbolic) representation and manipulation of games and strategies as first class citizens. Chatterjee et al. (2006) develop a symbolic algorithm for games with imperfect information based on fixed point iteration over antichains. Cassez (2007) gives an efficient on–the–fly algorithm for solving games of imperfect information. Kuijper and van de Pol (2009b) propose an antichain based, counter example driven algorithm for computing weakest strategies in safety games of imperfect information. The latter algorithm is antichain based but it is not an antichain algorithm in the classical sense. In particular the algorithm manipulates contravariant antichains which have more structure (indeed in this thesis we will develop that representation further and refer to these structures as *allow lattices*). Doyen and Raskin (2010) generalize their earlier results on antichain algorithms to antichains over arbitrary simulation relations, in addition new types of antichain algorithms based on promising sets are introduced.

As discussed before, compositionality adds another dimension to the synthesis problem: for reasons of scalability it is desirable to solve the synthesis problem in an incremental manner, treating first subproblems in isolation before combining their results. In general this requires a form of assume–guarantee reasoning. There exists an important line of related work that addresses such issues. One such recent development by de Alfaro and Henzinger (2001) that aims to deal with component based designs is the work on interface automata. This work introduces *interfaces* as a set of behavioral assumptions/guarantees between components. Composition of interfaces is *optimistic*: two interfaces are *compatible* iff there exists an environment in which they could be made to work together. In this thesis, we work from a similar optimistic assumption while deriving local control constraints, i.e.: a local transition should not be disabled as long as there exists a safe context which would allow it to be taken. A synchronous, bidirectional interface model is presented by Chakrabarti et al. (2002). Our component model is similar, but differs on the input/output partition of the variables to be able to handle partially observable systems. Interfaces also have nice algorithmic properties allowing for instance automated refinement and compatibility checking. Several algorithms for interface synthesis are discussed by Beyer et al. (2007).

Chatterjee and Henzinger (2007) describe a co–synthesis method based on assume–guarantee reasoning. Their solution is interesting in that it addresses non–zero–sum games where processes compete, but not at all cost. Chatterjee et al. (2008) explore ways in which to compute environment assumptions for specifications involving liveness properties, where removal of unsafe transitions constitutes a pre–processing step.

In another line of work by Ricker and Rudie (2007), Basu et al. (2009), Bensalem et al. (2010), and Katz et al. (2011) on *knowledge based distributed control* the goal is to exploit local knowledge in order to derive distributed controllers that are able to execute concurrently. This can mean controllers run either completely without synchronization or with only minimal synchronization between them. In this thesis we do not consider such a requirement, in fact we aim for quite the opposite which is a completely integrated controller. In this sense it is interesting to see that in the distributed case controllers are combined *disjunctively*, whereas in our case controllers are combined *conjunctively*. Partial observability in knowledge based distributed control seems only to play a part between the various local processes and their peers, authors generally assume full observability of the neighborhood of a (set of) process(es). In Bensalem et al. (2010) there are a number of interesting observations on the incorporation of the desired safety invariant in the model *before* computing local knowledge.

In Katz et al. (2011) the computation of a global control strategy is considered as a pre–processing step, to be applied before the computation of the distributed control.

1.5 This Thesis

The impetus for this work was formed by the observation that, although the synthesis problem seems to be solved, in theory, for very rich classes of systems and properties, there is very little practical application of the results. The limitation seems to be that scalability issues prevent straightforward application of the existing algorithms. It may seem that this cannot be avoided, since control synthesis with imperfect information is essentially an intractable problem, even when restricted to simple safety properties.

Yet, sometimes intractable problems have practical solutions. As an example, we mention circuit verification, where symbolic methods based on BDD's have proven to be very successful. What seems to be the key there is that BDD's manage to exploit tacit structure that is often present in circuits. Note that circuits are, after all, highly engineered artefacts. At face value it seems not unreasonable to extrapolate this to the realm of system engineering and expect that, with a sufficiently strong symbolic method, synthesis with imperfect information can be made practical.

In this thesis we contribute a game theoretic framework that allows safety control problems to be formalized and solved. For this we develop a new symbolic representation of control strategies and a new game solving algorithm. We employ this new game solving algorithm in a compositional setting which allows us to exploit the component structure present in a given plant specification. In particular, the use of compositional methods in combination with imperfect information presents some unique challenges that will be addressed in the thesis.

Our results concerning the use of compositional methods for control synthesis are positive in the sense that they confirm our working hypothesis. In particular we are able to show how the compositional approach, on certain natural problem instances consisting of a variable number of components, scales linearly in the number of components whereas the monolithic approach on the same instances scales super–exponentially in the number of components.

1.6 Structure of the Thesis

This thesis is divided into two parts. The first part, based on Kuijper and van de Pol (2009b), focusses on synthesis and the second part, based on Kuijper and van de Pol (2009a), focusses on compositionality.

More specifically, the rest of the thesis is structured as follows. In Chapter 2 we develop a game theoretic framework for representing and solving individual instances of the type of controller synthesis problems we are interested in. In Chapter 3 we extend our result to encompass synthesis of finite state machines. In Chapter 4 we use all the results from the first part of thesis in developing a framework and algorithm for compositional synthesis of safety controllers. In Chapter 5 we validate our approach using several experiments. Finally, in Chapter 6 we summarize our results, draw conclusions and give perspectives on future work.

Chapter 2

Knowledge Based Control Synthesis

In this chapter we introduce a game theoretic framework to stage and solve safety control problems. The chapter is structured as follows. In Section 2.1 we give a short introduction. In Section 2.2 we define safety games and give some auxiliary definitions and background. In Section 2.3 we define a novel symbolic datastructure for representation of knowledge based strategies for safety games of imperfect information. In Section 2.4 we develop a symbolic algorithm for solving safety games of imperfect information. Finally, in Section 2.5 we discuss some efficiency concerns and optimizations.

2.1 Introduction

Controlling a system can be formalized as a game played by the *controller* against the *plant*. In Figure 2.1 we illustrate this view. The controller, repre-

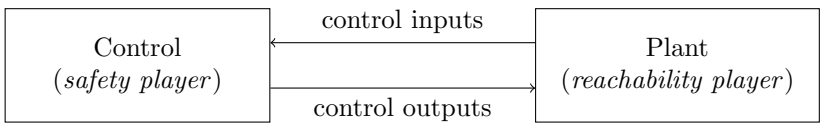


Figure 2.1: Control and Plant connected through control outputs and inputs.

sented by the *safety player*, and the plant, represented by the *reachability player*, are connected through a fixed, finitary interface of control outputs and inputs. It is the job of the controller to accomplish some useful task in the plant without ever reaching an unsafe state. We conservatively assume the plant to be totally uncooperative so we can say it is the aim of the plant to reach an unsafe state.

Note that the name "plant", comes from the analogy with an industrial plant, which is one of the obvious applications of controller synthesis. However, in this context, it is good to think of the plant as comprising everything that is not controllable. The plant, for example, could incorporate actions performed by a user of the system, which are clearly not controllable. The plant could also incorporate external, uncontrollable factors that influence the system, like transmission errors or failure of components.

In this chapter we develop a game–theoretical framework to solve such safety control problems. The framework will be especially geared towards solving games for compositional synthesis. This can be seen, for instance, in the fact that we formalize a *move* for the safety player as a *set* of allowed control outputs (as opposed to just any single, concrete control output). The latter is important when partially constraining subgames before composing them. Roughly speaking, for the composed game the controller will have to play at most the intersection of the allow sets for the subgames. This will be the topic of Chapter 4. In this chapter we consider the problem of solving individual game instances.

2.2 Safety Games

The definitions below rely on important concepts from epistemology, dating back to, at least, Hintikka (1962), and game theory, dating back to, at least, Von Neumann (1928). Particularly important will be the game–theoretic concept of an *information set*. For recent surveys on these subjects cf. Halpern (1995) and Halpern (2003). In this section we will first focus on the basic definition of a safety game.

Definition 2.1 (Safety Games) A *safety game of imperfect information* G is a tuple

$$G = (L, C^{\text{out}}, C^{\text{in}}, \alpha, \beta, \delta, i^{\text{init}})$$

consisting of a finite set of *game locations* L, a finite set of *control outputs* C^{out}, a finite set of *control inputs* C^{in}, an *output labeling* $\alpha : L \to C^{\text{out}}$, an *input labeling* $\beta : L \to C^{\text{in}}$, a *game board* $\delta \subseteq L \times L$, and a set of *initial locations* $i^{\text{init}} \subseteq L$ (also

called the *initial information set*). We let $L^{\text{dead}} = \{\ell \in L \mid \nexists \ell' \in L.(\ell, \ell') \in \delta\}$ denote the set of *deadlock locations*. We define $\mathcal{O} = C^{\text{out}} \times C^{\text{in}}$ as the set of *observations*, an observation $o \in \mathcal{O}$ is written as $o = c^{\text{out}}/c^{\text{in}}$. As a convenience we define labeling $\gamma : L \to \mathcal{O}$ such that $\gamma(\ell) = \alpha(\ell)/\beta(\ell)$. We define $A = \mathbb{P}(C^{\text{out}})$ as the set of *allow sets*. We define $\alpha^{-1}(a) = \{\ell \in L \mid \exists c^{\text{out}} \in a.\alpha(\ell) = c^{\text{out}}\}$, and $\gamma^{-1}(o) = \{\ell \in L \mid \gamma(\ell) = o\}$. These maps show how allow sets and observations partition the set of locations. Formally we need to distinguish an allow set or an observation from the set of locations that support it. However, since it is always clear from the context where a set of locations is required, most of the time we will leave the conversions $\alpha^{-1}(\cdot)$ and $\gamma^{-1}(\cdot)$ implicit. $\lhd$

A safety game of imperfect information should be interpreted as a game between two players: the safety player and the reachability player. The game is played by the players moving a token on a game board which is a directed graph for which the vertices are called *locations*, and the edges are called *transitions*. Initially the reachability player will place a token on some location of the game board that is in the set of initial locations $\ell \in i^{\text{init}}$. During the play the token moves from location to location, along the edges of the game board. The objective for the safety player is to keep the game running forever. The objective for the reachability player is to reach a deadlock state, i.e.: a location ℓ' in which it holds $\delta(\ell') = \varnothing$. Note that, in this way, it is possible to encode an arbitrary safety property into the game board by removing all the outgoing transitions from locations that violate the safety property.

The next location on the game board is decided by the moves of the players: first the safety player picks an *allow set* which is the set of control outputs that she wants to allow, next the reachability player will resolve all remaining non–determinism by moving the token along a game transition onto a new location that is labeled with a control output that the safety player allows. In this way the safety player can allow more than one concrete control output at any given point in the play. For example, if the token is on game location $\ell \in L$ and the safety player chooses move $a \in A$, the reachability player must move the token to a successor location from the *forcing set* which is defined as $\delta(\ell) \cap a$. Note that, as mentioned in Definition 2.1, formally we should write $\delta(\ell) \cap \alpha^{-1}(a)$ but for brevity we consistently keep the conversion α^{-1} implicit.

It is the responsibility of the safety player to ensure that her forcing set $\delta(\ell) \cap a$ never becomes empty. We give an example to illustrate this.

Example 2.1 (Pennymatching) In Figure 2.2 we illustrate the simple game of *pennymatching*. In this game, at each round, both players choose a side to a penny. If the safety player forfeits her choice by playing $a = \{\mathrm{h}, \mathrm{t}\}$ (heads *or*

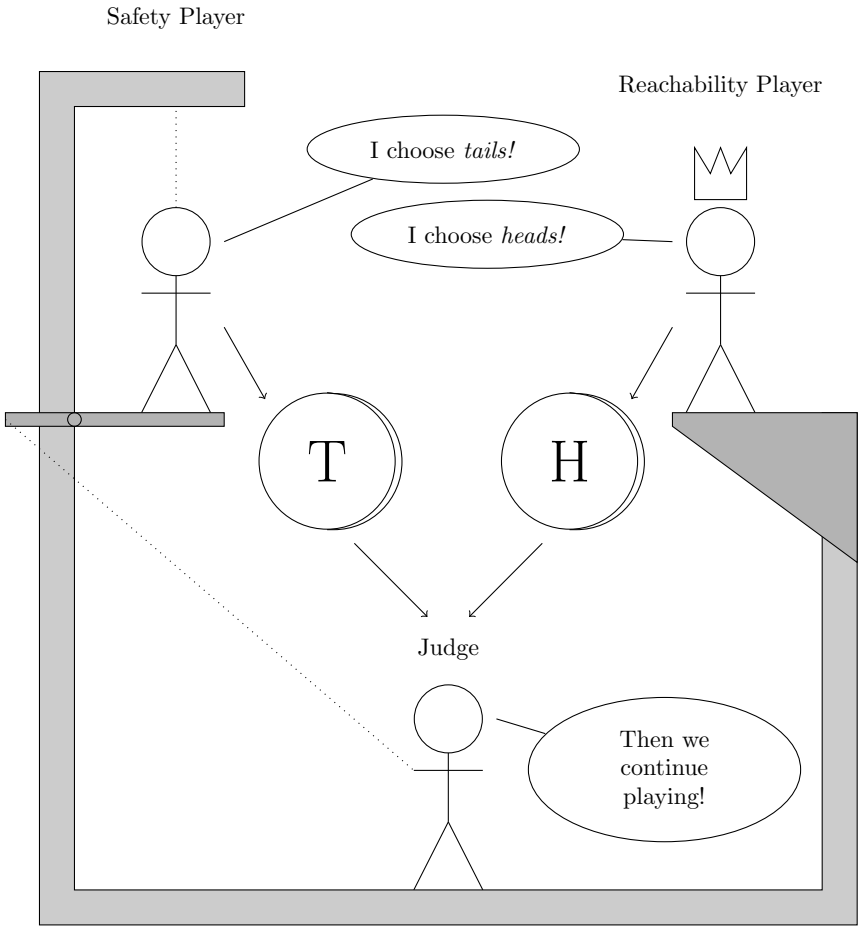


Figure 2.2: An illustration of the Penny Matching game.

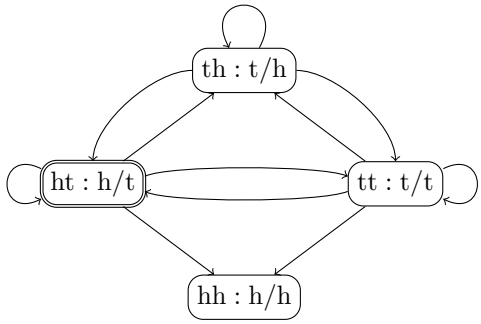


Figure 2.3: Game board for the pennymatching game of Example 2.1.

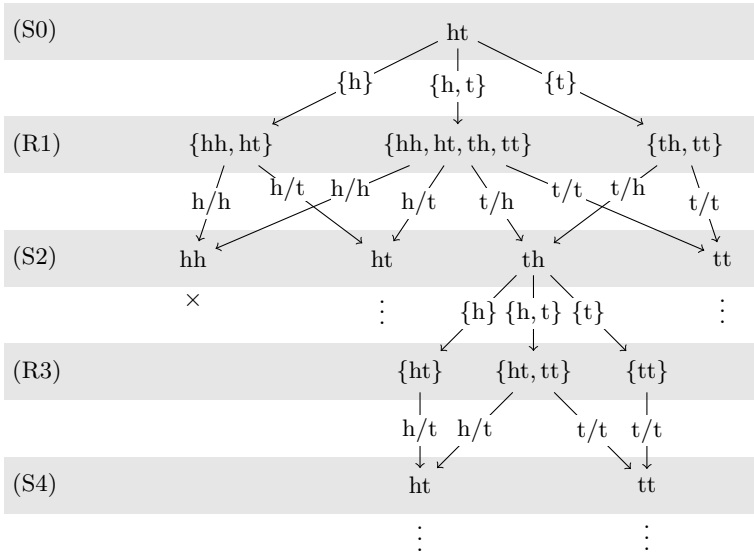


Figure 2.4: Game DAG for the pennymatching game.

tails) the reachability player will choose for her. After both players have made their moves the game progresses as follows: if both players have played heads the game is over and it is a win for the reachability player, in all other cases the game simply continues. To make the game slightly more interesting we stipulate that the reachability player cannot surprise the safety player by playing heads twice in a row. In accordance with Definition 2.1 we may model this game as follows:

$$\begin{array}{rclcrclcrcl} L & = & \{\mathrm{h},\mathrm{t}\} \times \{\mathrm{h},\mathrm{t}\} & & C^{\mathrm{in}} & = & \{\mathrm{h},\mathrm{t}\} & & \alpha(sr) & = & s \\ i^{\mathrm{init}} & = & \{\mathrm{ht}\} & & C^{\mathrm{out}} & = & \{\mathrm{h},\mathrm{t}\} & & \beta(sr) & = & r \end{array}$$

$$\delta = \{(sr, s'r') \in L \times L \mid \neg(s = r = \mathrm{h}) \wedge (r = \mathrm{h} \rightarrow r' \neq \mathrm{h})\}$$

Note that a location $(s, r) \in L$ is consistently shortened to a juxtaposition sr.$\lhd$

Figure 2.3 shows the game board for the pennymatching game, and Figure 2.4 shows a fragment of the unraveling of the possible paths through the game board into a directed acyclic graph. The chains in this DAG are called *plays*. During a play, the players move in strict alternation. The safety player moves at even levels of the DAG and the reachability player moves at odd levels of the DAG.

At level (S0) the safety player picks her first set of control outputs that she wants to allow. Based on her choice we land in one of three possible forcing sets. At level (R1) the reachability player moves the token to one of the possible locations in the forcing set.

To be safe from (S0) the safety player has to make a move for which all the possible moves of the reachability player lead to a safe state in (S2). Since the location hh is the only location that the safety player has to avoid, this is equivalent to saying that she must fix an allow set for which the forcing set at level (R1) does not contain hh. As can be seen, this leaves only $\{\mathrm{t}\}$ as a safe alternative to play at level (S0).

If the reachability player subsequently decides to move the token to th at level (S2) then the safety player gains more options. As can be seen, from location th she can allow both control outputs $\{\mathrm{h},\mathrm{t}\}$ and the structure of the underlying game board makes sure that the reachability player cannot pick the deadlock location at level (R3), so the token moves to either ht or tt at level (S4), which are both still safe.

In general, we will see that, for this type of safety game, we have the nice property that at any node in the game DAG at which the safety player is to

move, there exists a well–defined *greatest allow set* that is safe to play. We will come back to this in Section 2.2.6.

2.2.1 Concurrency and Alternation

There is concurrency inherent in the game's description as given in the previous section. This is apparent from the fact that we do not distinguish locations for the safety player and the reachability player: in the game graph there is only one type of location and it is labeled with a pair consisting of a control output and a control input. We resolve this concurrency on the level of *plays* where we do make the safety player and the reachability player move in strict alternation. In this context it is important to note that, on the level of plays, the moves are not individual, concrete control outputs and inputs, rather the moves for the safety player correspond to the picking of an allow set, and the moves for the reachability player correspond to the picking of a concrete successor location from the forcing set. The forcing set can be seen as labeling the missing intermediate node. Summing up we can say that the individual control outputs and inputs are assumed to occur *concurrently*. Then at the level of plays we resolve this concurrency though the intermediate concept of a forcing set.

It may be surprising to some readers that our game graphs are not bipartite graphs where each path forms a play directly. In our case there are good reasons for maintaining concurrency on the level of individual, concrete control outputs and inputs. This will become more clear in the second part of the thesis where we consider compositionality. Because the necessary formal concepts are not in place, at this moment, we can only hint at the underlying intuitions. The reason why it is important to treat control outputs and inputs completely symmetrically is because this will give us more flexibility in adapting the control interface. For example it becomes possible to increase the forcing power of the safety player by projecting part of the control inputs onto the control outputs, or to decrease the forcing power of the safety player by projecting part of the control outputs onto the control inputs. This type of partial side–switching with respect to what is input and what is output is important when synthesizing *context* (cf. Section 4.7). Note that this would become technically problematic if we would assume strict *output–after–input* or *input–after–output* alternation.

From a modeling perspective, the type of concurrency that we introduce on the level of control outputs and inputs is an abstraction. Arguably, it may be the case that the control outputs and the inputs happen at precisely the same time, however more likely it is the case that the control outputs and the control inputs are initiated at slightly different moments in time. Yet, for the purpose

of our model, we choose to abstract over the exact ordering and duration of events as far as they occur in the same control cycle.

As an example of this consider control output a and control input b, the observation a/b should then be understood as saying: "at some point during this control cycle it happened that the control sent a to the plant and at some point during this control cycle it happened that the plant sent b to the control". Nothing is said about the duration or relative ordering of these events within the scope of that single control cycle, and nothing is said about any relation of causality that may or may not exist between these events within the scope of that single control cycle.

As a more concrete example consider the penny matching game graph in Figure 2.3. More specifically consider the three safe locations: $L_{\text{safe}} = \{\text{ht}, \text{th}, \text{tt}\}$. Next consider their observations: $\gamma(\text{ht}) = \text{h/t}$, $\gamma(\text{th}) = \text{t/h}$ and $\gamma(\text{tt}) = \text{t/t}$. This set of possible observations can be characterized symbolically as follows: $O_{\text{safe}} = \{s/r \in C^{\text{out}} \times C^{\text{in}} \mid s = \text{h} \rightarrow r = \text{t}\}$. So we see that the set of possible observations can be characterized by a logical implication in what is, essentially, a simple propositional logic. Since we know that any safe strategy, when implemented by the safety player, will keep the game confined to these three observations this means the given implication will have to be an invariant of the resulting system. However, the fact that this invariant is an implication still does not say anything about the existence of a causality relation between the two propositions. To see this more clearly, just note that, by contraposition, the set of safe observations can equally well be characterized symbolically as follows: $O_{\text{safe}} = \{s/r \in C^{\text{out}} \times C^{\text{in}} \mid r = \text{h} \rightarrow s = \text{t}\}$.

The situation with respect to consecutivity of events in time is similar to the situation with respect to causality. As an example that appeals more closely to intuition consider the sentence: "if my neighbor gets his newspaper today then I get my newspaper today". In a pure propositional logic, in absence of temporal modalities, this sentence can be formalized as a simple implication: $p \rightarrow p'$. The sentence talks about two correlated events that both pertain to the same day. The sentence does not say anything about the duration or temporal relationship of these two events within the course of that day. It might be the case that I receive my newspaper before my neighbor receives his newspaper, but then again, it might also be the case that my neighbor receives his newspaper before I receive my newspaper. The situation all depends on the route that the paperboy is taking. It might even be the case we obtain our newspapers at precisely the same moment, for instance because we are sharing the same doormat. The sentence is simply not precise enough to distinguish among any of these alternatives.

The crucial observation to make about the newspaper delivery example is that, in the scope of a single day, we may just treat both events as propositions that pertain to that single day. This is an adequate abstraction to make as long as the events occur *concurrently* at the finest time granularity about which we choose to reason. In the case of the newspaper delivery example this is a single day, in the case of an actual controller for an embedded system this time interval will most likely be several orders of magnitude smaller. However the principle of abstraction is the same in both cases.

2.2.2 Imperfect Information

So far we have not explicitly dealt with the fact that the safety player has only a limited number of observations at her disposal. The fact that the safety player can only make a limited observation of the current state is commonly referred to as *partial observability*. Partial observability leads to the safety player having only *imperfect information* about the exact location of the token on the game board. The impact of imperfect information in the analysis of games is huge due to the fact that the game board δ is not *observation deterministic*. This means that distinct branchings in δ are not always distinguishable for the safety player, i.e. there exist $\ell \in L$ and $o \in \mathcal{O}$ for which $\delta(\ell) \cap \gamma^{-1}(o)$ contains more than one location. We illustrate this phenomenon with the example below.

Example 2.2 (Blind Pennymatching) We introduce a variant of the pennymatching game from Example 2.1 where the coin of the reachability player remains completely hidden for the safety player. Formally, we model this completely analogous to Example 2.1, except that we set $C^{\text{in}} = \{\mathbf{x}\}$ and for all $sr \in L$ we set $\beta(sr) = \mathbf{x}$. $\lhd$

Figure 2.5 shows the game board for the blind pennymatching game, and Figure 2.6 shows a fragment of the unraveling of the possible plays for this game into a game DAG.

Consider now the following play. At level (S0) the safety player chooses {t}. At level (R1) the reachability player chooses th as the successor location. The safety player observes t/$\mathbf{x}$, i.e. she sees her own coin but not the coin of the reachability player. Now, technically, it should be safe for her to move {h} since this is safe from location th. However, note that the reachability player at level (R1) might as well have chosen tt as the successor location, and this would have given the safety player the same observation t/$\mathbf{x}$. This means that, at level (S2) the locations th and tt are *indistinguishable* for the safety player. In Figure 2.6 this is indicated with a dashed line.

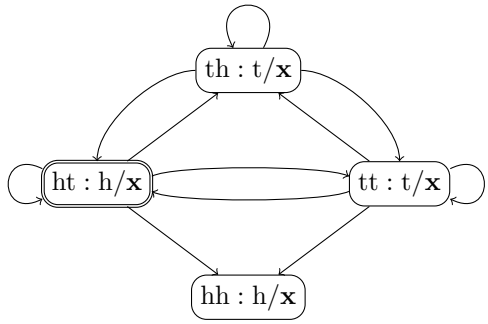


Figure 2.5: Game board for the blind pennymatching game of Example 2.2.

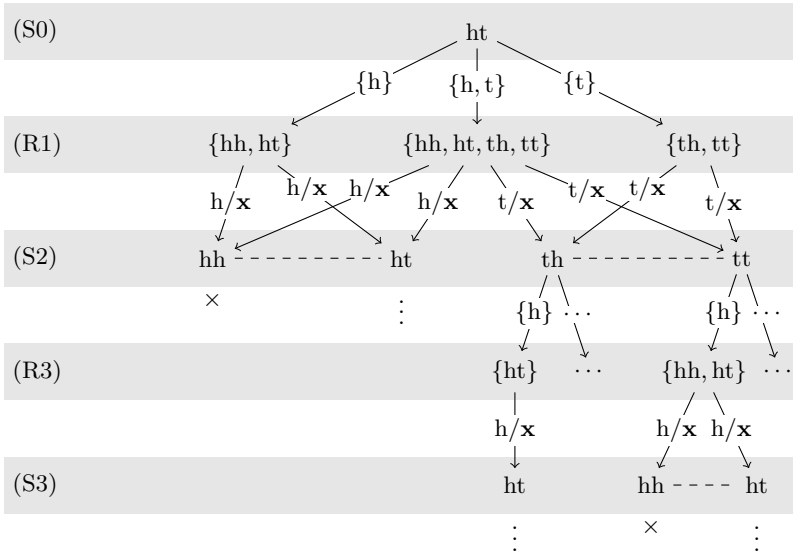


Figure 2.6: Game DAG for the blind pennymatching game.

If two locations are indistinguishable for the safety player she cannot allow any control output that would lead to an unsafe position from either of the indistinguishable locations. In the next section we show how to deal with this formally.

2.2.3 Knowledge Based Subset Construction

The type of uncertainty for the safety player about the true location of the token which we described in the previous section can be resolved by applying a subset construction. Intuitively, we move from the set of concrete game locations to the set of *information sets* which are sets of game locations. This construction will make the game graph observation deterministic again.

Roughly speaking, a player that has limited observational powers has, at any moment in the game, several possible states of affairs that it deems possible. The actual state of the game will be one among these possible states of affairs. This can be seen as a simplified version of the possible worlds semantics from epistemic logic. Please note, however, that for the treatment in this thesis we will not need the usual epistemic modalities to deal with introspective qualities, reasoning about other players' knowledge, common knowledge, etc. In our case, it suffices to think of an information set in purely game theoretical terms as a set of *possible locations* for the token. We give the definitions below. Recall that, for a given location $\ell \in L$, with $\gamma(\ell) = o$ we denote the observable information on ℓ, in this sense the set of observations $\mathcal{O}$ partitions L.

Definition 2.2 (Knowledge Subset Construction) For a given safety game, with I we denote the set of *information sets* defined as $I = \mathbb{P}(L)$, and with Δ we denote the *knowledge based subset construction* which is defined as a graph over information sets $\Delta \subseteq I \times I$ as follows:

$$\Delta = \{(i, i') \in I \times I \mid \exists o \in \mathcal{O}.i' = \delta(i) \cap o\}$$

Note that the image of δ on i is $\delta(i) = \bigcup_{\ell \in i} \delta(\ell)$, now $i' = \delta(i) \cap o$ represents the strongest knowledge the safety player has about the successor location upon observing o with knowledge i about the source location. $\triangleleft$

Using Δ to define a new game where all imperfect information has been resolved is now a straightforward albeit technical exercise. We illustrate this with the following example.

Example 2.3 (Knowledge Subset Construction) Figure 2.7 on page 42 is the knowledge based subset construction of the game board for the blind variant

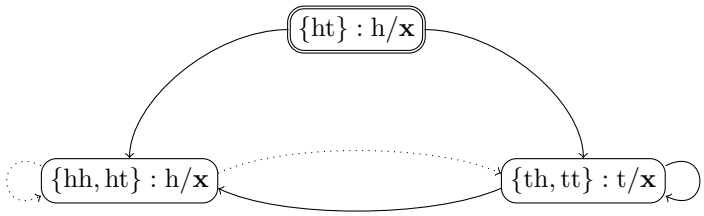


Figure 2.7: Knowledge based subset construction of the game board for the blind pennymatching game of Example 2.2.

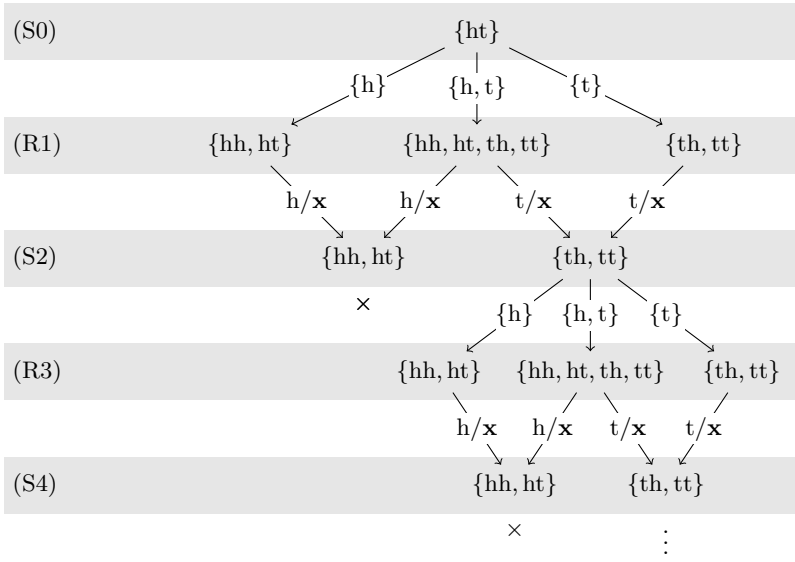


Figure 2.8: Partial unravelling of the game in Figure 2.7 into a DAG.

of the pennymatching game (cf. Figure 2.5 on page 40). As can be seen, we restrict to information sets that are reachable from the initial information set. The dotted edges are transitions in Δ that we must remove because they leave from an information set that contains a deadlock location. Figure 2.8 on page 42 shows a fragment of the knowledge based subset construction for the game dag of the blind game. $\triangleleft$

The knowledge based subset construction makes the game *observation deterministic* meaning that for each information set $i \in I$ and each observation $o \in \mathcal{O}$ there is a *unique* information set $i' \in I$ that represents the knowledge of the safety player after observing o with information i, namely: $i' = \delta(i) \cap o$.

2.2.4 Knowledge Based Strategies

Now that we have formally dealt with the problems related to imperfect information, we can introduce the concept of a *knowledge based strategy* for the safety player. This definition also determines the winning condition formally: the safety player wins the game iff she has a knowledge based strategy to force an infinite play.

Definition 2.3 (Knowledge Based Strategy) For a given game, a *knowledge based strategy* is a function $f : I \to A$. With KBS we denote the set of all knowledge based strategies. For a given strategy $f \in \mathsf{KBS}$ and information set $i_0 \in I$, with $\mathsf{outcome}(G, f, i_0)$ we denote the *outcome of f on G starting from i_0* as a set of non-empty traces of game locations annotated with information states: $\mathsf{outcome}(G, f, i_0) \subseteq (L \times I)^+$. This is defined as follows:

$$\begin{aligned}\mathsf{outcome}(G, f, i_0) = \{\ell_0 i_0 \dots \ell_n i_n \mid\ & \forall m \leq n.\ell_m \in i_m, \\ & \forall m < n.(\ell_m, \ell_{m+1}) \in \delta \\ & \quad \text{and } (i_m, i_{m+1}) \in \Delta \\ & \quad \text{and } \alpha(\ell_{m+1}) \in f(i_m)\ \}\end{aligned}$$

These are all possible finite (partial) plays that may arise when our safety player is playing according to knowledge based strategy f. An outcome is *safe* iff no play ends in a deadlock (every finite play has a proper extension). We say that a strategy f is *safe* for G iff for all $i \in I$ either $\mathsf{outcome}(G, f, i)$ is safe, or $f(i) = \varnothing$. A strategy is *winning* iff it is safe and $f(i^{\text{init}}) \neq \varnothing$. A game is *solvable* iff it permits a winning strategy. $\triangleleft$

As mentioned before, in game theoretic terms, information based strategies are *positional*. Meaning they assign a move to each position of the game. Where a *position*, in our case correspond to an information set representing the knowledge of the safety player about the possible locations of the token on the game board.

2.2.5 Inductive Characterization of Safety

In the previous section we defined the notion of *safety* as it applies to a knowledge based strategy of the safety player. We did this using the notion of *outcome*. This is an intuitive definition because it corresponds as closely as possible to how we feel the safety games are played. In particular: (i) there is only one *true* position of the token, and (ii) the safety player cannot base her decision directly on the true position of the token, instead she must rely on her knowledge.

For the exposition in the remainder it is helpful to use an equivalent, inductive characterization of safety that is phrased more directly in terms of knowledge based strategies. This characterization is, on the one hand, less intuitive, but, on the other hand it is more direct because it bypasses the definition of outcome in Definition 2.3. We will prove the two notions to be equivalent in Lemma 2.5.

Our inductive characterization of safety uses the following two elementary properties of knowledge based strategies. The first property is *obstinacy*, intuitively a strategy is obstinate if it blocks completely on information sets for which an empty forcing set is possible, or, equivalently, it returns a non-empty allow set only if each of the states in the information set has at least one valid successor in the underlying game board intersected with the allow set. The second property is *observation–closedness*, intuitively a strategy is observation-closed if it can guarantee that non-blocking states will, for every possible observation, lead to non-blocking successor states.

One may think of these two properties as an inductive definition of safety where obstinacy forms the base case, and observation–closedness forms the inductive case.

Definition 2.4 (Inductive Safety) For a given game, a knowledge based strategy $f \in \mathsf{KBS}$ is *obstinate* iff for all $i \in I$ such that there exists $\ell \in i$ for which $\delta(\ell) \cap f(i) = \varnothing$ it holds $f(i) = \varnothing$. A knowledge based strategy $f \in \mathsf{KBS}$ is *observation–closed*for all $i \in I$ and $o \in \mathcal{O}$ such that $\delta(i) \cap o \cap f(i) \neq \varnothing$ it holds that $f(\delta(i) \cap o) \neq \varnothing$. $\lhd$

The following lemma establishes that *obstinacy* and *observation–closedness*

are necessary and sufficient conditions to characterize safety for knowledge based strategies.

Lemma 2.5 (Inductive Safety) For a given game, a strategy is safe iff it is both obstinate and observation–closed. ◁

Proof For the right to left direction. Assume that f is obstinate and observation–closed. Let $\ell_0 i_0 \dots \ell_n i_n \in \mathsf{outcome}(G, f, i_0)$ and $f(i_0) \neq \varnothing$. By induction on n, using observation–closedness, it follows $f(i_n) \neq \varnothing$. Then, by obstinacy, we have existence of at least one successor to ℓ_n that is allowed by $f(i_n)$. Which suffices to show that $\ell_0 i_0 \dots \ell_n i_n$ has a proper extension in $\mathsf{outcome}(G, f, i)$.

For the left to right direction. Assume that f is safe. First assume, for contradiction, that f is not obstinate. By definition this implies there exists $i \in I$ and $\ell \in i$ such that $f(i) \neq \varnothing$ but $\delta(\ell) \cap f(i) = \varnothing$. This entails $\ell i \in \mathsf{outcome}(G, f, i)$ with no proper extension, which contradicts our assumption that f is safe. Next assume, for contradiction, that f is not observation–closed. By definition this implies there exists $i \in I$ and $o \in \mathcal{O}$ such that (i) $\delta(i) \cap o \cap f(i) \neq \varnothing$ but (ii) $f(\delta(i) \cap o) = \varnothing$. Let $i' = \delta(i) \cap o$. Now note that (i) entails there exists $\ell \in i$ and $\ell' \in \delta(\ell)$ such that $\gamma(\ell') = o = c^{\text{out}}/c^{\text{in}}$ and $c^{\text{out}} \in f(i)$, which, in turn, implies $\ell i \ell' i' \in \mathsf{outcome}(G, f, i)$, But now (ii) entails that this play has no proper extension, which contradicts our assumption that f is safe.□

2.2.6 Weakest Strategies

In the previous sections we have consistently defined *a* solution to a safety game as *any* winning strategy. In this section we sharpen this to *the weakest*, or *most permissive* winning strategy. Intuitively, a winning strategy is the most permissive winning strategy if for all plays the strategy always yields the largest possible allow set that is sufficient for keeping the future play safe. Formally, this means we introduce an ordering on KBS with respect to which we may select the greatest element in the subset of safe strategies.

Definition 2.6 (Permissivity) For a given game, we define a weak partial order $\sqsupseteq$ on KBS such that $f' \sqsupseteq f$ iff for all $i \in I$ it holds $f'(i) \supseteq f(i)$. We say f' is *weaker* or *more permissive* than f. A strategy $f \in \mathsf{KBS}$ is *antitone* iff for all $i, i' \in I$ it holds: $i \subseteq i'$ implies $f(i) \supseteq f(i')$. For two strategies $f_1, f_2 \in \mathsf{KBS}$ we define the join $f' = f_1 \sqcup f_2$ such that for all $i \in I$ we have $f'(i) = f_1(i) \cup f_2(i)$.◁

We first show that for obtaining weakest, safe strategies, we can restrict our attention to antitone strategies. Intuitively, if the safety player *knows more* then she can *allow more*. The following lemma makes this precise.

Lemma 2.7 (Antitonicity) For a given game, for any safe strategy f there exists a safe, antitone strategy f' such that $f' \sqsupseteq f$. $\lhd$

Proof Given a strategy f that is safe, we can define f' such that for all $i \in I$ it holds $f'(i) = \bigcup\{f(i') \mid i \subseteq i'\}$. It is easily seen that f' is antitone, f' is weaker than f, and f' is safe. $\square$

Lemma 2.8 (Lattice) For a given game, it holds that the set of safe, antitone strategies ordered by $\sqsubseteq$ forms a complete lattice. $\lhd$

Proof For any two safe, knowledge based strategies f_1 and f_2 it is easily seen that their join $f' = f_1 \sqcup f_2$ is safe, and forms the least upper bound of f_1 and f_2. Since the set of safe, knowledge based strategies is finite, this implies that every subset of safe, knowledge based strategies has a least upper bound. Hence the set of safe, knowledge based strategies forms a complete lattice. $\square$

By Lemmas 2.7 and 2.8 the following is now well–defined.

Definition 2.9 (Weakest Safe Strategy) With f_G we denote the weakest, safe strategy on game G. $\lhd$

We may sum up the discussion with the following theorem. The theorem states the correspondence between the solvability of a game G and the weakest safe, antitone strategy for G.

Theorem 2.10 For any game G it holds that G is solvable iff $f_G(i^{\text{init}}) \neq \varnothing$. $\lhd$

Proof For the right to left direction. Assume $f_G(i^{\text{init}}) \neq \varnothing$, since by definition f_G is safe it follows immediately that f_G is winning for G.

For the left to right direction. Assume G is solvable, this means there exists a strategy f that is safe for G and, moreover, $f(i^{\text{init}}) \neq \varnothing$, by definition 2.9 we have that f_G is weaker than any safe strategy, in particular $f_G \sqsupseteq f$, and it follows $f_G(i^{\text{init}}) \neq \varnothing$. $\square$

2.3 Symbolic Data Structure for Strategies

In the previous section we explored the use of knowledge based strategies for winning safety games of imperfect information. As mentioned before, in game theoretic terms, information based strategies are *positional*. Meaning they assign a move to each position of the game. However, as we have seen in Section 2.2.4, the positions in a game of imperfect information actually correspond

to information sets that represent the player's current knowledge about the true location of the game. This means that the domain of the knowledge based strategy is exponential in the number of game board locations $|L|$. Clearly then, representing the function $f : I \to A$ explicitly, by enumerating its graph, will not scale to larger games. In this section we explore other possibilities of representing a knowledge based strategy that are not intrinsically exponential.

2.3.1 Exploiting Contravariance

To get to a compact representation for a knowledge based strategy we may use the fact that, as we have shown in Section 2.2.6, the functions we are interested in are always antitone. This means we need only keep a subset of the function's graph to be able to compute the function value for every possible information set.

We can illustrate this based on the pennymatching game from Example 2.1. The weakest knowledge based strategy for this example is the function with the following graph:

$$
\begin{array}{rcl}
\{\mathrm{hh},\mathrm{ht},\mathrm{th},\mathrm{tt}\} & \mapsto^* & \varnothing \\
\{\mathrm{hh},\mathrm{ht},\mathrm{th}\} & \mapsto & \varnothing \\
\{\mathrm{hh},\mathrm{th},\mathrm{tt}\} & \mapsto & \varnothing \\
\{\mathrm{hh},\mathrm{ht},\mathrm{tt}\} & \mapsto & \varnothing \\
\{\mathrm{hh},\mathrm{ht}\} & \mapsto & \varnothing \\
\{\mathrm{hh},\mathrm{th}\} & \mapsto & \varnothing \\
\{\mathrm{hh},\mathrm{tt}\} & \mapsto & \varnothing \\
\{\mathrm{hh}\} & \mapsto & \varnothing
\end{array}
\qquad
\begin{array}{rcl}
\{\mathrm{ht},\mathrm{th},\mathrm{tt}\} & \mapsto^* & \{\mathrm{t}\} \\
\{\mathrm{ht},\mathrm{th}\} & \mapsto & \{\mathrm{t}\} \\
\{\mathrm{ht},\mathrm{tt}\} & \mapsto & \{\mathrm{t}\} \\
\{\mathrm{ht}\} & \mapsto & \{\mathrm{t}\} \\
\{\mathrm{th},\mathrm{tt}\} & \mapsto & \{\mathrm{t}\} \\
\{\mathrm{tt}\} & \mapsto & \{\mathrm{t}\}
\end{array}
\qquad
\begin{array}{rcl}
\{\mathrm{th}\} & \mapsto^* & \{\mathrm{h},\mathrm{t}\} \\
\varnothing & \mapsto & \{\mathrm{h},\mathrm{t}\}
\end{array}
$$

We have marked with a $(\cdot \mapsto^* \cdot)$ all the domain/co–domain pairs that are not implied by pairs with a weaker information set. Because the function is antitone all the pairs that have a stronger information set can be derived from these maximal pairs. We say these derived pairs are *allow subsumed* by the pairs *above* them. Formally, this is defined as follows.

Definition 2.11 (Allow Subsumed) We define an *info/allow* tuple as a pair $\langle i, a\rangle \in I \times A$. We define a pre–order $\subseteq^{\langle \mathrm{s},\cdot\rangle}$ on the set of info/allow pairs such that $\langle i, a\rangle \subseteq^{\langle \mathrm{s},\cdot\rangle} \langle i', a'\rangle$ iff $i \subseteq i'$, i.e. the order is only on the *source element* (the information set). We say $\langle i, a\rangle$ is *below* $\langle i', a'\rangle$, or, vice versa, $\langle i', a'\rangle$ is *above* $\langle i, a\rangle$. We now define a partial order $\subseteq^{\langle \mathrm{s},\mathrm{t}\rangle}$ on info/allow pairs such that $\langle i, a\rangle \subseteq^{\langle \mathrm{s},\mathrm{t}\rangle} \langle i', a'\rangle$ iff $i \subseteq i'$ and $a \subseteq a'$. With $\subset^{\langle \mathrm{s},\mathrm{t}\rangle}$ we denote the corresponding strict partial order. We say $\langle i, a\rangle$ is *allow–subsumed* by $\langle i', a'\rangle$. $\lhd$

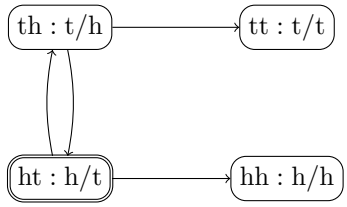


Figure 2.9: Game board for the contramatching game of Example 2.4.

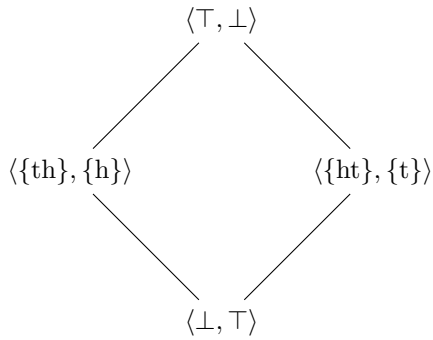


Figure 2.10: Weakest, safe Allow Lattice for the game from Figure 2.9.

To achieve a compact representation we can represent the function by the set of $\subseteq^{\langle s,t\rangle}$–maximal pairs from its graph. For the example this becomes:

$$\begin{array}{rcl} \langle\{\mathrm{hh},\mathrm{ht},\mathrm{th},\mathrm{tt}\} & , & \varnothing\rangle \\ \langle\{\mathrm{ht},\mathrm{th},\mathrm{tt}\} & , & \{\mathrm{t}\} \\ \langle\{\mathrm{th}\} & , & \{\mathrm{h},\mathrm{t}\}\rangle \end{array}$$

We call this a contravariant chain because the chain restricted to the left hand sides constitutes a strictly downward decreasing chain and the chain restricted to the right hand sides constitutes an upward increasing chain. It is not true that there is always only one maximal contravariant chain in the strategy. In the next example we demonstrate how to deal with multiple maximal contravariant chains.

Example 2.4 (Contramatching) We introduce yet another variant of the popular pennymatching game as first introduced in Example 2.1. In the *contramatching* variant the reachability player never chooses the same side twice. All the safety player has to do is to play the opposite side at each round. We define this game completely analogous to Example 2.1 except for the game board, which now becomes:

$$\delta = \{(sr, s'r') \in L \times L \mid r \neq r' \text{ and } s \neq r\}$$

In Figure 2.9 this game board is presented graphically. ◁

If we work out the weakest, safe strategy for the contramatching game we see that we obtain the function with the following graph:

$$\begin{array}{rcl} \{\mathrm{hh},\mathrm{ht},\mathrm{th},\mathrm{tt}\} & \mapsto^* & \varnothing \\ \{\mathrm{hh},\mathrm{ht},\mathrm{th}\} & \mapsto & \varnothing \\ \{\mathrm{hh},\mathrm{th},\mathrm{tt}\} & \mapsto & \varnothing \\ \{\mathrm{hh},\mathrm{ht},\mathrm{tt}\} & \mapsto & \varnothing \\ \{\mathrm{hh},\mathrm{ht}\} & \mapsto & \varnothing \\ \{\mathrm{hh},\mathrm{th}\} & \mapsto & \varnothing \\ \{\mathrm{hh},\mathrm{tt}\} & \mapsto & \varnothing \\ \{\mathrm{hh}\} & \mapsto & \varnothing \\ \{\mathrm{ht},\mathrm{th},\mathrm{tt}\} & \mapsto & \varnothing \\ \{\mathrm{ht},\mathrm{th}\} & \mapsto & \varnothing \\ \{\mathrm{ht},\mathrm{tt}\} & \mapsto & \varnothing \\ \{\mathrm{th},\mathrm{tt}\} & \mapsto & \varnothing \\ \{\mathrm{tt}\} & \mapsto & \varnothing \end{array} \qquad \begin{array}{rcl} \{\mathrm{ht}\} & \mapsto^* & \{\mathrm{t}\} \\ \{\mathrm{th}\} & \mapsto^* & \{\mathrm{h}\} \end{array} \qquad \varnothing \mapsto^* \{\mathrm{h},\mathrm{t}\}$$

Again we have marked with a $(\cdot \mapsto^* \cdot)$ all the maximal pairs that are not implied by pairs that have a weaker information set. Like for the previous example, we now look at possible contravariant chains in order to obtain a more concise representation. For this example we identify two distinct, maximal contravariant chains:

$$\begin{array}{rcl} \langle\{\text{hh},\text{ht},\text{th},\text{tt}\} & , & \varnothing\rangle \\ \langle\{\text{ht}\} & , & \{\text{t}\} \\ \langle\varnothing & , & \{\text{h},\text{t}\}\rangle \end{array} \qquad \begin{array}{rcl} \langle\{\text{hh},\text{ht},\text{th},\text{tt}\} & , & \varnothing\rangle \\ \langle\{\text{th}\} & , & \{\text{h}\} \\ \langle\varnothing & , & \{\text{h},\text{t}\}\rangle \end{array}$$

These contravariant chains, taken together will form an *allow lattice*. In Figure 2.10 we show the allow lattice for the contramatching game. For brevity, from now on, we will use symbolic notation for the top $\langle\top,\bot\rangle$ and bottom $\langle\bot,\top\rangle$ pair of the allow lattice respectively. Note that such top and bottom info/allow pairs will be present in the graph of any weakest, safe knowledge based strategy for a non–trivial game. The top pair expresses that, in the absence of any knowledge about the system state no action can be derived to be safe, the bottom pair expresses that, in the presence of inconsistent knowledge about the system state any action can be derived to be safe.

The allow lattice is a compact representation of the knowledge based strategy. To compute the function value for a given information set i, we need to take the union over all the allow sets a' associated to information sets i' such that $i \subseteq i'$, i.e. *i implies i'*. Under this semantics, if we view information sets as predicates on the underlying statespace of game locations, and allow sets as predicates on the underlying set of control outputs, then the allow lattice may be viewed as a predicate transformer.

In the next section we formalize the datastructure of allow lattices, and the operations that we may carry out on them.

2.3.2 Allow Lattices

Definition 2.12 (Allow Lattice) A given set of info/allow pairs $g \subseteq I \times A$ denotes the knowledge based strategy $[\![g]\!] : I \to A$ such that for all $i \in I$ it holds $[\![g]\!](i) = \bigcup\{a' \mid \exists i'.\langle i', a'\rangle \in g \text{ and } i \subseteq i'\}$. We will often use shorthand $g(i) = [\![g]\!](i)$ for denoting evaluation of an allow set through a given set of pairs. We define the *source set* of g as $\mathsf{Src}(g) = \{i \in I \mid \exists a.\langle i, a\rangle \in g\}$. We say $g \subseteq I \times A$ is *contravariant* iff for any two info/allow pairs $\langle i, a\rangle, \langle i', a'\rangle \in g$ such that $i \subseteq i'$ it holds $a \supseteq a'$. When $g \subseteq I \times A$ is contravariant it follows $\subseteq^{\langle s,\cdot\rangle}$ is a partial order on $\mathsf{Src}(g)$. We say g is *bounded* iff it has a top and bottom pair in the $\subseteq^{\langle s,\cdot\rangle}$–ordering. We say $g \subseteq I \times A$ is *complete* iff every subset of

pairs has a least upper bound in the $\subseteq^{\langle \mathrm{s},\cdot\rangle}$–ordering. If $g \subseteq I \times A$ is bounded, contravariant and complete we call g an *allow lattice*. With AL we denote the set of all allow lattices. We define $i_g^{\mathrm{init}} = \bigcap\{i' \mid \langle i', a'\rangle \in g \text{ s.t. } i' \supseteq i^{\mathrm{init}}\}$ as the best approximation of the initial information set i^{init} that occurs in g. $\lhd$

Note that for $g \subseteq I \times A$, contravariance entails that g is a functional relation, which entails that $\subseteq^{\langle \mathrm{s},\cdot\rangle}$ is antisymmetric, which, in turn, entails that $\subseteq^{\langle \mathrm{s},\cdot\rangle}$ is a partial order on g. The following example illustrates this definition.

Example 2.5 (Allow Lattice) Consider Pennymatching game from Example 2.1. The following set of info/allow pairs represents a safe, winning strategy for this game:

$$g_1 = \{\langle\{\mathrm{th}\}, \{\mathrm{h}\}\rangle, \langle\{\mathrm{ht}, \mathrm{th}, \mathrm{tt}\}, \{\mathrm{t}\}\rangle\}$$

Note for instance, if we want to compute the value $g_1(i)$ for the knowledge $i = \{\mathrm{th}\}$ we need to take the union of all allow sets associated with information sets that are implied by this knowledge, this yields $g_1(\{\mathrm{ht}\}) = \{\mathrm{h}\} \cup \{\mathrm{t}\}$. Note that g_1 is neither contravariant nor complete. The following equivalent set of pairs is contravariant

$$g_2 = \{\langle\{\mathrm{th}\}, \{\mathrm{h}, \mathrm{t}\}\rangle, \langle\{\mathrm{ht}, \mathrm{th}, \mathrm{tt}\}, \{\mathrm{t}\}\rangle\}$$

If we now add a suitable top and bottom element we will obtain an equivalent allow lattice:

$$g_3 = \{\langle\top, \bot\rangle, \langle\{\mathrm{th}\}, \{\mathrm{h}, \mathrm{t}\}\rangle, \langle\{\mathrm{ht}, \mathrm{th}, \mathrm{tt}\}, \{\mathrm{t}\}\rangle, \langle\bot, \top\rangle\}$$

Note that the allow lattice contains a subset of the graph of the function it represents. $\lhd$

The following lemma ensures that every antitone knowledge based strategy can be represented as an allow lattice.

Lemma 2.13 (Allow Lattice) For a given antitone knowledge based strategy $f \in \mathsf{KBS}$ its graph g_f is an allow lattice, $[\![g_f]\!] = f$. $\lhd$

Proof That g_f is contravariant follows directly from the fact that f is antitone, g_f is trivially complete as $\mathsf{Src}(g_f) = I = \mathbb{P}(L)$ is a powerset lattice. To see that, indeed, $[\![g_f]\!] = f$, let $i \in I$, we need $[\![g_f]\!](i) = f(i)$. We expand: $[\![g_f]\!](i) = \bigcup\{a' \mid \exists i'.\langle i', a'\rangle \in g_f \text{ and } i \subseteq i'\}$. Then we note that, as a special case $\langle i, f(i)\rangle \in g_f$ which entails $[\![g_f]\!](i) \supseteq f(i)$. Now since g_f is contravariant we have for all $\langle i', a'\rangle \in g_f$ such that $i \subseteq i'$ that $a' \subseteq f(i)$ it follows $[\![g_f]\!](i) \subseteq f(i)$.$\square$

This shows that allow lattices are adequate to represent any antitone knowledge based strategy but we have still to show formally how the more compact representants that we saw in the previous section can be systematically derived.

2.3.3 A Normal Form for Allow Lattices

We introduce a normal form for allow lattices in two steps.

1. Identify the *principal* pairs, which are all the pairs that must be present in any contravariant set of pairs that represents the same function.
2. To the principal set, keep adding pairs until every subset of pairs has a join–pair and a meet–pair. This set will then constitute the smallest possible allow lattice.

We first identify the *principal* pairs for a given set of pairs which are all the pairs that are not implied by pairs with a weaker allow set. As it turns out this is exactly the minimal set of pairs that must be present in any allow lattice that can represent the same function.

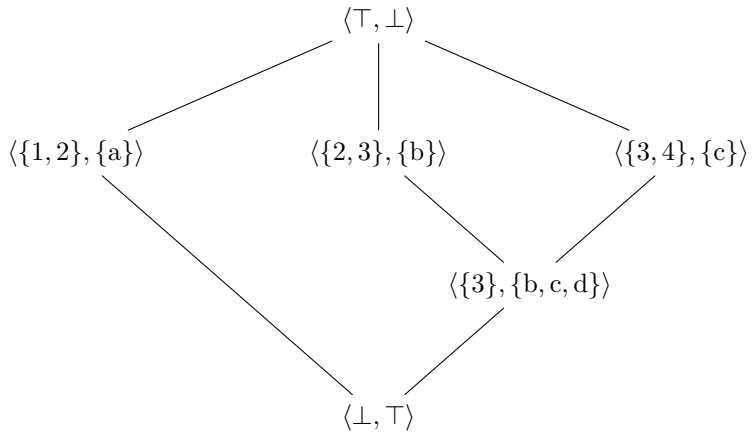
Definition 2.14 (Principal Pairs) For an arbitrary given set of pairs $g \subseteq I \times A$ we define $\mathsf{Contravariant}(g) = \{\langle i, g(i)\rangle \mid i \in \mathsf{Src}(g)\}$ For a given contravariant $g \subseteq I \times A$ we call an info/allow pair $\langle i, a\rangle \in g$ *principal* iff $a \supset \bigcup_{i' \supset i} g(i')$. For a given contravariant $g \subseteq I \times A$ we define $\mathsf{Principal}[g]$ as the subset of principal pairs. For an arbitrary $g \subseteq I \times A$ we define the *sparse normal form* as

$$\mathsf{SNF}[g] = \{\langle \top, g(\top)\rangle\} \cup \mathsf{Principal}[\mathsf{Contravariant}(g)] \cup \{\langle \bot, g(\bot)\rangle\} \tag{2.1}$$

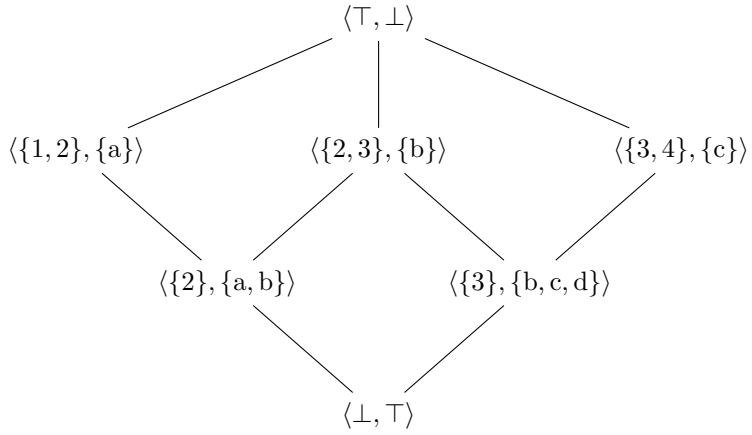
Note that the sparse normal form, besides the top and bottom pair, only contains principal pairs from the graph of the function that is being represented. Formally, the sparse normal form constitutes a bounded partially ordered set (poset). In determining the set of principal pairs the following definition is useful, for a given $i \in \mathsf{Src}(g)$ we define $(g \Uparrow i)$ as the set of pairs in g that have an information set that is strictly weaker than i, formally: $(g \Uparrow i) = \{\langle i', a'\rangle \in g \mid i' \supset i\}$. ◁

It is not hard to see that $\langle i, a\rangle \in g$ is principal iff $a \supset [\![g \Uparrow i]\!](i)$. So we need only look at the pairs that are directly above $\langle i, a\rangle$ in the diagram to see if $\langle i, a\rangle$ makes a principal contribution to the function that is being represented.

Example 2.6 (SNF) As an example of an allow lattice in sparse normal form consider Figure 2.11(a). Note that the pair $\langle\{3\}, \{\mathrm{b}, \mathrm{c}, \mathrm{d}\}\rangle$ is principal because it contains control output d which is not allowed by any of the pairs above it. ◁



(a) An allow lattice in sparse normal form (with one pair implicit).



(b) An allow lattice in lattice normal form (closed under intersection).

Figure 2.11: Allow lattice in sparse and lattice normal form, the locations/control outputs have no particular meaning in this case and are chosen just to illustrate the structure.

The following lemma shows that the restriction to principal pairs is sound, in the sense that it preserves the function that is being represented.

Lemma 2.15 (Soundness) For all $g \subseteq I \times A$ it holds $[\![\mathsf{SNF}[g]]\!] = [\![g]\!]$. $\lhd$

Proof The $\sqsubseteq$–direction is trivial, just note that the sparse normal form only contains pairs from the graph of $[\![g]\!]$ hence $[\![\mathsf{SNF}[g]]\!]$ cannot be stronger than $[\![g]\!]$. The $\sqsupseteq$–direction is the only interesting case. Wlog, let $i \in I$ and $c \in [\![g]\!](i)$ we show $c \in [\![\mathsf{SNF}[g]]\!](i)$. Let $\langle i, a\rangle \in g$ such that $c \in a$ and $\nexists\langle i', a'\rangle \in g$ such that $i \subset i'$ and $c \in a'$. I.e. $\langle i, a\rangle$ is a maximal pair for which the allow set contains c. Note that, because g is finite, there always exists at least one such pair. Clearly then this will be a principal pair, hence $\langle i, a\rangle \in \mathsf{SNF}[g]$, and it follows $c \in [\![\mathsf{SNF}[g]]\!](i)$. $\square$

As can be seen the lemma does not crucially rely on the forced presence of a top and bottom pair, in fact it will turn out to be more of a technical convenience in presenting the algorithms and examples that follow.

We may sum up the discussion concerning the sparse normal form with the following theorem.

Theorem 2.16 (Canonicity) For $g_1, g_2 \subseteq I \times A$ such that $[\![g_1]\!] = [\![g_2]\!]$ it holds $\mathsf{SNF}[g_1] = \mathsf{SNF}[g_2]$. $\lhd$

Proof Assume $[\![g_1]\!] = [\![g_2]\!]$ and $\langle i, a\rangle \in \mathsf{SNF}[g_1]$, the fact that $\langle i, a\rangle$ is principal in g_1 implies that $\exists c \in a$ such that $c \notin \bigcup_{i' \supset i} [\![g_1]\!](i')$ it follows $c \notin \bigcup_{i' \supset i} [\![g_2]\!](i')$. Now assume, for contradiction, $i \notin \mathsf{Src}(g_2)$, it follows $[\![g_2]\!](i) = \bigcup_{i' \supset i} [\![g_2]\!](i')$, hence $c \notin [\![g_2]\!](i)$ whereas clearly $c \in [\![g_1]\!](i)$, since we assumed $[\![g_1]\!] = [\![g_2]\!]$ this leads to a contradiction, hence we must conclude $i \in \mathsf{Src}(g_2)$ and $\langle i, a\rangle \in \mathsf{SNF}[g_2]$. $\square$

We now describe how to complete the set of principal pairs into a proper allow lattice.

Definition 2.17 (Completion) For a given contravariant $g \subseteq I \times A$ we define $\mathsf{Complete}(g) = \{\langle i, g(i)\rangle \mid \exists I' \subseteq \mathsf{Src}(g).i = \bigcap_{i' \in I'} i'\}$. For a given $g \subseteq I \times A$ we define *the lattice normal form of* g as

$$\mathsf{LNF}[g] = \mathsf{Complete}(\mathsf{SNF}[g])$$

i.e. we take the completion of the principal pairs of g. $\lhd$

Example 2.7 (LNF) As an example of an allow lattice in lattice normal form consider Figure 2.11(b). Note that the pair $\langle\{2\}, \{\mathrm{a}, \mathrm{b}\}\rangle$ is not principal because it is implied by the pairs above it. $\lhd$

The following lemma ensures the soundness of the completion construction.

Lemma 2.18 For any set of pairs $g \subseteq I \times A$ the lattice normal form, $\mathsf{LNF}[g]$, is an allow lattice and $[\![\mathsf{LNF}[g]]\!] = [\![g]\!]$. $\lhd$

Proof By construction, the source set of $\mathsf{LNF}[g]$ is closed under intersection so the result is indeed a complete lattice. We prove, $[\![\mathsf{LNF}[g]]\!] = [\![g]\!]$. For the $\sqsupseteq$–direction note that $\mathsf{LNF}[g] \supseteq \mathsf{Principal}[g]$, then by Lemma 2.15 it follows $[\![\mathsf{LNF}[g]]\!] \sqsupseteq [\![g]\!]$. For the $\sqsubseteq$–direction just note that $\mathsf{LNF}[g]$ contains only pairs from the graph of $[\![g]\!]$. $\square$

Because the sparse normal form is always smaller than the lattice normal form yet carries essentially the same information, we will prefer to keep the knowledge based strategy in sparse normal form, so as a poset instead of a complete lattice. If it is necessary to close the diagram under intersection this can always be done in a lazy fashion, i.e. by introducing a new pair on the fly whenever the meet of several pairs is required. For the purpose of exposition, however, in the remainder we will use the LNF which has the nice property of being closed under finite meets. In Section 2.5, we will revisit these efficiency concerns, and show how it is possible to use SNF's, efficiently.

2.3.4 Allow Lattice with Transitions

In the previous section we introduced allow lattices as a compact representation of knowledge based strategies. These positional strategies will map a given knowledge in the form of an information set $i \in I$ to the greatest safe allow set $a \in A$ that suffices to keep the future play safe. One aspect that does not receive an explicit representation in this way is how a knowledge i gets updated to a knowledge $i' = \delta(i) \cap o$ upon observing some $o \in \mathcal{O}$ — recall that $\mathcal{O} = C^{\mathrm{out}} \times C^{\mathrm{in}}$.

In this section, we will enrich our datastructure by placing a transition structure on top of the allow lattice. The transitions will be labeled with observations $o \in \mathcal{O}$. In the subsequent section, we will use this transition relation to show how to refine any given allow lattice until it represents the weakest safe strategy for the underlying game.

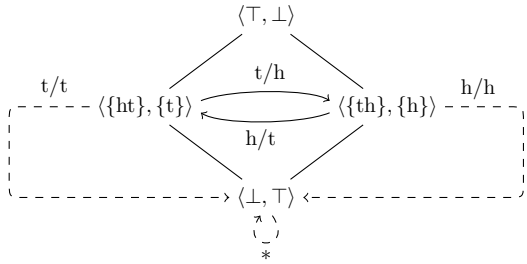


Figure 2.12: Allow Lattice for the Contramatching Example 2.4 with the induced knowledge update relation.

Definition 2.19 (Knowledge Update Relation) For a given contravariant $g \subseteq I \times A$ we define the *knowledge update relation* $R_g \subseteq g \times \mathcal{O} \times g$ such that $\langle i, a\rangle \overset{c^{\text{out}}/c^{\text{in}}}{\longrightarrow} \langle i', a'\rangle \in R_g$ iff $i' \supseteq \delta(i) \cap \gamma^{-1}(c^{\text{out}}/c^{\text{in}})$ and $c^{\text{out}} \in a$. We use shorthand $i \xrightarrow{o} i' \in R_g$ to mean $\exists a, a'.\langle i, a\rangle \xrightarrow{o} \langle i', a'\rangle \in R_g$. We call a triple $i \xrightarrow{o} i' \in R_g$ *right–minimal* iff there is no $i \xrightarrow{o} j' \in R_g$ such that $j' \subset i'$. With $R_{\overline{g}}$ we denote the subset of right–minimal triples in R_g. We call a triple $i \xrightarrow{o} i' \in R_g$ *left–maximal* iff there is no $j \xrightarrow{o} i' \in R_g$ such that $j \supset i$. With $R_{\underline{g}}$ we denote the subset of left–maximal triples in R_g. If a triple is both right–minimal and left–maximal we say it is a *principal triple*. With $R_{\overline{\underline{g}}}$ we denote the subset of principal triples in R_g. ◁

In general we will keep the allow lattice with the directed edges of the (right–minimal) knowledge update relation on top of this partially ordered structure. The following example illustrates this.

Example 2.8 (Knowledge Update Relation) In Figure 2.12 we show the allow lattice for the Contramatching game of Example 2.4. The induced knowledge update relation for this structure is shown on top of the lattice structure using the directed edges. The transitions that lead to $\perp$ are shown using dashed edges. These transitions to $\perp$ are special, in the sense that they are not supposed to occur anymore for a system that is under control. In a pragmatic sense $\perp$ can be thought of a trip–state. In a real controller typically a fail–safe override procedure will be started when such unexpected observations occur. ◁

2.4 Symbolic Algorithm for Safety Games

In this section we develop a counter example driven refinement procedure called CEDAR with which we can compute the weakest safe strategy for a given safety game of imperfect information. The procedure works by approximating, from above, an obstinate and observation closed fixed point in the lattice of knowledge based strategies, which, in turn, are represented as allow lattices.

2.4.1 Balancing Permissivity and Knowledge

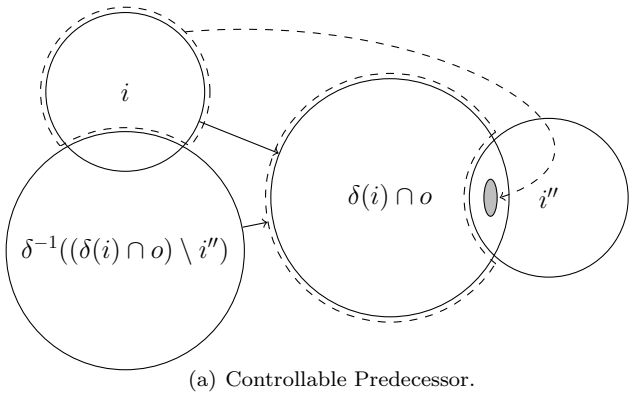
Before we treat the algorithm proper we first explain what are the crucial refinement steps that the algorithm will rely on. The final algorithm will then basically be a straightforward closure under these refinement steps. For this we recall our characterization of safety in terms of obstinacy and observation–closedness from Definition 2.4. The condition of obstinacy is easy to maintain. We should simply take care never to enter any info/allow pair into the strategy that contains a potential deadlock. The condition of observation–closedness then remains for the algorithm to work towards. In developing a fixed point we will start with the most permissive obstinate strategy possible and gradually refine this by treating counterexamples that violate observation–closedness. A counterexample to observation–closedness consists of an information set $i \in I$ and an observation $o \in \mathcal{O}$ such that the strategy blocks on the successor information set $i' = \delta(i) \cap o$. The following definition makes this precise.

Definition 2.20 (Explains) For a given contravariant $g \subseteq I \times A$, $i, i'' \in I$ and $o = (c^{\text{out}}/c^{\text{in}}) \in \mathcal{O}$ we say i'' *explains* (i, o) iff $\delta(i) \cap o \subseteq i''$ and $g(i'') \neq \varnothing$. If $c^{\text{out}} \in g(i)$ and there is no explanation for (i, o) we say (i, o) is an unexplained observation. ◁

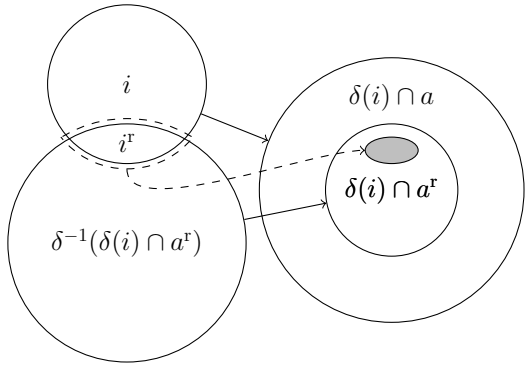
Note that, with this definition, the strategy is observation closed iff there are no unexplained observations.

To obtain this we need a procedure to treat a given counterexample against observation–closedness. The procedure will need to put a weakest (necessary) constraint on the strategy so that the given counterexample transition can be explained by some information set to which the strategy still assigns a non–empty allow set. To do this we rely on the observation that there are always exactly two ways in which we can resolve such a given counterexample transition:

1. We can suppress the transition by removing all locations that lead us into a loosing (blocking) target information set i' from the source information set i.



(a) Controllable Predecessor.



(b) Restricted Successor.

Figure 2.13: Illustration of Controllable Predecessor and Restricted Successor as used in Algorithm 1 (CEDAR)

2. We can suppress the transition by removing its control output from the source allow set a.

This can be paraphrased by saying that to solve an observation conflict we either require more knowledge (Solution 1) or we become less permissive (Solution 2). We refer to Solution 1 as the *controllable predecessor strategy* and to Solution 2 as the *restricted successor strategy.*

So let (i, o) be an unexplained observation. And let i'' be a target information set that is a candidate to explain the counterexample. We can now find the weakest controllable predecessor for i by strengthening i to the weakest $i^{\mathrm{c}} \subset i$ such that i'' *explains* (i^{c}, o). This will constitute one possible way to resolve the counterexample by requiring more knowledge about the predecessor location. If we do this systematically for all potential explanations i''. We obtain all possible controllable predecessors pairs $\langle i^{\mathrm{c}}, a\rangle$ (cf. Theorem 2.26 in Section 2.4.4).

In Figure 2.13(a) we illustrate how to compute the controllable predecessor strategy in general, for a given i, o and i''. The goal is to compute the largest $i^{\mathrm{c}} \subset i$ such that $\delta(i^{\mathrm{c}}) \cap o \subseteq i''$, i.e.: i'' *explains* (i^{c}, o). We do this by first computing what would be the strongest knowledge that the player will obtain when observing o with knowledge i this is $\delta(i) \cap o$. We then subtract i'' from this set of locations to get the smallest set of conflict locations that fall outside of i''. We then find the weakest constraint needed to suppress these conflict locations by taking the pre–image and subtracting this from the original information set: $i^{\mathrm{c}} = i \setminus \delta^{-1}((\delta(i) \cap o) \setminus i'')$. Note that, in the diagram, i^{c} is the left dashed region, and the gray region denotes $\delta(i^{\mathrm{c}}) \cap o$. In Example 2.9 we show how this works out in a concrete instance.

Example 2.9 (Controllable Predecessor) Consider the Pennymatching game as introduced in Example 2.1. Now consider the info/allow pair $\langle\{\mathrm{ht}, \mathrm{th}, \mathrm{tt}\}, \{\mathrm{h}, \mathrm{t}\}\rangle$. This says we allow both heads and tails from any of the locations $\{\mathrm{ht}, \mathrm{th}, \mathrm{tt}\}$. This strategy has an unexplained observation h/h. In Figure 2.14 we show how to compute the controllable predecessor for this particular counterexample.

First we compute the strongest knowledge the safety player will have when observing h/h from the source information set $\{\mathrm{ht}, \mathrm{th}, \mathrm{tt}\}$, this is $\{\mathrm{hh}\}$. We subtract from this set the target explanation — the i'' in Figure 2.13(a) — which, in our case, is equal to the original source information set. The result will still be $\{\mathrm{hh}\}$, we are now sure that, indeed, hh is a conflict location that we need to avoid in order for the successor information set to stay below the intended set of locations (the intended *explanation*): $\{\mathrm{ht}, \mathrm{th}, \mathrm{tt}\}$. So we take the pre–image of this singleton set of conflict locations, this will be $\{\mathrm{ht}, \mathrm{tt}\}$. And to avoid the conflict location we subtract the pre–image from the original source

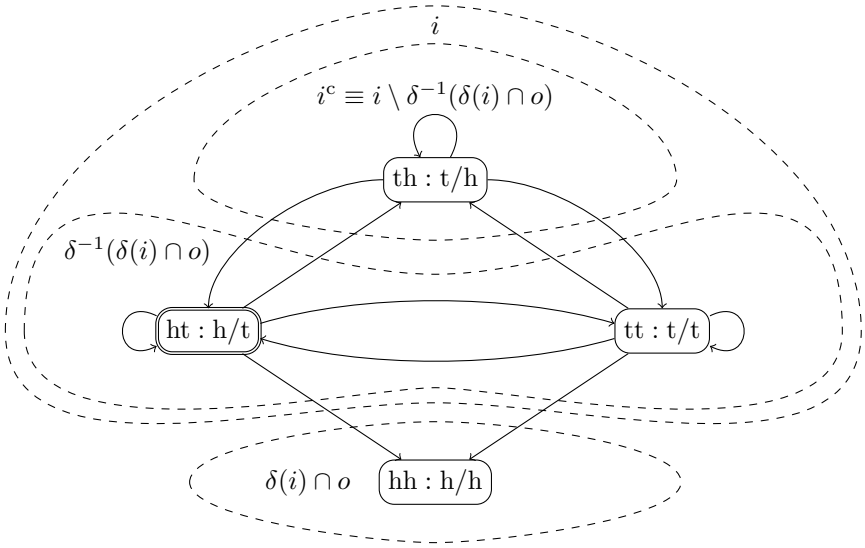


Figure 2.14: Controllable predecessors for the Pennymatching Example 2.9

information set. This yields {th}, which is the largest set of locations from which we can safely play h and still land within the intended set of locations.◁

Computing the restricted successor strategy consists in removing the offending control output from the allow set. In doing so we need to be careful to maintain obstinacy. Because we are removing a control output from the allow set the obstinacy–condition requires us to also remove any source locations from the information set that would otherwise deadlock under this smaller allow set.

In Figure 2.13(b) we illustrate how to compute the restricted successor strategy, for a given pair $\langle i, a\rangle$ and $a^{\mathrm{r}} = a \setminus \{c^{\mathrm{out}}\}$. The goal is to compute the largest $i^{\mathrm{r}} \subseteq i$ such that every $\ell \in i^{\mathrm{r}}$ still has at least one successor in $\delta(i) \cap a^{\mathrm{r}}$. We do this by first computing all the successor locations that are reachable under the restricted allow set, this is $\delta(i) \cap a^{\mathrm{r}}$. We then take the pre–image to obtain all the source locations that have at least one transition into this set: $\delta^{-1}(\delta(i) \cap a^{\mathrm{r}})$. We intersect the result with i to obtain $i^{\mathrm{r}} = i \cap \delta^{-1}(\delta(i) \cap a^{\mathrm{r}})$. Now $\langle i^{\mathrm{r}}, a^{\mathrm{r}}\rangle$ is the weakest obstinate restricted successor pair. In Example 2.10 we show how this works out in a concrete instance.

Example 2.10 (Restricted Successor) Consider the Contramatching game

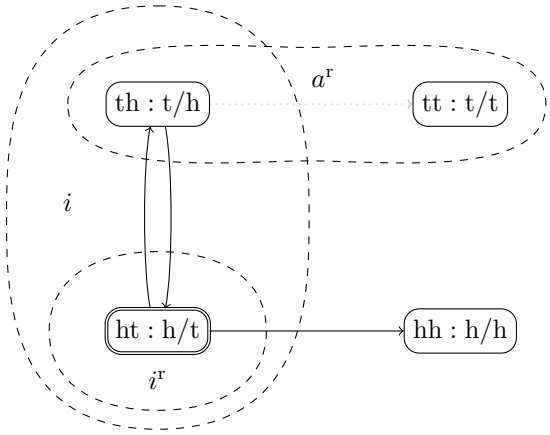


Figure 2.15: Controllable predecessors for the Contramatching Example 2.10

as introduced in Example 2.4. To make the example more interesting we remove the transition th $\longrightarrow$ tt from the game board. This can be seen as a partially implemented strategy for the safety player, the constraint being: "never play tails after the adversary has played heads". Now consider the pair $\langle\{\mathrm{ht}, \mathrm{th}\}, \{\mathrm{h}, \mathrm{t}\}\rangle$. This says we allow both heads and tails from any of the locations $\{\mathrm{ht}, \mathrm{th}\}$. This strategy has an unexplained observation h/h. In Figure 2.15 we show how to compute the restricted successor for this particular counterexample.

First we compute the forcing set under the restricted allow set. In our case this is $\{\mathrm{th}\}$. Next we take the pre–image of this set to obtain all the locations that have at least one successor into this set. In our case this is $\{\mathrm{ht}\}$. We intersect this with the original source set, which yields $\{\mathrm{ht}\}$. Now $\{\mathrm{ht}\}$ is the largest set from which we can still play $\{\mathrm{t}\}$ without risking deadlock. $\lhd$

2.4.2 Counter Example Driven Antichain Refinement

Before we can present the Algorithm proper we need the following auxiliary definitions. From the previous section recall the notion of *unexplained* observation. The following definition shows how to detect unexplained observations by looking at the knowledge update relation as defined in Section 2.3.4.

Definition 2.21 (Counterexamples) For a given contravariant $g \subseteq I \times A$ we call a triple $\langle i, a\rangle \xrightarrow{o} \langle i', a'\rangle \in R_{\overline{g}}$ a *counterexample to observation–closedness*

iff $a' = \bot$. With $\mathsf{Counter}(g)$ we denote the set of all *principal* counterexamples to observation–closedness, i.e. $\mathsf{Counter}(g) = \{\langle i, a\rangle \xrightarrow{o} \langle i', a'\rangle \in R_{\overline{\overline{g}}} \mid a' = \bot\}$. For a given i with $\mathsf{Children}_g(i)$ we denote the antichain of direct children of i in g, defined as $\mathsf{Children}_g(i) = \{i' \in \mathsf{Src}(g) \mid i' \subset i \text{ and } \nexists j \in \mathsf{Src}(g).i' \subset j \subset i\}$. $\lhd$

We recall once more our characterization of safety in terms of obstinacy and observation–closedness from Definition 2.4. The definition of $\mathsf{Counter}(g)$ now ensures that $[\![g]\!]$ is observation closed if $\mathsf{Counter}(g) = \varnothing$ (cf. Lemma 2.23 in Section 2.4.4). So if we take care to maintain obstinacy as an invariant we can use the refinement procedure from Section 2.4.1 and iteratively treat counterexamples to observation closedness until we reach a fixed point where $\mathsf{Counter}(g) = \varnothing$. Since we approximate the fixed point from above it follows that the resulting strategy will be indeed the weakest safe knowledge based strategy for the underlying game.

Now note that, for a given counterexample to observation closedness $\langle i, a\rangle \xrightarrow{o} \langle i', \bot\rangle \in \mathsf{Counter}(g)$, the weakest candidate explanations for the observation o are exactly the direct children of i' in the allow lattice. To see this just note that, due to contravariance of the normal form, we have for all $\langle i'', a''\rangle \in g$ such that $\langle i'', a''\rangle \subset^{\langle \mathrm{s}, \cdot\rangle} \langle i', \bot\rangle$ it holds $a'' \supset \bot$. So indeed the direct children of i' constitute an antichain of not–yet–loosing positions in the current approximation of the fixed point strategy. Clearly then for any $i''' \subset i''$ such that $i''' \neq \bot$ it holds that i''' is also a candidate explanation for (i, o) however, if i''' explains (i, o) then this implies that i'' explains (i, o) so it is sound for us to restrict to considering only the weakest such potential explanations.

With these auxiliary definitions in place we are in a position to present the counterexample driven antichain refinement (CEDAR) algorithm. We give a short description of Algorithm 1. In line 1 the allow lattice is initialized to be fully uninformed, except that the system is not initially in a deadlock state, and maximally permissive, i.e. it allows *all* control outputs. The while loop in line 2 runs until there are no more counterexamples to observation closedness. In line 3 we select a counterexample to observation closedness. In lines 4, 5 and 6 we compute the most conservative refinement needed to make the allow lattice observation-closed for the selected counterexample observation.

After one iteration of the while loop we have obtained the next allow lattice which is strictly below the previous one but always above-or-equal-to f_G in the strategy subsumption ordering $\sqsupseteq$. The algorithm terminates when there are no more counterexamples to observation–closedness. Since the other requirement on f_G, obstinacy, is an invariant of the algorithm, it follows that, after termination, $[\![g]\!] = f_G$. In line 7 we do a final test to see if g is winning from the initial

Algorithm 1: CEDAR — Counterexample Driven Antichain Refinement

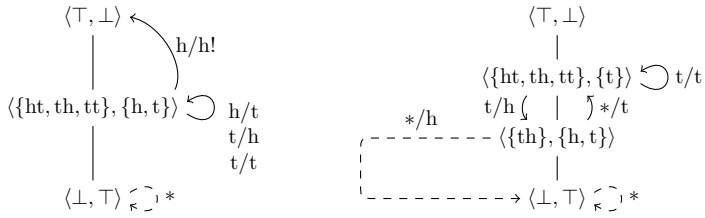
Data: A game $G = (L, C^{\text{out}}, C^{\text{in}}, \alpha, \beta, \delta, i^{\text{init}})$.
Result: Returns the weakest winning strategy for G, or $\bot$ in case there exists no winning strategy.

$g \leftarrow \mathsf{LNF}[\ \{\langle L \setminus L^{\text{dead}}, A\rangle\}\ ]$
while $\mathsf{Counter}(g) \neq \varnothing$ **do**
  let $\langle i, a\rangle \xrightarrow{o} \langle \top, \bot\rangle \in \mathsf{Counter}(g)$ **where** $o = c^{\text{out}}/c^{\text{in}}$
  $\mathsf{RSucc} \leftarrow \{\langle i^{\text{r}}, a^{\text{r}}\rangle \mid a^{\text{r}} = a \setminus \{c^{\text{out}}\}, i^{\text{r}} = i \cap \delta^{-1}(\delta(i) \cap a^{\text{r}})\}$
  $\mathsf{CPre} \leftarrow \{\langle i^{\text{c}}, a\rangle \mid i'' \in \mathsf{Children}_g(\top), i^{\text{c}} = i \setminus \delta^{-1}((\delta(i) \cap o) \setminus i'')\}$
  $g \leftarrow \mathsf{LNF}[(g \setminus \{\langle i, a\rangle\}) \cup \mathsf{RSucc} \cup \mathsf{CPre}]$
if $g(i^{\text{init}}) \neq \bot$ **then**
  return g
else
  return $\bot$

information set, if so then the results are useful and g is returned otherwise the game is unsolvable and we return $\bot$.

2.4.3 Example Runs of CEDAR

Example 2.11 (CEDAR) In Figure 2.16 we show the successive approximations of the weakest safe strategy for the pennymatching game from Example 2.1 as computed by CEDAR, in Figure 2.17 we show the same for the blind pennymatching game of Example 2.2. in Figure 2.18 we show the same for the contramatching game of Example 2.4. ◁



(a) Initial AL. Counterexample: h/h.

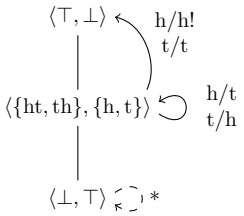
(b) Contravariant chain splits. Done.

Figure 2.16: Example run of CEDAR on Pennymatching Example 2.1.

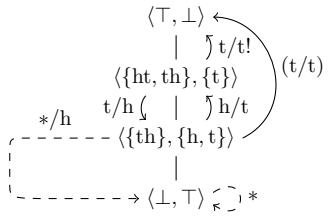
(a) Initial AL. Counterexample: h/**x**

(b) Done. Note: {th} is not reachable.

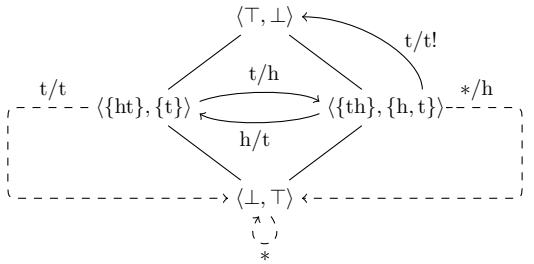
Figure 2.17: Example run of CEDAR on Blind Pennymatching Example 2.2.



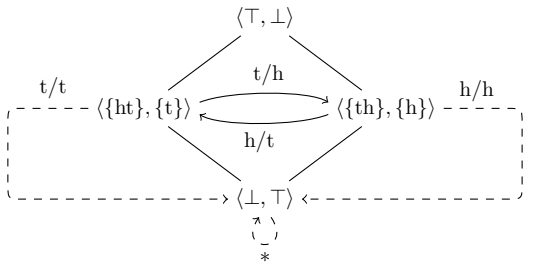
(a) Initial AL. Counterexample: h/h!



(b) Chain splits vertically. Treat: t/t



(c) Chain splits horizontally, into non–trivial antichain. Treat: t/t



(d) Last counterexample is now suppressed. Done.

Figure 2.18: Example run of CEDAR on the Contramatching Example 2.4

2.4.4 Correctness of CEDAR

In this section we give a formal proof of correctness for CEDAR. For the proof we use the inductive characterization of safety as given in section 2.2.5. We recall Lemma 2.5 which establishes that *obstinacy* and *observation–closedness* (cf. Definition 2.4) are necessary and sufficient conditions for safety of a given knowledge based strategy.

The general outline of the proof is then as follows. First we establish that obstinacy is an invariant of the algorithm so we always stay within the subset of obstinate strategies in the lattice of all knowledge based strategies. Next we establish that the absence of counterexamples implies observation–closedness, and that by treating a counterexample we strictly go down in the lattice of knowledge based strategies. Finally we establish that we never put any constraint on the strategy that is not necessary, in other word we stay above f_G (the weakest safe strategy for game G) in the lattice of knowledge based strategies. This suffices since the lattice of knowledge based strategies is finite we can only go down a finite number of times before we hit f_G.

In the remainder of this section we define g_n for $n \in \mathbb{N}$ to be the value of g in Algorithm 1 after n iterations of the while loop, or the final value of g if the while loop terminates in less than n iterations.

Lemma 2.22 (Monotonicity) For all $n \geq 0$ it holds $\mathsf{Counter}(g_n) \neq \varnothing$ implies $[\![g_n]\!] \sqsupset [\![g_{n+1}]\!]$. $\lhd$

Proof Let $t = (\langle i, a\rangle \xrightarrow{o} \langle \top, \bot\rangle) \in \mathsf{Counter}(g_n)$ where $o = c^{\text{out}}/c^{\text{in}}$. Note that this implies, in particular, $c^{\text{out}} \in [\![g_n]\!](i)$. Which, in turn, implies that $\langle i, a\rangle$ is principal, to see this note that for all $\langle j, b\rangle \supset^{\langle \mathrm{s},\cdot\rangle} \langle i, a\rangle$ it holds $c^{\text{out}} \notin b$, otherwise we would have had $t' = (\langle j, b\rangle \xrightarrow{o} \langle \top, \bot\rangle) \in \mathsf{Counter}(g_n)$ which would imply that t is not left–maximal and hence not a principal counterexample. Now note that g_{n+1} has the principal pair $\langle i, a\rangle$ removed, and replaced with a number of pairs that are all strictly below $\langle i, a\rangle$. Namely: $\langle j, b\rangle \in \mathsf{RSucc}$ implies $c^{\text{out}} \notin b$ and $\langle j, b\rangle \in \mathsf{CPre}$ implies $j \subset i$. It follows that $c^{\text{out}} \notin [\![g_{n+1}]\!](i)$, and as we had $c^{\text{out}} \in [\![g_n]\!](i)$, and clearly $[\![g_n]\!] \sqsupseteq [\![g_{n+1}]\!]$, it follows $[\![g_n]\!] \sqsupset [\![g_{n+1}]\!]$. $\square$

Lemma 2.23 (Observation Closure) For all contravariant $g \subseteq I \times A$ it holds that $\mathsf{Counter}(g) = \varnothing$ implies $[\![g]\!]$ is observation closed. $\lhd$

Proof Assume $\mathsf{Counter}(g) = \varnothing$. Then, because g is in normal form, there does not exist any transition $t = (\langle j, b\rangle \xrightarrow{o} \langle \top, \bot\rangle) \in R_{\overline{=}_g}$. Now assume, for contradiction, $[\![g]\!]$ is not observation–closed. This implies existence of a witness

$i \in I$ and $o = (c^{\text{out}}/c^{\text{in}}) \in \mathcal{O}$ such that $c^{\text{out}} \in [\![g]\!](i)$ and $\delta(i) \cap o \neq \varnothing$ yet $[\![g]\!](\delta(i) \cap o) = \bot$. Now let $j' \in \mathsf{Src}(g)$ be a weakest information set in the diagram such that $j' \supseteq i$ and $c^{\text{out}} \in g(j')$ by the definition of allow lattices and our assumption that $c^{\text{out}} \in [\![g]\!](i)$ such a j' must exist. Clearly then there must also exist a transition $t' = (\langle j', b'\rangle \xrightarrow{o} \langle \top, \bot\rangle) \in R_{\overline{\overline{g}}}$ which establishes the required contradiction. □

Lemma 2.24 (Obstinacy) For all $n \geq 0$ it holds $[\![g_n]\!]$ is obstinate. ◁

Proof By induction on n. For the base case just note that $L \setminus L^{\text{dead}}$ are exactly all locations for which there exists at least one successor location, and $[\![g_0]\!](L \setminus L^{\text{dead}}) = A$ and $[\![g_0]\!](L') = \bot$ in case $L' \not\subseteq L \setminus L^{\text{dead}}$ this means g_0 will be the weakest possible obstinate strategy for G. For the inductive case the only potentially problematic part is line 1 because we are adding a new pair to the strategy with a strictly smaller allow set than the pair we just removed. But note that the construction of this new pair ensures that all the locations that are included in the information set have at least one successor in the underlying game board intersected with this smaller allow set, cf. Figure 2.13(b). □

Lemma 2.25 (Completeness) For all $n \geq 0$ it holds $[\![g_n]\!] \sqsupseteq f_G$. ◁

Proof By induction on n. For the base case, we need: $[\![g_0]\!] \sqsupseteq f_G$. Assume, for contradiction, $[\![g_0]\!] \not\sqsupseteq f_G$, this gives us a witness $i^{\text{w}} \in I$ and $c^{\text{w}} \in f_G(i^{\text{w}})$ such that $c^{\text{w}} \notin [\![g_0]\!](i^{\text{w}})$. The only way for this to be possible is $i^{\text{w}} \not\subseteq L \setminus L^{\mathsf{dead}}$, which implies $i^{\text{w}} \cap L^{\mathsf{dead}} \neq \varnothing$, but this would imply f_G is not obstinate, hence not safe, which establishes the required contradiction.

For the inductive case, we need: $[\![g_{n+1}]\!] \sqsupseteq f_G$. First let $t = (\langle i, a\rangle \xrightarrow{o} \langle \top, \bot\rangle) \in \mathsf{Counter}(g_n)$ such that $o = c^{\text{o}}/c^{\text{i}}$. Now assume, for contradiction, $[\![g_{n+1}]\!] \not\sqsupseteq f_G$, this gives us a witness $i^{\text{w}} \in I$ and $c^{\text{w}} \in f_G(i^{\text{w}})$ such that $c^{\text{w}} \notin [\![g_{n+1}]\!](i^{\text{w}})$. The only way for this to be possible is $i^{\text{w}} \subseteq i$ and $c^{\text{w}} \in a$. This is because $\langle i, a\rangle$ is the only pair we took away from g_n and, by inductive hypothesis $c^{\text{w}} \in [\![g_n]\!](i^{\text{w}})$. Now let $\langle j, b\rangle$ be the meet of all $\langle j', b'\rangle \in g_n$ such that $j' \supseteq i^{\text{w}}$. Because we are using the lattice normal form it holds $\langle j, b\rangle \in g_n$. We have $i^{\text{w}} \subseteq j \subseteq i$ and $c^{\text{w}} \in b$, we make a case distinction on (i) $j \subset i$ or (ii) $j = i$. For (i) we just note that $\langle j, b\rangle \in g_{n+1}$ hence $c^{\text{w}} \in [\![g_{n+1}]\!](i^{\text{w}})$, so this establishes the required contradiction for this case. For case (ii), we make a new case distinction (1) $c^{\text{o}} \in f_G(i^{\text{w}})$, or (2) $c^{\text{o}} \notin f_G(i^{\text{w}})$. For case (1) it is easily seen that observation–closedness of f_G entails that for at least one of the information sets i^{c} of newly–introduced CPre–pairs we have $i^{\text{w}} \subseteq i^{\text{c}}$, hence $c^{\text{w}} \in [\![g_{n+1}]\!](i^{\text{w}})$,

which establishes the required contradiction in this case. For case (2) we have $c^{\mathrm{w}} \neq c^{\mathrm{o}}$, then we can show that obstinacy of f_G entails that $i^{\mathrm{w}} \subseteq i^{\mathrm{r}}$ where i^{r} is the information set of the newly introduced RSucc–pair, and since $c^{\mathrm{w}} \neq c^{\mathrm{o}}$ it must follow $c^{\mathrm{w}} \in [\![g_{n+1}]\!](i^{\mathrm{w}})$, which establishes the required contradiction. □

Theorem 2.26 (Correctness) Algorithm 1 always terminates with $[\![g]\!] = f_G$, i.e. the weakest safe strategy for game G. ◁

Proof Since the lattice of knowledge based strategies is finite, by Lemma 2.22 we have that the while loop must eventually terminate. By Lemma 2.23 we have that, $[\![g]\!]$ is observation–closed. By Lemma 2.24 we have $[\![g]\!]$ is obstinate. By Lemma 2.5 the latter two facts suffice to show that $[\![g]\!]$ is safe. Since f_G is the weakest such safe strategy we have $[\![g]\!] \sqsubseteq f_G$. By Lemma 2.25 we have $[\![g]\!] \sqsupseteq f_G$. It follows $[\![g]\!] = f_G$. □

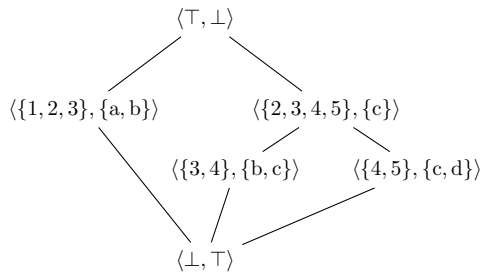
2.5 Efficiency Concerns and Optimizations

In the previous sections we gave an abstract presentation of a refinement algorithm for computing weakest knowledge based strategies in safety games of imperfect information. In this section we revisit two aspects of the definitions from an efficiency perspective: sparse normal forms and knowledge update transitions. When implemented naively, the operations on the knowledge update relation, and on the underlying allow lattice that CEDAR relies on, may become expensive to compute. For this reason we outline, in this section, optimized versions of these operations.

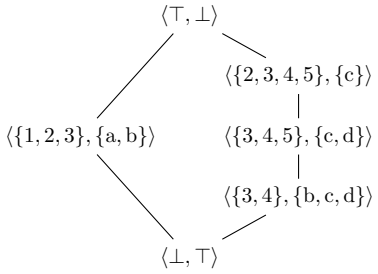
The optimizations we will discuss in the remainder of this section are conceptually quite straightforward yet can get very rich in technical detail very quickly, when completely formalized. In order to avoid an overly technical treatment of what should be essentially algorithmic ideas, these sections will have a less formal character than the previous. For the most part, the correctness of the proposed optimizations rests on the properties of the sparse normal form which were already established in Section 2.3.3.

In the previous sections we mostly assumed the lattice normal form (LNF) for storing strategies. In this section we show how it is possible to optimize this and use the sparse normal (SNF) form instead.

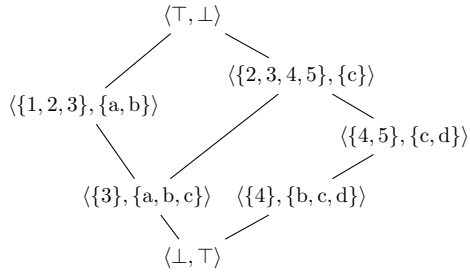
The most important thing to note when switching from LNF to SNF is that removing a pair from the lattice becomes more difficult because the poset will not be closed under finite meets and several non–trivial meet pairs of the pair–to–be–removed might be implicit in the poset.



(a) Initial Allow lattice in SNF.



(b) Initial with $\langle\{3, 4, 5\}, \{d\}\rangle$ added.



(c) Initial with $\langle\{3, 4\}, \{b, c\}\rangle$ removed.

Figure 2.19: Sparse Normal Forms as used in Example 2.12 and Example 2.13.

In the remainder we discuss the two key operations involved in handling allow lattices in sparse normal form: *adding* pairs and *removing* pairs. Both operations can be performed incrementally, exploiting the partial order structure of the diagram. And both operations can be performed in $O(n)$ steps where n is the size of the sparse lattice.

Adding a pair consists of two steps: (1) determining the set of direct ancestors, called *parents*, and the set of direct descendants, called *children*, of the new pair in the $\subseteq^{\langle s,\cdot\rangle}$–order, and (2) updating the allow sets of all the pairs that are below the newly added pair in order to maintain contravariance, and removing all the pairs that are subsumed by the new pair in the process.

For step (1), the simplest way to determine the parents (children) of the new pair is to run the following antichain closure procedure. Start with the singleton antichain that is the top (bottom) pair of the bounded poset and keep going

down (up) in the lattice of antichains until the antichain contains exactly the set of smallest (greatest) pair above (below) the to–be–added pair, this gives us the antichain of parents (children) of the to–be–added pair.

Once the parents of the new pair are known its allow set can be weakened with the join of the allow sets of the parents, the result can be checked for subsumption against the join of the allow sets of the parents. In this way we are checking the resulting pair to see if it is principal or not. If its allow set is subsumed by its joined parents, it is not principal and the pair need not be added. If its allow set is not subsumed by the joined allow set of its parents, it is principal and it must be added. Since the children of the new pair are determined, the place it will occupy in the poset is known so it can be added and the parent and child relations can be updated accordingly.

For step (2) contravariance can be ensured by another antichain closure procedure that starts with the children of the newly added pair and works its way down through the poset updating the allow set of all the pairs that are met on the way by joining them with the allow set of the new pair and removing subsumed pairs in the process.

Example 2.12 (Adding a Pair) Consider the initial allow lattice in SNF in Figure 2.19(a). To this initial lattice we will add a new pair: $\langle\{3,4,5\},\{\mathrm{d}\}\rangle$. The end–result is shown in Figure 2.19(b).

For step (1) we compute the parents of the new pair. We start with the antichain $\{\langle\top,\bot\rangle\}$ we then check the diagram for the children of $\langle\top,\bot\rangle$, those are: $\langle\{1,2,3\},\{\mathrm{a},\mathrm{b}\}\rangle$ and $\langle\{2,3,4,5\},\{\mathrm{c}\}\rangle$. Of these only $\langle\{2,3,4,5\},\{\mathrm{c}\}\rangle$ falls above the new pair, so we add this pair to the next antichain (restricting to maximal elements) this way we obtain: $\{\langle\{2,3,4,5\},\{\mathrm{c}\}\rangle\}$. Another iteration will not yield any new elements so the closure algorithm will stabilize on this antichain, meaning it is indeed the (singleton) antichain of parents. We then run the dual closure procedure to determine the children of the new pair. This time we start from $\{\langle\bot,\top\rangle\}$, and arrive, again after a single iteration, at $\{\langle\{3,4\},\{\mathrm{b},\mathrm{c}\}\rangle,\langle\{4,5\},\{\mathrm{c},\mathrm{d}\}\rangle\}$ which is the antichain of children.

For step (2) we weaken the allow set of the new pair by joining it with the allow sets of its parents, in this case it means we weaken the allow set from $\{\mathrm{d}\}$ to $\{\mathrm{c},\mathrm{d}\}$. Since this is not subsumed by the parents, we now enter the new pair, with weakened allow set, into the poset making sure that the diagram below the new pair remains contravariant and sparse. In this case it means we weaken the allow set associated with $\{3,4\}$ further from $\{\mathrm{b},\mathrm{c}\}$ to $\{\mathrm{b},\mathrm{c},\mathrm{d}\}$ and we prune out the pair with information set $\{4,5\}$ because its allow set $\{\mathrm{c},\mathrm{d}\}$ has become subsumed by the new pair. The end–result is shown in Figure 2.19(b). $\lhd$

Removing a pair from the poset consists of two steps: (i) removing the pair from the poset, and (ii) saturating the poset with all the non–trivial meets of the removed pair with some other pair in the diagram. Note that step (ii) is only necessary when we remove principal pairs, not when we prune out subsumed pairs, like for step (2) above.

For step (i), we may simply update the child and parent relation among the parents and children of the removed pair, i.e. each parent of the removed pair and each child of the removed pair will now stand in a parent and child relation, respectively, themselves. This effectively removes the pair from the poset, but also all the implied non–trivial meet pairs that would have been present if the poset would have been saturated into a complete lattice. This will be solved in the next step.

For step (ii), we run an antichain closure procedure starting from the top of the lattice working our way down. At each stage in the closure process we take the meet of the information set of the existing pair in the poset and the information set of the newly–removed pair, and the join of the allow set of the existing pair with the join of the allow set of the newly–removed pair. If the resulting information set is a strict subset of both, and the resulting allow set is a strict superset of both we know that we have a new non–trivial meet pair that has been implicit in the sparse normal form. So we add it to the poset to ensure that it remains a proper sparse reflection of the underlying lattice (which would be closed under finite meets). Note that we will never add the newly–removed pair itself because of the strict–subset/strict–superset requirement, the end result will be the same as when we first would have saturated the poset under finite meets, next removed the pair, and finally pruned away all the subsumed pairs to obtain the sparse normal form again.

Example 2.13 (Removing a Pair) Consider the initial allow lattice in SNF in Figure 2.19(a). From this initial lattice we will remove the pair $\langle\{3,4\},\{\mathrm{b},\mathrm{c}\}\rangle$. The end–result is shown in Figure 2.19(c).

For step (i) we simply remove the pair from the poset, updating the parent–child relations.

For step (ii) we start with $\langle\top,\bot\rangle$, for this pair we meet the information set $\top$ with $\{3,4\}$ and, obviously, we get $\{3,4\}$, we join the allow set $\bot$ with $\{\mathrm{b},\mathrm{c}\}$ and, obviously, we get $\{\mathrm{b},\mathrm{c}\}$. In other words we get the pair itself and hence the strict–subset/strict–superset requirement is not (yet) fulfilled. We therefore continue the closure–process from the antichain $\{\langle\top,\bot\rangle\}$ and look at the children of $\langle\top,\bot\rangle$, those are: $\langle\{1,2,3\},\{\mathrm{a},\mathrm{b}\}\rangle$ and $\langle\{2,3,4,5\},\{\mathrm{c}\}\rangle$. For the first child pair we meet $\{1,2,3\}$ with $\{3,4\}$ obtaining $\{3\}$, and we join $\{\mathrm{a},\mathrm{b}\}$ with

$\{b, c\}$ obtaining $\{a, b, c\}$, this gives us the new pair $\langle\{1, 2, 3\}, \{a, b, c\}\rangle$. It holds $\{3\} \subset \{2, 3\}$ and $\{a, b, c\} \supset \{b, c\}$ so the strict–subset/strict–superset requirement is fulfilled, and we add the new pair $\langle\{3\}, \{a, b, c\}\rangle$ to the diagram. For the second child pair we meet $\{2, 3, 4, 5\}$ with $\{3, 4\}$ and we obtain $\{3, 4\}$ so the strict subset requirement fails, we therefore continue with $\{\langle\{2, 3, 4, 5\}, \{c\}\rangle\}$ as the new antichain. We continue the closure process and look at the children of $\langle\{2, 3, 4, 5\}, \{c\}\rangle$ that is: $\langle\{4, 5\}, \{c, d\}\rangle$ — recall that the pair itself, $\langle\{3, 4\}, \{b, c\}\rangle$, has already been removed from the poset — for the still remaining child pair we meet $\{4, 5\}$ with $\{3, 4\}$ obtaining $\{4\}$, and we join $\{c, d\}$ with $\{b, c\}$ obtaining $\{b, c, d\}$, this gives us the new pair $\langle\{4\}, \{b, c, d\}\rangle$. It holds $\{4\} \subset \{3, 4\}$ and $\{b, c, d\} \supset \{b, c\}$ so the strict–subset/strict–superset requirement is fulfilled, and we add the new pair $\langle\{4\}, \{b, c, d\}\rangle$ to the diagram. The end–result is shown in Figure 2.19(c). $\lhd$

Chapter 3

State Based Control Synthesis

3.1 Introduction

In the previous chapter we have developed a symbolic refinement algorithm for computing a weakest strategy in a safety game of imperfect information. The resulting strategy is represented in the form of a contravariant allow lattice. In this chapter we discuss an extension of CEDAR that allows us to transform the allow lattice into a concrete finite state machine.

3.2 Basic Control Loop

To motivate more thoroughly the need for state based control synthesis we introduce the *basic control loop* as a reference architecture for a rudimentary reactive controller. We first illustrate the basic control loop with full, symbolic information update. This demonstrates how to directly apply the results of chapter 2 in a working controller that correctly implements the weakest, safe strategy as computed by CEDAR. We next illustrate the basic control loop with concrete information update. This gets us closer to the classical notion of a controller as a finite state machine. Possible advantages of the latter technique as opposed to the full information update are:

1. A faster information update that can be carried out on the simplest em-

bedded devices as it does not involve any (symbolic) computation.

2. The possibility to reify the control constraints *back* into a symbolic form.

The first advantage is mainly about practical considerations. The second advantage has more immediate theoretical impact, for, as we will see, compositionality requires us to work with partly constrained systems where (sub)controllers have become part of the symbolic representation of the plant behavior. We will come back to this in Chapter 4.

3.2.1 Control Loop with Full Information Update

The *basic control loop with full information update* describes a simple architecture for a control executing a knowledge based control strategy in the form of an allow lattice. The body of the loop consists of the *basic control cycle*. In each cycle the control will write its output, read its input, and, based on the output/input pair exchanged, update its information state.

Throughout this chapter we will assume to be working in the context of a single safety control problem in the form of a safety game. We recall Definition 2.1 where we introduced δ and i^{init} as the game board and the initial information set of this underlying control game. In particular note that $\delta(i)$ denotes the set of all successor locations of i and $\delta(i) \cap o$ denotes the set of all successor locations of i that are consistent with observation o.

In Algorithm 2 the basic control loop is presented in pseudocode. In Table 3.1 we show a sample trace of the basic control loop on an example control game. Before we treat this in more detail we first discuss the algorithm line by line.

In line 1 we initialize the internal information state of the control to the initial information set i^{init}. We then enter the main while–loop, of which the body is explained below. The while–condition states that the only way to break out of the control loop is when the controller enters into the *inconsistent* information state. This may happen when the control makes an observation that is not consistent with any of the successor states allowed by the plant model. When this happens the control loop can no longer ensure safety and must throw an exception[1].

[1]For an adequately modelled system the goal is, of course, for this exception never to occur. Even equipment failure should preferably be handled by modeling this as a special location, and recovering gracefully. This can be done, for example, by including a special control input modeling user intervention. However, since this is a completely open-ended problem (double failures, triple failures, etc.) there should always be the possibility for breaking out of the controller and entering into a fail-safe state that is engaged as a last resort.

Algorithm 2: Basic Control Loop with Full Knowledge Update

Data: A safe strategy in the form of an allow lattice $g \subseteq I \times A$, and the initial information set i^{init}

```
1  initialize i ← i^init
2  while i ≠ ⊥ do
3  |  compute allow set a ← g(i)
4  |  pick output c^out ← steer(a)
5  |  do
6  |  |  wait for next output to be acceptable
7  |  |  write c^out → outputports
8  |  in parallel with
9  |  |  wait for next input to be available
10 |  |  read c^in ← inputports
11 |  merge o ← c^out/c^in
12 |  update information state i ← Update(i, o)
13 throw exception
```

At the start of each control cycle, in line 3 the control computes the largest set of currently allowed control outputs. In line 4 the control selects a concrete control output from the symbolic set of allowed outputs. For generality, we make no assumptions on the order of arrival and departure of the control outputs and inputs, instead we assume that the outputs and inputs occur concurrently. In line 6 the control waits for the next control output to be due. In line 7 the chosen output is written to the control ports. Concurrently, in line 9 the control waits for the next control input to stabilize and become available. In line 10 the control reads the next control input from the input ports. In line 11 the two threads merge their data into the output/input pair that was exchanged. This output/input pair constitutes the observation that was made on the plant state. In line 12 we use the observation to update the internal information state of the control. This function is defined as follows.

Definition 3.1 (Full Knowledge Update Function) We define the *full knowledge update function* $\mathsf{Update} : I \to \mathcal{O} \to I$ as follows:

$$\mathsf{Update}(i, o) = \delta(i) \cap o$$

This update function computes the strongest possible successor information set based on the current information state and the observation. ◁

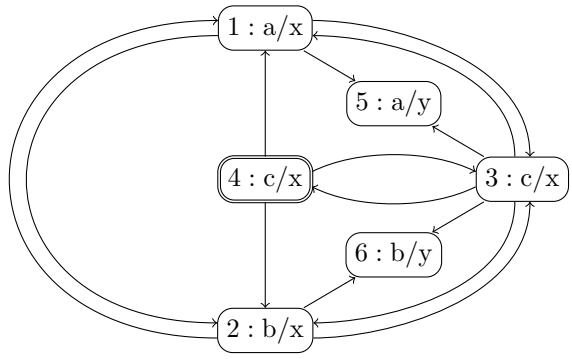


Figure 3.1: Game board for the racetrack system of Example 3.1.

To illustrate the basic control loop with full knowledge update we introduce the following control game example.

Example 3.1 (Racetrack Game) In Figure 3.1 we show the game board for the *racetrack game*. The system modeled in this control game is loosely based on the idea of a toy electric racetrack. It is chosen to be as compact as possible yet exhibit several interesting features concerning information retention. The racecar modeled in this control game can be in one of six locations. Locations $1, 2, 3$ and 4 are safe locations and locations 5 and 6 are failure locations. Let us say these correspond to the racecar going off the track while trying to exit/enter to/from the pit stop at location 4. The system will return at each cycle one of two control inputs: x or y. The control input x notifies a safe location and the control input y notifies a failure location. The system accepts at each cycle one of three control outputs: a, b or c. Control outputs a and b lead us to re-enter through the top and bottom of the track, respectively. Note that control outputs a and b can never be given twice in a row without ending up in a failure location. Control output c can always be given safely and results in alternation between locations 3 and 4. Due to this alternating behavior it becomes important *how often* the control output c occurred. Counting from location 4 an *odd* number of c's means the system is in location 3 in which it is *unsafe* to play either a or b, and an *even* number of c's means the system is in location 4 in which it is *safe* to play either a, b, or c. Because there is no observation distinguishing locations 3 and 4 the control strategy must retain at least one bit of information to distinguish between them. This is why the racetrack game is interesting from a state based control perspective. ◁

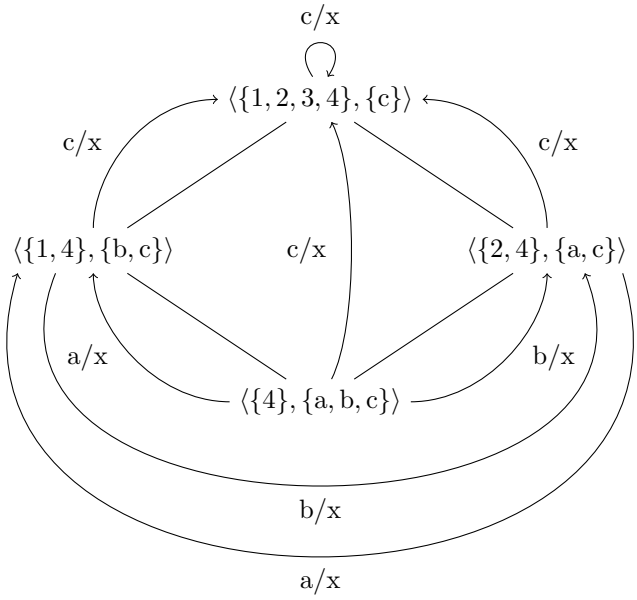


Figure 3.2: Strategy computed by CEDAR on the racetrack system of Example 3.1, the top and bottom pair are omitted for brevity.

In Figure 3.2 we show the weakest, safe, allow lattice as computed by CEDAR on the racetrack game of Example 3.1. In Table 3.1 we show a sample trace of the basic control loop for the racetrack game of Example 3.1. As can be seen, at each iteration, we know exactly what is the largest set of safe control outputs a. After seeing the observation sequence $\mathrm{a/x}, \mathrm{c/x}, \mathrm{c/x}$ we return once more to the state $i = \{4\}$ where $a = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$, i.e.: all control outputs are allowed.

3.2.2 Control Loop with Concrete Knowledge Update

The control loop introduced in the previous section provides a fully functional implementation of a safety controller where the control states are computed on-the-fly using the symbolic δ successor-set operation. We would now like to extend our approach to provide an equivalent controller that avoids the use of the δ successor-set operation and, instead, is based on the more traditional notion of a finite state machine.

To bridge our notion of an allow lattice to the more traditional notion of a

Table 3.1: Sample run of the basic control loop (Algorithm 2) on Example 3.1.

line **1**;	initialization:	$i \leftarrow \{4\}$
line **3**;	compute allow set:	$a \leftarrow \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$
line **4**;	pick control output:	$c^{\mathrm{out}} \leftarrow \mathrm{a}$
line **10**;	read control input:	$c^{\mathrm{in}} \leftarrow \mathrm{x}$
line **11**;	merge into observation:	$c^{\mathrm{out}}/c^{\mathrm{in}} \leftarrow \mathrm{a/x}$
line **12**;	update knowledge:	$i \leftarrow \{1\}$
line **3**;	compute allow set:	$a \leftarrow \{\mathrm{b}, \mathrm{c}\}$
line **4**;	pick control output:	$c^{\mathrm{out}} \leftarrow \mathrm{c}$
line **10**;	read control input:	$c^{\mathrm{in}} \leftarrow \mathrm{x}$
line **11**;	merge into observation:	$c^{\mathrm{out}}/c^{\mathrm{in}} \leftarrow \mathrm{c/x}$
line **12**;	update knowledge:	$i \leftarrow \{3\}$
line **3**;	compute allow set:	$a \leftarrow \{\mathrm{c}\}$
line **4**;	pick control output:	$c^{\mathrm{out}} \leftarrow \mathrm{c}$
line **10**;	read control input:	$c^{\mathrm{in}} \leftarrow \mathrm{x}$
line **11**;	merge into observation:	$c^{\mathrm{out}}/c^{\mathrm{in}} \leftarrow \mathrm{c/x}$
line **12**;	update knowledge:	$i \leftarrow \{4\}$
line **3**;	compute allow set:	$a \leftarrow \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$

finite state machine we make use of the knowledge update transitions as they were already introduced in the previous chapter. We will interpret the pairs in the allow lattice as the states in an automaton, and we will interpret the knowledge update relation as the transition relation of this automaton. For a correct interpretation of the allow lattice as a finite state machine we will need two extra conditions to be fulfilled.

As a first condition, the allow lattice must be closed under finite meets. In practice we would always proceed in a forward manner starting from the initial state and scanning observations to find successor states in the automaton using a simple forward saturation algorithm. We will proceed this way in order to avoid having to potentially close the entire diagram under finite meets, however for the purpose of this exposition we will leave such an optimization implicit and simply assume the entire diagram to be closed under finite meets.

The main reason to close the diagram under finite meets is to make the $R_{\overrightarrow{g}}$ relation *deterministic*. Note that $R_{\overrightarrow{g}}$, the right-minimal knowledge update relation, is the only suitable relation to use as the transition relation for a control automaton. This is so because the right-minimal knowledge update relation always selects the *strongest* possible successor knowledge in the diagram based on our current knowledge and the observation.

As a second condition, we must make sure that there are enough information sets present in the lattice to *retain all necessary information*. We will work this out formally in the next section. For Example 3.1 we remarked that playing the racetrack game requires the player to always retain at least one bit of information (whether an even/odd number of `c`'s were played). It is this type of information retention that requires additional information sets to be introduced into the diagram. To illustrate this further we first introduce the basic control loop with concrete knowledge update. Next, we will show how the naive translation (just taking the allow lattice as it is computed by CEDAR) does not contain enough states to retain all necessary information. In the next sections we will show how the allow lattice can be refined up to a point that the two approaches (full versus concrete information update) coincide.

Algorithm 3 differs from Algorithm 2 in lines 1 and 12. In line 1 the initialization is now performed by setting the internal information state of the control to the *best approximation* of the initial information set in g (cf. Definition 2.12). In line 12, the current knowledge is updated to reflect the observation using the *concrete knowledge update function* defined as follows:

Definition 3.2 (Concrete Knowledge Update Function) For a given allow lattice $g \subseteq I \times A$ we define the *concrete knowledge update function*, denoted:

Algorithm 3: Basic Control Loop with Concrete Knowledge Update

Data: A safe strategy in the form of an allow lattice $g \subseteq I \times A$, and the initial information set i^{init}

initialize $i \leftarrow i_g^{\text{init}}$
while $i \neq \bot$ **do**
  compute allow set $a \leftarrow g(i)$
  pick output $c^{\text{out}} \leftarrow \mathsf{steer}(a)$
  do
    wait for next output to be acceptable
    write $c^{\text{out}} \rightarrow \mathsf{outputports}$
  in parallel with
    wait for next input to be available
    read $c^{\text{in}} \leftarrow \mathsf{inputports}$
  merge $o \leftarrow c^{\text{out}}/c^{\text{in}}$
  update information state $i \leftarrow \mathsf{ConcreteUpdate}_g(i, o)$
throw exception

$\mathsf{ConcreteUpdate}_g : I \to \mathcal{O} \to I$, such that for a given information set $i \in I$ and a given observation $o \in \mathcal{O}$ it holds:

$$\mathsf{ConcreteUpdate}_g(i, o) = \bigcap \{i' \in g \mid i' \supseteq \mathsf{Update}(i, o)\}$$

Note that, for an allow lattice that closed under finite meets there is always precisely one such i' in the diagram, as long as o is allowed $\mathsf{ConcreteUpdate}_g$, coincides with the knowledge update relation $R_{\overline{g}}$. $\lhd$

In Table 3.2 we show a sample trace of the basic control with concrete knowledge updates for the same plant model and allow lattice as we used in Table 3.1. As can be seen the knowledge after observing the observation sequence $\mathrm{a/x, c/x, c/x}$ has degenerated to $i = \{1, 2, 3, 4\}$ and the associated allow set becomes $a = \{\mathrm{c}\}$. Contrast this to Table 3.1 where the knowledge and allow set, after the same observation sequence, were $i = \{3\}$ and $a = \{\mathrm{a, b, c}\}$, respectively. This shows that the naive translation of simply taking the allow lattice as computed by CEDAR does not suffice for the case of the concrete knowledge update.

Table 3.2: Sample run of the basic control loop with concrete knowledge updates (Algorithm 3) on Example 3.1 to be contrasted to Table 3.1.

line **1**;	initialization:	$i \leftarrow \{4\}$
line **3**;	compute allow set:	$a \leftarrow \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$
line **4**;	pick control output:	$c^{\mathrm{out}} \leftarrow \mathrm{a}$
line **10**;	read control input:	$c^{\mathrm{in}} \leftarrow \mathrm{x}$
line **11**;	merge into observation:	$c^{\mathrm{out}}/c^{\mathrm{in}} \leftarrow \mathrm{a/x}$
line **12**;	update knowledge:	$i \leftarrow \{1, 4\}$
line **3**;	compute allow set:	$a \leftarrow \{\mathrm{b}, \mathrm{c}\}$
line **4**;	pick control output:	$c^{\mathrm{out}} \leftarrow \mathrm{c}$
line **10**;	read control input:	$c^{\mathrm{in}} \leftarrow \mathrm{x}$
line **11**;	merge into observation:	$c^{\mathrm{out}}/c^{\mathrm{in}} \leftarrow \mathrm{c/x}$
line **12**;	update knowledge:	$i \leftarrow \{1, 2, 3, 4\}$
line **3**;	compute allow set:	$a \leftarrow \{\mathrm{c}\}$
line **4**;	pick control output:	$c^{\mathrm{out}} \leftarrow \mathrm{c}$
line **10**;	read control input:	$c^{\mathrm{in}} \leftarrow \mathrm{x}$
line **11**;	merge into observation:	$c^{\mathrm{out}}/c^{\mathrm{in}} \leftarrow \mathrm{c/x}$
line **12**;	update knowledge:	$i \leftarrow \{1, 2, 3, 4\}$
line **3**;	compute allow set:	$a \leftarrow \{\mathrm{c}\}$

3.2.3 Introducing History States into the Control Loop

In the previous section we gave a counterexample where the naive interpretation of the information contained in the allow lattice computed by CEDAR as the basis for representing the knowledge of a controller falls short of being maximally permissive. This means that we need to introduce auxiliary information sets into the allow lattice that maintain enough information about the history of the observation sequence to keep the controller maximally permissive. There should be enough information sets that are identified in the allow lattice such that the concrete knowledge updates retain just enough information for the controller to remain maximally permissive for *any* possible observation sequence.

As a first step to achieving this with a symbolic saturation algorithm we take a closer look at the racetrack game of Example 3.1. In particular we look at Figure 3.2 showing the weakest, safe allow lattice computed by CEDAR for this game. As is shown in the trace in Table 3.2 the problem occurs when we jump from knowledge $i = \{1, 4\}$ through observation c/x to $i = \{1, 2, 3, 4\}$. The full knowledge update would allow us to conclude that output c from location 1 or 4 always leads to location 3. However there is no information set $\{3\}$ present in the allow lattice. The reason that it is not present is because it is not relevant to the allow set (it is not principal), so the next knowledge defaults to the much weaker information $\{1, 2, 3, 4\}$. Even though the information set $\{3\}$ is not *principally* (0–step) relevant we say it is *historically* $(0, 1, 2, \ldots, n$–step) relevant.

In Figure 3.3 we show the same allow lattice now with the information set $\{3\}$ added to it. As can be seen the transition structure has been changed by the addition of the history state $\{3\}$. If we now run the basic control loop using this new allow lattice, again on the same game and observation sequence, we obtain the trace in Table 3.3. When we contrast this to Tables 3.2 and 3.1 we see that the implemented strategy is now maximally permissive, at least for this example.

The question remains what is the exact formal criterion by which the state $\{3\}$ qualifies as *historical*, and subsequently how to systematically introduce historical states so that the full information retention strategy and the concrete information update strategy coincide. We will formalize this more thoroughly in the next section. But first we sketch the general outline of the refinement step by which the diagram in Figure 3.3 may be obtained from the diagram in Figure 3.2.

Historical states are introduced using a generalization of the controllable predecessor step as used in the CEDAR refinement algorithm. Whether a right–

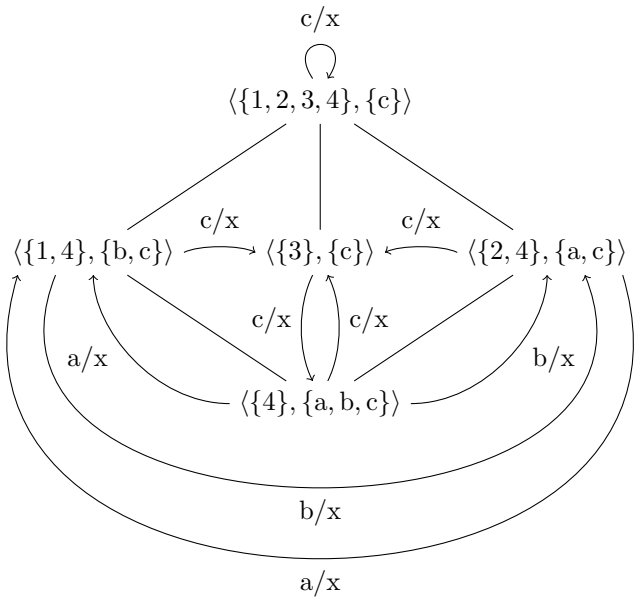


Figure 3.3: Same strategy as Figure 3.2, with the addition of history state {3}.

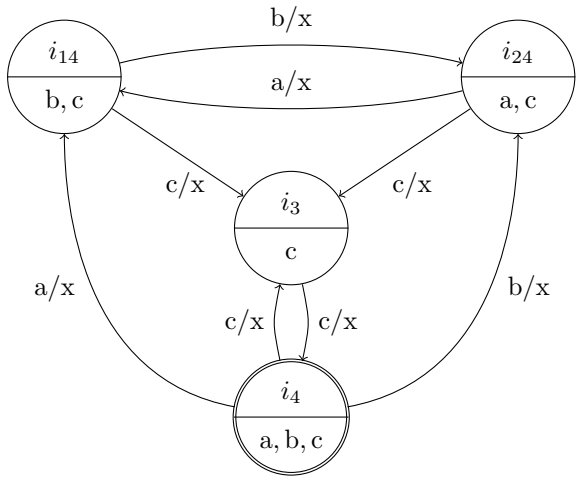
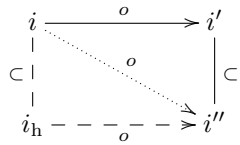


Figure 3.4: Reachable part of Figure 3.3 presented as a Moore/Mealy machine

Table 3.3: Sample run of the basic control loop (Algorithm 3) using the allow lattice in Figure 3.3 to be contrasted to Table 3.2.

line **1**;	initialization:	$i \leftarrow \{4\}$
line **3**;	compute allow set:	$a \leftarrow \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$
line **4**;	pick control output:	$c^{\mathrm{out}} \leftarrow \mathrm{a}$
line **10**;	read control input:	$c^{\mathrm{in}} \leftarrow \mathrm{x}$
line **11**;	merge into observation:	$c^{\mathrm{out}}/c^{\mathrm{in}} \leftarrow \mathrm{a/x}$
line **12**;	update knowledge:	$i \leftarrow \{1, 4\}$
line **3**;	compute allow set:	$a \leftarrow \{\mathrm{b}, \mathrm{c}\}$
line **4**;	pick control output:	$c^{\mathrm{out}} \leftarrow \mathrm{c}$
line **10**;	read control input:	$c^{\mathrm{in}} \leftarrow \mathrm{x}$
line **11**;	merge into observation:	$c^{\mathrm{out}}/c^{\mathrm{in}} \leftarrow \mathrm{c/x}$
line **12**;	update knowledge:	$i \leftarrow \{1, 2, 3, 4\}$
line **3**;	compute allow set:	$a \leftarrow \{\mathrm{c}\}$
line **4**;	pick control output:	$c^{\mathrm{out}} \leftarrow \mathrm{c}$
line **10**;	read control input:	$c^{\mathrm{in}} \leftarrow \mathrm{x}$
line **11**;	merge into observation:	$c^{\mathrm{out}}/c^{\mathrm{in}} \leftarrow \mathrm{c/x}$
line **12**;	update knowledge:	$i \leftarrow \{4\}$
line **3**;	compute allow set:	$a \leftarrow \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$

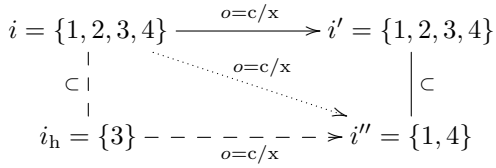
minimal, knowledge update transition $i \xrightarrow{o} i'$ is a candidate for refinement depends on the direct, non–trivial children of i' in the allow lattice. If there exists a direct, non–trivial child $i'' \subset i'$ in the allow lattice and $i'' \cap o \neq \varnothing$ we call $i \xrightarrow{o} i'$ a *duplicable edge*. In this case we can compute the largest set $i_{\mathrm{h}} \subset i$ such that $i_{\mathrm{h}} \xrightarrow{o} i''$, i.e. we try to strengthen the source information set to make the knowledge update transition land at the stronger target information set. This results in the following diagram:



(3.1)

Where i_{h} forms the newly introduced historical state. In effect we are "pulling–out" the source state i of the transition $i \xrightarrow{o} i'$ with the aim of strengthening the target state to $i'' \subset i'$. The dotted transition from i to i'' is called the *potential knowledge update transition*.

For the example in Figure 3.2 this applies because we have the following situation:



So indeed, we arrive at $\{3\}$ as the newly introduced historical state, yielding Figure 3.3. By symmetry, the same result can be obtained using $i'' = \{2, 4\}$.

Note that this is a generalization of the controllable predecessor step as we are using it in the CEDAR algorithm. The generalization lies in the fact that we not only consider unsafe, right–minimal knowledge update transitions that lead to $\langle \top, \bot \rangle$ but we consider right–minimal knowledge update transitions that have arbitrary target states as long as these target states have non–trivial children states supporting the same observation. In Section 3.3 we will make this refinement step precise. But first we formally introduce the necessary, additional semantical concepts.

3.3 State Based Control

In the previous section we have outlined an idea of how to come from knowledge based strategies to history based strategies by identifying a small, finite basis of information. We also outlined how to compute such a basis of information by refining the allow lattice further than what is achieved in a basic run of the CEDAR refinement algorithm. In this section and the following we will give a more formal treatment of this idea, first on a semantical level.

3.3.1 History Based Strategies

We start by defining the standard game theoretical notion of a *history based strategy*. This will be a suitable extension of the knowledge based semantics that, in addition to the initial information set, takes into a account the *sequence of observations* leading to a new, updated information set.

Definition 3.3 (HBS) A *history based strategy* is a function $h : I \to \mathcal{O}^* \to A$ mapping an initial information set and a history of observations to an allow set. The set of all history based strategies is denoted HBS. We define $h_\bot$ as the strategy that maps all initial information sets and histories to $\bot$ and $h_\top$ as the strategy that maps all initial information sets and histories to $\top$, and $h_1 \sqcup h_2$ as the strategy h' that maps a given information set $i \in I$ and history $\sigma \in \mathcal{O}^*$ to $h'(i,\sigma) = h_1(i,\sigma) \cup h_2(i,\sigma)$. We define $\sqcap$ analogously using intersection of allow sets. Finally we define a partial order $\sqsubseteq$ on HBS such that $h \sqsubseteq h'$ iff for all $i \in I$ and $\sigma \in \mathcal{O}^*$ it holds $h(i,\sigma) \subseteq h'(i,\sigma)$. $\lhd$

It is quite straightforward to see that $(\mathsf{HBS}, \sqsubseteq)$ forms a complete lattice. The strategies $h_\top$ and $h_\bot$ form the top and bottom elements, respectively. And the operations $\sqcup$ and $\sqcap$ form the natural join and meet, respectively. Note that every history based strategy $h \in \mathsf{HBS}$ naturally corresponds to a knowledge based strategy $f \in \mathsf{KBS}$ such that for all $i \in I$ it holds $f(i) = h(i,\epsilon)$.

3.3.2 History Aware Semantics

Analogous to Section 2.3.2 where we introduced the knowledge based semantics for pairsets we will now introduce the two new history based semantics for pairsets that we already informally introduced in Sections 3.2.1 and 3.2.2.

Definition 3.4 (Full Knowledge Update Semantics) We lift the *full knowledge update function* as defined in Definition 3.1 to possibly empty sequences of

observations $\mathsf{Update}^* : I \to \mathcal{O}^* \to I$ with the following recurrence:

$$\begin{aligned}\mathsf{Update}^*(i, \epsilon) &= i\\ \mathsf{Update}^*(i, o^\frown\sigma) &= \mathsf{Update}^*(\mathsf{Update}(i, o), \sigma)\end{aligned}$$

Based on this we define for a given $g \subseteq I \times A$ the *full knowledge update strategy* $\mathsf{Allow}_g : I \to \mathcal{O}^* \to A$ as the history based strategy such that for all $i \in I$ and $\sigma \in \mathcal{O}^*$ it holds:

$$\mathsf{Allow}_g(i, \sigma) = [\![g]\!](\mathsf{Update}^*(i, \sigma))$$

This strategy uses the full update function to determine the strongest successor knowledge, next it uses the knowledge based strategy $[\![g]\!]$ to specify which allow set should be played. ◁

Definition 3.5 (Concrete Update Semantics) For a given set $g \subseteq I \times A$, we lift the *concrete knowledge update function* as defined in Definition 3.2 to possibly empty sequences of observations $\mathsf{ConcreteUpdate}^*_g : I \to \mathcal{O}^* \to I$ with the following recurrence:

$$\begin{aligned}\mathsf{ConcreteUpdate}^*_g(i, \epsilon) &= \bigcap\{i' \in g \mid i' \supseteq i\}\\ \mathsf{ConcreteUpdate}^*_g(i, o^\frown\sigma) &= \mathsf{ConcreteUpdate}^*_g(\mathsf{ConcreteUpdate}_g(i, o), \sigma)\end{aligned}$$

Based on this we define the *concrete knowledge update strategy* as the history based strategy $\mathsf{ConcreteAllow}_g : I \to \mathcal{O}^* \to A$ such that for all $i \in I$ and $\sigma \in \mathcal{O}^*$ it holds:

$$\mathsf{ConcreteAllow}_g(i, \sigma) = [\![g]\!](\mathsf{ConcreteUpdate}^*_g(i, \sigma))$$

This strategy starts with the best approximation knowledge of i, it then uses the concrete update function to compute the best approximation of the successor knowledge and finally it uses the knowledge based strategy $[\![g]\!]$ to specify which allow set should be played. ◁

The following lemmas illustrate the connection between the full and the concrete knowledge update semantics:

Lemma 3.6 For all $g \subseteq I \times A$ it holds $\mathsf{ConcreteAllow}_g \sqsubseteq \mathsf{Allow}_g$. ◁

Proof For all $i \in I$ and all $o \in \mathcal{O}$ it holds $\mathsf{ConcreteUpdate}_g(i, o) \supseteq \mathsf{Update}(i, o)$, i.e.: $\mathsf{ConcreteUpdate}_g$ overapproximates Update. Then by a simple induction

on the length of observation sequences we obtain the same for the iterated knowledge update functions, i.e.: $\mathsf{ConcreteUpdate}_g^*$ overapproximates Update^* Then by contravariance of $[\![g]\!]$ it follows that $\mathsf{ConcreteAllow}_g$ underapproximates Allow_g. □

Lemma 3.7 For every allow lattice $g \subseteq I \times A$ it holds that there exists a refinement $g' \supseteq g$ such that $\mathsf{ConcreteAllow}_{g'} = \mathsf{Allow}_g$. ◁

Proof We may take $g' = \{\langle i, [\![g]\!](i)\rangle \mid i \in I\}$, i.e. the full powerset lattice. For this g' we have for all $i \in I$ and $o \in \mathcal{O}$ it holds $\delta(i) \cap o \in g'$ and hence for all $i \neq \varnothing$ it holds $\mathsf{ConcreteUpdate}_{g'}(i, o) = \mathsf{Update}(i, o)$. □

The latter existence proof relies on the full powerset construction. For a more practical result we need a way to compute a *sufficient* refinement $g' \supseteq g$ that retains enough information such that $\mathsf{ConcreteAllow}_{g'} = \mathsf{Allow}_g$ without immediately requiring the full powerset construction.

3.3.3 State Based Control Synthesis Algorithm

In the previous chapter we developed an algorithm to symbolically compute the weakest, safe allow set for any given information set in a safety game. We did this using a counter example driven refinement step. In this section we look at a new type of refinement step that allows us to refine the strategy further than what is achieved in a basic run of the CEDAR algorithm. The refinement step will be based on the notion of a duplicable edge observation which generalizes the controllable predecessor refinement.

Definition 3.8 (Potential Knowledge Update Relation) In the remainder, will often use shorthand notation writing $i \in g$ instead of $\exists a.\langle i, a\rangle \in g$ and $(i \xrightarrow{o} i') \in R$ instead of $\exists a, a' \in A.\langle i, a\rangle \xrightarrow{o} \langle i', a'\rangle \in R$ wherever this is clear from the context. For a given $g \subseteq I \times A$ we define the *potential knowledge update relation* $R_g^{\mathsf{pot}} \subseteq g \times \mathcal{O} \times g$ such that $i \xrightarrow{o} i'' \in R_g^{\mathsf{pot}}$ iff there exists a *duplicable edge* $i \xrightarrow{o} i' \in R_{\overline{g}}$ such that $i' \supset i''$. ◁

For a diagram illustrating the above definition cf. Page 85 Figure 3.1. Intuitively a potential knowledge update transition can be turned into a true knowledge update transition by constraining the source information set. We can then treat the potential knowledge update relation by inserting this strengthened predecessor into g. This brings us in a position to define the DEODAR algorithm, formally, as a fixed–point iteration.

Definition 3.9 (DEODAR Fixed Point) For a given $g \subseteq I \times A$ we define the successive *historical approximations* of g using the following recursion:

$$g_0 = g$$
$$g_{n+1} = g_n \cup \{\langle i_{\mathrm{h}}, g(i_{\mathrm{h}})\rangle \mid \exists (i \xrightarrow{o} i'') \in R^{\mathsf{pot}}_{g_n} \text{ and } i_{\mathrm{h}} = i \setminus \delta^{-1}((\delta(i) \cap o) \setminus i'')\}$$

We define the *historical fixed point* $\overline{g} = \lim_{n\to\infty} g_n$. ◁

The purpose of the DEODAR algorithm is to compute a diagram that contains enough information sets so that the concrete knowledge update semantics coincides with the full knowledge update semantics. This is expressed in the following theorem.

Theorem 3.10 For any allow lattice $g \subseteq I \times A$ it holds that

$$\mathsf{ConcreteAllow}_{\overline{g}} = \mathsf{Allow}_g$$

i.e.: the history based strategy with concrete information updates, on the saturated diagram, coincides with the history based strategy with full information updates on the original diagram. ◁

Proof Let σ be some arbitrary sequence of observations, more precisely, wlog:

$$\sigma = o_0 \dots o_n \in \mathcal{O}^*$$

Let $i_0 i_1 \dots i_n$ be the successive knowledge information sets for a player (called the *full player*) that follows the *full knowledge update strategy*, more precisely, we define:

$$i_m = \mathsf{Update}^*(i^{\mathrm{init}}, o_0 \dots o_m)$$

Finally let $i'_0 \dots i'_n$ be the successive knowledge information sets for a player (called the *concrete player*) that follows the *concrete knowledge update strategy*, more precisely, we define:

$$i'_m = \mathsf{ConcreteUpdate}^*(i^{\mathrm{init}}, o_0 \dots o_m)$$

We now need the following:

1. At each point in the sequence both players' allow sets will be the same:

$$\text{for all } 0 \le m \le n \text{ it holds } [\![g]\!](i_m) = [\![\overline{g}]\!](i'_m)$$

We proceed by defining a simulation function $\mathsf{sim} : I \to \overline{g}$ that assigns each possible information set its strongest approximation in $\overline{g}$, more precisely for all $i \in I$ we define

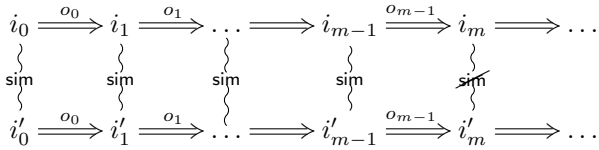
$$\mathsf{sim}(i) = \bigcap \{i' \in \overline{g} \mid i' \supseteq i\}$$

Now, since we only added pairs of the form $\langle i_{\mathrm{h}}, [\![g]\!](i_{\mathrm{h}})\rangle$ to $\overline{g}$, it is not hard to see that $[\![g]\!](i_m) = [\![\overline{g}]\!](\mathsf{sim}(i_m))$. This means that for proving 1 it would suffice to show the following:

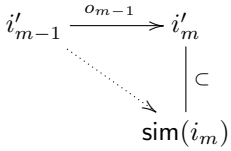
2. At each point in the sequence the concrete player's knowledge information set will be the strongest approximation of the full player's information set:

$$\text{for all } 0 \leq m \leq n \text{ it holds } i'_m = \mathsf{sim}(i_m)$$

Now to prove 2 we proceed by contradiction. For contradiction assume that 2 does not hold. Now, without loss of generality, let m be the first index at which it no longer holds $i'_m = \mathsf{sim}(i_m)$. By definition we have $i'_0 = \mathsf{sim}(i_0)$ so it must follow $m > 0$, in particular this proves there is at least one predecessor for which it still holds $i'_{m-1} = \mathsf{sim}(i_{m-1})$. The situation can be summed up in the following diagram:

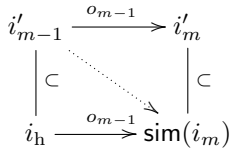


Where the upper double arrows denote an application of the full Update function, and the lower double arrows denote an application of the ConcreteUpdate function. Since we have $i'_m \supseteq \mathsf{sim}(i_m)$ (otherwise the concrete player's information would be inconsistent with the full player's information) and $i'_m \neq \mathsf{sim}(i_m)$ it follows that $i'_m \supset \mathsf{sim}(i_m)$. Now we have the following situation:



In particular it follows $i'_{m-1} \xrightarrow{o_{m-1}} \mathsf{sim}(i_m) \in R^{\mathsf{pot}}_{\overline{g}}$, i.e.: it is a potential knowledge update transition. Then, by closure of $\overline{g}$ under DEODAR we get the existence of

an additional historical information set $i_{\mathrm{h}} \in \overline{g}$ such that:



Then we get, by construction of the controllable predecessor i_{h}, that it must hold: $i'_{m-1} \supset i_{\mathrm{h}} \supseteq i_{m-1}$. This contradicts our assumption that i'_{m-1} is the best approximation of i_{m-1} in $\overline{g}$. □

3.4 Efficiency Concerns and Optimizations

In this section we discuss an important optimization for DEODAR. We start from the observation that, in many cases, it is possible to have a reduction in the number of control states if we relax the strict requirement of generating the *weakest strategy*, to any strategy that has *the same outcome as the weakest strategy*. To see this consider the following example.

Example 3.2 (Orthogonal Game) Consider the game board shown in Figure 3.5. This very simple game is called the *orthogonal game* because the control input and output are orthogonal in the sense that the output has no bearing on the input and vice versa. In particular, the only safe control output is 'a' regardless of what is the input, and the input alternates strictly between 'x' and 'y' regardless of what is the output.

In Figure 3.6 we show the derivation of a weakest strategy for this control game. In Figure 3.6(a) we show the initial allow lattice, in Figure 3.6(b) we show the allow lattice yielded by CEDAR, and in Figure 3.6(c) we show the allow lattice yielded by DEODAR.

Intuitively the result in Figure 3.6(b) is sufficiently strong as a winning strategy: the control output is restricted to always be safe, and the control input is, as we noticed earlier, not important to the control strategy. Yet as we can see in Figure 3.6(c), the history based strategy as computed by DEODAR does require an additional state distinction. The reason for this are the dashed transitions leading to the inconsistent state: $\bot$. In order to detect when such an inconsistent observation occurs the strategy needs to distinguish locations ℓ_0 and ℓ_2. ◁

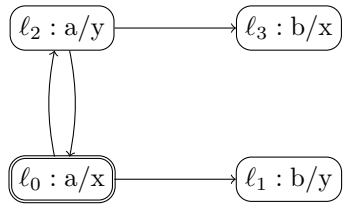


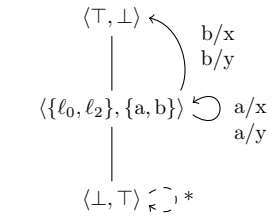
Figure 3.5: Ortogonal game board.

As can be seen from the previous example: sometimes state distinctions made by DEODAR are only present to predict what are *inconsistent observations*. This type of distinctions are fundamentally different from state distinctions that are necessary to predict the result of *consistent observations*. In particular the latter type of distinction may influence the outcome whereas the former type of distinction does not influence the outcome.

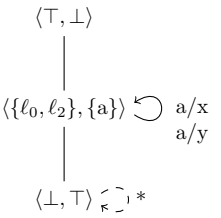
When we relax the requirement of a strictly weakest strategy to a strategy that has the same outcome as the weakest strategy on the underlying plant model, we obtain the possibility for an important optimization of the DEODAR algorithm. The optimization consists in an additional requirement on the *potential knowledge update relation* $i \xrightarrow{p} i'' \in R_g^{\text{pot}}$ where we require i'' to be nonempty. We call this optimization *loose inconsistency tracking* because it entails that not every inconsistent observation will be detected in general, yet a finite state controller derived in this way is potentially a lot smaller than a finite state controller that does *strict inconsistency tracking*.

Example 3.3 (Loose Inconsistency Tracking) If we consider CEDAR+DEODAR applied to the game board shown in Figure 3.5, this time with loose inconsistency tracking enabled the end result will be Figure 3.6(b) instead of Figure 3.6(c). If we then consider an inconsistent observation sequence, for instance: $\mathrm{a/y}, \mathrm{a/y}$ we see that, instead of ending up in the bottom state $\langle \bot, \top \rangle$ like in Figure 3.6(c) we stay in the state $\langle \{\ell_0, \ell_2\}, \{\mathrm{a}\} \rangle$. $\lhd$

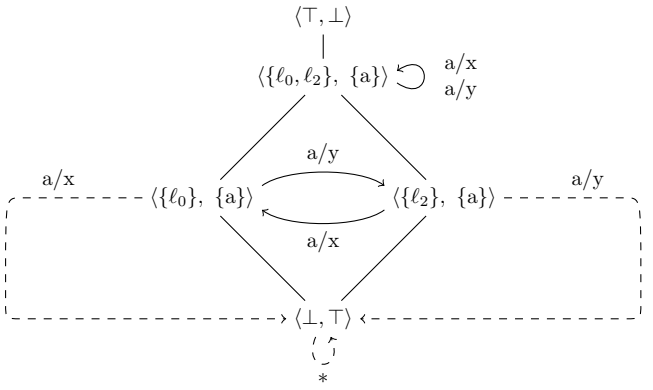
As can be seen from the previous example, in some cases it is possible to significantly simplify the allow lattice (less state distinctions) in the case of loose inconsistency tracking. Instead of "accept–all" semantics for inconsistent observation sequences we obtain more of a "don't–care" semantics. This type of simplification is somewhat analogous to the situation for BDDs. Also BDD structures can sometimes be significantly simplified based on a "don't–care" subset of their domain. Clearly the principle is related, although in our case



(a) Initial allow lattice.



(b) Result after CEDAR.



(c) Result after CEDAR+DEODAR.

Figure 3.6: Example run of CEDAR+DEODAR on the orthogonal game.

we are dealing with languages of inconsistent observation sequences rather than simple value sets.

Part II

Compositionality

Chapter 4

Compositional Control Synthesis

4.1 Introduction

In the previous chapters we have discussed symbolic methods for solving safety control problems over plant models that feature partial observability. In this chapter we show how to employ these techniques in a compositional framework.

Compositionality refers to the property of a constraint derived on a component level to remain valid when this component is subsequently composed with other components to form a larger system.

We are interested in compositionality for reasons of scalability. By relying on compositionality we are able to derive control constraints on small, individual components of the plant model first, and then to gradually increase the scope before, finally, synthesizing an integrated controller capable of controlling the entire plant.

The compositional approach may be contrasted to the monolithic approach where we would treat the entire, uncontrolled plant model at once, without first restricting the behavior of sublocalities. Our working hypothesis is that the compositional approach gives better scaling behavior than the monolithic approach.

How the idea of compositionality works out in the case of control synthesis is illustrated in Figure 4.1. As in Figure 2.1 from Chapter 2 the diagram shows the two principal domains involved: the control and the plant. However, in

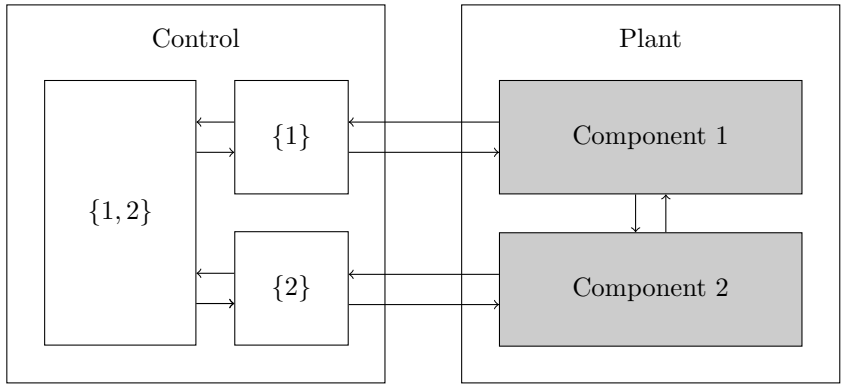


Figure 4.1: Two plant components, and three control localities.

addition we now show some more of the fine–grained structure that is present in the plant model. In this case the plant model decomposes into two distinct *plant components*.

The main problem of compositionality is to derive control constraints for a given *subset* of the full set of plant components. We refer to such a subset as a *locality*. In Figure 4.1 there are two plant components so there are three relevant control localities: $\{1\}, \{2\}$ and $\{1, 2\}$. Locality $\{1\}$ is responsible for the safety of plant component 1 insofar it can be locally enforced, control locality $\{2\}$ is responsible for the safety of plant component 2 insofar it can be locally enforced, and finally control locality $\{1, 2\}$ is responsible for filtering out any unsafe behavior that cannot be enforced on the level of either control locality $\{1\}$ or $\{2\}$. In this chapter we discuss how this can be achieved without loss of generality.

The rest of this chapter is structured as follows. In Section 4.2 we introduce a running example that is used in the remainder. In Section 4.3 we introduce the notion of a propositional transition system, which will form the semantic object of interest for the remainder of the chapter. In Sections 4.4 and 4.5 we make precise the link between the symbolic framework and the algorithms that we have developed in Chapters 2 and 3. And finally, in Section 4.7 we present a compositional algorithm for solving safety control problems.

4.2 Motivating Example

We consider the *parcel plant* illustrated in Figure 4.2(a). This fictive plant consists of a *feeder* and two *stamps* connected together with a conveyor belt. A parcel is fed onto the belt by the feeder. The belt transports the parcel over to stamp 1 which prints a tracking code. The belt then transports the parcel over to stamp 2 which stamps the shipping company's logo.

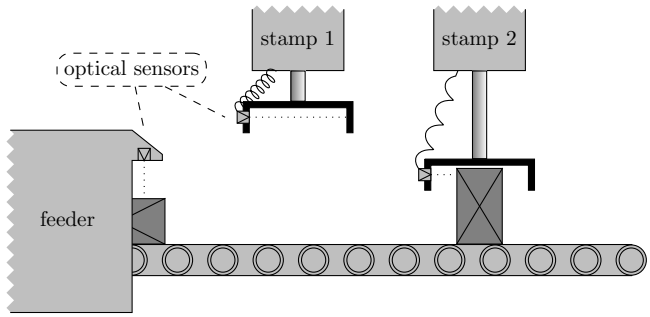
In Figure 4.2(b) we show the decomposition structure of the model. As can be seen the three physical units that make up the plant are modeled as three separate plant components. The interface between the various components is made up of output and input propositions. Propositions are single bits of information that encode a particular piece of state in the plant model. Some propositions are readable by the control, these are *control input propositions*, some propositions are writable by the control, these are *control output propositions*, and some propositions are neither readable nor writable, these are *uncontrollable* (plant) propositions.

In Figure 4.2(c) we list all the propositions we used to model the various components of this particular example. For each proposition we indicate whether it is a control output or input, and whether or not it holds for the current state of the plant as it is shown in the picture in Figure 4.2(a).

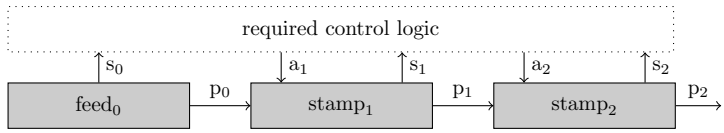
4.3 Propositional Transition Systems

In order to formalize the parcel plant example, as introduced in the previous section, we need to specify the internal behavior of the various plant components involved. As a simple and light–weight modeling framework we use *propositional transition systems* to specify the behavior of plant components.

Definition 4.1 (PTSs) We let $\mathcal{X}$ be a background set of *propositions*. For a given subset $X \subseteq \mathcal{X}$ a function $s : X \to \{0,1\}$ mapping each proposition to a truth value is called a *state* or *valuation* over X, with $\mathcal{S}[X]$ we denote the full set of all possible states over X. A *propositional transition system* (PTS) is a tuple $P = (L, X, \gamma, \delta, L^{\text{init}})$, consisting of a finite set of *locations* L, a finite set of *relevant propositions* $X \subseteq \mathcal{X}$, a *propositional labeling* $\gamma : L \to \mathcal{S}[X]$, a *transition relation* $\delta \subseteq L \times L$, and a set of *initial locations* $L^{\text{init}} \subseteq L$. With $\mathcal{P}$ we denote the set of all PTSs. We define an operation $\mathsf{Reach} : \mathcal{P} \to \mathcal{P}$ such that $\mathsf{Reach}(P)$ denotes the restriction of P to *reachable locations* which are all locations which can be reached by at least one path of transitions starting from some initial location. For a given PTS P we define $\mathsf{Traces}(P)$ as the language of



(a) Illustration of the parcel stamping plant.



(b) Decomposition structure of the parcel stamping plant model.

Prop.:	Description:	Type:	Fig. (a):
p_0	parcel present at belt location 0	uncontrollable	true
s_0	optical sensor 0 triggered	control input	true
p_1	parcel present at belt location 1	uncontrollable	false
a_1	stamping arm 1 activated	control output	false
s_1	optical sensor 1 triggered	control input	false
p_2	parcel present at belt location 2	uncontrollable	true
a_2	stamping arm 2 activated	control output	true
s_2	optical sensor 2 triggered	control input	true

(c) Legend of all the propositions used in modeling the parcel stamping plant.

Figure 4.2: A modular parcel stamping plant, illustration (a), decomposition structure (b) and legend of propositions used in the modeling (c).

finite or infinite traces $\sigma \in \mathcal{S}[\mathcal{X}]^* \cup \mathcal{S}[\mathcal{X}]^\omega$ of maximal states that are consistent with some maximal, finite or infinite path of locations through P starting at some initial location. $\lhd$

Propositional transition systems can be seen as a specialization of the more general *labeled transition systems*, where the set of edge labels is degenerate (singleton) and the set of state labels is the set of propositional valuations. Alternatively PTSs can be seen as *Kripke structures*, except for the fact that we do not require the property of *left totality* (or deadlock freedom). Note that, after synthesis this property will hold on the resulting system. So that, indeed, after synthesis the resulting system (plant composed with controller) will be a Kripke structure.

The reason for fixing our semantics at the level of PTSs is twofold. On the one hand, they are rich enough to describe all the semantic objects that we will need in the remainder (plant, component, game board, controller). On the other hand, they are technically simpler than the full blown notion of a labeled transition system. In particular PTSs permit a natural, well known symbolic representation using binary decision diagrams.

To make the relationship between PTSs and symbolic notation precise we will introduce a basic primed propositional logic as a light–weight formalism.

Definition 4.2 (Primed Propositional Logic) Recall the definition of $\mathcal{X}$ as the set of *atomic propositions*. We now define the set of *primed atomic propositions* $\mathcal{X}' = \{x' \mid x \in \mathcal{X}\}$, i.e. if $p \in \mathcal{X}$ then $p' \in \mathcal{X}'$. The set of *primed propositional logic formulas* is then defined using the following production rule:

$$\phi ::= \top \mid \phi \wedge \phi \mid \neg\phi \mid x \mid x'$$

where x may be any atomic proposition $x \in \mathcal{X}$, and x' may be any *primed* atomic proposition $x' \in \mathcal{X}'$. In addition to the primitives $\top$, $\wedge$ and $\neg$ we define the usual derived boolean connectives $\phi \vee \psi \equiv \neg(\neg\phi \wedge \neg\psi)$, $\phi \rightarrow \psi \equiv \neg\phi \vee \psi$, $\phi \leftrightarrow \psi \equiv (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$, $\bot \equiv \neg\top$. We define a *state formula* as any formula $\phi[X]$ that does not contain primed propositions. We define a *transition formula* as any formula $\phi[X, X']$ that may contain both primed as well as unprimed propositions.

The truth of a state formula $\phi[X]$ on some state $s \in \mathcal{S}[X]$ is defined induc-

tively on the structure of ϕ as follows.

$$
\begin{array}{ll}
s \vDash \top & \\
s \vDash x & \text{iff } s(x) = 1 \\
s \vDash \phi \wedge \psi & \text{iff } s \vDash \phi \text{ and } s \vDash \psi \\
s \vDash \neg\phi & \text{iff not } s \vDash \phi
\end{array}
$$

The truth of a transition formula $\phi[X, X']$ on some pair of states $(s, s') \in \mathcal{S}[X] \times \mathcal{S}[X]$ is defined similarly, inductively on the structure of ϕ as follows.

$$
\begin{array}{ll}
(s, s') \vDash \top & \\
(s, s') \vDash x & \text{iff } s(x) = 1 \\
(s, s') \vDash x' & \text{iff } s'(x) = 1 \\
(s, s') \vDash \phi \wedge \psi & \text{iff } (s, s') \vDash \phi \text{ and } (s, s') \vDash \psi \\
(s, s') \vDash \neg\phi & \text{iff not } (s, s') \vDash \phi
\end{array}
$$

For a given PTS P over X we say a given state formula $\phi[X]$ holds on P (notation $P \vDash \phi$) iff for all $\ell \in L_{\mathsf{Reach}(P)}$ it holds $\gamma(\ell) \vDash \phi$, and we say a given transition formula $\phi[X, X']$ holds on P iff for all $(\ell, \ell') \in \delta_{\mathsf{Reach}(P)}$ it holds $(\gamma(\ell), \gamma(\ell')) \vDash \phi$.

The presence of both primed as well as unprimed propositions allows us to symbolically relate the *current state* of the system (using the propositions in $\mathcal{X}$) and the *next state* of the system (using the propositions in $\mathcal{X}'$). As an example of this $\mathrm{p} \rightarrow \mathrm{p}'$ would denote a transition invariant saying that if p holds in the current state then p holds in the next state. So p and p' essentially denote the same proposition except at different moments in time.

If we want a state formula $\phi[X]$ to be interpreted as a transition formula (despite the fact that it does not contain primed variables) we append the special postfix operator $(\cdot)^\dagger$. As an example the formula $p^\dagger$ is read as: *p–or–deadlock*, because this formula when interpreted as transition formula only holds on PTSs where from all the states where p is *not true* there are no successors. The $(\cdot)^\dagger$ notation is especially useful for introducing *safety requirements* because, in our framework, deadlock states are by definition considered to be *unsafe states*. $\lhd$

At this point the reader may wonder about the separation of concerns regarding the activities of *modeling* and *specifying*. It is true that by incorporating the required safety invariant into the model we are mixing the modeling of the *possible* plant behavior with the *required* plant behavior. On the one hand this can be seen as a formal convenience. It is entirely possible to have two separate

models: one for the possible behavior of the plant component and one for the required safety invariant of the plant component. In that case we could then simply compose these two separate models to obtain the final plant component model. On the other hand we may submit that actually mixing the specification of the possible behavior with the required safety invariant, at least in some cases, is *more* natural than to enforce separation of concerns. The reasons for this are twofold:

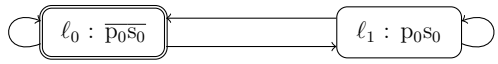
1. It is very difficult to model, by hand, possible behavior that is guaranteed deadlock free, especially when dealing with the parallel composition of multiple components.

2. When enumerating the possible successor states from some source state it is very natural to sometimes map to $\bot$ with the intuitive meaning: "I don't know (care) what the precise behavior is in this particular state, therefore it should be considered unsafe by lack of information (motivation)".

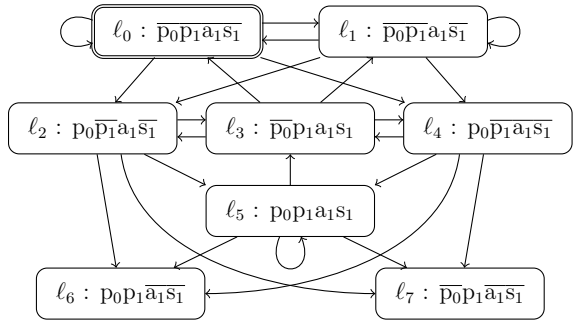
For brevity and because of the aforementioned reasons we will adhere to the convention of incorporating the safety invariant in the plant component models throughout the rest of the thesis.

Using PTSs and primed propositional formulas to define properties of PTSs we are in a position to formalize and explain the PTSs that make up our parcel stamp example.

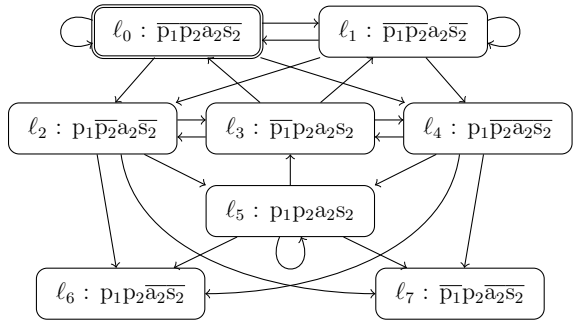
Example 4.1 (PTSs) In Figure 4.3 we show the behavior of the three components in the parcel stamp using PTSs over the atomic propositions as we introduced them in Figure 4.2. In our notation, we use a horizontal bar to denote the negation of a proposition letter, i.e.: $\overline{\mathrm{a}_1}$ should be read as $\neg \mathrm{a}_1$ which, in turn, is interpreted as *stamp number 1 has not activated its stamping arm.* The PTS P_{feed_0} in Figure 4.3(a) models the feeder plant component. We recall, from Figure 4.2(b), that the feeder plant component communicates with the control through a single control input proposition s_0 modeling the optical sensor that measures whether or not a parcel is present on the belt at the feeding station, furthermore the feeder component communicates with the next plant component through a single plant proposition p_0 which models the actual physical presence of a parcel on the belt. The model itself then stipulates the sensor proposition holds iff the parcel proposition holds. This gives two possible locations: ℓ_0 where the parcel is not present and hence the sensor is not triggered, and ℓ_1 where the parcel is present and hence the sensor is triggered. Note that this simple PTS satisfies the following state formula:



(a) PTS P_{feed_0} modeling the feeder.



(b) PTS P_{stamp_1} modeling the first stamp.



(c) PTS P_{stamp_2} modeling the second stamp.

Figure 4.3: PTSs modeling the feeder (a), the first stamp (b) and the second stamp (c) of the parcel plant in Figure 4.2.

- $s_0 \leftrightarrow p_0$, i.e.: optical sensor 0 triggers if–and–only–if there is a parcel on the belt under the feeder. It can be checked that, at each location that makes s_0 true, p_0 is also true and vice versa.

The PTS P_{stamp_1} in Figure 4.3(b) models the first parcel stamp. We recall from Figure 4.2(b) that the stamp component communicates with the two other plant components through propositions p_0 and p_1 modeling the physical presence of a parcel at the feeder and under the stamp, respectively. Furthermore the stamp communicates to the control through propositions a_1 and s_1 modeling the vertical actuator that makes up the stamping arm and the optical sensor attached to the stamping arm, respectively. The model itself then consists of eight locations. Rather than explaining all these locations separately we will explain the model using some primed propositional logic formulas. In particular we note that the following properties are satisfied by the stamp model:

- $p_0 \leftrightarrow p_1'$, i.e.: if there is a parcel in the feeder, in the next state there will be a parcel under the stamp. It can be checked that, for each location that makes p_0 true, all the successor locations make p_1 true, and vice versa.

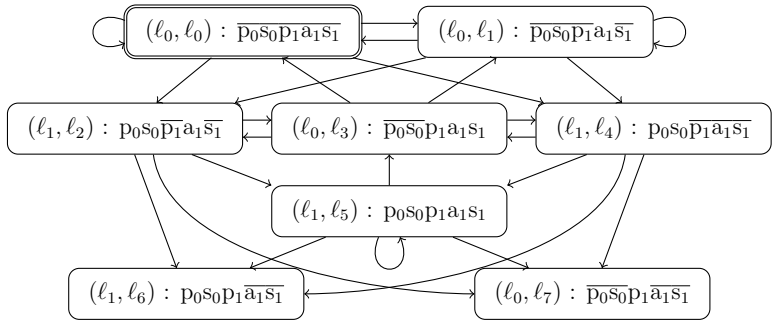
- $p_1 \wedge a_1 \leftrightarrow s_1$, i.e.: if there is a parcel under the stamp, and the stamping arm is activated then the optical sensor is triggered. It can be checked that, at each location that makes both p_1 and a_1 true, s_1 is also true, and vice versa.

- $(p_1 \rightarrow a_1)^\dagger$, i.e.: if there is a parcel present, the stamp must be activated otherwise the next state is undefined. It can be checked that, for each location that makes p_1 true and a_1 false, there are no successor locations.

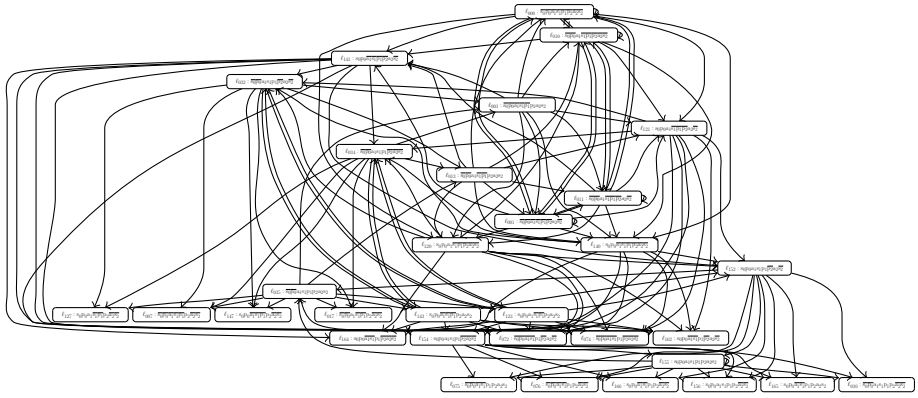
This last transition formula effectively defines a safety property encoded in the stamp model. The safety property here simply says that the stamp *must* activate whenever there is a parcel present on the belt, otherwise the set of safe control outputs becomes empty which is reflected in the fact that locations ℓ_6 and ℓ_7, where this safety condition is violated, have no successor locations. We may say that the set of *control alternatives for the next state* is empty in those locations, hence they are *unsafe*. ◁

4.3.1 Composition of PTSs

A great advantage of working with PTSs is the fact that these structures allow a clean and intuitive form of composition. Formally this is defined as follows:



(a) PTS $P_{\mathrm{feed}_0} \,||\, P_{\mathrm{stamp}_1}$.



(b) PTS $P_{\mathrm{feed}_0} \,||\, P_{\mathrm{stamp}_1} \,||\, P_{\mathrm{stamp}_2}$. This product is not human–readable anymore: we show it here to illustrate the fast–growing complexity of multiple products of PTSs.

Figure 4.4: Composed PTSs.

Definition 4.3 (PTS Composition) For any two PTSs P_1, P_2 the *composition* $P_{12} = P_1 \| P_2$ is defined as follows:

$$L_{12} = \{(\ell_1, \ell_2) \in L_1 \times L_2 \mid \gamma_1(\ell_1) \downharpoonright (X_1 \cap X_2) = \gamma_2(\ell_2) \downharpoonright (X_1 \cap X_2)\}$$

$$X_{12} = X_1 \cup X_2$$

$$\delta_{12} = \{((\ell_1, \ell_2), (\ell_1', \ell_2')) \mid \ell_1 \delta_1 \ell_1' \text{ and } \ell_2 \delta_2 \ell_2'\}$$

$$L_{12}^{\text{init}} = L_{12} \cap (L_1^{\text{init}} \times L_2^{\text{init}})$$

And for all $(\ell_1, \ell_2) \in L_{12}$ and all $x \in X_{12}$ we define:

$$\gamma_{12}(\ell_1, \ell_2)(x) = \begin{cases} \gamma_1(\ell_1)(x) & \text{if } x \in X_1 \\ \gamma_2(\ell_2)(x) & \text{if } x \in X_2 \end{cases}$$

Note that L_{12} and L_{12}^{init} contain all the pairs of (initial) locations in the PTSs that are *consistent*, meaning that they agree on all *shared propositions* $X_1 \cap X_2$.◁

Example 4.2 In Figure 4.4(a) we show the composition $P_{\text{feed}_0} \mid\mid P_{\text{stamp}_1}$ of the feeder component with the stamp component. As can be seen the new PTS uses the union of the set of propositions of both substructures, and the subset of consistent pairs of locations from the cartesian product of both substructures. Note that, in general, the complexity of the PTSs goes up rapidly as we increase the number of components. As an illustration of this, in Figure 4.4(b) we show the composition $P_{\text{feed}_0} \mid\mid P_{\text{stamp}_1} \mid\mid P_{\text{stamp}_2}$, which is the plant model in its entirety consisting of all three plant components. The number of locations and transitions between them is too large, making the diagram too verbose to be human–readable. ◁

In general, complex models can only be understood when they are decomposed into small components. Another aid in dealing with the complexity of such models is an intuitive, symbolic notation. Using formulas to characterize models means that we do not need to work with the models in their extensive form which, as Figure 4.4(b) demonstrates, quickly becomes unwieldy.

4.3.2 Simulation of PTSs

In the remainder we will use PTSs as our unifying semantic object for tying together the notions of a plant model, control game, and controller. By composing

a plant component with a controller we are constraining the behavior that is modeled by the plant component. For this reason it is important to formalize what it means to *constrain* the behavior that is modeled by a given PTS.

Most of the results in this Section can be considered sanity conditions on our definitions. These results are almost completely analogous to similar classical results on Kripke structures, finite automata and labeled transition systems modulo some slight technical ideosyncracies of our particular setting.

In general the result of applying any constraint on a given PTS will be a new PTS that is *simulated* by the original PTS. Meaning the original structure allows *at least* all the behavior that is allowed by the constrained structure.

Definition 4.4 (PTS Simulation) For two given PTSs P_1 and P_2 a relation $Z \subseteq L_1 \times L_2$ between the locations of P_1 and the locations of P_2 is called a *simulation relation* from P_1 to P_2 iff $X_2 \subseteq X_1$ and the following conditions hold for all $(\ell_1, \ell_2) \in Z$:

1. $\gamma_1(\ell_1) \downharpoonright X_2 = \gamma_2(\ell_2)$, i.e.: ℓ_1 and ℓ_2 are consistent with respect to all the shared propositions in X_2.

2. for all successors $\ell_1' \in L_1$ such that $(\ell_1, \ell_1') \in \delta_1$ it holds there exists $\ell_2' \in L_2$ such that $(\ell_2, \ell_2') \in \delta_2$ and $(\ell_1', \ell_2') \in Z$, i.e.: every move in P_1 can be matched by P_2.

If there exists a simulation relation Z such that $(\ell_1, \ell_2) \in Z$ we say ℓ_2 *simulates* ℓ_1, notation: $\ell_1 \leq \ell_2$ If for all $\ell_1 \in L_1^{\text{init}}$ there exists $\ell_2 \in L_2^{\text{init}}$ such that $\ell_1 \leq \ell_2$ we say P_2 simulates P_1, notation: $P_1 \leq P_2$. ◁

PTS composition is now an operation that always decreases with respect to this simulation pre–order. The following Lemma makes this precise and shows that composing two PTSs can be seen as restricting the behavior allowed by either.

Lemma 4.5 (PTS Composition and Simulation) For two PTSs P_1 and P_2 it holds $P_1 \mid\mid P_2 \leq P_1$ and $P_1 \mid\mid P_2 \leq P_2$. ◁

Proof We show $P_1 \mid\mid P_2 \leq P_2$ and $P_1 \mid\mid P_2 \leq P_1$ will follow by symmetry. We construct $Z \subseteq (L_1 \times L_2) \times L_2$ such that $Z = \{((\ell_1, \ell_2), \ell_2) \mid (\ell_1, \ell_2) \in L_{P_1||P_2}\}$. Now it holds that Z is a simulation relation from $P_1 \mid\mid P_2$ to P_2. For Condition 1, note that, by definition of PTS composition $\gamma_{P_1||P_2}(\ell_1, \ell_2) \downharpoonright X_2 = \gamma_{P_2}(\ell_2)$. For Condition 2, assume there exists $(\ell_1', \ell_2') \in L_{P_1||P_2}$ such that $(\ell_1, \ell_2)\delta_{P_1||P_2}(\ell_1', \ell_2')$, by definition of PTS composition it follows that $\ell_2 \delta_{P_2} \ell_2'$,

and by definition of Z it follows $((\ell_1', \ell_2'), \ell_2') \in Z$. For the initiality condition, assume $(\ell_1, \ell_2) \in L^{\text{init}}_{P_1||P_2}$ then, by definition of PTS composition, it follows $\ell_2 \in L^{\text{init}}_{P_2}$. □

In the literature there exist many variations on the basic notion of a simulation relation as we have defined it here. One that we cannot fail to mention is the *bisimulation relation*. The latter is particularly interesting from our perspective because it is a natural notion of equivalence for PTSs, i.e.: two (deterministic) models encode essentially the same behavior iff they are bisimilar.

Definition 4.6 (Bisimulation) Let P_1, P_2 be two PTSs, and let $\ell_1 \in L_1$ and $\ell_2 \in L_2$, let Z be a simulation relation from L_1 to L_2 such that $(\ell_1, \ell_2) \in Z$, if it now holds that Z^{-1} is also a simulation relation (from L_2 to L_1) then we say Z is a *bisimulation relation*. We say two locations ℓ_1 and ℓ_2 are *bisimilar*, notation: $\ell_1 \sim \ell_2$, iff there exists a bisimulation relation linking them. If for all $\ell_1 \in L^{\text{init}}_1$ there exists $\ell_2 \in L^{\text{init}}_2$ such that $\ell_1 \sim \ell_2$ and for all $\ell_2 \in L^{\text{init}}_2$ there exists $\ell_1 \in L^{\text{init}}_1$ such that $\ell_1 \sim \ell_2$ we say P_1 and P_2 are bisimilar, notation: $P_1 \sim P_2$. ◁

Example 4.3 We would like to have that two locations in a PTS are bisimilar iff they are mutually simulating each other. However, in general, this is not the case. As a counterexample consider Figure 4.5. The PTSs P_{sim} and P'_{sim} are mutually simulating each-other, yet they are not bisimilar. First, in order to see that $P_{\text{sim}} \leq P'_{\text{sim}}$, consider:

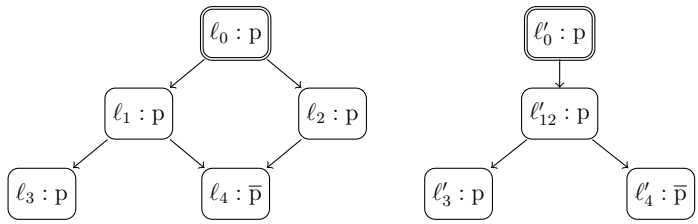
$$Z = \{(\ell_0, \ell_0'), (\ell_1, \ell_{12}'), (\ell_2, \ell_{12}'), (\ell_3, \ell_3'), (\ell_4, \ell_4')\}$$

Now it holds that Z is a simulation relation from P_{sim} to P'_{sim}. Next, in order to see that $P'_{\text{sim}} \leq P_{\text{sim}}$, consider:

$$Z' = \{(\ell_0', \ell_0), (\ell_{12}', \ell_1), (\ell_3', \ell_3), (\ell_4', \ell_4)\}$$

Now it holds that Z' is a simulation relation from P'_{sim} to P_{sim}. However it does not hold that $Z^{-1} = Z'$, in fact there exists no single simulation relation from P_{sim} to P'_{sim} for which the inverse is also a simulation relation. In other words: there exists no bisimulation relation between P_{sim} and P'_{sim}. The problem is the non–deterministic choice in location ℓ_0. In general, in the presence of non–determinism, we cannot say that two locations that are mutually simulating are also, necessarily, bisimilar. ◁

The latter example shows that we need a suitable notion of determinism for PTSs, as is given by the following definition.



(a) P_{sim} (b) P'_{sim}

Figure 4.5: Counterexample showing why bisimulation and mutual simulation are not the same: P_{sim} and P'_{sim} are mutually simulating but not bisimilar.

Definition 4.7 (Deterministic PTSs) A PTS P is *initially deterministic* iff for all $\ell_1, \ell_2 \in L_P^{\mathrm{init}}$ such that $\gamma(\ell_1) = \gamma(\ell_2)$ it holds $\ell_1 = \ell_2$. A PTS P is *successor deterministic* iff for all $\ell \in L$ and for all $\ell'_1, \ell'_2 \in \delta(\ell)$ such that $\gamma(\ell'_1) = \gamma(\ell'_2)$ it holds $\ell'_1 = \ell'_2$. A PTS is *deterministic* iff it is both initially deterministic and successor deterministic. With $\mathcal{P}^{\mathrm{det}}$ we denote the set of all deterministic PTSs. $\lhd$

Example 4.4 (Deterministic PTSs) As an example of a deterministic PTS consider P'_{sim} in Figure 4.5(b). Note that P'_{sim} is a strongest deterministic overapproximation of P_{sim} in the simulation order. $\lhd$

Theorem 4.8 (Simulation and Bisimulation) For two deterministic PTSs P_1 and P_2 it holds $P_1 \sim P_2$ iff $P_1 \leq P_2$ and $P_2 \leq P_1$. $\lhd$

Proof It is easy to see that, for deterministic PTSs P_1 and P_2 it holds $P_1 \leq P_2$ iff $X_1 \supseteq X_2$ and $\mathsf{Traces}(P_1) \subseteq \mathsf{Traces}(P_2)$, and $P_1 \sim P_2$ iff $X_1 = X_2$ and $\mathsf{Traces}(P_1) = \mathsf{Traces}(P_2)$. $\square$

We will only be interested in PTSs upto bisimulation: if two PTSs are bisimilar we consider them equal, for all intents and purposes. For that reason, in the remainder we will work on the quotient structure $\mathcal{P}^{\mathrm{det}}/\!\sim$ of $\sim$–equivalence classes of deterministic PTSs. To this end we lift the pre-order $\leq$ to a partial order $\preceq$ on the quotient structure $\mathcal{P}^{\mathrm{det}}/\!\sim$.

Definition 4.9 (PTS Lattice) We lift the pre-order $\leq$ on $\mathcal{P}^{\text{det}}$ to a partial order $\preceq$ on $\mathcal{P}^{\text{det}}/\sim$ such that for all $E, E' \in \mathcal{P}^{\text{det}}/\sim$ it holds $E \preceq E'$ iff for all $P \in E$ there exists $P' \in E'$ s.t. $P \leq P'$. For a given $P \in \mathcal{P}^{\text{det}}$ we define $[P] \in \mathcal{P}^{\text{det}}/\sim$ as the equivalence class of P, i.e.: $[P] = \{P' \in \mathcal{P}^{\text{det}} \mid P \sim P'\}$, now, in particular, $[P] \preceq [B]$ iff $P \leq B$ $\lhd$

Theorem 4.10 It holds $\mathcal{P}^{\text{det}}/\sim$ together with the partial order $\preceq$ forms a complete lattice. $\lhd$

Proof We show that the partial order of equivalence classes of deterministic PTSs can be embedded into the contravariant product lattice of trace–sets and proposition–sets. It is not hard to show that for two given deterministic PTSs P, P' we have $[P] \preceq [P']$ iff $\mathsf{Traces}(P) \subseteq \mathsf{Traces}(P')$ and $X_P \supseteq X_{P'}$. It is also not hard to see that $\mathsf{Traces}(P \parallel P') = \mathsf{Traces}(P) \cap \mathsf{Traces}(P')$ and we have by definition $X_{P||P'} = X_P \cup X_{P'}$. $\square$

4.3.3 Specifying Control Problems using PTSs

In order to specify control problems using PTSs we need to identify the *signature* of control input and output propositions. The following definition makes this precise:

Definition 4.11 (Control Signature) For a given plant PTS P a *control signature* is a pair $[X^{\text{out}}/X^{\text{in}}]$ consisting a set of *control output propositions* $X^{\text{out}} \subseteq X_P$, and a set of *control input propositions* $X^{\text{in}} \subseteq X_P$. We require these sets to be disjoint: $X^{\text{in}} \cap X^{\text{out}} = \varnothing$. The combination of a plant PTS and a control signature is called a *control problem.* $\lhd$

A *solution* to a given control problem $P_{\text{plant}}[X^{\text{out}}/X^{\text{in}}]$ will consist of a *control PTS* P_{ctrl} intended to constrain the plant PTS. The goal is to obtain a resulting system $P_{\text{ctrl}} \parallel P_{\text{plant}}$ that is *safe* in the sense that no deadlock locations are reachable anymore. The following definition makes this precise:

Definition 4.12 (Control PTS) A PTS P is *input enabled* for a given input proposition $x^{\text{in}} \in X_P$ iff for all $(\ell, \ell') \in \delta_P$ it holds there exists $(\ell, \ell'') \in \delta_P$ such that $\gamma_P(\ell')(x^{\text{in}}) = 1 - \gamma_P(\ell'')(x^{\text{in}})$, i.e.: if one permutation is reachable than so is the other. We say P is input enabled for a given set of input propositions $X^{\text{in}} \subseteq X_P$ iff P is input enabled for all $x^{\text{in}} \in X^{\text{in}}$. We say a PTS P respects the signature $[X^{\text{out}}/X^{\text{in}}]$ iff $X_P = X^{\text{in}} \cup X^{\text{out}}$ and P is input enabled for X^{in}. A control PTS P_{ctrl} is *permissible* for a given control problem consisting of a

plant PTS P_{plant} and a control signature $[X^{\text{out}}/X^{\text{in}}]$ iff it is deterministic, and it respects the control signature $[X^{\text{out}}/X^{\text{in}}]$. With $P_{\text{plant}}[X^{\text{out}}/X^{\text{in}}]$ we denote the set of all permissible control PTSs for P_{plant} in combination with $[X^{\text{out}}/X^{\text{in}}]$. A PTS P is safe iff $\mathsf{Reach}(P)$ does not contain any deadlock locations. With $\mathcal{P}^{\text{safe}}$ we denote the set of all safe PTSs. A control PTS $P_{\text{ctrl}} \in P_{\text{plant}}[X^{\text{out}}/X^{\text{in}}]$ is *safe for* P_{plant} iff $P_{\text{ctrl}} \mid\mid P_{\text{plant}} \in \mathcal{P}^{\text{safe}}$. ◁

In this definition, the input enabledness condition prevents a controller from placing constraints on the input propositions. Note that, with this definition there is always a trivially safe controller, namely the one that has an empty set of initial locations. For this controller the set of reachable locations will always be empty and therefore, trivially, no deadlock locations will be reachable. We deal with this by defining the notion of a maximally and a minimally permissive control in the next section.

4.3.4 Maximally and Minimally Permissive Control PTSs

Definition 4.12 lays out the conditions that any permissible control PTS must fulfill. Rather than synthesizing an *arbitrary* control PTS we are mainly interested in deriving either *minimally permissive* or *maximally permissive* control PTSs.

Deriving a *maximally permissive* control PTS is useful in a setting where we have a system that is not safe and we want to compute a controller for this system that is safe yet constrains it in the least possible way. We say that maximally permissive control embodies the *weakest sufficient control*. The following definition makes this precise.

Definition 4.13 (Maximally Permissive Control PTS) A control PTS P_1 is *strictly more permissive* than another control PTS P_2 iff it holds $P_1 > P_2$. For a given control problem $P_{\text{plant}}[X^{\text{out}}/X^{\text{in}}]$ we define the set of *maximally permissive control PTSs* :

$$P_{\text{plant}}\lceil X^{\text{out}}/X^{\text{in}}\rceil = \lceil\{P_{\text{ctrl}} \in P_{\text{plant}}[X^{\text{out}}/X^{\text{in}}] \mid P_{\text{ctrl}} \mid\mid P_{\text{plant}} \in \mathcal{P}^{\text{safe}}\}\rceil$$

which are all the maximal control PTSs in the $\geq$ ordering that are safe for P_{plant}. ◁

Deriving a *minimally permissive* control PTS is useful in a setting where we have a system that is already safe by construction (but does not respect the control signature yet) and we want to approximate a controller for this system. We say that minimally permissive control embodies the *strongest necessary control*. The following definition makes this precise.

Definition 4.14 (Minimally Permissive Control PTS) For a given safe PTS $P_{\text{safeplant}}$ and control signature $[X^{\text{out}}/X^{\text{in}}]$ we define:

$$P_{\text{safeplant}}\lfloor X^{\text{out}}/X^{\text{in}}\rfloor = \lfloor\{P_{\text{ctrl}} \in P_{\text{safeplant}}[X^{\text{out}}/X^{\text{in}}] \mid P_{\text{ctrl}} \geq P_{\text{safeplant}}\}\rfloor$$

as the set of *minimally permissive control PTSs* which are all the minimal control PTSs in the $\geq$ ordering that are still more permissive than $P_{\text{safeplant}}$. $\lhd$

The following two theorems show that in both cases, *maximally* as well as *minimally* permissive control, we have that the resulting control PTS is well–defined upto bisimulation.

Theorem 4.15 For any given control problem $P_{\text{plant}}[X^{\text{out}}/X^{\text{in}}]$ it holds for all $P_1, P_2 \in P_{\text{plant}}\lfloor X^{\text{out}}/X^{\text{in}}\rfloor$ that $P_1 \sim P_2$. $\lhd$

Proof Let $P_1, P_2 \in P_{\text{plant}}\lfloor X^{\text{out}}/X^{\text{in}}\rfloor$ as in the theorem, we define the meet P_{12} such that P_{12} accepts the intersection of the languages of P_1 and P_2: $P_{12} = P_1 \mid\mid P_2$ it is not hard to see that P_{12} is also permissible for $P_{\text{plant}}[X^{\text{out}}/X^{\text{in}}]$ and P_{12} is also above P_{plant} in the $\geq$ ordering, hence $P_{12} \in P_{\text{plant}}[X^{\text{out}}/X^{\text{in}}]$. By Lemma 4.5 we have $P_1 \geq P_{12}$ and $P_2 \geq P_{12}$, and it cannot be the case $P_1 > P_{12}$ or $P_2 > P_{12}$ because P_1 and P_2 are minimal. It must follow that $P_1 \sim P_{12} \sim P_2$. $\square$

Theorem 4.16 For any given control problem $P_{\text{plant}}[X^{\text{out}}/X^{\text{in}}]$ it holds for all $P_1, P_2 \in P_{\text{plant}}\lceil X^{\text{out}}/X^{\text{in}}\rceil$ that $P_1 \sim P_2$. $\lhd$

Proof Let $P_1, P_2 \in P_{\text{plant}}\lceil X^{\text{out}}/X^{\text{in}}\rceil$ as in the theorem, we define the join P_{12} such that P_{12} accepts the union of the languages of P_1 and P_2:

$$\begin{aligned} L_{12} =& \{(\ell_1, \ell_2) \mid \ell_1 \in L_1, \ell_2 \in L_2, \gamma_1(\ell_1) = \gamma_2(\ell_2)\} \\ & \cup \{(\ell_1, \ell_\perp) \mid \ell_1 \in L_1\} \\ & \cup \{(\ell_\perp, \ell_2) \mid \ell_2 \in L_2\} \end{aligned}$$

$$X_{12} = X^{\text{out}} \cup X^{\text{in}} \qquad \gamma_{12}(\ell_1, \ell_2) = \begin{cases} \gamma(\ell_1) & \text{if } \ell_1 \neq \ell_\perp \\ \gamma(\ell_2) & \text{otherwise} \end{cases}$$

The transition relation is constructed in such a way that it will accept any sequence that is consistent with either P_1 or P_2 or with both. This means that we use the fresh $\ell_\perp$ helper locations to model the fact that either one of the

two PTSs is no longer consistent. Yet, the constructed PTS keeps on accepting observations as long as at least one of the two constituent PTSs is consistent.

$$\begin{aligned}\delta_{12} =&\{((\ell_1,\ell_2),(\ell_1',\ell_2')) \mid (\ell_1,\ell_1') \in \delta_1, (\ell_2,\ell_2') \in \delta_2\}\\ &\cup \{((\ell_1,\ell_2),(\ell_1',\ell_\perp)) \mid (\ell_1,\ell_1') \in \delta_1, \nexists(\ell_2,\ell_2') \in \delta_2.\gamma(\ell_2') = \gamma(\ell_1')\}\\ &\cup \{((\ell_1,\ell_2),(\ell_\perp,\ell_2')) \mid \nexists(\ell_1,\ell_1') \in \delta_1.\gamma(\ell_1') = \gamma(\ell_2'), (\ell_2,\ell_2') \in \delta_2\}\\ &\cup \{((\ell_1,\ell_\perp),(\ell_1',\ell_\perp)) \mid (\ell_1,\ell_1') \in \delta_1\}\\ &\cup \{((\ell_\perp,\ell_2),(\ell_\perp,\ell_2')) \mid (\ell_2,\ell_2') \in \delta_2\}\end{aligned}$$

$$L_{12}^{\text{init}} = \{(\ell_1,\ell_2) \mid \ell_1 \in L_1^{\text{init}}, \ell_2 \in L_2^{\text{init}} \text{ and } \gamma_1(\ell_1) = \gamma_2(\ell_2) = s\}$$

It is now not hard to verify that P_{12} is also permissible and safe, hence $P_{12} \in P_{\text{plant}}\lceil X^{\text{out}}/X^{\text{in}}\rceil$. It is also easy to see that $P_1 \leq P_{12}$ and $P_2 \leq P_{12}$. It follows that $P_1 \sim P_{12} \sim P_2$. □

In general maximally permissive control is more intuitive than minimally permissive control. Typically what we will do is to define a minimally permissive approximation *over* a system that is already under maximally permissive control (such a system is safe by construction). Hence, it is natural to first look at maximally permissive control and then to move on to treat minimally permissive control. For this reason we defer an example of a minimally permissive control to Section 4.5. Below we give a preliminary example of a maximally permissive control.

Example 4.5 (Maximally Permissive Control PTS) As an example of a maximally permissive control PTS consider Figure 4.6. This PTS is a maximally permissive, safe controller for the control problem $P_{\text{stamp}_1}[\text{a}_1/\text{s}_1]$. For reference, the underlying plant PTS is shown in Figure 4.3(b). It can be verified that this control PTS is indeed deterministic and input enabled for s_1. In the remainder of this section we will demonstrate that this controller is indeed maximally permissive. In Figure 4.7 we show the product of this controller with the plant model. ◁

4.4 Computing Maximally Permissive Control

In the previous section we have laid out the foundations of a symbolic framework allowing us to formally deal with composition of plant components and

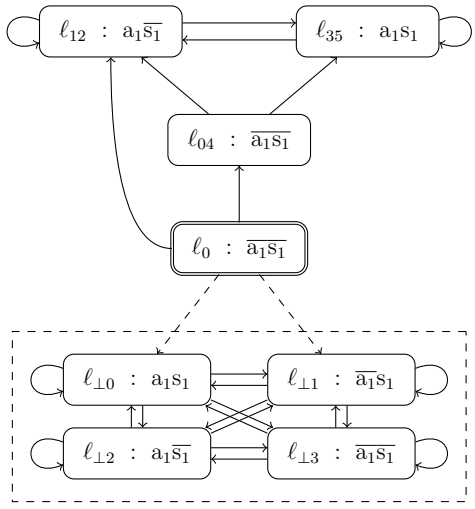


Figure 4.6: $P_{\mathrm{ctrlstamp}_1}$ maximally permissive control PTS for $P_{\mathrm{stamp}_1}[\mathrm{a}_1/\mathrm{s}_1]$.

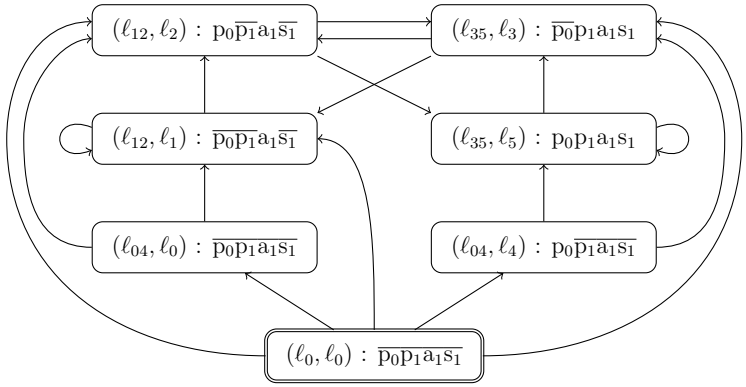


Figure 4.7: The resulting system $P_{\mathrm{ctrlstamp}_1} \mid\mid P_{\mathrm{stamp}_1}$.

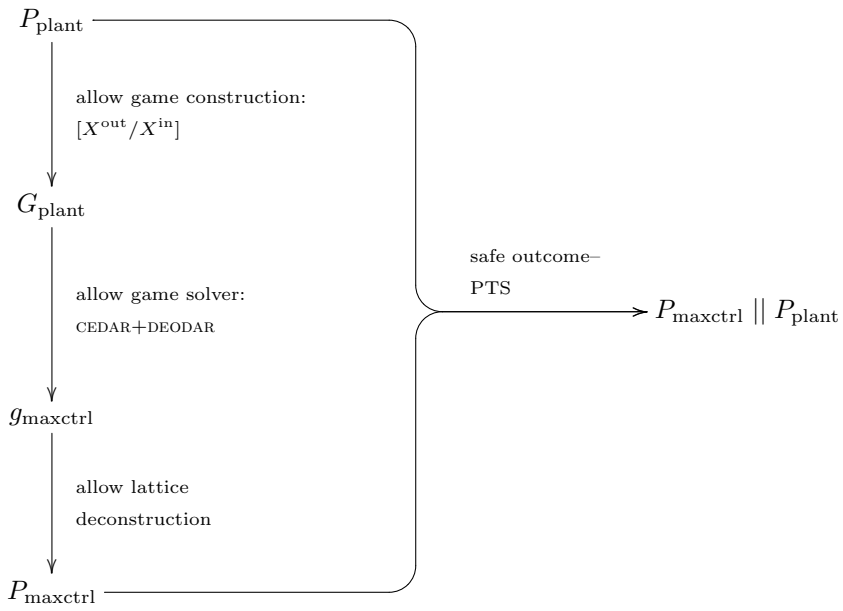


Figure 4.8: Computing maximally permissive controllers.

controllers. In Definition 4.13 we have given a declarative definition of one of the objects of interest that is the maximally permissive control PTS for a given plant PTS (and signature). In this section we show how this object of interest can be effectively synthesized.

In particular we will reduce the derivation of a maximally permissive safe control PTS for a given plant PTS with signature to the solving of a safety game of imperfect information. The reduction of PTS control problems to safety games of imperfect information gives us an effective way to synthesize control PTSs using the symbolic game solving algorithms CEDAR+DEODAR. The outline of the reduction is given in Figure 4.8.

4.4.1 Constructing PTS Games

In this section we formalize the connection between PTSs (Definition 4.1) and safety games of imperfect information (Definition 2.1). In particular, we will show how a PTS together with a signature defines a safety game where the valuations over the input and output propositions form the control input and outputs respectively.

To keep the exposition more focussed, in the remainder of this section, we assume to be working in the context of a given, solvable background problem consisting of a PTS P_{plant} and signature $[X^{\text{out}}/X^{\text{in}}]$. All the definitions in this section should be understood in the context of this single background problem. For an overview cf. Figure 4.8.

Definition 4.17 (PTS Game Board Construction) Let $P_{\text{plant}}[X^{\text{out}}/X^{\text{in}}]$ be a given background control problem. We define the corresponding plant game G_{plant} through the following *game board construction*:

- $L_{G_{\text{plant}}} = L_{P_{\text{plant}}}$, i.e.: the set of locations of the game corresponds to the set of locations of the plant PTS.
- $C^{\text{out}}_{G_{\text{plant}}} = \mathcal{S}[X^{\text{out}}]$, i.e.: the set of control outputs is the set of all valuations over the control output propositions.
- $C^{\text{in}}_{G_{\text{plant}}} = \mathcal{S}[X^{\text{in}}]$, i.e.: the set of control inputs is the set of all valuations over the control input propositions.
- $\alpha_{G_{\text{plant}}}(\ell) = \gamma_{P_{\text{plant}}}(\ell) \downharpoonright X^{\text{out}}$, i.e.: the control output associated with a location is the valuation restricted to the output propositions.
- $\beta_{G_{\text{plant}}}(\ell) = \gamma_{P_{\text{plant}}}(\ell) \downharpoonright X^{\text{in}}$, i.e.: the control input associated with a location is the valuation restricted to the input propositions.
- $\delta_{G_{\text{plant}}} = \delta_{P_{\text{plant}}}$, i.e.: the transitions in the game correspond to the transitions in the PTS.
- $\imath^{\text{init}}_{G_{\text{plant}}} = \varnothing$

In the context of this game board we define $X^{\text{hidden}} = X_{P_{\text{plant}}} \setminus (X^{\text{in}} \cup X^{\text{out}})$ as the set of *hidden propositions* consisting of all the propositions that were projected away from the original plant PTS. $\lhd$

The definition of game board is almost one–to–one, with the exception of the initial information set. We set the initial information set to $\varnothing$ because we want to compute the weakest allow lattice *for any information set*. In this way we can accomodate the fact that a PTS essentially has a *family* of initial information sets rather than one well–defined initial information set. In particular each *initial observation* defines an initial information set. The following definition makes this precise:

Definition 4.18 (Family of Initial Information Sets) For a given initial observation $c^{\text{out}}/c^{\text{in}}$ we define:

$$L^{\text{init}}_{P_{\text{plant}}}(c^{\text{out}}/c^{\text{in}}) = \{\ell \in L^{\text{init}}_{P_{\text{plant}}} \mid \gamma_{P_{\text{plant}}}(\ell) \lfloor X^{\text{in}} = c^{\text{in}}, \gamma_{P_{\text{plant}}}(\ell) \lfloor X^{\text{out}} = c^{\text{out}}\}$$

For the game G_{plant} we define $i^{\text{init}}_{G_{\text{plant}}}(c^{\text{out}}/c^{\text{in}}) = L^{\text{init}}_{P_{\text{plant}}}(c^{\text{out}}/c^{\text{in}})$, i.e.: the initial information consisting of all the initial locations in the underlying plant model that are consistent with the initial observation. $\lhd$

An advantage of using PTSs and signatures for representing games is that we may quickly identify several games that are played on the same PTS but use a different signature. In Section 4.7 we use this flexibility in the game signature for ensuring compositionality of control constraints.

Example 4.6 As a first example of a game board based on a PTS we take the PTS P_{stamp_1} as defined in Example 4.1. As the control signature we fix $[\text{a}_1/\text{s}_1]$, i.e.: we take the activation of the stamping arm a_1 as the only output proposition and the triggering of the optical sensor s_1 as the only input proposition. In Figure 4.9 we show the resulting game board.

Figure 4.10 shows the knowledge based subset construction for this game for the initial information set $i^{\text{init}}_{\text{stamp}_1}(\overline{\text{a}_1}/\overline{\text{s}_1})$. From this graph it can be seen that imperfect information plays an important role in this game. In particular when playing $\overline{\text{a}_1}$ the information of the safety player will eventually always deteriorate up to the point that the set of safe control alternatives becomes empty.

As a second example of a game board based on a PTS we take the composition $P_{\text{feed}_0} \mid\mid P_{\text{stamp}_1}$ as defined in Example 4.1. As the game signature we fix $[\text{a}_1/\text{s}_0\text{s}_1]$, i.e.: we added the input of the optical sensor on the feeder s_0 to the game signature of the previous example. In Figure 4.11 we show the resulting game board.

We do not show the knowledge based subset construction for this second game. This is not necessary since it is isomorphic to the original game graph.

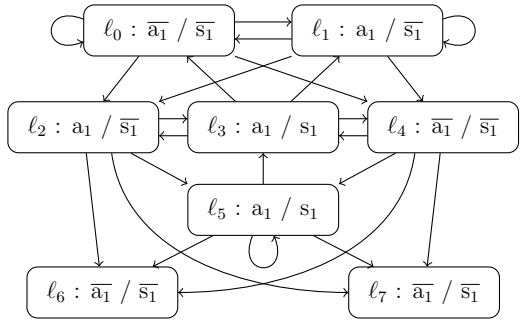


Figure 4.9: Game board for $P_{\mathrm{stamp}_1}[a_1/s_1]$.

The extended control signature prevents the deterioration of the knowledge that we saw in the previous example led to the set of safe control alternatives becoming empty. This then enables a more permissive control strategy. ◁

4.4.2 Solving PTS Games Symbolically

In the previous section we introduced a construction that allows us to reduce a PTS control problem to a safety game. Now that we have defined this construction we are in a position to employ the CEDAR + DEODAR algorithm that we developed in the first part of the thesis to solve these control problems. The following definition makes this precise:

Definition 4.19 (Control Lattice) We define the allow lattice g_{maxctrl} as the result computed by CEDAR+DEODAR on the plant game G_{plant}. ◁

The combination of CEDAR + DEODAR constitutes a symbolic algorithm in the sense that it manipulates (representations of) *sets of states* rather than *concrete states*. This makes it a suitable algorithm for solving safety games defined over PTSs.

In particular the top–down refinement approach followed by CEDAR + DEODAR prevents it from having to generate the knowledge based subset construction (as in Figure 4.10). This subset construction can easily become exponen-

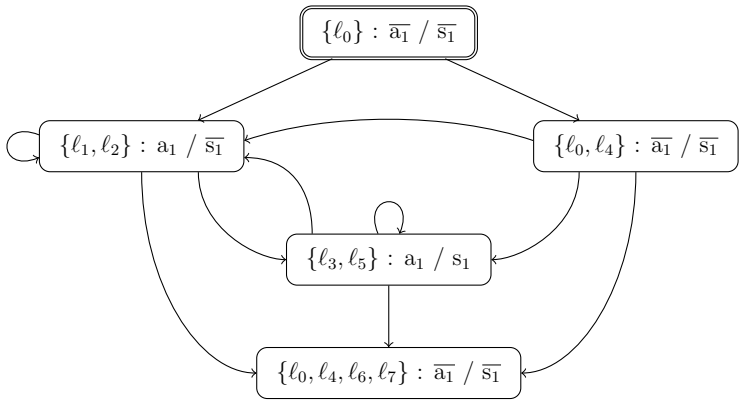


Figure 4.10: Knowledge based subset construction of Figure 4.9 for initial information $i^{\text{init}}_{\text{stamp}_1}(\overline{a_1}/\overline{s_1})$

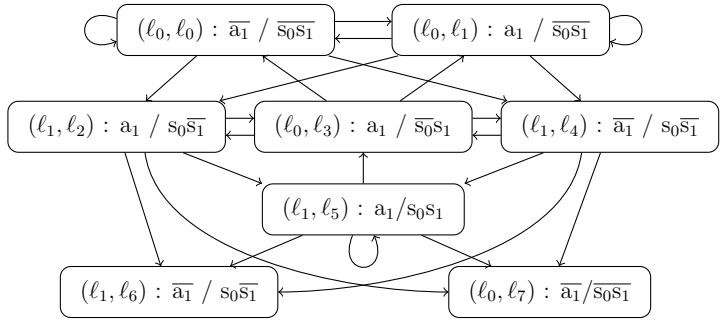
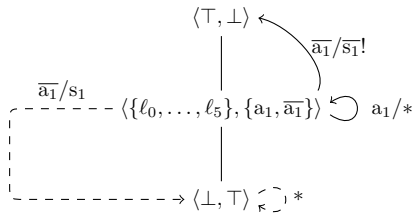
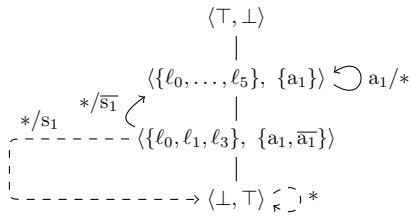


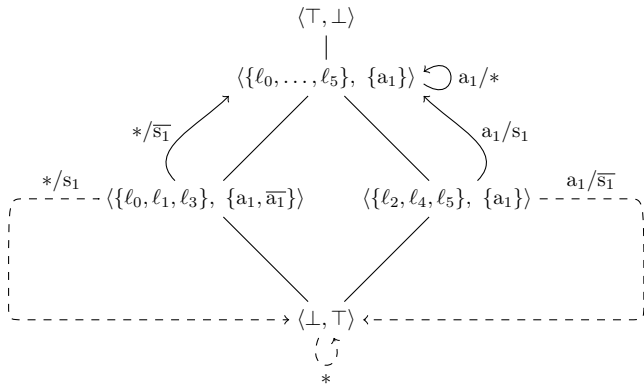
Figure 4.11: Game board for $(P_{\text{feed}_0} \mid\mid P_{\text{stamp}_1})[a_1/s_0 s_1]$.



(a) Initial allow lattice.



(b) Result after CEDAR.



(c) Result after CEDAR+DEODAR.

Figure 4.12: Example run of CEDAR+DEODAR on control game $P_{\text{stamp}_1}[a_1/s_1]$

tially large, especially when generated over a PTS that is, itself, the product of several component PTSs.

In addition, the top–down refinement approach followed by CEDAR + DEODAR prevents the algorithm from having to explicitly generate the underlying game graph (as in Figure 4.9). Instead the algorithm works directly on the *symbolic representation* of the underlying game graph. How this works out in practice is illustrated by the following examples.

Example 4.7 (Solving PTS Games) In Figure 4.12 we show how CEDAR + DEODAR solves the safety game for $P_{\text{stamp}_1}[a_1/s_1]$ of which the game board is shown in Figure 4.9. ◁

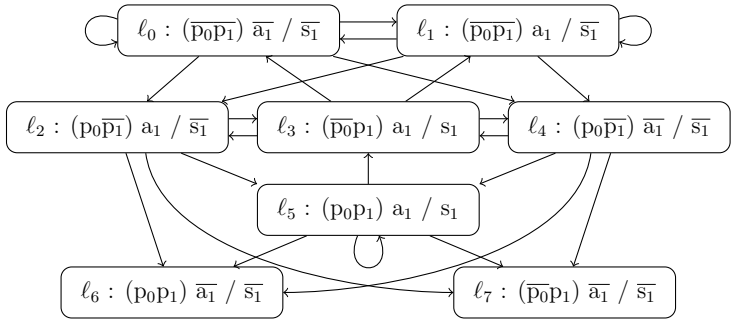


Figure 4.13: Hidden propositions for $P_{\text{stamp}_1}[a_1/s_1]$.

The previous example clearly shows that the top–down refinement approach avoids forward exploration of the game board or its knowledge based subset construction. The example also shows how the top–down refinement approach finds the relevant sets of locations in the form of the information sets present in the final diagram. These relevant information sets can be represented and manipulated symbolically preventing the algorithm from having to explicitly enumerate over the locations that make up these information sets.

It is well known that Kripke structures can be represented and manipulated symbolically, using BDD's or some other suitable symbolic representation. This has been well–studied in the field of symbolic model checking for example. Although related, our approach is sufficiently different to merit some additional explanation about how sets of locations are represented symbolically during a run of CEDAR+DEODAR. To avoid excessive technical detail we do so using an example.

Example 4.8 (Using Hidden Propositions) The game board in Figure 4.9 has eight locations but only three different propositional labels. This means that we cannot characterize arbitrary sets of locations though their propositional theory because there are several locations that we cannot distinguish based on the value of output and input propositions alone.

In order to characterize arbitrary subsets of game board locations we take into account also the propositions that fall outside the control signature, we refer to these as the *hidden propositions*. In Figure 4.13 we show once again the

game board for $P_{\text{stamp}_1}[\text{a}_1/\text{s}_1]$ this time labeled with the hidden propositions in addition to the input/output propositions.

Taking into account hidden propositions we obtain the property that each location has a unique propositional labeling. This means that we can now characterize arbitrary subsets of locations through their propositional theory. In particular we can characterize the three relevant information sets from Example 4.7 as follows:

1. The information set $\{\ell_0, \ell_1, \ell_2, \ell_3, \ell_4, \ell_5\}$ can be characterized by the formula: $\text{p}_1 \rightarrow \text{a}_1$. It can be checked that each of the locations that makes p_1 true also makes a_1 true. The other locations ℓ_6 and ℓ_7 both make p_1 true and a_1 false.

2. The information set $\{\ell_0, \ell_1, \ell_3\}$ can be characterized by: $\overline{\text{p}_0} \wedge (\text{p}_1 \rightarrow \text{a}_1)$. It can be checked that each of these locations makes p_0 false.

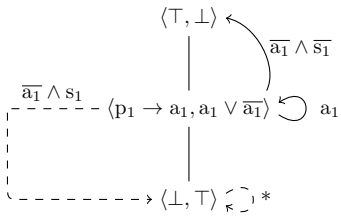
3. The information set $\{\ell_2, \ell_4, \ell_5\}$ can be characterized by: $\text{p}_0 \wedge (\text{p}_1 \rightarrow \text{a}_1)$. It can be checked that each of these locations makes p_0 true.

The latter constitutes just one of many possible ways of representing information sets symbolically. Note that this choice of representation is completely orthogonal to the resulting strategy that is being computed. In particular, it is not the case that the hidden propositions will somehow become visible or influenceable for the player. We only rely on hidden propositions for characterizing information sets. We do not rely on hidden propositions for characterizing observation sets or allow sets.

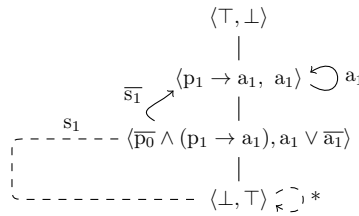
How this works out for CEDAR + DEODAR is shown in Figure 4.14. This figure shows essentially the same run as was shown in Figure 4.12 except for the fact that now all the information sets are substituted with the formulas that characterize them. What this figure shows most clearly is how, during a run of CEDAR + DEODAR, all the relevant structural elements (information sets, observation sets, allow sets) are represented and manipulated symbolically. ◁

4.4.3 Allow Lattice Deconstruction

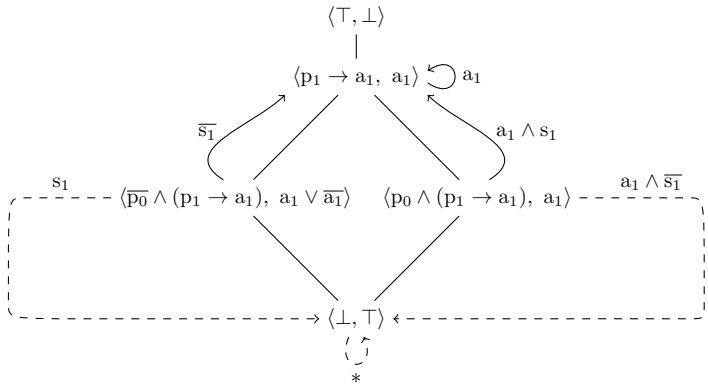
In this section we discuss the last step of the procedure shown in Figure 4.8. This last step consists in a reification of the computed allow lattice back into the form of a PTS. We call this *allow lattice deconstruction* to emphasize the fact that we go in the opposite direction of the *game board construction* as described in Section 4.4.1.



(a) Initial allow lattice.



(b) Result after CEDAR



(c) Result after CEDAR+DEODAR

Figure 4.14: Example run of CEDAR+DEODAR with symbolic sets.

The deconstruction step requires us to encode the information sets in the allow lattice into a PTS. The following definition makes this precise.

Definition 4.20 (Allow Lattice Deconstruction) For a given initial observation $c^{\text{out}}/c^{\text{in}}$ we define the *best approximation initial information set* $i^{\text{init}}_{g_{\text{maxctrl}}}(o)$ such that $i^{\text{init}}_{g_{\text{maxctrl}}}(o) = \bigcap\{i \in g \mid i \supseteq i^{\text{init}}_{G_{\text{plant}}}(o)\}$, i.e.: $i^{\text{init}}_{g_{\text{maxctrl}}}(o)$ consists of the best approximation in g_{maxctrl} of the initial information set for the initial observation o. We define $a^{\text{init}}_{g_{\text{maxctrl}}}(o) = g(i^{\text{init}}_{g_{\text{maxctrl}}}(o))$ as the allow set for the given initial observation. We define the control PTS P_{maxctrl} as the reachable restriction $P_{\text{maxctrl}} = \mathsf{Reach}(P)$ of a PTS $P = (L, X, \gamma, \delta, L^{\text{init}})$ where:

- $L = \{(i, c^{\text{out}}/c^{\text{in}}) \mid i \in g_{\text{maxctrl}} \text{ and } c^{\text{out}} \in C^{\text{out}}_{G_{\text{plant}}} \text{ and } c^{\text{in}} \in C^{\text{in}}_{G_{\text{plant}}}\}$ i.e.: the set of locations for the control PTS is the set of pairs consisting of a current information set from the allow lattice and a *last seen observation.*
- $X = X^{\text{out}} \cup X^{\text{in}}$, i.e.: the relevant variables are the control output and input variables.
- $\gamma(i, c^{\text{out}}/c^{\text{in}}) = c^{\text{out}} \cup c^{\text{in}}$, i.e.: the labeling for the control PTS reflects the last seen observation.
- $\delta = \{((i, o), (i', o')) \mid (i \xrightarrow{o'} i') \in R_{\overline{g}_{\text{maxctrl}}}\}$, i.e.: the transition relation for the control PTS reflects the knowledge update relation of the allow lattice (cf. Definition 2.19).
- $L^{\text{init}} = \{(i, o) \mid i = i^{\text{init}}_{g_{\text{maxctrl}}}(o) \text{ and } a^{\text{init}}_{g_{\text{maxctrl}}}(o) \neq \bot\}$, i.e.: the set of initial locations consists of all the locations that start from the best approximation initial information set in the allow lattice for a given initial observation that is not loosing.

Note that initial locations are potentially accessible also from other (internal) locations in the PTS, hence initial locations can be (re)visited, hence we need to keep last seen observations also for initial locations. ◁

Example 4.9 In Figure 4.15 we show the deconstruction of the allow lattice for the control problem $P_{\text{stamp}_1}[\text{a}_1/\text{s}_1]$. As can be seen this control PTS is equivalent to a maximally permissive control PTS as given in Figure 4.6. The allow lattice on which this control PTS was based is shown in Figure 4.12(b). The underlying game is shown in Figure 4.9. ◁

Lemma 4.21 (Permissible Control PTS) It holds that P_{maxctrl} is a permissible control PTS for P_{plant}. ◁

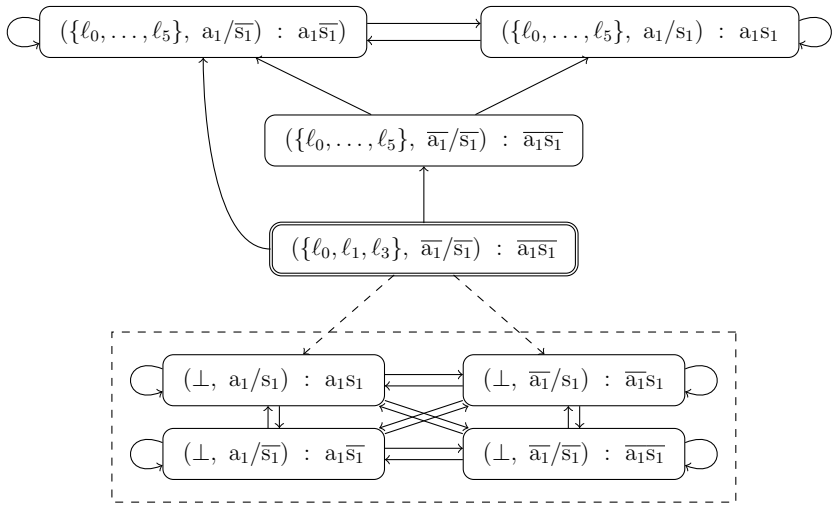


Figure 4.15: Deconstructed allow lattice for control problem $P_{\mathrm{stamp}_1}[\mathrm{a}_1/\mathrm{s}_1]$.

Proof The lemma requires that:

1. P_{maxctrl} is deterministic.
2. P_{maxctrl} is input enabled for X^{in}.

Condition 1. follows directly from determinism of the knowledge update relation $R^{-}_{g_{\mathrm{maxctrl}}}$ (cf. Definition 2.17). Condition 2. follows from the fact that from any source information set, any allowed control output $c^{\mathrm{out}} \in \mathcal{S}[X^{\mathrm{out}}]$ and any control input $c^{\mathrm{in}} \in \mathcal{S}[X^{\mathrm{in}}]$ the knowledge update relation always maps to a well defined target information set. □

Lemma 4.22 (Safe Control PTS) It holds that P_{maxctrl} is a safe control PTS for P_{plant}. ◁

Proof For contradiction, assume P_{maxctrl} is not safe for P_{plant} by definition this means there must exist a path through the product system $P_{\mathrm{maxctrl}} \parallel P_{\mathrm{plant}}$ that ends in a deadlock location. Let $((i_0, o_0), \ell_0)((i_1, o_1), \ell_1) \ldots ((i_n, o_n), \ell_n)$ be such a path where $((i_n, o_n), \ell_n)$ is a deadlock location. From this path we can

define a trace $\ell_0 i_0 \ell_1 i_1 \ldots \ell_n i_n$ consisting of an alternation of concrete plant/game locations and information sets. It is not hard to see that this constitutes a loosing trace in the outcome (cf. Definition 2.3) of the strategy defined by the allow lattice g_{maxctrl} played on the game G_{plant}. This would entail that the strategy defined by g_{maxctrl} is not a safe strategy for the control game G_{plant}. The latter contradicts our assumption that g_{maxctrl} is a safe allow lattice and a solution to the control game G_{plant}. □

Theorem 4.23 (Maximally Permissive Control PTS) It holds that P_{maxctrl} is a maximally permissive safe control PTS for P_{plant}. ◁

Proof Sketch By Lemmas 4.21 and 4.22 we have that P_{maxctrl} is permissible and safe for P_{plant}. It remains to show that P_{maxctrl} is indeed maximally permissive. We first note that P_{maxctrl} is based on a maximally permissive (or: weakest) history based strategy represented by the control lattice g_{maxctrl}. In order to obtain, from this fact, that P_{maxctrl} is also a maximally permissive control PTS it suffices to show that the lattice of (bisimulation equivalence classes of) permissible control PTSs can be embedded in the lattice of history based strategies. This would allow us to conclude that, since g_{maxctrl} is maximally permissive then P_{maxctrl} must be maximally permissive.

So we proceed by defining this embedding. Let P be an arbitrary permissible control PTS for P_{plant}. We first define a function $\mathsf{Succ} : L^{\text{init}}_{P_{\text{maxctrl}}} \to \mathcal{O}^* \to \mathbb{P}(L_P)$ mapping a sequence of observations to the set of successor control locations that are possible to reach in one step after seeing the observation sequence. This is defined as follows:

$$\begin{aligned} \mathsf{Succ}(\ell)(\epsilon) &= \delta_P(\ell) \\ \mathsf{Succ}(\ell)(\pi^\frown c^{\text{out}}/c^{\text{in}}) &= \{\ell' \in \delta_P(\mathsf{Succ}(\ell)(\pi)) \mid \\ &\qquad \gamma_P(\ell') \lfloor X^{\text{in}} = c^{\text{in}}, \\ &\qquad \gamma_P(\ell') \lfloor X^{\text{out}} = c^{\text{out}}\} \end{aligned}$$

Based on this function we define a history based strategy $h_P : I \to \mathcal{O}^* \to P$ such that for a given initial location $\ell \in L^{\text{init}}_{P_{\text{maxctrl}}}$, labeled with control output $c^{\text{out}} = \gamma(\ell) \lfloor X^{\text{out}}$, and control input $c^{\text{in}} = \gamma(\ell) \lfloor X^{\text{in}}$, and a given initial information set $i \subseteq L^{\text{init}}_{P_{\text{plant}}}(c^{\text{out}}/c^{\text{in}})$ it holds:

$$h_P(i)(\pi) = \{c^{\text{out}} \mid \exists \ell' \in \mathsf{Succ}(\ell)(\pi) \text{ and } \gamma_P(\ell') \lfloor X^{\text{out}} = c^{\text{out}}\}$$

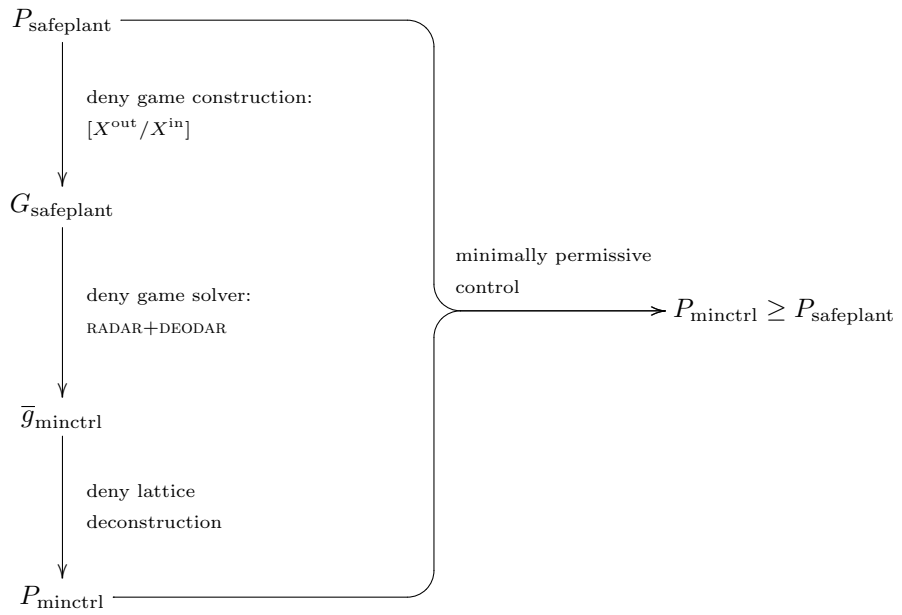


Figure 4.16: Computing minimally permissive controllers.

for all other information sets i we define:

$$h_P(i)(\pi) = \varnothing$$

A straightforward albeit rather technical argument shows that for any two control PTSs P_1, P_2 it holds $P_1 \sim P_2$ iff $h_{P_1} = h_{P_2}$. This allows us to define a one to one homomorphism that is an embedding from the lattice of bisimulation equivalence classes of control PTSs to the lattice of history based strategies. □

4.5 Computing Minimally Permissive Control

In the previous section we saw how to compute a *maximally permissive controller* by reducing a control problem to a safety game of imperfect information. In this section we discuss the second type of control derivation which is the *minimally*

permissive controller as introduced in Definition 4.14. In particular we give an example of a minimally permissive controller and we sketch an algorithm for computing minimally permissive controllers.

This section will be less formal than the previous section on maximally permissive controllers. This is so partly to avoid excessive technical detail, and partly because the derivation of a minimally permissive control is much *easier* than the derivation of a maximally permissive control. To compute a minimally permissive controller we use the same types of symbolic representation and symbolic techniques that we have used throughout the thesis so far, yet they are employed in a more direct fashion.

The outline of the derivation is given Figure 4.16. All the steps involved are completely analogous to the steps we took for computing maximally permissive control (cf. Figure 4.8). The only exception is the new RADAR (*reverse allowset driven antichain refinement*) algorithm. In the remainder of this section we will focus mainly on explaining the new RADAR algorithm, although we will also illustrate the other steps using the example.

In the next section we use maximally and minimally permissive control derivation in an alternating fashion to facilitate compositionality. At this point however, it is important to realize that the two types of derivation are used for opposite purposes. In particular maximally permissive control is useful for deriving the *weakest sufficient control* whereas minimally permissive control is useful for deriving the *strongest necessary control*. As such the latter presupposes that we already have some representation of the maximally permissive, safe, behavior of the system. The following example illustrates this.

Example 4.10 (Minimally Permissive Control) In Figure 4.17 we show the safe portion of the plant model $P_{\text{safestamp}_1}$. This system is safe by construction, i.e.: we simply assume that, somehow, all the unsafe locations are suppressed. In Figure 4.18 we show the game board $G_{\text{safestamp}_1}$ over the safe plant model $P_{\text{safestamp}_1}$ and control signature $[\text{a}_1/\text{s}_1]$. The minimally permissive control for this game consists of the control PTS that at each point in the game allows the smallest set of control outputs without ever overconstraining the underlying system.

In Figure 4.19 we show the minimally permissive control PTS for $P_{\text{safestamp}_1}$. It can be checked that $P_{\text{minctrlstamp}_1}$ is permissible for $P_{\text{safestamp}_1}$. In addition, it can be checked that $P_{\text{minctrlstamp}_1}$ does not restrict $P_{\text{safestamp}_1}$. The only restrictions are present in locations $\ell_{\perp 0}$ and $\ell_{\perp 1}$ which are, as the labels suggest, inconsistent locations that are not reachable in the composition $P_{\text{safestamp}_1} \mid\mid P_{\text{minctrlstamp}_1}$. In the remainder of this section we will show how

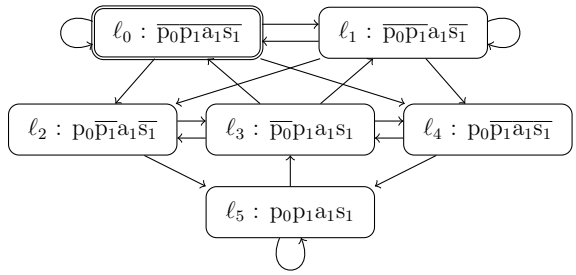


Figure 4.17: $P_{\text{safestamp}_1}$ safe portion of the plant model in Figure 4.3(b)

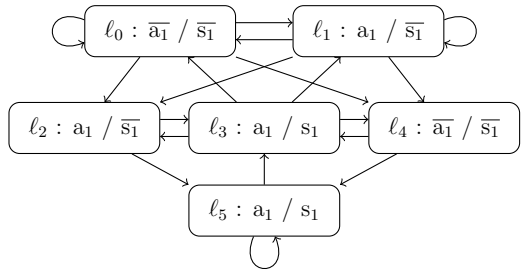


Figure 4.18: $G_{\text{safestamp}_1}$ game board based on Figure 4.17

$P_{\text{minctrlstamp}_1}$ can be derived systematically, and, in doing so, we will demonstrate that $P_{\text{minctrlstamp}_1}$ is indeed minimally permissive. ◁

To see how we can compute a minimally permissive controller efficiently we introduce the notion of a *deny strategy*. A deny strategy is similar to an allow strategy in the sense that it maps information sets to sets of control outputs, the only difference is that we now interpret the set of control outputs as a *deny set*, i.e.: a set of control outputs that are *not allowed*. Crucially, it is the case that the *minimally permissive controller* can be constructed based on the *weakest deny strategy*. Moreover the weakest deny strategy will be an *antitone* function: with *stronger* knowledge we can play a *weaker* deny set. Because of this nice property we can represent a deny strategy using a contravariant lattice of domain/co–domain pairs, just as we have been doing for allow strategies.

In order to compute this deny lattice efficiently we develop a symbolic algo-

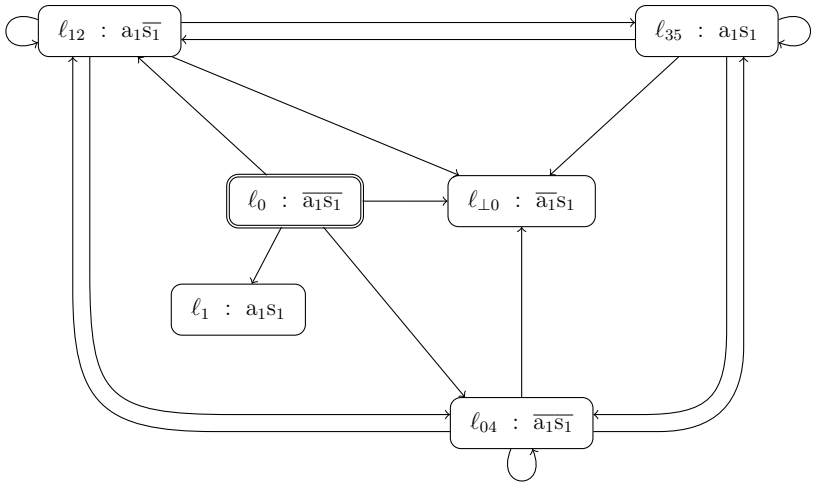


Figure 4.19: Minimally permissive control $P_{\text{minctrlstamp}_1}$ for $P_{\text{safestamp}_1}$

Table 4.1: Computation of the deniable predecessor sets for Figure 4.18.

$\overline{a}$	$\delta^{-1}(\overline{a})$	$L \setminus \delta^{-1}(\overline{a})$
$\varnothing$	$\varnothing$	$\{\ell_0, \ell_1, \ell_2, \ell_3, \ell_4, \ell_5\}$
$\{\overline{\mathrm{a}_1}\}$	$\{\ell_0, \ell_1, \ell_3\}$	$\{\ell_2, \ell_4, \ell_5\}$
$\{\mathrm{a}_1\}$	$\{\ell_0, \ell_1, \ell_2, \ell_3, \ell_4, \ell_5\}$	$\varnothing$
$\{\overline{\mathrm{a}_1}, \mathrm{a}_1\}$	$\{\ell_0, \ell_1, \ell_2, \ell_3, \ell_4, \ell_5\}$	$\varnothing$

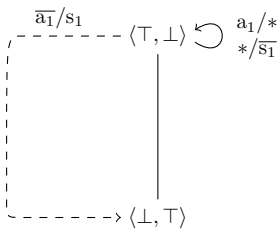
rithm called RADAR (*reverse allowset driven antichain refinement*). In its most basic form the algorithm consists in computing the complement of the weakest predecessors for each possible deny set and entering the resulting info/deny pair into the contravariant control lattice, now aptly called *deny lattice*. Computing the *deniable predecessors* of a given deny set $\overline{a} \in A$ is done using a single pre–image operation followed by a complementation: $\mathsf{DPre}(\overline{a}) = L \setminus \delta^{-1}(\overline{a})$. This is the weakest information set from which the player can deny all the control outputs in the deny set without restricting the underlying system. For the example this works out as follows.

Example 4.11 (RADAR) We derive the minimally permissive controller for the control game in Figure 4.18 using RADAR. In Figure 4.20 we show the entire derivation. The algorithm consist in computing the weakest predecessor information sets for each possible deny set. Table 4.1 lists the pre–image and the deniable predecessor sets for the example. When we compare the resulting info/deny pairs, we obtain the following maximal contravariant chain:

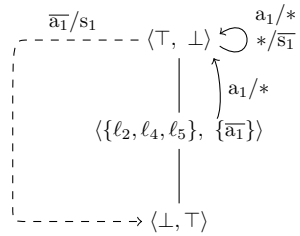
$$\begin{aligned} \langle \{\ell_0, \ell_1, \ell_2, \ell_3, \ell_4, \ell_5\} &\ , \ \varnothing \rangle \\ \langle \{\ell_2, \ell_4, \ell_5\} &\ , \ \{\overline{\mathrm{a}_1}\} \rangle \\ \langle \varnothing &\ , \ \{\overline{\mathrm{a}_1}, \mathrm{a}_1\} \rangle \end{aligned}$$

Note that the pair $\langle \varnothing, \{\mathrm{a}_1\} \rangle$ is subsumed by the pair $\langle \varnothing, \{\overline{\mathrm{a}_1}, \mathrm{a}_1\} \rangle$. In Figure 4.20(b) we show the resulting deny lattice. In Figure 4.20(c) we show the deny lattice after applying DEODAR. In Figure 4.21 we show the minimally permissive control through the deconstructed deny lattice. ◁

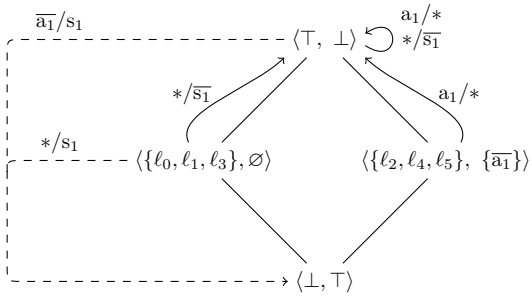
As can be seen, the resulting minimally permissive control PTS in Figure 4.21, does not do much more than to detect inconsistencies. The reason for this is the fact that the control signature is too *narrow*. This means that



(a) Initial deny lattice.



(b) Result after RADAR.



(c) Result after RADAR+DEODAR.

Figure 4.20: Example run of RADAR+DEODAR on deny game $P_{\text{safestamp}_1}[a_1/s_1]$

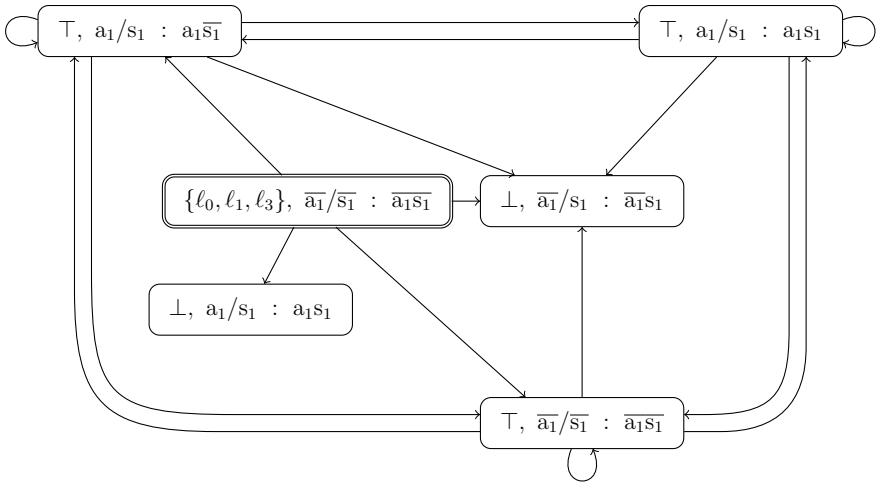


Figure 4.21: Deconstructed deny lattice for control problem $P_{\text{safestamp}_1}[a_1/s_1]$.

our approximation in the form of the minimally permissive controller can not actually *enforce* the safe behavior represented by $P_{\text{safestamp}_1}$.

In the next section we will discuss a method for compositional control synthesis that allows us to systematically *widen* the control signature up to the point where the maximally permissive control and the minimally permissive control coincide.

4.6 Plant Under Control

The compositional algorithm that we define in the next section acts on a *plant under control*, which is a plant model that consist of a number of *plant components*. The following definition makes this precise.

Definition 4.24 (Plant Under Control) We define $\mathfrak{P}$ to be some finite, background index set $\mathfrak{P} = \{\mathfrak{p}_1, \ldots, \mathfrak{p}_m\}$ of *plant component descriptors*. We define P_{plant} to be some background PTS:

$$P_{\text{plant}} = P_{\mathfrak{p}_1} \mid\mid \ldots \mid\mid P_{\mathfrak{p}_m}$$

as the *plant under control*, which is a plant model consisting of a finite number of indexed *plant components*. For each plant component $\mathfrak{p} \in \mathfrak{P}$ we define $X_{\mathfrak{p}}$ as the set of relevant propositions used by $P_{\mathfrak{p}}$, we define $X^{\text{in}}_{\mathfrak{p}}$ as the largest subset of relevant propositions in $X_{\mathfrak{p}}$ for which $P_{\mathfrak{p}}$ is input enabled, we define $X^{\text{out}}_{\mathfrak{p}} = X_{\mathfrak{p}} \setminus X^{\text{in}}_{\mathfrak{p}}$, we call $[X^{\text{out}}_{\mathfrak{p}}/X^{\text{in}}_{\mathfrak{p}}]$ the *component signature of* $\mathfrak{p}$. In addition to the individual component signatures, we assume some background signature $[X^{\text{out}}_{\text{ctrl}}/X^{\text{in}}_{\text{ctrl}}]$ as the *global control signature*. ◁

Example 4.12 (Plant Under Control) For our running example we define the set of plant descriptors as $\mathfrak{P}_{\text{plant}} = \{\text{feed}_0, \text{stamp}_1, \text{stamp}_2\}$ We define the plant model as $P_{\text{plant}} = P_{\text{feed}_0} \mid\mid P_{\text{stamp}_1} \mid\mid P_{\text{stamp}_2}$. The component signatures then become as follows: $X^{\text{out}}_{\text{feed}_0} = \{\text{s}_0, \text{s}_0\}$, $X^{\text{in}}_{\text{feed}_0} = \varnothing$, $X^{\text{out}}_{\text{stamp}_1} = \{\text{s}_1, \text{a}_1, \text{p}_1\}$, $X^{\text{in}}_{\text{stamp}_1} = \{\text{p}_0\}$, $X^{\text{out}}_{\text{stamp}_2} = \{\text{s}_2, \text{a}_2, \text{p}_2\}$, $X^{\text{in}}_{\text{stamp}_2} = \{\text{p}_1\}$. It can be checked that, indeed, P_{stamp_1} and P_{stamp_2} are input enabled for p_0 and p_1, respectively. Finally, we define the global control signature as $X^{\text{out}}_{\text{ctrl}} = \{\text{a}_1, \text{a}_2\}$ and $X^{\text{in}} = \{\text{s}_0, \text{s}_1, \text{s}_2\}$. For a graphical overview of this plant under control cf. Figure 4.2(b). ◁

Before we turn to the compositional algorithm proper we first consider the target result on the global level. We consider the plant model as a whole and

define the maximally and the minimally permissive control in line with the previous sections. The following definitions make this precise.

Definition 4.25 (Global Maximally Permissive Control) For the remainder of this section we let P_{maxctrl} be some background PTS such that it holds $P_{\text{maxctrl}} \in P_{\text{plant}}\lceil X_{\text{ctrl}}^{\text{out}} / X_{\text{ctrl}}^{\text{in}} \rceil$ as the *global maximally permissive control.* $\lhd$

We will refer to the global maximally permissive control in the definitions that follow. Yet, is important to note that we will not rely on the direct computation of the global maximally permissive control in the compositional algorithm that we will present in the next section. Instead we will demonstrate how it is possible to synthesize a controller that has the same effect as the global maximally permissive control whilst proceeding in a compositional fashion.

Example 4.13 (Global Maximally Permissive Control) The global maximally permissive control PTS for our example cannot be easily presented in its raw form. Just like the graph in Figure 4.4(b) the result would not be human–readable anymore. Yet, it is possible to *visualize* this graph to gain a better understanding of the type of structure that we are dealing with. In Figure 4.22 we show a visualization of the maximally permissive control PTS that was generated using a variant of the circular bi–connected graph visualization algorithm by Six and Tollis (1999) and Kaufmann and Wiese (2002). This layout algorithm works by decomposing the graph into maximal connected components, laying out each connected component in a circular fashion and finally laying out the individual connected components according to a spanning tree.

As can be seen P_{maxctrl} for the parcelplant example consists of two maximal connected components. The upper part represents the *steady states* and the lower part consist of the *inconsistent states.* It is possible to transition from the steady state part into the inconsistent state part but, once entered into the inconsistent state part, it is not possible to transition back into the steady state part. Each of the gray transitions that forms part of the fiber–like trunk is such a transition leading down into an inconsistent state. The inconsistent states amongst themselves form a *fully connected component* meaning that *all* behavior is simulated.

The steady states that together form the upper connected component and the transitions among them can be characterized symbolically. In particular we have, for our example, that the steady state satisfies the primed propositional formula: $\phi^{\circlearrowleft} \equiv \mathrm{s}_0 \rightarrow \mathrm{a}_1' \wedge \mathrm{s}_1 \rightarrow \mathrm{a}_2'$ i.e.: if a sensor triggers in the current state then the corresponding actuator will activate in the next state. $\lhd$

The formula $\phi^{\circlearrowleft}$ from the example above is called a *steady state invariant*. We will use the concept of a steady state invariant in the examples that follow. In particular steady state invariants are very useful to characterize controllers in a compact, symbolic way without having to consider the underlying PTS explicitly.

Definition 4.26 (Global Minimally Permissive Control) For the remainder of this section we let P_{minctrl} be some background PTS such that it holds $P_{\text{minctrl}} \in (P_{\text{maxctrl}} \mid\mid P_{\text{plant}})\lfloor X^{\text{out}}_{\text{ctrl}}/X^{\text{in}}_{\text{ctrl}} \rfloor$ as the *global minimally permissive control*. ◁

Example 4.14 (Global Minimally Permissive Control) In Figure 4.23 we show a visualization of P_{minctrl} for the parcelplant example. As can be seen also this control PTS consists of a steady state part and an inconsistent state part. The only difference with P_{maxctrl} as shown in Figure 4.22 is the fact that the inconsistent state part now consists of many individual deadlock locations rather than one large, fully connected component. ◁

In between the extrema of the global maximally and minimally permissive control there potentially exist many control PTSs P' such that $P_{\text{minctrl}} \leq P' \leq P_{\text{maxctrl}}$. These alternative control PTSs can still be interesting controllers for the plant under control even when they are not strictly speaking maximally or minimally permissive. In particular it is possible to do *loose inconsistency tracking* where not every inconsistent observation is necessarily detected (cf. Section 3.4). The advantage of this is that the controllers are potentially a lot smaller because they need not keep track of all possible inconsistent observations. To keep the presentation focussed, in the remainder of this section we will continue to work with the strict definition of maximally and minimally permissive control.

4.7 Algorithm For Compositional Synthesis

In the previous section we defined the plant under control consisting of a number of plant components. We also defined the global maximally and minimally permissive control over this model. In this section we consider the problem of deriving such a global controller for the given plant under control in a compositional fashion. In particular we will approximate the global control from above by deriving control constraints for a gradually increasing subset of the full plant model. We call such a subset of the full set of plant component a *locality*. The following definition makes this precise.

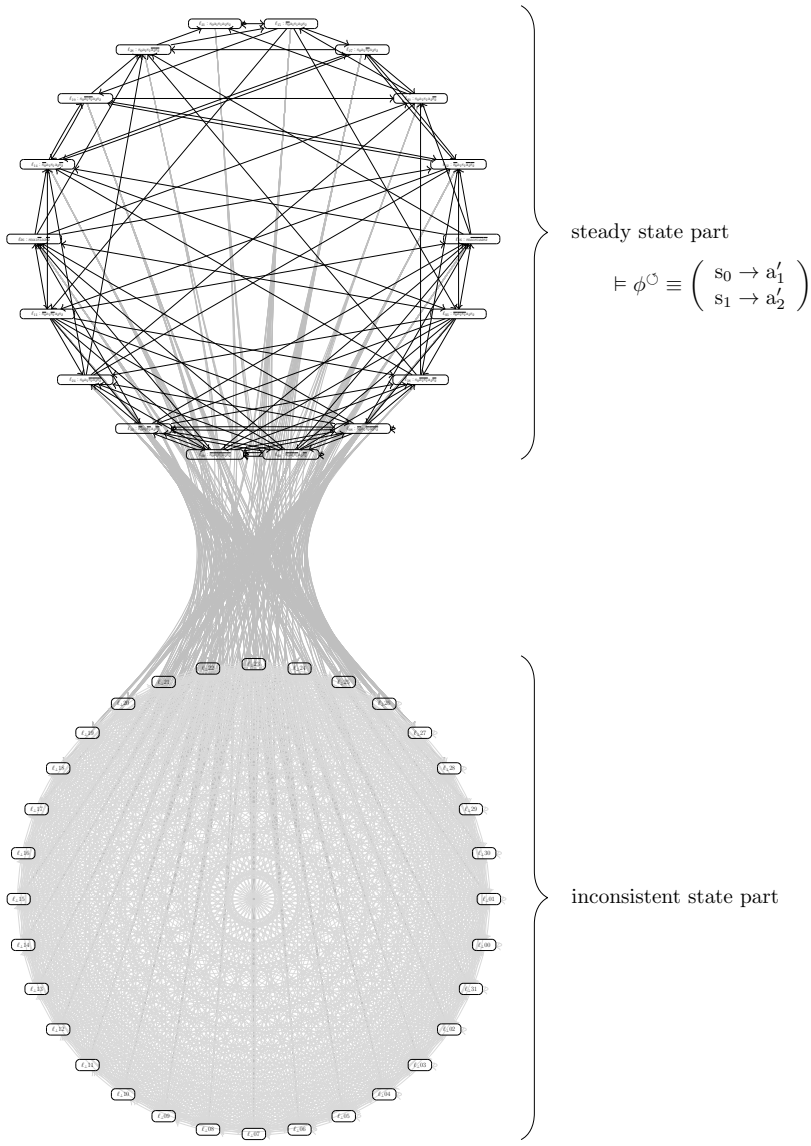


Figure 4.22: Visualization of P_{maxctrl} for the parcelplant example.

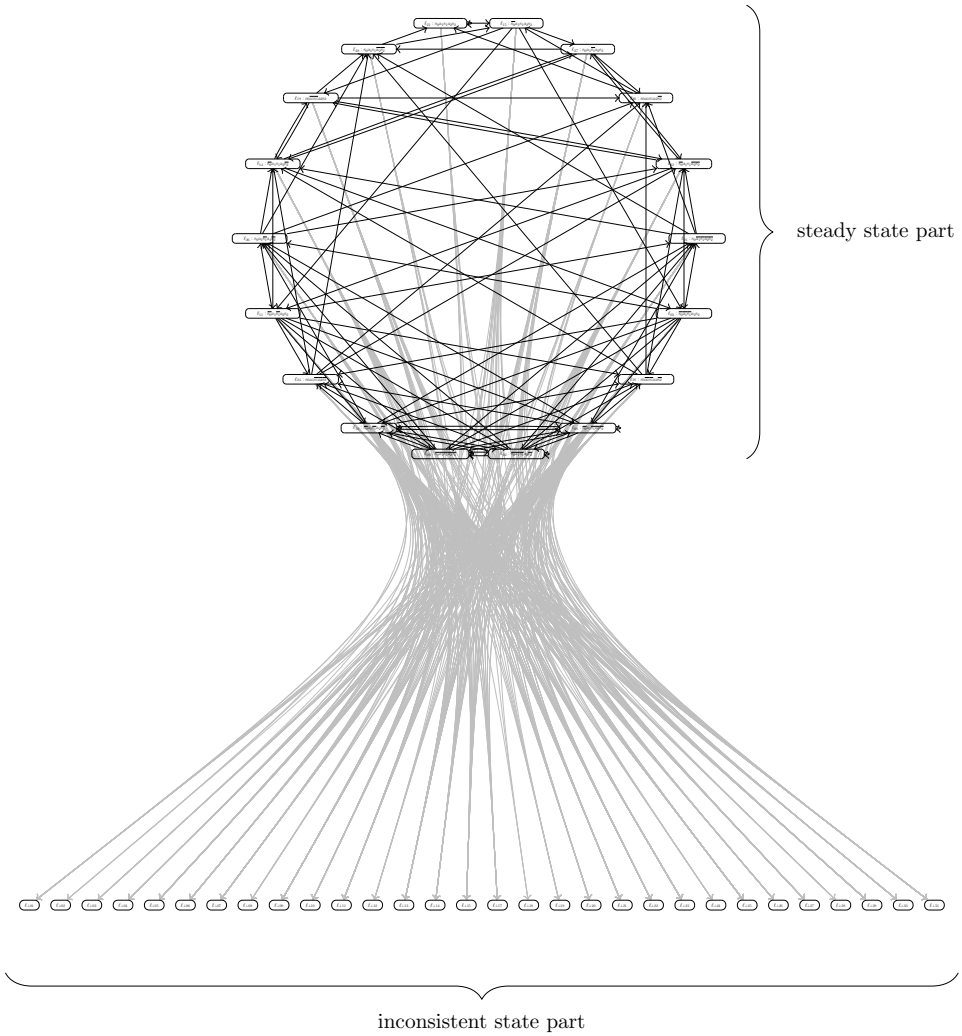


Figure 4.23: Visualization of P_{minctrl} for the parcelplant example.

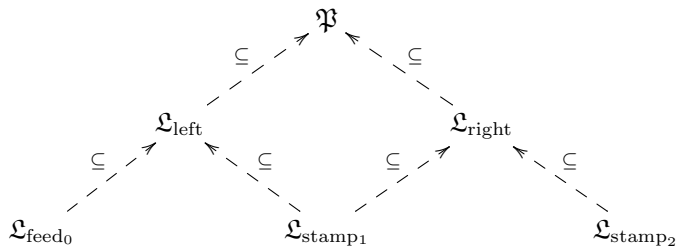


Figure 4.24: Directed acyclic graph based on locality inclusion.

Definition 4.27 (Localities) We define a *locality* $\mathfrak{L}$ as a subset $\mathfrak{L} \subseteq \mathfrak{P}$ of plant components. For a given locality $\mathfrak{L}$ we define: $P_{\mathfrak{L}} = \Pi_{\mathfrak{p} \in \mathfrak{L}} P_{\mathfrak{p}}$ as the product of the plant components in $\mathfrak{L}$, $X_{\mathfrak{L}} = \bigcup_{\mathfrak{p} \in \mathfrak{L}} X_{\mathfrak{p}}$ as the set of all relevant propositions of the plant components in $\mathfrak{L}$, $X^{\text{out}}_{\mathfrak{L}} = \bigcup_{\mathfrak{p} \in \mathfrak{L}} X^{\text{out}}_{\mathfrak{p}}$ as the set of all *output* propositions of the plant components in $\mathfrak{L}$, $X^{\text{in}}_{\mathfrak{L}} = X_{\mathfrak{L}} \setminus X^{\text{out}}_{\mathfrak{L}}$ as the set of all *input* propositions of plant components in $\mathfrak{L}$, and $\overline{\mathfrak{L}} = \mathfrak{P} \setminus \mathfrak{L}$ as the *periphery* of $\mathfrak{L}$ which are all the plant components in the model that fall outside of $\mathfrak{L}$. We assume a background set of *relevant localities* $\mathsf{Localities} \subseteq \mathbb{P}(\mathfrak{P})$. We assume that at least the plant in its entirety is a relevant locality: $\mathfrak{P} \in \mathsf{Localities}$. We further assume that the empty locality is never relevant: $\varnothing \notin \mathsf{Localities}$. For a given relevant locality $\mathfrak{L} \in \mathsf{Localities}$ we define the *relevant child localities* $\downdownarrows \mathfrak{L}$ of $\mathfrak{L}$ such that

$$\downdownarrows \mathfrak{L} = \lceil \{\mathfrak{L}' \in \mathsf{Localities} \mid \mathfrak{L}' \subset \mathfrak{L}\} \rceil$$

i.e.: all the largest sublocalities of $\mathfrak{L}$. $\lhd$

Example 4.15 (Localities) The relevant localities for our running example are:

$$\mathsf{Localities} = \{\mathfrak{P}, \mathfrak{L}_{\text{left}}, \mathfrak{L}_{\text{right}}, \mathfrak{L}_{\text{feed}_0}, \mathfrak{L}_{\text{stamp}_1}, \mathfrak{L}_{\text{stamp}_2}\}$$

where $\mathfrak{P} = \{\text{feed}_0, \text{stamp}_1, \text{stamp}_2\}$ the full index set is the locality covering the plant in its entirety, $\mathfrak{L}_{\text{left}} = \{\text{feed}_0, \text{stamp}_1\}$ is the locality covering the left part of the plant with the feeder and first parcel stamp, $\mathfrak{L}_{\text{right}} = \{\text{stamp}_1, \text{stamp}_2\}$ is the right part of the plant with the first and the second parcel stamp,

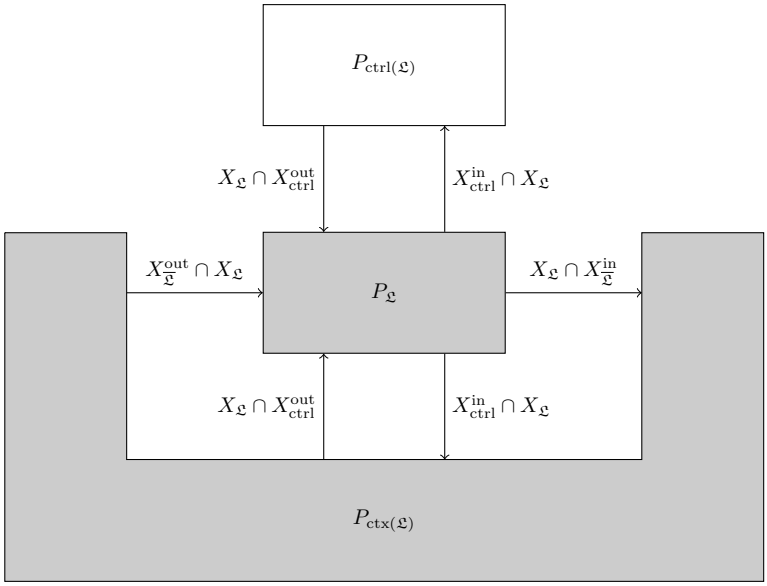


Figure 4.25: A locality with its context and local control and their signatures.

$\mathfrak{L}_{\mathrm{feed}_0} = \{\mathrm{feed}_0\}$ is the singleton locality containing only the feeder in isolation, $\mathfrak{L}_{\mathrm{stamp}_1} = \{\mathrm{stamp}_1\}$ is the singleton locality containing only the first parcel stamp in isolation, $\mathfrak{L}_{\mathrm{stamp}_2} = \{\mathrm{stamp}_2\}$ is the singleton locality containing only the second parcel stamp in isolation. Figure 4.24 shows the corresponding locality DAG. ◁

In order to arrive at a compositional algorithm, we need the ability to derive a *local controller* for a given locality $\mathfrak{L} \subseteq \mathfrak{P}$ *without factoring in the periphery* $\overline{\mathfrak{L}}$. If we were to consider the product with the periphery we would obtain the entire plant model $\mathfrak{P} = \mathfrak{L} \cup \overline{\mathfrak{L}}$, which is exactly what we are trying to avoid when deriving a controller at a local level. For this reason we first define the *local control signature* as the restriction of the global control signature to the relevant propositions for the locality. The following definition makes this precise.

Definition 4.28 (Local Control Signature) For a given locality $\mathfrak{L} \subseteq \mathfrak{P}$ we define the *local control output propositions* $X^{\mathrm{out}}_{\mathrm{ctrl}(\mathfrak{L})} = X^{\mathrm{out}}_{\mathrm{ctrl}} \cap X_{\mathfrak{L}}$, and the *local control input propositions* $X^{\mathrm{in}}_{\mathrm{ctrl}(\mathfrak{L})} = X^{\mathrm{in}}_{\mathrm{ctrl}} \cap X_{\mathfrak{L}}$. ◁

Table 4.2: Local control signatures for the parcelplant example.

$\mathfrak{L}$	$X_{\mathfrak{L}}$	$X^{\text{out}}_{\text{ctrl}(\mathfrak{L})}/X^{\text{in}}_{\text{ctrl}(\mathfrak{L})}$
$\mathfrak{P}$	$\mathrm{p}_0\mathrm{s}_0\mathrm{p}_1\mathrm{a}_1\mathrm{s}_1\mathrm{p}_2\mathrm{a}_2\mathrm{s}_2$	$\mathrm{a}_1\mathrm{a}_2/\mathrm{s}_0\mathrm{s}_1\mathrm{s}_2$
$\mathfrak{L}_{\text{left}}$	$\mathrm{p}_0\mathrm{s}_0\mathrm{p}_1\mathrm{a}_1\mathrm{s}_1$	$\mathrm{a}_1/\mathrm{s}_0\mathrm{s}_1$
$\mathfrak{L}_{\text{right}}$	$\mathrm{p}_0\mathrm{p}_1\mathrm{a}_1\mathrm{s}_1\mathrm{p}_2\mathrm{a}_2\mathrm{s}_2$	$\mathrm{a}_1\mathrm{a}_2/\mathrm{s}_1\mathrm{s}_2$
$\mathfrak{L}_{\text{feed}_0}$	$\mathrm{p}_0\mathrm{s}_0$	$\varnothing/\mathrm{s}_0$
$\mathfrak{L}_{\text{stamp}_1}$	$\mathrm{p}_0\mathrm{p}_1\mathrm{a}_1\mathrm{s}_1$	$\mathrm{a}_1/\mathrm{s}_1$
$\mathfrak{L}_{\text{stamp}_2}$	$\mathrm{p}_1\mathrm{p}_2\mathrm{a}_2\mathrm{s}_2$	$\mathrm{a}_2/\mathrm{s}_2$

Example 4.16 (Local Control Signatures) The local control signatures for our running example are given in Table 4.2. For each locality the control signature contains only the relevant control outputs and inputs. For the top locality $\mathfrak{P}$ the control signature coincides with the context signature (cf. Example 4.17).$\lhd$

Perhaps, intuitively, it may seem that it is possible to derive a local controller directly based on the local control signature. However, the problem that we face is the fact that this local control signature may be *too narrow*, resulting in a local controller that is too restrictive. In particular it may be the case that information can flow to and from the controller through the *open periphery–in and outputs*: $X^{\text{in}}_{\mathfrak{L}} \cap X_{\mathfrak{L}}$ and $X^{\text{out}}_{\mathfrak{L}} \cap X_{\mathfrak{L}}$ as is illustrated in Figure 4.25. The solution that we take is to simultaneously abstract the periphery and the controller into a *context*. Because this context sees all the open periphery inputs (in addition to the control inputs) and controls all the open periphery outputs (in addition to the control outputs) we obtain a context that is *conservative* in the restrictions that it imposes on the plant. The following definition makes this precise.

Definition 4.29 (Conservativity) A (context/local control) PTS P is *conservative* iff it holds that:

$$P \geq P_{\text{maxctrl}} \,||\, P_{\text{plant}}$$

i.e.: P allows at least all the behavior that is allowed by the plant under maximally permissive control. $\lhd$

This definition makes use of the object P_{maxctrl}, the global maximally permissive control. For our purposes, this means that this is not an effective definition yet, as we want to avoid direct construction of the global maximally

Table 4.3: Locality context signatures for the parcelplant example.

$\mathfrak{L}$	$X_{\mathfrak{L}}$	$X^{\text{out}}_{\text{ctx}(\mathfrak{L})}/X^{\text{in}}_{\text{ctx}(\mathfrak{L})}$
$\mathfrak{P}$	$\text{p}_0\text{s}_0\text{p}_1\text{a}_1\text{s}_1\text{p}_2\text{a}_2\text{s}_2$	$\text{a}_1\text{a}_2/\text{s}_0\text{s}_1\text{s}_2$
$\mathfrak{L}_{\text{left}}$	$\text{p}_0\text{s}_0\text{p}_1\text{a}_1\text{s}_1$	$\text{a}_1/\text{s}_0\text{s}_1\text{p}_1$
$\mathfrak{L}_{\text{right}}$	$\text{p}_0\text{p}_1\text{a}_1\text{s}_1\text{p}_2\text{a}_2\text{s}_2$	$\text{p}_0\text{a}_1\text{a}_2/\text{s}_1\text{s}_2$
$\mathfrak{L}_{\text{feed}_0}$	p_0s_0	$\varnothing/\text{s}_0\text{p}_0$
$\mathfrak{L}_{\text{stamp}_1}$	$\text{p}_0\text{p}_1\text{a}_1\text{s}_1$	$\text{p}_0\text{a}_1/\text{s}_1\text{p}_1$
$\mathfrak{L}_{\text{stamp}_2}$	$\text{p}_1\text{p}_2\text{a}_2\text{s}_2$	$\text{p}_1\text{a}_2/\text{s}_2$

permissive control. For this reason we will *overapproximate* $P_{\text{maxctrl}} \parallel P_{\text{plant}}$ at a local level. Once we have such a local overapproximation we may use this local overapproximation to derive a conservative local controller over it. The way we make the overapproximation at a local level is by applying the context abstraction as we discussed above. The following definition makes this precise.

Definition 4.30 (Locality Context Signature) For a given locality $\mathfrak{L} \subseteq \mathfrak{P}$ we define the *context output propositions* $X^{\text{out}}_{\text{ctx}(\mathfrak{L})} = (X^{\text{out}}_{\text{ctrl}} \cup X^{\text{out}}_{\overline{\mathfrak{L}}}) \cap X_{\mathfrak{L}}$, and the *context input propositions* $X^{\text{in}}_{\text{ctx}(\mathfrak{L})} = ((X^{\text{in}}_{\text{ctrl}} \cup X^{\text{in}}_{\overline{\mathfrak{L}}}) \cap X_{\mathfrak{L}}) \setminus X^{\text{out}}_{\text{ctx}(\mathfrak{L})}$. ◁

Example 4.17 (Locality Context Signatures) The relevant locality context signatures for our running example are given in Table 4.3. The first thing to note from this table is how, for each locality, the context signature contains both the control outputs and inputs for that locality as well as the open periphery outputs and inputs. The second thing to note is the fact that, for the top locality $\mathfrak{P}$, the context signature coincides with the global control signature (cf. Example 4.16). ◁

Now that we have all prerequisites in place we are in a position to present Algorithm 4. The *compositional controller synthesis algorithm*, or COCOS for short, works by applying the context abstraction followed by a local control derivation (as discussed above). The algorithm does this for each relevant locality. In particular the algorithm starts from the *leaf localities* and gradually works its way up the locality hierarchy until it reaches the *top locality* (cf. Figure 4.24). The exact order in which the localities are treated is unspecified as long as all child localities are treated before their parent locality is treated.

Algorithm 4: COCOS — Compositional Control Synthesis

Data: A set of plant component descriptors $\mathfrak{P} = \{\mathfrak{p}_1, \ldots, \mathfrak{p}_m\}$, a plant model $P_{\text{plant}} = P_{\mathfrak{p}_1} \parallel \ldots \parallel P_{\mathfrak{p}_m}$, a global control signature $[X^{\text{out}}_{\text{ctrl}}/X^{\text{in}}_{\text{ctrl}}]$, and a set of relevant localities $\mathsf{Localities} \subseteq \mathbb{P}(\mathfrak{P})$, for which it holds that, at least, $\mathfrak{P}$ (the plant in its entirety) is a relevant locality, and $\varnothing$ (the empty locality) is not a relevant locality.

Result: Returns a conservative, safe controller for P_{plant}

1 $\mathsf{Treated} \leftarrow \varnothing$

2 **while** $\mathfrak{P} \notin \mathsf{Treated}$ **do**

3 **let** $\mathfrak{L} \in \mathsf{Localities}$ **such that** $\mathfrak{L} \notin \mathsf{Treated}$ **and** $\downdownarrows \mathfrak{L} \subseteq \mathsf{Treated}$

4 **synthesize** $P_{\text{ctx}(\mathfrak{L})} \in (\Pi_{\mathfrak{L} \in \downdownarrows \mathfrak{L}} P_{\text{ctrl}(\mathfrak{L})} \parallel P_{\mathfrak{L}})\lceil X^{\text{out}}_{\text{ctx}(\mathfrak{L})}/X^{\text{in}}_{\text{ctx}(\mathfrak{L})}\rceil$

5 **synthesize** $P_{\text{ctrl}(\mathfrak{L})} \in (P_{\text{ctx}(\mathfrak{L})} \parallel \Pi_{\mathfrak{L} \in \downdownarrows \mathfrak{L}} P_{\text{ctrl}(\mathfrak{L})} \parallel P_{\mathfrak{L}})\lfloor X^{\text{out}}_{\text{ctrl}(\mathfrak{L})}/X^{\text{in}}_{\text{ctrl}(\mathfrak{L})}\rfloor$

6 $\mathsf{Treated} \leftarrow \mathsf{Treated} \cup \{\mathfrak{L}\}$

7 **return** $P_{\text{ctrl}(\mathfrak{P})}$

We give a short description of the basic steps of the algorithm. In line 1 the set of treated localities is initialized. In line 2 we enter the main loop. The termination condition checks whether the top locality (the locality that encompasses the plant in its entirety) has been treated. If this is not yet the case in line 3 we select a locality for which all the direct child localities (if any) have already been treated.

In line 4 we synthesize a context for the given locality taking into account local controllers for the direct child localities (if any) that have previously been synthesized. Note that the keyword **synthesize** presupposes some underlying game solving algorithm that is able to construct a maximally permissive control PTS as per Definition 4.13. We recall Section 4.4 where we showed how CEDAR+DEODAR can be used for this purpose. As such, in the remainder we assume that CEDAR+DEODAR is used for the synthesis step in line 4.

In line 5 we synthesize a conservative local controller based on the safe system consisting of the plant model, the previously synthesized controllers for the child localities and the additional constraints imposed by the newly synthesized context. We recall Section 4.5 where we showed how RADAR+DEODAR can be used for this purpose. As such, in the remainder we assume RADAR+DEODAR will be used for the synthesis step in line 5.

In line 6 we add the locality to the treated set. As soon as the top locality has been treated we enter into line 7 where the local controller for the top locality is returned.

Example 4.18 (Sample Run of COCOS) We consider a run of the COCOS algorithm on the parcelplant example:

1. We assume that the first selected leaf locality is $\mathfrak{L}_{\text{stamp}_1}$. In Figure 4.26 we show the intermediate results for this locality. We use steady state invariants $\phi^{\circlearrowleft}$ to indicate the control constraints found in the form of the locality context and the local control. As can be seen, the locality context places the constraint $\phi^{\circlearrowleft} \equiv (\text{p}_0 \rightarrow \text{a}_1')$, the local control does not place any constraint. The reason for this is the fact that the control signature is too narrow. In particular the proposition p_0 is part of the periphery, and not a control input. It is therefore impossible to enforce the control constraint $\phi^{\circlearrowleft} \equiv (\text{p}_0 \rightarrow \text{a}_1)$ through the local control signature. The only solution that is conservative is a local controller that is vacuous, i.e.: that places no control constraint on the plant whatsoever.

2. We skip the other leaf localities $\mathfrak{L}_{\text{feed}_0}$ and $\mathfrak{L}_{\text{stamp}_2}$ because these are handled completely analogously.

3. Next, we assume the selected locality is $\mathfrak{L}_{\text{left}}$. In Figure 4.27 we show the intermediate results for this locality. As can be seen from the steady state invariants, the locality context places the constraint $\phi^{\circlearrowleft} \equiv (\text{s}_0 \rightarrow \text{a}_1')$, note that p_0 is no longer part of the context signature, it is now an *internal proposition* for the locality, the local control places the same constraint $\phi^{\circlearrowleft} \equiv (\text{s}_0 \rightarrow \text{a}_1')$. This is the first instance in which we can actually see a control constraint being derived in a local context.

4. We skip the locality $\mathfrak{L}_{\text{right}}$ because it is handled completely analogously.

5. We then arrive at the top locality $\mathfrak{P}$. In Figure 4.28 we show the final results for the top locality. As can be seen from the steady state invariants, the locality context places *no* constraint, the reason for this is the fact that all the necessary control constraints are already in place, the local control places the constraint $\phi^{\circlearrowleft} \equiv (\text{s}_0 \rightarrow \text{a}_1') \wedge (\text{s}_1 \rightarrow \text{a}_2')$ as expected.

In Figure 4.29 we show the final result applied directly to the plant under control. ◁

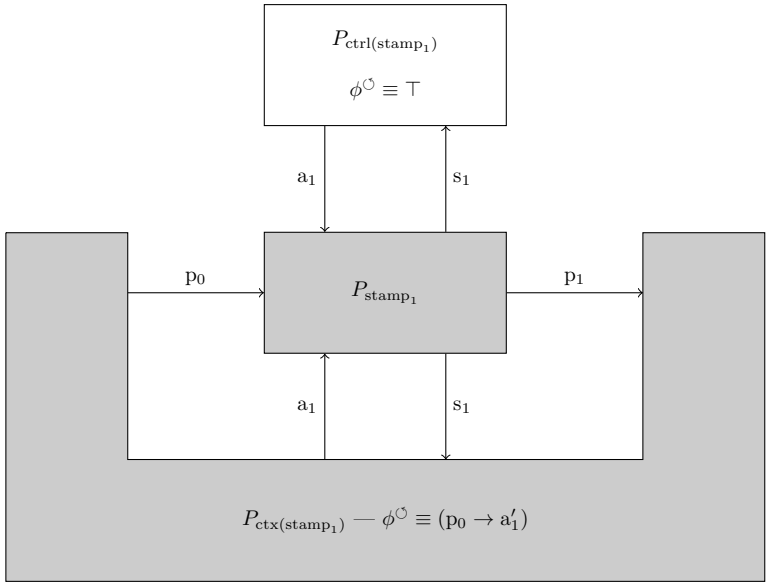


Figure 4.26: Intermediate results of COCOS for locality $\mathfrak{L}_{\text{stamp}_1}$

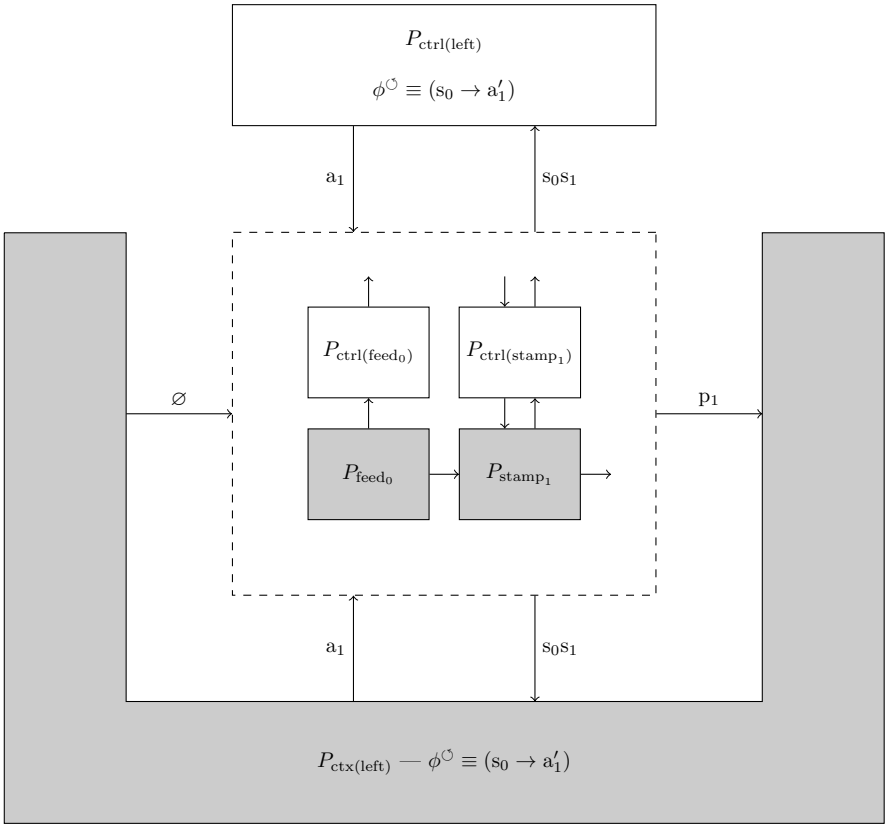


Figure 4.27: Intermediate results of COCOS for locality $\mathfrak{L}_{\text{left}}$

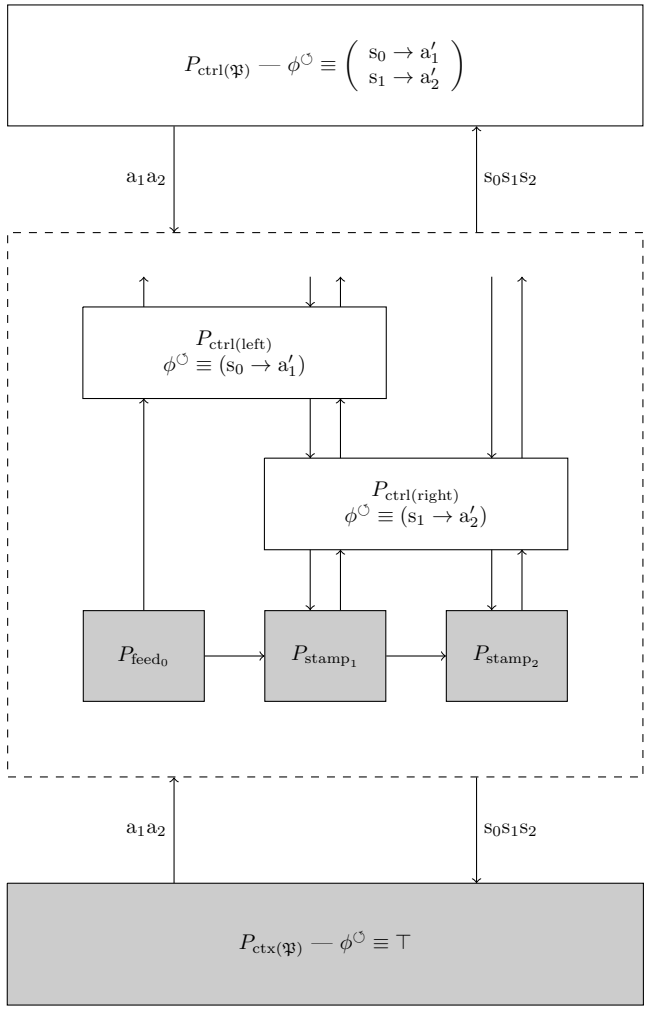


Figure 4.28: Final results of COCOS for the top locality $\mathfrak{P}$

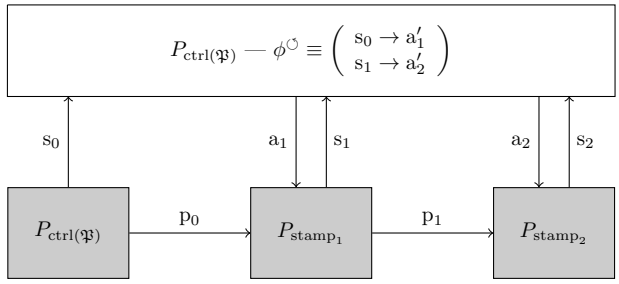


Figure 4.29: Final result of COCOS applied directly to the plant under control.

The crucial property on which the correctness of COCOS rests is the fact that all the intermediate results $P_{\mathrm{ctx}(\mathfrak{L})}$ and $P_{\mathrm{ctrl}(\mathfrak{L})}$ are conservative. This is the content of the following lemma.

Lemma 4.31 (Conservativity) It is an invariant of Algorithm 4 that for all $\mathfrak{L} \in \mathsf{Localities}$ it holds $P_{\mathrm{ctx}(\mathcal{L})} \geq P_{\mathrm{maxctrl}} \mid\mid P_{\mathrm{plant}}$ and $P_{\mathrm{ctrl}(\mathcal{L})} \geq P_{\mathrm{maxctrl}} \mid\mid P_{\mathrm{plant}}$, i.e.: both the context and the local controllers are conservative. $\lhd$

Proof By induction on the size of $\mathfrak{L}$. For the base case $\mathfrak{L} = \varnothing$ the lemma follows trivially as $\varnothing \notin \mathsf{Localities}$. For the inductive case assume $\mathfrak{L} \in \mathsf{Localities}$ such that $|\mathfrak{L}| = n$. Assume, by inductive hypothesis, that the lemma holds for all localities of size $n-1$, in particular the lemma holds for $\Pi_{\mathfrak{L} \in \downdownarrows \mathfrak{L}} P_{\mathrm{ctrl}(\mathfrak{L})}$. In line 4 we synthesize $P_{\mathrm{ctx}(\mathfrak{L})}$ as a most permissive, safe control PTS for $(\Pi_{\mathfrak{L} \in \downdownarrows \mathfrak{L}} P_{\mathrm{ctrl}(\mathfrak{L})} \mid\mid P_{\mathfrak{L}})\lceil X^{\mathrm{out}}_{\mathrm{ctx}(\mathfrak{L})} / X^{\mathrm{in}}_{\mathrm{ctx}(\mathfrak{L})} \rceil$. Now we construct:

$$P_{\mathrm{witness}} \in (P_{\mathrm{maxctrl}} \mid\mid P_{\overline{\mathfrak{L}}})\lfloor X^{\mathrm{out}}_{\mathrm{ctx}(\mathfrak{L})} / X^{\mathrm{in}}_{\mathrm{ctx}(\mathfrak{L})} \rfloor$$

i.e.: we take the combination of the global maximally permissive control and the periphery of $\mathfrak{L}$ and project out all the propositions from the control signature that are not part of the context signature. It is not hard to see that P_{witness} is a safe control PTS for the control problem $(\Pi_{\mathfrak{L} \in \downdownarrows \mathfrak{L}} P_{\mathrm{ctrl}(\mathfrak{L})} \mid\mid P_{\mathfrak{L}})\lceil X^{\mathrm{out}}_{\mathrm{ctx}(\mathfrak{L})} / X^{\mathrm{in}}_{\mathrm{ctx}(\mathfrak{L})} \rceil$. It follows $P_{\mathrm{ctx}(\mathfrak{L})} \geq P_{\mathrm{witness}}$ and, transitively, $P_{\mathrm{ctx}(\mathfrak{L})} \geq P_{\mathrm{maxctrl}} \mid\mid P_{\mathrm{plant}}$ (recall that $P_{\mathrm{plant}} = P_{\mathfrak{L}} \mid\mid P_{\overline{\mathfrak{L}}}$). We continue with line 5 where we synthesize $P_{\mathrm{ctrl}(\mathfrak{L})}$ as a minimally permissive control PTS above $P_{\mathrm{ctx}}(\mathfrak{L}) \mid\mid \Pi_{\mathfrak{L} \in \downdownarrows \mathfrak{L}} P_{\mathrm{ctrl}(\mathfrak{L})} \mid\mid P_{\mathfrak{L}}$ We already showed $P_{\mathrm{ctx}}(\mathfrak{L})$ to be conservative so it must follow by definition that $P_{\mathrm{ctrl}(\mathfrak{L})}$ is conservative. $\square$

Theorem 4.32 (COCOS**)** After completion of Algorithm 4 it holds that:

$$P_{\mathrm{ctrl}(\mathfrak{P})} \sim P_{\mathrm{minctrl}}$$

i.e.: the algorithm returns the global minimally permissive control for P_{plant}. $\lhd$

Proof By definition $P_{\mathfrak{P}} = P_{\mathrm{plant}}$ and $X^{\mathrm{out}}_{\mathrm{ctx}(\mathfrak{P})} = X^{\mathrm{out}}_{\mathrm{ctrl}}$ and $X^{\mathrm{in}}_{\mathrm{ctx}(\mathfrak{P})} = X^{\mathrm{in}}_{\mathrm{ctrl}}$ this implies that, in the final iteration of the algorithm, line 4 reduces to:

synthesize $P_{\mathrm{ctx}(\mathfrak{P})} \in (\Pi_{\mathfrak{L} \in \downdownarrows \mathfrak{L}} P_{\mathrm{ctrl}(\mathfrak{L})} \mid\mid P_{\mathrm{plant}})\lceil X^{\mathrm{out}}_{\mathrm{ctrl}} / X^{\mathrm{in}}_{\mathrm{ctrl}} \rceil$

By Lemma 4.31 we have that the PTS $\Pi_{\mathfrak{L} \in \downdownarrows \mathfrak{L}} P_{\mathrm{ctrl}(\mathfrak{L})}$ is conservative. Now recall that P_{maxctrl} is defined such that $P_{\mathrm{maxctrl}} \in P_{\mathrm{plant}}\lceil X^{\mathrm{out}}_{\mathrm{ctrl}} / X^{\mathrm{in}}_{\mathrm{ctrl}} \rceil$. It is not hard to see that this implies that:

$$(P_{\mathrm{ctx}(\mathfrak{P})} \mid\mid \Pi_{\mathfrak{L} \in \downdownarrows \mathfrak{L}} P_{\mathrm{ctrl}(\mathfrak{L})} \mid\mid P_{\mathrm{plant}}) \sim (P_{\mathrm{maxctrl}} \mid\mid P_{\mathrm{plant}})$$

This, in turn, implies that, in the final iteration, line 5 can be equivalently written as:

synthesize $P_{\text{ctrl}(\mathfrak{P})} \in (P_{\text{maxctrl}} \mid\mid P_{\text{plant}}) \lfloor X^{\text{out}}_{\text{ctrl}} / X^{\text{in}}_{\text{ctrl}} \rfloor$

Which means, by definition, $P_{\text{ctrl}(\mathfrak{P})} \sim P_{\text{minctrl}}$. □

4.8 Efficiency Concerns and Optimizations

In this section we discuss an important optimization for COCOS. We start from the observation that, in many cases, it is possible to find *successor independencies* between outputs for a given plant model. Successor independence refers to the fact that often two or more output propositions are always suppressed or allowed *independently* of each other in the successor state. The following definition makes this precise.

Definition 4.33 (Extended Signature) We define an *extended signature*:

$$[X^{\text{out}}_1, \ldots, X^{\text{out}}_n / X^{\text{in}}]$$

as a *partitioned set of output propositions* $X^{\text{out}} = X^{\text{out}}_1 \cup \cdots \cup X^{\text{out}}_n$ in combination with a set of *input* propositions X^{in} such that $X^{\text{out}} \cap X^{\text{in}} = \varnothing$. For a given X^{out}_i we define $X^{\text{out}}_{\bar{i}}$ as the complement $X^{\text{out}} \setminus X^{\text{out}}_i$. We say a PTS P *respects* the extended signature iff it can be decomposed into $P \sim P_1 \mid\mid \ldots \mid\mid P_n$ such that each P_i respects the signature $[X^{\text{out}}_i / X^{\text{in}} \cup X^{\text{out}}_{\bar{i}}]$. ◁

Successor independence occurs very naturally when we are considering a plant under control that consists of several plant components. The extended signature can be induced from the individual plant component signatures. In addition it is possible to work with user supplied information, i.e.: a user directly indicates that the controller she wants to synthesize should respect some extended signature.

Whenever we know that we are synthesizing a controller that should respect an extended signature we can use this fact to our advantage. We do this by an important optimization of the CEDAR algorithm underlying COCOS and an important optimization of the RADAR+DEODAR algorithm underlying COCOS.

For the CEDAR algorithm this consists in an optimization of the restricted successor step. In particular it is possible to replace the single restricted successor pair that we introduce into the allow lattice with a set of n restricted successor pairs, one for each subset in the output partition. With the symbolic

framework that we introduced in this chapter this becomes very straightforward. Basically we make n projections of the observation, one for each of the subsets in the output partition, instead of just one. This is valid since we know that, any restriction on the output we will have to be done independently for each of the subsets in the output partition.

For the RADAR+DEODAR algorithm the optimization goes even further in the sense that it is possible to synthesize n individual controllers, one for each subset in the output partition. To obtain the end–result we then take the composition of these individual control PTSs. This is valid because the resulting set of controllers adheres precisely to Definition 4.33.

Chapter 5

Experiments

5.1 Introduction

In this chapter we evaluate further the compositional framework that was developed in Chapter 4. We will validate the compositional aspects of the framework through several experiments. For running these experiments we will also rely on the results from the first part of the thesis. In particular we will use the CEDAR and DEODAR algorithms that were developed in Chapters 2, and 3, respectively.

The chapter is structured as follows. In Section 5.2 we briefly discuss our implementation. In Section 5.3 we discuss the hypothesis that there is a positive relation between compositionality and scalability. In Section 5.4 we discuss the ability of the COCOS compositional algorithm to derive residual constraints for higher localities. In Section 5.5 we discuss some conclusions based on these experiments.

5.2 Implementation

All the experiments in this chapter were carried out with a single threaded implementation of COCOS with solvers CEDAR+DEODAR and RADAR+DEODAR. The implementation makes full use of the optimizations of sparse allow lattices (cf. Section 2.5), loose inconsistency tracking (cf. Section 3.4), and successor independence (cf. Section 4.8).

The tool is written in Java. All the symbolic BDD operations are performed through a modified version JavaBDD which is a Java wrapper around Buddy (a

native C implementation). The modifications consist in integration of the Buddy reference counting scheme with the Java virtual machine garbage collector. The latter incurs a slight variation in running times that is amortized by averaging each experiment over several runs. The experiments were run on a dual core 2.4 GHz machine.

5.3 Compositionality and Scalability

In this section we test the hypothesis that compositionality has a positive effect on scalability. Here, scalability should be understood in terms of the relation between some suitable measure of problem size and some suitable measure of performance. As a measure of problem size we will simply take the number of plant components. As a measure of performance we will simply take the running time of the entire COCOS+CEDAR+DEODAR synthesis procedure. In addition, we will measure the number of refinement steps performed in the CEDAR phase. The latter measure has the advantage that it does not depend on the performance of the underlying symbolic engine.

In order to be able to vary the model size in a convenient, reproducible fashion we will test our algorithms on a parameterized version of the parcel plant example from Chapter 4. The parameter we introduce is the number N of plant components in the plant model. We can then obtain scalability results by testing how the algorithms perform for increasing N.

5.3.1 Parameterized Parcel Plant Model

For parameterizing our parcel plant model we use an indexed version of the plant component $\text{stamp}_{i,j}$ where i is the index of the incoming p–proposition (which is the proposition that is acted upon by the left neighbor component) and j is the index of the outgoing p–proposition (which is the proposition that is acted upon by the actuator and sensor propositions). We then arrange these components in a circular topology as follows:

$$P^N_{\text{stamp}_{i,j}} = P_{\text{stamp}_{1,2}} || P_{\text{stamp}_{2,3}} || \ldots || P_{\text{stamp}_{N-1,N}} || P_{\text{stamp}_{N,1}}$$

To illustrate the topology of the model, as an example, in Figure 5.1 we show the component graph for $N = 5$. Note that the resulting model no longer has a feeder component. For this reason we also weaken the initial state invariant to make sure that the initial arrangement of parcels on the belt is random. If we would keep the initial state invariant as it was defined in Chapter 4 the control

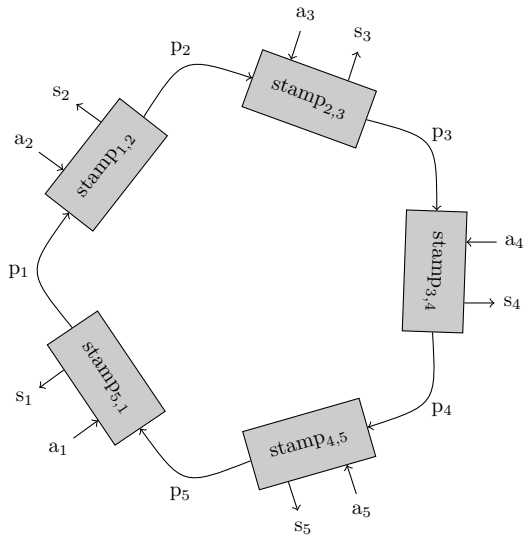


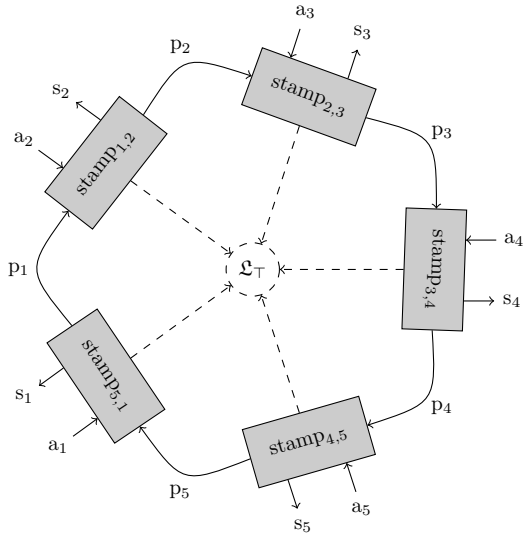
Figure 5.1: The circular topology of parcel stamp components for $N = 5$.

problem would become trivial since in that case there would never be any parcel entering into the system.

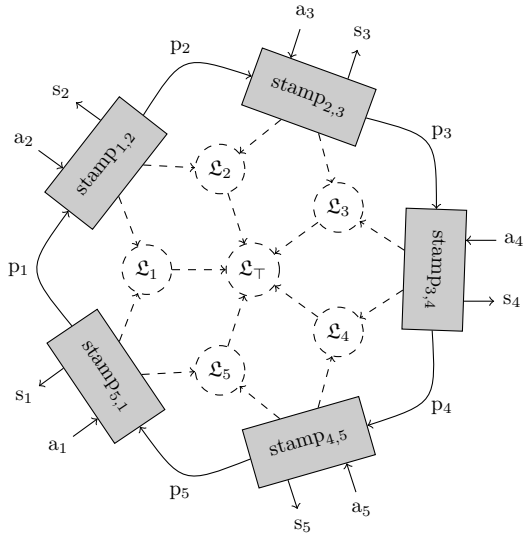
5.3.2 Monolithic Variant and Compositional Variant

We now define two variants of this model with respect to the locality structure. Both variants are illustrated in Figure 5.2. These diagrams use dashed arrows to indicate membership of a component into a locality or inclusion of a locality into a higher locality. This gives us another way of presenting the familiar ordering based on locality inclusion, in this case as a directed acyclic graph fanning out radially from the center.

We will refer to the first variant as shown in Figure 5.2(a) as the *monolithic* variant, and we will refer to the second variant as shown in Figure 5.2(b) as the *compositional* variant. The compositional variant is so called because it features a basic decomposition structure into N localities consisting of successive pairs of plant components. The monolithic variant is so called because it features only the top locality that includes all the plant components at once. By contrasting the results for both these variants we will be able to test the relative importance



(a) The *monolithic* variant.



(b) The *compositional* variant.

Figure 5.2: Two variants based on the circular topology in Figure 5.1

Table 5.1: Running times of COCOS+CEDAR+DEODAR for the monolithic variant and the compositional variant. Each measurement is averaged over 100 runs.

	running time			
N	monolithic		compositional	
2	0.48	$\pm$0.01 s	0.47	$\pm$0.01 s
3	0.72	$\pm$0.02 s	0.82	$\pm$0.01 s
4	11.83	$\pm$1.39 s	1.07	$\pm$0.07 s
5	timeout	$>$ 5 minutes	1.33	$\pm$0.04 s
6	-		1.70	$\pm$0.10 s
7	-		1.90	$\pm$0.06 s
8	-		2.28	$\pm$0.14 s

of the granularity of the decomposition structure to our algorithms.

In Table 5.1 we list the running times of our COCOS compositional synthesis algorithm backed by CEDAR and DEODAR symbolic refinement algorithms. As can be seen, for increasing N the running times increase sharply for the monolithic variant whereas the algorithm handles the compositional variant much better. This constitutes a basic confirmation of the hypothesis.

Note that the results in table 5.2 have been obtained using fixed, optimized variable orderings for the underlying BDD engine. This makes sure that the influence of the underlying symbolic operations on the running time is as small as possible.

The results show that the presence of decomposition structure has a significant influence on what the tool can handle. In order to understand better what is driving the combinatorial explosion that seems to be going on in the monolithic case we need to look at the CEDAR phase of the compositional algorithm. For the monolithic case there is only a single invocation of CEDAR where most of the work for computing the controller is handled. In contrast, for the compositional case there are as many invocations as there are localities.

In Table 5.2 we list the cumulative number of counterexamples treated over all invocations of CEDAR. For the monolithic case there is always only a single invocation regardless of the value of N. For the compositional case we need to distinguish two situations. The common situation is $N > 2$, in this case the number of localities is $N + 1$ (N pairs + 1 top locality). The degenerate situation is $N = 2$, in this case the number of localities is 1 (there is only one pair which is also the top locality).

Table 5.2: Number of counterexamples treated by CEDAR for the monolithic variant and the compositional variant. Each measurement is averaged over 100 runs.

	counterexamples			
N	monolithic		compositional	
2	6.6	±0.8	6.5	±0.8
3	56.0	±4.8	47.3	±4.3
4	629.9	±34.9	61.3	±4.1
5	-		79.5	±5.4
6	-		97.7	±6.0
7	-		107.6	±6.0
8	-		120.5	±6.0

We see that for the monolithic case, again, the number of counterexamples rises sharply. In Figure 5.3 we see that for the compositional case, the number of counterexamples rises linearly. The reason for this is the fact that, in our example, safety violations can occur independently for each component. This means that the number of observations in the allow lattice that go to the unsafe state is exponential in N. In the compositional case the algorithm sustains this much better because it deals with these observations in a local context. In a local context the number of components involved is much smaller therefore the number of possible interactions is much smaller, and, as a result, the total number of counterexamples that has to be treated is much smaller.

5.4 Residual Constraints

In the previous section we saw how the compositional approach scales much better by deriving control constraints on a local level. Although this is already a nice result it leaves two important questions unanswered. The first question is about how well our compositional method handles *residual constraints*. The second question is more specific and is about how well our method handles *dependent constraints*.

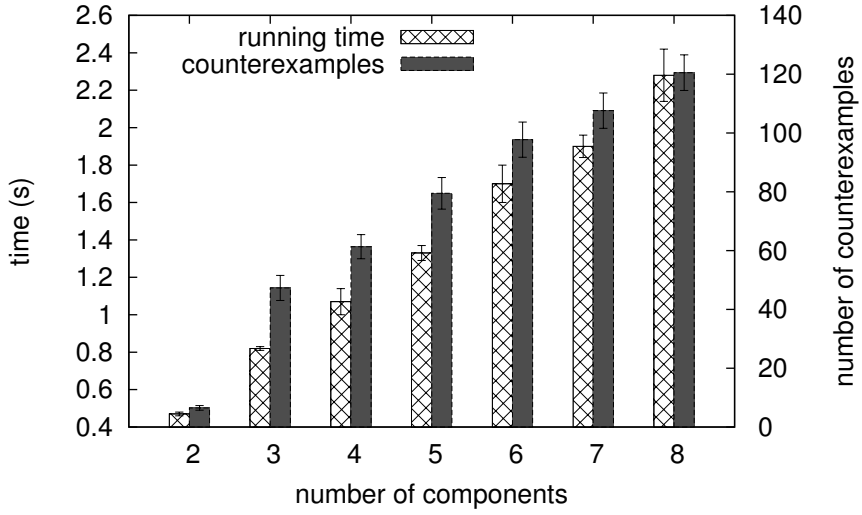


Figure 5.3: A combined graph containing the running time as well as the number of counterexamples treated by CEDAR for the compositional variant. This data is listed in Tables 5.2 and 5.1

5.4.1 Dependent and Independent Residual Constraints

We submit the concepts of residual constraints and dependent constraints as notions that should be interpreted relative to a particular point in the decomposition hierarchy. For our running example we conceptually position ourselves in between the *child localities* $\mathfrak{L}_1, \mathfrak{L}_2$, etc. and the *parent locality* $\mathfrak{L}_\top$. We can now reason about a constraint being derived at *child level* or a constraint being derived at *parent level*.

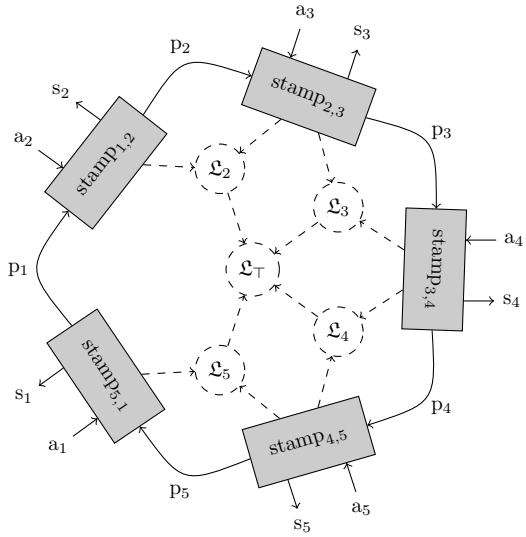
Residual constraints are constraints that cannot be derived at child level and hence must be derived at parent level. Dependent constraints are constraints that intrinsically involve all child localities. Dependent constraints will become residual constraints by definition. The converse is not true: it is also possible to have independent residual constraints in those cases where the modeler simply failed to provide enough granularity in the locality structure.

5.4.2 Gap Variant and Busy Variant

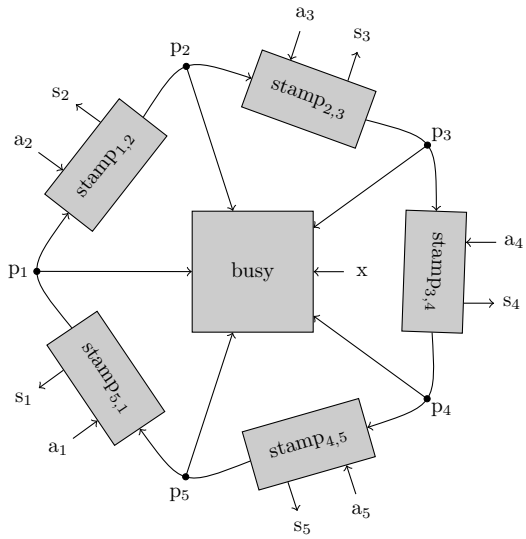
In Figure 5.4(a) we illustrate a third variant of our running example. This variant is the same as the compositional variant shown in Figure 5.2(b), except for the fact that we left out the first child locality $\mathfrak{L}_1$. We call this variant the *gap* variant. In Figure 5.4(b) we illustrate a fourth variant of our running example. This variant is the same as the compositional variant shown in Figure 5.2(b), except for the fact that we introduce an extra component "busy" that models the condition that there is at least one parcel on the belt being processed. We call this variant the *busy* variant.

The "busy" component models a "busy–light", informing any human operator that the plant is busy processing at least one parcel. Formally this is implemented by having the busy component impose a safety requirement over plant variables p_1, p_2, ..., p_N and control output x stating that x can only be set by the controller if there is at least one parcel on the belt. Note that this does *not* entail direct observability of p_1, p_2, ..., p_N: the controller will still have to rely on its observable propositions s_1, s_2, etc.

The interesting fact about the *busy* variant is that it features a *dependent residual* constraint that is intrinsically spanning all the child localities. It is not possible to decompose the busy constraint further as a product of independent constraints. In contrast, the interesting fact about the *gap* variant is that it features an *independent residual* constraint. The constraint concerning $\mathrm{stamp}_{1,2}$ will have to be derived at $\mathfrak{L}_\top$ because, in absence of $\mathfrak{L}_1$, $\mathfrak{L}_\top$ is the only locality that includes all the necessary control inputs and outputs for controlling



(a) *gap* variant



(b) *busy* variant

Figure 5.4: Two more variants of the circular parcel plant example.

$\text{stamp}_{1,2}$.

In Table 5.3 we list the running times on the gap variant and the busy variant, respectively. The algorithms have much more difficultly with the independent residual constraint than with the dependent residual constraint. This is surprising when we realize that using an approach like *partial order reduction* the situation would have been quite the opposite: partial order reduction relies on *independence* to reduce the amount of work the model checker has to perform. Here we rely on *dependence* to reduce the amount of work our synthesis procedure has to perform.

In order to understand why this is so we look at Table 5.4. Here we see that the number of counterexamples, although initially higher for the busy variant, is eventually overtaken by the gap variant.

There are two other important facts to notice from Figures 5.6 and 5.5. The first thing to notice is the fact that the number of counterexamples, in both cases, is rising exponentially in N. The second thing to notice is the fact that the running time is rising super–exponentially in N. In some sense this scaling behavior is to be expected for the majority of models. Note that the decision problem we are solving is of exponential complexity already for safety games of imperfect information that are represented as concrete state graphs. In addition, we are representing our game graphs symbolically which means that the concrete state graph would be, again, exponential in the size of N. The underlying symbolic representation can prevent this second exponential in some important cases, however, exponential blow–up of the representation can of course not be avoided in the general case.

For the gap variant it is quite obvious why the number of counterexamples is rising exponentially with N. This is simply a moderate version of the problem we already saw for the monolithic case. Since we removed a locality we see that the problem of interaction between independent observations crops up again, although, this time much less severely so than in the fully monolithic case.

For the busy variant it is not so obvious what is causing the number of counterexamples to rise exponentially with N. Upon inspection of the synthesis results it becomes clear however that the exponential is caused by the fact that each stamp is free to activate even when there is no parcel present. This means the number of observations that the controller can see when detecting the required condition ("there is no single parcel on the belt") is exponential in N. In some sense the number of observations is artificially blown up by the "frivolous" non–determinism of parcel stamps turning on for no apparent reason, we may say the model is *ill–disciplined*.

Table 5.3: Running times of COCOS+CEDAR+DEODAR for the gap variant and the busy variant. Each measurement is averaged over 10 runs.

	running time			
N	gap		busy	
2	0.48	±0.01 s	0.59	±0.02 s
3	0.69	±0.02 s	0.73	±0.02 s
4	1.08	±0.03 s	0.87	±0.01 s
5	5.17	±0.58 s	1.25	±0.04 s
6	33.90	±9.74 s	2.06	±0.07 s
7	timeout	> 5 minutes	4.56	±0.21 s
8			9.12	±0.25 s

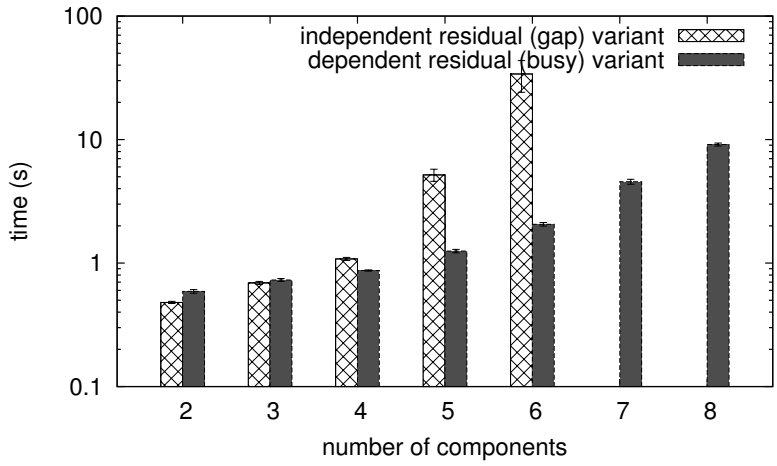


Figure 5.5: Number of counterexamples treated by CEDAR for the gap variant and the busy variant. Each measurement is averaged over 10 runs. This data is listed in Table 5.3.

Table 5.4: Number of counterexamples treated by CEDAR for the gap variant and the busy variant. Each measurement is averaged over 10 runs.

	counterexamples			
N	gap		busy	
2	6.9	±0.9	33.9	±3.0
3	39.8	±3.3	58.9	±4.7
4	84.1	±4.9	77.7	±5.7
5	234.3	±12.8	111.9	±6.6
6	1386.6	±258.2	161.0	±4.0
7			232.0	±4.0
8			387.3	±5.2

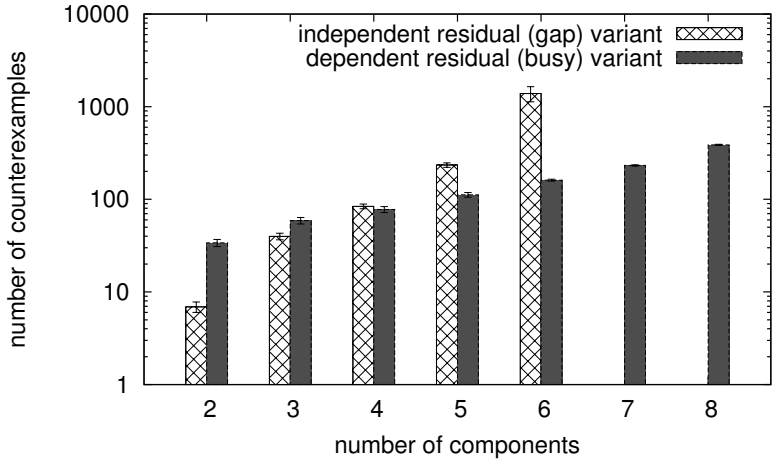


Figure 5.6: Number of counterexamples treated by CEDAR for the gap variant and the busy variant. Each measurement is averaged over 10 runs. This data is listed in Table 5.4.

5.4.3 Disciplined Variant

To demonstrate how the frivolous non–determinism, as discussed in the previous section, can be removed we introduce a final variant of the running example, called the *disciplined variant*. This variant is the same as Figure 5.4(b) except for the fact that each stamp is required to stay idle in case there is no parcel queueing.

In Table 5.5 we list the results. As can be seen in Figure 5.7 the scaling behavior is now linear again.

5.5 Conclusions

Based on the results in Section 5.3 we can answer the question of whether compositionality can have a beneficial effect on scaleability of the controller synthesis problem in the affirmative.

In particular, we have demonstrated that there exist natural examples where compositionality has the ability to reduce, what would otherwise be an exponential number of counterexamples at the global level, to a linear number of counterexamples spread over many localities.

On the other hand, in Section 5.4 we have shown how easy it is to break this linearity and to end up, again, with an exponential number of counterexamples and, consequently, a super–exponential running time. These results can be seen as limitative, or cautionary.

Our results, in some sense, can be compared to the results concerning the size of BDD representations of boolean functions. The size of a BDD representation of an arbitrary boolean function is exponential in general. However, for many practical examples this blow–up seems not to occur, or at least not to occur with such intensity that would make them unsuitable as a symbolic representation.

We can put together the results of Sections 5.3 and 5.4 and conclude with the qualitative relationships in Table 5.6. The table says that for deriving local constraints *independence* is *good* because it is exactly the independent constraints that can often be enforced on a local level. At the same time *dependence* is bad because it prevents constraints from being expressible and enforceable in a local context.

The table also says that for deriving residual constraints *independence* is *bad* because it can blow up the number of counterexamples. At the same time *dependence* is *good* because it can limit the number of counterexamples to a small number of very specific observations, this is clearly demonstrated by the

Table 5.5: Running time and number of counterexamples treated by CEDAR for the disciplined variant. Each measurement is averaged over 10 runs.

	disciplined variant			
N	running time		counterexamples	
2	0.77	±0.02	62.10	±1.37
3	1.01	±0.03	113.30	±2.49
4	1.16	±0.11	144.40	±3.77
5	1.44	±0.15	180.50	±4.25
6	1.85	±0.15	229.50	±3.77

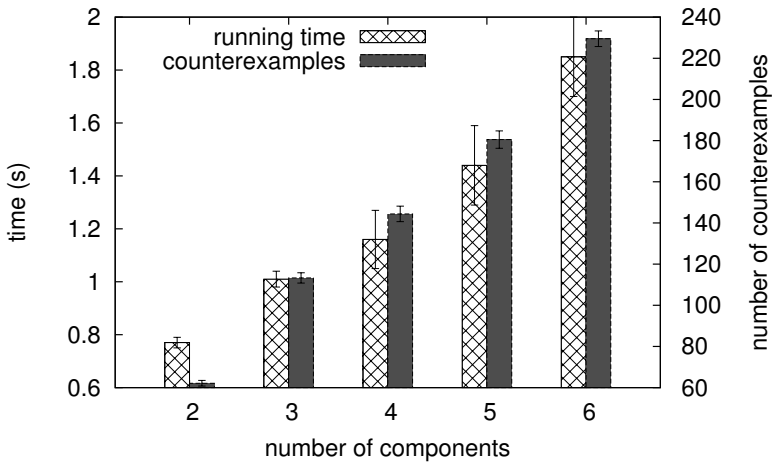


Figure 5.7: Running time and number of counterexamples treated by CEDAR for the disciplined variant. Each measurement is averaged over 10 runs. This data is listed in Table 5.5.

Table 5.6: Qualitative relationships between the effect of local versus residual control constraint derivation and the presence of control constraint dependencies.

	independence	dependence
local	good	bad
residual	bad	good

results we got on the disciplined variant in Section 5.4.

Summing up these insights we can conclude that for successfully applying a compositional method we need the dependency structure of the required constraints to, as best as possible, mirror the locality structure of the model, and vice versa.

Chapter 6

Conclusion

This concluding chapter is structured as follows. In Section 6.1 we give a summary of the results achieved in the thesis. In Section 6.2 we give some perspectives on future work.

6.1 Summary of Results

In this thesis we look at the synthesis problem for safety controllers under the assumption of partial observability with a distinct emphasis on compositionality. Our starting point is the working hypothesis that compositionality has a beneficial effect on scalability. Where scalability, in this context, should be understood as the ability of the synthesis algorithm to perform efficiently also with increasing problem size.

In order to reach a comprehensive solution, we divide the problem of compositional synthesis of safety controllers into two subproblems. The first subproblem is how to solve individual safety games, symbolically. The second subproblem is how to split a control problem of a single plant model into many safety games and, subsequently, how to compose the resulting symbolic strategies back into a single integrated controller.

For the first subproblem we propose a novel symbolic representation for control strategies and a novel symbolic game solving algorithm. We summarize this contribution separately in Section 6.1.1 below. For the second subproblem we propose a compositional framework and a novel compositional synthesis algorithm. We summarize this contribution separately in Section 6.1.2 below.

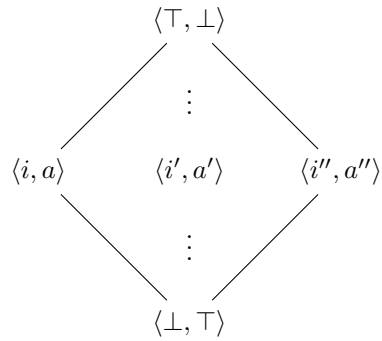


Figure 6.1: Illustration of the concept of an *allow lattice*.

In addition to the aforementioned theoretical contributions we also make a practical contribution in the form of a tool that implements the compositional synthesis algorithm. Using this tool we validate our solution and obtain a positive confirmation of the working hypothesis through our experimental results.

The experimental results that we obtain with the compositional synthesis algorithm are very encouraging. In particular we show how, for a natural example, it is possible to reduce what would otherwise be an exponential number of counterexamples to be treated at the global level to a linear number of counterexamples to be treated at the local level, spread over many localities. We show that this is true even if there are so called *residual constraints* that can only be inferred and applied at the global level due to dependencies spanning multiple localities

6.1.1 Safety Games

We all play games. For some games it is best to take risks, but for the type of games that we consider in this thesis it is best to play it safe. The games that we are interested in involve a *controller* and a *plant*, the objective is for the controller to keep the plant in a safe state at all times.

An important concept that we use in our solution approach is the concept of an *information set* as the set of states that the controller can know the plant to be in based on observations. This concept is crucial, especially because we are working under the assumption of partial observability. A second important concept is the concept of an *allow set* as the (largest) set of control outputs that

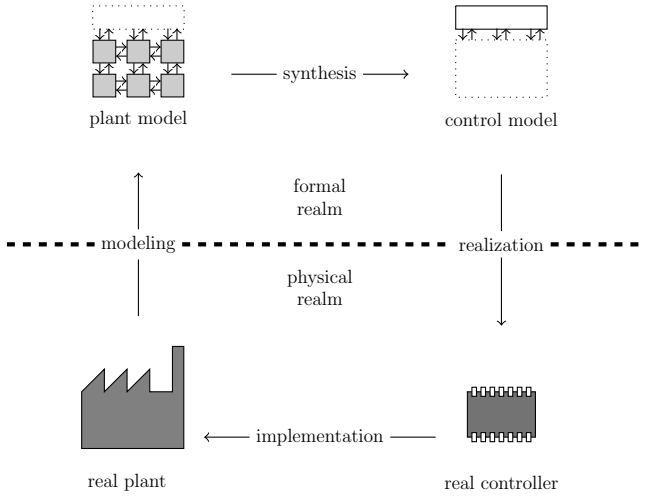


Figure 6.2: Division of responsibilities between modeling and synthesis.

the controller can safely choose given a particular information set.

In order to solve safety games symbolically we consider information sets and allow sets simultaneously in a single integrated lattice of info/allow pairs as illustrated in Figure 6.1. An important technique that becomes possible now that we have the representation in terms of an allow lattice, is to infer the relevant *control states* based purely on the relevant information sets in the allow lattice.

As a first example of a control state based on an info/allow pair, consider the top element in the allow lattice: $\langle \top, \bot \rangle$. The left–hand–side of this info/allow pair, $\top$, represents the weakest possible knowledge about the state of the plant, i.e.: the plant could be in *any state* (including unsafe states). The right–hand–side of this info/allow pair, $\bot$, represents the empty allow set, i.e.: the controller allows *nothing*. Hence, this info/allow pair can be paraphrased as saying: in absence of any knowledge about the state of plant, the controller should block and allow nothing.

As a second example of a control state based on an info/allow pair, consider the bottom element in the allow lattice: $\langle \bot, \top \rangle$. The left–hand–side of this info/allow pair, $\bot$, represents the inconsistent knowledge about the state of the plant, i.e.: there is no state that is consistent with the control observations.

The right–hand–side of this info/allow pair, $\top$, represents the largest possible allow set, i.e.: the controller allows *everything*. Hence, this info/allow pair can be paraphrased as saying: in the presence of inconsistent knowledge about the plant, the controller should give up and allow anything. This is completely analogous to the situation in most logics where from falsum anything follows.

In this context it is interesting to see how we are able to distinguish between two important, degenerate cases that may seem similar yet should not be confused. In particular: the top pair represents the *unsafe* control state, and the bottom pair represents the *inconsistent* control state. The unsafe control state is the state in which the controller ends up after observing something that it knows might lead the plant to an unsafe state. The inconsistent control state is the state in which the controller ends up after observing something that it assumed to be *impossible* based on the underlying plant model.

The latter two extrema can also be viewed as the formal reflection of the meta–theoretical distinction between what is the responsibility of the automated synthesis procedure and what is the responsibility of the *user* of the automated synthesis procedure. We will refer to the user of the automated synthesis procedure as the *modeler*, since it is a *model* of the plant that forms the principal input on which the automated synthesis procedure actually works. This is illustrated in Figure 6.2. Our automated synthesis procedure is able to guarantee that, after synthesis, the *unsafe* control state will never be reached. In contrast, it is the responsibility of the modeler to ensure that, when the synthesized controller is actually realized and implemented in a physical system, the *inconsistent* control state will never be reached.

6.1.2 Compositional Synthesis

One often hears the phrase "divide and conquer" being applied to solving problems of a technical nature. What this means is that breaking up a problem into smaller chunks makes the individual chunks more easily manageable, which in turn increases our chances of success. We could also apply this phrase to the type of control problems that we consider in this thesis.

It is rather unfortunate, and perhaps, telling, that the phrase "divide and conquer" originally stems from the latin phrase "divide et impera" in which form it rather refers to a politics of repression intended to prevent an enemy from rising up from amongst an empire. For solving problems of a technical nature, a more apt phrase would perhaps be "*integrate* and conquer" since making divisions is *easy*, solving small problems is also *easy*, yet integrating the sub–solutions of these sub–problems into an overarching super–solution is *hard*.

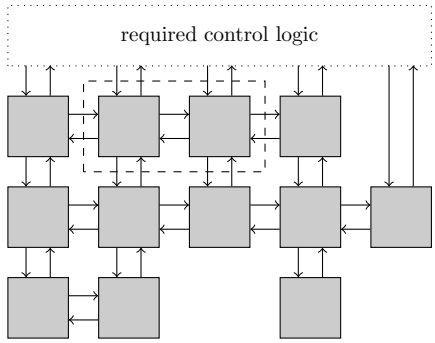


Figure 6.3: A plant model consisting of several interacting plant components and a *locality* (the subset of two plant components shown as a dashed rectangle).

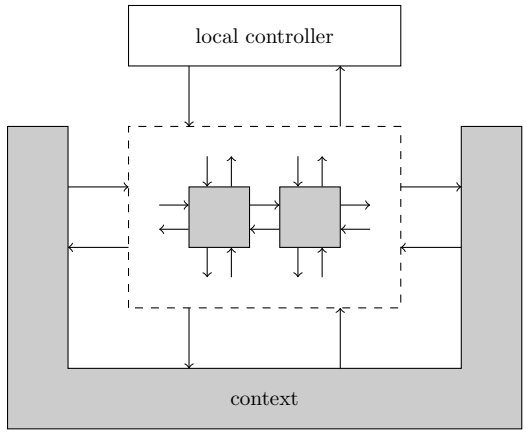


Figure 6.4: A *local controller* and a *context* for the locality shown in Figure 6.3.

Why is integration hard? Integration is, essentially, a form of *composition of sub–solutions*. Where integration often fails is when the choices embodied in two or more sub–solutions are in mutual conflict with each other. A way out of this that we propose in this thesis is to make the sub–solutions *conservative*, meaning they do not commit to any local choice unless it can be proven that this choice *must* be made also on a global level. The technical challenge in this solution approach then lies in how to prove *locally* that something is necessary *globally*. This is where the concepts of *locality*, *periphery* and *context* become important.

The concept of *locality* can be defined when we think about a *plant model* as consisting of several *plant components*. A locality is then simply a subset of these components. This is illustrated in Figure 6.3. Usually we would choose connected subsets, but essentially an arbitrary subset can form a locality. The concept of *periphery* is then defined simply as the complement of a locality so all the plant components that are *not* part of the locality. Finally a *context* is then a *simultaneous abstraction of the periphery and the control*. This is illustrated in Figure 6.4.

The crucial observation is that, by synthesizing a context, we are able to put a lowerbound on what safe behavior may be allowed at a global level. This then in turn allows us to put an upperbound on the behavior that must be allowed on a local level. The constraints that we derive in this way make up a *conservative local controller*.

A conservative local controller constrains the locality for which it was derived but it never *overconstrains* it, in the sense that it does not rule out any safe controller at the global level. We really put this concept to work using a compositional algorithm that starts out synthesizing local controllers for small localities and gradually expands the scope, eventually arriving at the locality that covers the plant model in its entirety. The huge advantage is that by the time the plant model is treated in its entirety many, if not all, unsafe behaviors are already suppressed by the previously derived local controllers.

6.2 Perspectives

In this section we give some perspectives on how to extend the present work.

A first thing to note is that the work in this thesis has a very discrete flavor to it. We work in a setting of propositions, states, locations and also winning conditions (safety versus unsafety) that are all *discrete*. It is widely recognized that for embedded systems the most natural models are often a combination

of discrete and continuous dynamics. Sometimes these models are called *hybrid models* precisely because of this combination. In this sense a logical extension of the work in this thesis would be to investigate if and under which assumptions it is possible to apply these techniques in a hybrid setting. In particular the challenge would be to incorporate results and techniques from *control theory*.

A similar perspective exists in extending the results towards *probabilistic systems*. There a unique challenge would be to deal with the fact that also the winning conditions are no longer discrete, but rather *gradual*. In this context one can think of a quality measure that says that the probability of reaching an unsafe state within a certain number of time steps would be below some small bound. One possible direction here would be to incorporate techniques from *optimization theory*.

Bibliography

Asarin, E., Bournez, O., Dang, T., Maler, O., Pnueli, A., 2000. Effective synthesis of switching controllers for linear systems. Proceedings of the IEEE 88 (7), 1011–1025.
URL http://dx.doi.org/10.1109/5.871306

Asarin, E., Maler, O., Pnueli, A., 1995. Symbolic controller synthesis for discrete and timed systems. In: Hybrid Systems II. Vol. 999 of LNCS. Springer-Verlag, pp. 1–20.
URL http://dx.doi.org/10.1007/3-540-60472-3_1

Basu, A., Bensalem, S., Peled, D., Sifakis, J., 2009. Priority scheduling of distributed systems based on model checking. In: Bouajjani, A., Maler, O. (Eds.), Computer Aided Verification. Vol. 5643 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 79–93.
URL http://dx.doi.org/10.1007/978-3-642-02658-4_10

Bensalem, S., Bozga, M., Graf, S., Peled, D., Quinton, S., 2010. Methods for knowledge based controlling of distributed systems. In: Bouajjani, A., Chin, W.-N. (Eds.), Automated Technology for Verification and Analysis. Vol. 6252 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 52–66.
URL http://dx.doi.org/10.1007/978-3-642-15643-4_6

Beyer, D., Henzinger, T. A., Singh, V., 2007. Algorithms for interface synthesis. In: Computer Aided Verification. Vol. 4590 of LNCS. Springer, pp. 4–19.
URL http://dx.doi.org/10.1007/978-3-540-73368-3_4

Buchi, J. R., Landweber, L. H., Apr. 1969. Solving sequential conditions by Finite-State strategies. Transactions of the American Mathematical Society 138, 295–311.
URL http://www.jstor.org/stable/1994916

Cassez, F., 2007. Efficient On-the-Fly algorithms for partially observable timed games. In: Formal Modeling and Analysis of Timed Systems. Vol. 4763 of LNCS. Springer, pp. 5–24.
URL http://dx.doi.org/10.1007/978-3-540-75454-1_3

Chakrabarti, A., de Alfaro, L., Henzinger, T. A., Mang, F. Y. C., 2002. Synchronous and bidirectional component interfaces. In: Computer Aided Verification. Vol. 2404 of LNCS. Springer, pp. 711–745.
URL http://dx.doi.org/10.1007/3-540-45657-0_34

Chatterjee, K., Doyen, L., Henzinger, T. A., Raskin, J., Sep. 2006. Algorithms for Omega-Regular games with imperfect information,. In: Computer Science Logic. Vol. 4207 of LNCS. Springer-Verlag, pp. 287–302.
URL http://dx.doi.org/10.1007/11874683_19

Chatterjee, K., Henzinger, T., Jobstmann, B., 2008. Environment assumptions for synthesis. In: Concurrency Theory. Vol. 5201 of LNCS. pp. 147–161.
URL http://dx.doi.org/10.1007/978-3-540-85361-9_14

Chatterjee, K., Henzinger, T. A., 2007. Assume-Guarantee synthesis. In: Tools and Algorithms for the Construction and Analysis of Systems. Vol. 4424 of LNCS. Springer, pp. 261–275.
URL http://dx.doi.org/10.1007/978-3-540-71209-1_21

Church, A., 1957. Application of recursive arithmetic to the problem of circuit synthesis. Summaries of the Summer Institute of Symbolic Logic, Cornell Univ., Ithaca, NY 1, 3–50.

Clarke, E. M., Emerson, E. A., 1982. Design and synthesis of synchronization skeletons using Branching-Time temporal logic. In: Logic of Programs, Workshop. Springer-Verlag, pp. 52–71.
URL http://portal.acm.org/citation.cfm?id=747438

de Alfaro, L., Henzinger, T. A., 2001. Interface automata. SIGSOFT Softw. Eng. Notes 26 (5), 109–120.
URL http://portal.acm.org/citation.cfm?doid=503271.503226

de Graauw, T., Helmich, F. P., Jul. 2001. Herschel-HIFI: The Heterodyne Instrument for the Far-Infrared. In: Pilbratt, G. L., Cernicharo, J., Heras, A. M., Prusti, T., Harris, R. (Eds.), The Promise of the Herschel Space Observatory. Vol. 460 of ESA Special Publication. p. 45.

de Graauw, T., Helmich, F. P., Phillips, T. G., Stutzki, J., Caux, E., Whyborn, N. D., Dieleman, P., Roelfsema, P. R., Aarts, H., Assendorp, R., et al., Jul 2010. The herschel - heterodyne instrument for the far-infrared (hifi). Astronomy and Astrophysics 518, L6.
URL http://dx.doi.org/10.1051/0004-6361/201014698

Doyen, L., Raskin, J.-F., 2010. Antichain algorithms for finite automata. In: Esparza, J., Majumdar, R. (Eds.), Tools and Algorithms for the Construction and Analysis of Systems. Vol. 6015 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 2–22.
URL http://dx.doi.org/10.1007/978-3-642-12002-2_2

Halpern, J., 1995. Reasoning about knowledge: A survey. In: Handbook of Logic in Artificial Intelligence and Logic Programming: Volume 4: Epistemic and Temporal Reasoning. Handbook of logic in artificial intelligence and logic programming. Oxford University Press, USA, pp. 1–34.
URL http://books.google.nl/books?id=goZ7MgEACAAJ

Halpern, J. Y., 2003. A computer scientist looks at game theory. Games and Economic Behavior 45 (1), 114 – 131, ¡ce:title¿First World Congress of the Game Theory Society¡/ce:title¿.
URL http://www.sciencedirect.com/science/article/pii/S0899825602005298

Hintikka, J., 1962. Knowledge and Belief. Ythaca, N.Y.: Cornell University Press.

Katz, G., Peled, D., Schewe, S., 2011. Synthesis of distributed control through knowledge accumulation. In: Gopalakrishnan, G., Qadeer, S. (Eds.), Computer Aided Verification. Vol. 6806 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 510–525.
URL http://dx.doi.org/10.1007/978-3-642-22110-1_41

Kaufmann, M., Wiese, R., 2002. Maintaining the mental map for circular drawings. In: Goodrich, M., Kobourov, S. (Eds.), Graph Drawing. Vol. 2528 of LNCS. Springer Berlin / Heidelberg, pp. 235–255, 10.1007/3-540-36151-0_2.
URL http://dx.doi.org/10.1007/3-540-36151-0_2

Kuijper, W., van de Pol, J., 2009a. Compositional control synthesis for partially observable systems. In: Bravetti, M., Zavattaro, G. (Eds.), Concurrency Theory. Vol. 5710 of LNCS. Springer Berlin Heidelberg, pp. 431–447.
URL http://dx.doi.org/10.1007/978-3-642-04081-8_29

Kuijper, W., van de Pol, J., 2009b. Computing weakest strategies for safety games of imperfect information. In: Tools and Algorithms for the Construction and Analysis of Systems. Vol. 5505 of LNCS. Springer Berlin Heidelberg, pp. 92–106.
URL http://dx.doi.org/10.1007/978-3-642-00768-2_10

Kupferman, O., Piterman, N., Vardi, M. Y., 2006. Safraless compositional synthesis. In: Computer Aided Verification. Vol. 4144 of LNCS. Springer, pp. 31–44.
URL http://dx.doi.org/10.1007/11817963_6

Kupferman, O., Vardi, M. Y., 2000. Synthesis with incomplete information. In: Advances in Temporal Logic. Vol. 16 of Applied Logic Series. Kluwer, pp. 109–127.

Lin, F., Wonham, W. M., 1990. Decentralized control and coordination of discrete-event systemswith partial observation. IEEE Transactions on automatic control 35 (12), 1330–1337.

Maler, O., Nickovic, D., Pnueli, A., 2007. On synthesizing controllers from Bounded-Response properties. In: Computer Aided Verification. Vol. 4590 of LNCS. Springer, pp. 95–107.
URL http://dx.doi.org/10.1007/978-3-540-73368-3_12

Manna, Z., Wolper, P., 1984. Synthesis of communicating processes from temporal logic specifications. ACM Transactions on Programming Languages and Systems (TOPLAS) 6 (1), 68–93.

Marincic, J., Mader, A. H., Wieringa, R. J., 2008. Classifying assumptions made during requirements verification of embedded systems. In: Requirements Engineering: Foundation for Software Quality, 14th International Working Conference, REFSQ 2008, Montpellier, France. Vol. 5025. Springer Verlag, London, pp. 141–146.

Pnueli, A., Rosner, R., 1989a. On the synthesis of a reactive module. In: Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. ACM, Austin, Texas, United States, pp. 179–190.
URL http://portal.acm.org/citation.cfm?id=75293

Pnueli, A., Rosner, R., 1989b. On the synthesis of an asynchronous reactive module. In: Automata, Languages and Programming. Vol. 372 of LNCS.

Springer, pp. 652–671.
URL http://dx.doi.org/10.1007/BFb0035790

Rabin, M. O., 1972. Automata on infinite objects and Church's problem. American Mathematical Society.

Ramadge, P. J., Wonham, W. M., 1989. The control of discrete event systems. Proceedings of the IEEE 77 (1), 81–98.
URL http://dx.doi.org/10.1109/5.21072

Ricker, S. L., Rudie, K., 2007. Knowledge is a terrible thing to waste: Using inference in discrete-event control problems. IEEE Trans. Automat. Contr. 52 (3), 428–441.

Six, J., Tollis, I., 1999. Circular drawings of biconnected graphs. In: Goodrich, M., McGeoch, C. (Eds.), Algorithm Engineering and Experimentation. Vol. 1619 of LNCS. Springer Berlin / Heidelberg, pp. 662–662, 10.1007/3-540-48518-X_4.
URL http://dx.doi.org/10.1007/3-540-48518-X_4

Thomas, W., 1995. On the synthesis of strategies in infinite games. In: Symposium on Theoretical Aspects of Computer Science, volume 900 of LNCS. Vol. 900. p. 12.

Thomas, W., 2009. Facets of synthesis: Revisiting church's problem. In: Foundations of Software Science and Computational Structures. Springer-Verlag New York Inc, p. 1.

Toyota, Feb 2010. Toyota announces voluntary recall on 2010 model-year prius to update abs software. Press Release.
URL http://www.toyota.com/recall/abs.html

Von Neumann, J., 1928. Zur theorie der gesellschaftsspiele. Mathematische Annalen 100, 295–320.
URL http://dx.doi.org/10.1007/BF01448847

Index

Titles in the IPA Dissertation Series since 2006

E. Dolstra. *The Purely Functional Software Deployment Model*. Faculty of Science, UU. 2006-01

R.J. Corin. *Analysis Models for Security Protocols*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-02

P.R.A. Verbaan. *The Computational Complexity of Evolving Systems*. Faculty of Science, UU. 2006-03

K.L. Man and R.R.H. Schiffelers. *Formal Specification and Analysis of Hybrid Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2006-04

M. Kyas. *Verifying OCL Specifications of UML Models: Tool Support and Compositionality*. Faculty of Mathematics and Natural Sciences, UL. 2006-05

M. Hendriks. *Model Checking Timed Automata - Techniques and Applications*. Faculty of Science, Mathematics and Computer Science, RU. 2006-06

J. Ketema. *Böhm-Like Trees for Rewriting*. Faculty of Sciences, VUA. 2006-07

C.-B. Breunesse. *On JML: topics in tool-assisted verification of JML programs*. Faculty of Science, Mathematics and Computer Science, RU. 2006-08

B. Markvoort. *Towards Hybrid Molecular Simulations*. Faculty of Biomedical Engineering, TU/e. 2006-09

S.G.R. Nijssen. *Mining Structured Data*. Faculty of Mathematics and Natural Sciences, UL. 2006-10

G. Russello. *Separation and Adaptation of Concerns in a Shared Data Space*. Faculty of Mathematics and Computer Science, TU/e. 2006-11

L. Cheung. *Reconciling Nondeterministic and Probabilistic Choices*. Faculty of Science, Mathematics and Computer Science, RU. 2006-12

B. Badban. *Verification techniques for Extensions of Equality Logic*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2006-13

A.J. Mooij. *Constructive formal methods and protocol standardization*. Faculty of Mathematics and Computer Science, TU/e. 2006-14

T. Krilavicius. *Hybrid Techniques for Hybrid Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-15

M.E. Warnier. *Language Based Security for Java and JML*. Faculty of Science, Mathematics and Computer Science, RU. 2006-16

V. Sundramoorthy. *At Home In Service Discovery.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-17

B. Gebremichael. *Expressivity of Timed Automata Models.* Faculty of Science, Mathematics and Computer Science, RU. 2006-18

L.C.M. van Gool. *Formalising Interface Specifications.* Faculty of Mathematics and Computer Science, TU/e. 2006-19

C.J.F. Cremers. *Scyther - Semantics and Verification of Security Protocols.* Faculty of Mathematics and Computer Science, TU/e. 2006-20

J.V. Guillen Scholten. *Mobile Channels for Exogenous Coordination of Distributed Systems: Semantics, Implementation and Composition.* Faculty of Mathematics and Natural Sciences, UL. 2006-21

H.A. de Jong. *Flexible Heterogeneous Software Systems.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-01

N.K. Kavaldjiev. *A run-time reconfigurable Network-on-Chip for streaming DSP applications.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-02

M. van Veelen. *Considerations on Modeling for Early Detection of Abnormalities in Locally Autonomous Distributed Systems.* Faculty of Mathematics and Computing Sciences, RUG. 2007-03

T.D. Vu. *Semantics and Applications of Process and Program Algebra.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-04

L. Brandán Briones. *Theories for Model-based Testing: Real-time and Coverage.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-05

I. Loeb. *Natural Deduction: Sharing by Presentation.* Faculty of Science, Mathematics and Computer Science, RU. 2007-06

M.W.A. Streppel. *Multifunctional Geometric Data Structures.* Faculty of Mathematics and Computer Science, TU/e. 2007-07

N. Trčka. *Silent Steps in Transition Systems and Markov Chains.* Faculty of Mathematics and Computer Science, TU/e. 2007-08

R. Brinkman. *Searching in encrypted data.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-09

A. van Weelden. *Putting types to good use.* Faculty of Science, Mathematics and Computer Science, RU. 2007-10

J.A.R. Noppen. *Imperfect Information in Software Development Processes.* Faculty of Electrical Engi-

neering, Mathematics & Computer Science, UT. 2007-11

R. Boumen. *Integration and Test plans for Complex Manufacturing Systems.* Faculty of Mechanical Engineering, TU/e. 2007-12

A.J. Wijs. *What to do Next?: Analysing and Optimising System Behaviour in Time.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2007-13

C.F.J. Lange. *Assessing and Improving the Quality of Modeling: A Series of Empirical Studies about the UML.* Faculty of Mathematics and Computer Science, TU/e. 2007-14

T. van der Storm. *Component-based Configuration, Integration and Delivery.* Faculty of Natural Sciences, Mathematics, and Computer Science,UvA. 2007-15

B.S. Graaf. *Model-Driven Evolution of Software Architectures.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2007-16

A.H.J. Mathijssen. *Logical Calculi for Reasoning with Binding.* Faculty of Mathematics and Computer Science, TU/e. 2007-17

D. Jarnikov. *QoS framework for Video Streaming in Home Networks.* Faculty of Mathematics and Computer Science, TU/e. 2007-18

M. A. Abam. *New Data Structures and Algorithms for Mobile Data.* Faculty of Mathematics and Computer Science, TU/e. 2007-19

W. Pieters. *La Volonté Machinale: Understanding the Electronic Voting Controversy.* Faculty of Science, Mathematics and Computer Science, RU. 2008-01

A.L. de Groot. *Practical Automaton Proofs in PVS.* Faculty of Science, Mathematics and Computer Science, RU. 2008-02

M. Bruntink. *Renovation of Idiomatic Crosscutting Concerns in Embedded Systems.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-03

A.M. Marin. *An Integrated System to Manage Crosscutting Concerns in Source Code.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-04

N.C.W.M. Braspenning. *Model-based Integration and Testing of High-tech Multi-disciplinary Systems.* Faculty of Mechanical Engineering, TU/e. 2008-05

M. Bravenboer. *Exercises in Free Syntax: Syntax Definition, Parsing, and Assimilation of Language Conglomerates.* Faculty of Science, UU. 2008-06

M. Torabi Dashti. *Keeping Fairness Alive: Design and Formal Verification of Optimistic Fair Exchange*

Protocols. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2008-07

I.S.M. de Jong. *Integration and Test Strategies for Complex Manufacturing Machines*. Faculty of Mechanical Engineering, TU/e. 2008-08

I. Hasuo. *Tracing Anonymity with Coalgebras*. Faculty of Science, Mathematics and Computer Science, RU. 2008-09

L.G.W.A. Cleophas. *Tree Algorithms: Two Taxonomies and a Toolkit*. Faculty of Mathematics and Computer Science, TU/e. 2008-10

I.S. Zapreev. *Model Checking Markov Chains: Techniques and Tools*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-11

M. Farshi. *A Theoretical and Experimental Study of Geometric Networks*. Faculty of Mathematics and Computer Science, TU/e. 2008-12

G. Gulesir. *Evolvable Behavior Specifications Using Context-Sensitive Wildcards*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-13

F.D. Garcia. *Formal and Computational Cryptography: Protocols, Hashes and Commitments*. Faculty of Science, Mathematics and Computer Science, RU. 2008-14

P. E. A. Dürr. *Resource-based Verification for Robust Composition of Aspects*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-15

E.M. Bortnik. *Formal Methods in Support of SMC Design*. Faculty of Mechanical Engineering, TU/e. 2008-16

R.H. Mak. *Design and Performance Analysis of Data-Independent Stream Processing Systems*. Faculty of Mathematics and Computer Science, TU/e. 2008-17

M. van der Horst. *Scalable Block Processing Algorithms*. Faculty of Mathematics and Computer Science, TU/e. 2008-18

C.M. Gray. *Algorithms for Fat Objects: Decompositions and Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-19

J.R. Calamé. *Testing Reactive Systems with Data - Enumerative Methods and Constraint Solving*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-20

E. Mumford. *Drawing Graphs for Cartographic Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-21

E.H. de Graaf. *Mining Semi-structured Data, Theoretical and Experimental Aspects of Pattern Evaluation*. Faculty of Mathematics and Natural Sciences, UL. 2008-22

R. Brijder. *Models of Natural Computation: Gene Assembly and Membrane Systems.* Faculty of Mathematics and Natural Sciences, UL. 2008-23

A. Koprowski. *Termination of Rewriting and Its Certification.* Faculty of Mathematics and Computer Science, TU/e. 2008-24

U. Khadim. *Process Algebras for Hybrid Systems: Comparison and Development.* Faculty of Mathematics and Computer Science, TU/e. 2008-25

J. Markovski. *Real and Stochastic Time in Process Algebras for Performance Evaluation.* Faculty of Mathematics and Computer Science, TU/e. 2008-26

H. Kastenberg. *Graph-Based Software Specification and Verification.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-27

I.R. Buhan. *Cryptographic Keys from Noisy Data Theory and Applications.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-28

R.S. Marin-Perianu. *Wireless Sensor Networks in Motion: Clustering Algorithms for Service Discovery and Provisioning.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-29

M.H.G. Verhoef. *Modeling and Validating Distributed Embedded Real-Time Control Systems.* Faculty of Science, Mathematics and Computer Science, RU. 2009-01

M. de Mol. *Reasoning about Functional Programs: Sparkle, a proof assistant for Clean.* Faculty of Science, Mathematics and Computer Science, RU. 2009-02

M. Lormans. *Managing Requirements Evolution.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-03

M.P.W.J. van Osch. *Automated Model-based Testing of Hybrid Systems.* Faculty of Mathematics and Computer Science, TU/e. 2009-04

H. Sozer. *Architecting Fault-Tolerant Software Systems.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-05

M.J. van Weerdenburg. *Efficient Rewriting Techniques.* Faculty of Mathematics and Computer Science, TU/e. 2009-06

H.H. Hansen. *Coalgebraic Modelling: Applications in Automata Theory and Modal Logic.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-07

A. Mesbah. *Analysis and Testing of Ajax-based Single-page Web Applications.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-08

A.L. Rodriguez Yakushev. *Towards Getting Generic Programming*

Ready for Prime Time. Faculty of Science, UU. 2009-9

K.R. Olmos Joffré. *Strategies for Context Sensitive Program Transformation.* Faculty of Science, UU. 2009-10

J.A.G.M. van den Berg. *Reasoning about Java programs in PVS using JML.* Faculty of Science, Mathematics and Computer Science, RU. 2009-11

M.G. Khatib. *MEMS-Based Storage Devices. Integration in Energy-Constrained Mobile Systems.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-12

S.G.M. Cornelissen. *Evaluating Dynamic Analysis Techniques for Program Comprehension.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-13

D. Bolzoni. *Revisiting Anomaly-based Network Intrusion Detection Systems.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-14

H.L. Jonker. *Security Matters: Privacy in Voting and Fairness in Digital Exchange.* Faculty of Mathematics and Computer Science, TU/e. 2009-15

M.R. Czenko. *TuLiP - Reshaping Trust Management.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-16

T. Chen. *Clocks, Dice and Processes.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-17

C. Kaliszyk. *Correctness and Availability: Building Computer Algebra on top of Proof Assistants and making Proof Assistants available over the Web.* Faculty of Science, Mathematics and Computer Science, RU. 2009-18

R.S.S. O'Connor. *Incompleteness & Completeness: Formalizing Logic and Analysis in Type Theory.* Faculty of Science, Mathematics and Computer Science, RU. 2009-19

B. Ploeger. *Improved Verification Methods for Concurrent Systems.* Faculty of Mathematics and Computer Science, TU/e. 2009-20

T. Han. *Diagnosis, Synthesis and Analysis of Probabilistic Models.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-21

R. Li. *Mixed-Integer Evolution Strategies for Parameter Optimization and Their Applications to Medical Image Analysis.* Faculty of Mathematics and Natural Sciences, UL. 2009-22

J.H.P. Kwisthout. *The Computational Complexity of Probabilistic Networks.* Faculty of Science, UU. 2009-23

T.K. Cocx. *Algorithmic Tools for Data-Oriented Law Enforcement*. Faculty of Mathematics and Natural Sciences, UL. 2009-24

A.I. Baars. *Embedded Compilers*. Faculty of Science, UU. 2009-25

M.A.C. Dekker. *Flexible Access Control for Dynamic Collaborative Environments*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-26

J.F.J. Laros. *Metrics and Visualisation for Crime Analysis and Genomics*. Faculty of Mathematics and Natural Sciences, UL. 2009-27

C.J. Boogerd. *Focusing Automatic Code Inspections*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2010-01

M.R. Neuhäußer. *Model Checking Nondeterministic and Randomly Timed Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-02

J. Endrullis. *Termination and Productivity*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-03

T. Staijen. *Graph-Based Specification and Verification for Aspect-Oriented Languages*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-04

Y. Wang. *Epistemic Modelling and Protocol Dynamics*. Faculty of Science, UvA. 2010-05

J.K. Berendsen. *Abstraction, Prices and Probability in Model Checking Timed Automata*. Faculty of Science, Mathematics and Computer Science, RU. 2010-06

A. Nugroho. *The Effects of UML Modeling on the Quality of Software*. Faculty of Mathematics and Natural Sciences, UL. 2010-07

A. Silva. *Kleene Coalgebra*. Faculty of Science, Mathematics and Computer Science, RU. 2010-08

J.S. de Bruin. *Service-Oriented Discovery of Knowledge - Foundations, Implementations and Applications*. Faculty of Mathematics and Natural Sciences, UL. 2010-09

D. Costa. *Formal Models for Component Connectors*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-10

M.M. Jaghoori. *Time at Your Service: Schedulability Analysis of Real-Time and Distributed Services*. Faculty of Mathematics and Natural Sciences, UL. 2010-11

R. Bakhshi. *Gossiping Models: Formal Analysis of Epidemic Protocols*. Faculty of Sciences, Department of Computer Science, VUA. 2011-01

B.J. Arnoldus. *An Illumination of the Template Enigma: Software Code Generation with Templates*. Faculty of Mathematics and Computer Science, TU/e. 2011-02

E. Zambon. *Towards Optimal IT Availability Planning: Methods and Tools.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-03

L. Astefanoaei. *An Executable Theory of Multi-Agent Systems Refinement.* Faculty of Mathematics and Natural Sciences, UL. 2011-04

J. Proença. *Synchronous coordination of distributed components.* Faculty of Mathematics and Natural Sciences, UL. 2011-05

A. Moralı. *IT Architecture-Based Confidentiality Risk Assessment in Networks of Organizations.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-06

M. van der Bijl. *On changing models in Model-Based Testing.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-07

C. Krause. *Reconfigurable Component Connectors.* Faculty of Mathematics and Natural Sciences, UL. 2011-08

M.E. Andrés. *Quantitative Analysis of Information Leakage in Probabilistic and Nondeterministic Systems.* Faculty of Science, Mathematics and Computer Science, RU. 2011-09

M. Atif. *Formal Modeling and Verification of Distributed Failure Detectors.* Faculty of Mathematics and Computer Science, TU/e. 2011-10

P.J.A. van Tilburg. *From Computability to Executability – A process-theoretic view on automata theory.* Faculty of Mathematics and Computer Science, TU/e. 2011-11

Z. Protic. *Configuration management for models: Generic methods for model comparison and model co-evolution.* Faculty of Mathematics and Computer Science, TU/e. 2011-12

S. Georgievska. *Probability and Hiding in Concurrent Processes.* Faculty of Mathematics and Computer Science, TU/e. 2011-13

S. Malakuti. *Event Composition Model: Achieving Naturalness in Runtime Enforcement.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-14

M. Raffelsieper. *Cell Libraries and Verification.* Faculty of Mathematics and Computer Science, TU/e. 2011-15

C.P. Tsirogiannis. *Analysis of Flow and Visibility on Triangulated Terrains.* Faculty of Mathematics and Computer Science, TU/e. 2011-16

Y.-J. Moon. *Stochastic Models for Quality of Service of Component Connectors.* Faculty of Mathematics and Natural Sciences, UL. 2011-17

R. Middelkoop. *Capturing and Exploiting Abstract Views of States in OO Verification.* Faculty of

Mathematics and Computer Science, TU/e. 2011-18

M.F. van Amstel. *Assessing and Improving the Quality of Model Transformations.* Faculty of Mathematics and Computer Science, TU/e. 2011-19

A.N. Tamalet. *Towards Correct Programs in Practice.* Faculty of Science, Mathematics and Computer Science, RU. 2011-20

H.J.S. Basten. *Ambiguity Detection for Programming Language Grammars.* Faculty of Science, UvA. 2011-21

M. Izadi. *Model Checking of Component Connectors.* Faculty of Mathematics and Natural Sciences, UL. 2011-22

L.C.L. Kats. *Building Blocks for Language Workbenches.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2011-23

S. Kemper. *Modelling and Analysis of Real-Time Coordination Patterns.* Faculty of Mathematics and Natural Sciences, UL. 2011-24

J. Wang. *Spiking Neural P Systems.* Faculty of Mathematics and Natural Sciences, UL. 2011-25

A. Khosravi. *Optimal Geometric Data Structures.* Faculty of Mathematics and Computer Science, TU/e. 2012-01

A. Middelkoop. *Inference of Program Properties with Attribute Grammars, Revisited.* Faculty of Science, UU. 2012-02

Z. Hemel. *Methods and Techniques for the Design and Implementation of Domain-Specific Languages.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-03

T. Dimkov. *Alignment of Organizational Security Policies: Theory and Practice.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-04

S. Sedghi. *Towards Provably Secure Efficiently Searchable Encryption.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-05

F. Heidarian Dehkordi. *Studies on Verification of Wireless Sensor Networks and Abstraction Learning for System Inference.* Faculty of Science, Mathematics and Computer Science, RU. 2012-06

K. Verbeek. *Algorithms for Cartographic Visualization.* Faculty of Mathematics and Computer Science, TU/e. 2012-07

D.E. Nadales Agut. *A Compositional Interchange Format for Hybrid Systems: Design and Implementation.* Faculty of Mechanical Engineering, TU/e. 2012-08

H. Rahmani. *Analysis of Protein-Protein Interaction Networks by Means of Annotated Graph Mining Algorithms.* Faculty of Mathematics and Natural Sciences, UL. 2012-09

S.D. Vermolen. *Software Language Evolution.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-10

L.J.P. Engelen. *From Napkin Sketches to Reliable Software.* Faculty of Mathematics and Computer Science, TU/e. 2012-11

F.P.M. Stappers. *Bridging Formal Models – An Engineering Perspective.* Faculty of Mathematics and Computer Science, TU/e. 2012-12

W. Heijstek. *Software Architecture Design in Global and Model-Centric Software Development.* Faculty of Mathematics and Natural Sciences, UL. 2012-13

C. Kop. *Higher Order Termination.* Faculty of Sciences, Department of Computer Science, VUA. 2012-14

A. Osaiweran. *Formal Development of Control Software in the Medical Systems Domain.* Faculty of Mathematics and Computer Science, TU/e. 2012-15

W. Kuijper. *Compositional Synthesis of Safety Controllers.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-16