Automated Theorem Proving for General Game Playing

Dissertation

zur Erlangung des akademischen Grades
Doktor rerum naturalium (Dr. rer. nat.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von

Diplom-Informatiker Sebastian Haufe

geboren am 01. 05. 1982 in Hoyerswerda

Betreuender Hochschullehrer und Erstgutachter	Prof. Dr. rer. nat. habil. Michael Thielscher School of Computer Science and Engineering The University of New South Wales, Sydney
Zweitgutachter	Prof. Dr. rer. nat. habil. Torsten Schaub Institut für Informatik Universität Potsdam
Tag der Einreichung	24. 11. 2011
Tag der Verteidigung	22. 06. 2012

Abstract

While automated game playing systems like Deep Blue perform excellent within their domain, handling a different game or even a slight change of rules is impossible without intervention of the programmer. Considered a great challenge for Artificial Intelligence, General Game Playing is concerned with the development of techniques that enable computer programs to play arbitrary, possibly unknown n-player games given nothing but the game rules in a tailor-made description language. A key to success in this endeavour is the ability to reliably extract hidden game-specific features from a given game description automatically. An informed general game player can efficiently play a game by exploiting structural game properties to choose the currently most appropriate algorithm, to construct a suited heuristic, or to apply techniques that reduce the search space. In addition, an automated method for property extraction can provide valuable assistance for the discovery of specification bugs during game design by providing information about the mechanics of the currently specified game description. The recent extension of the description language to games with incomplete information and elements of chance further induces the need for the detection of game properties involving player knowledge in several stages of the game.

In this thesis, we develop a formal proof method for the automatic acquisition of rich game-specific invariance properties. To this end, we first introduce a simple yet expressive property description language to address knowledge-free game properties which may involve arbitrary finite sequences of successive game states. We specify a semantic based on state transition systems over the Game Description Language, and develop a provably correct formal theory which allows to show the validity of game properties with respect to their semantic across all reachable game states. Our proof theory does not require to visit every single reachable state. Instead, it applies an induction principle on the game rules based on the generation of answer set programs, allowing to apply any off-the-shelf answer set solver to practically verify invariance properties even in complex games whose state space cannot totally be explored. To account for the recent extension of the description language to games with incomplete information and elements of chance, we correctly extend our induction method to properties involving player knowledge. With an extensive evaluation we show its practical applicability even in complex games.

Acknowledgements

I am deeply grateful to my excellent advisor Michael Thielscher for his many fruitful ideas and discussions, his constant encouragements and substantial contributions to writing papers, and his continuous availability for cordially answering questions and providing advice. I further want to thank our whole group for a welcoming and friendly atmosphere and especially my colleague Stephan Schiffel, who has always been there to troubleshoot technical problems, and who provided countless invaluable suggestions and discussions over the years and after intensively proofreading an earlier version of this thesis. I am also grateful to Torsten Schaub for accepting to be my second reviewer, and to Uwe Petersohn for accepting to be my “Fachreferent”.

Special thanks go to my beloved wife Sophie Haufe who had the strength to bear with me in difficult times and who enabled to keep me writing by taking care of so many other issues. I am also thankful to Florian Stenger who agreed to proofread the thesis and whose determined struggling through the hardly understandable passages of an earlier version generated a multitude of helpful comments.

Contents

Chapter 1

Introduction

Autonomous Computer Systems have become an integral part in everyday life, being entrusted with a wide range of tasks such as the surveillance of good production in factories, the automated steering of vehicles, the efficient scheduling of process cycles, or the trading at stock markets. All these scenarios have a common structure: they involve one or more participants which manipulate the current state by performing certain actions with the goal to eventually maximise their outcome. Several participants may compete against each other, persue common interests, or not be of mutual influence at all. This general structure can be referred to as a *game*. Games have been utilised to model scenarios from a wide range of sciences such as mathematics, economics, biology and philosophy, allowing to benefit from insights and results in the well-established field of *Game Theory* which is concerned with the formal analysis of games.

In recent years, many game-playing systems have grown to show convincing performance in practice. Especially in the field of classic board games, super-human performance can often be achieved, and some board games are even completely solved [Sch00]. For example, IBM's Deep Blue system [CHH02] defeated the reigning world champion Garry Kasparov in chess in 1997, and perfect play in Checkers was shown to lead to a draw in 2007 [SBB$^+$07]. A common approach to efficiently playing games automatically is the incorporation of knowledge concerning the specific domain as well as human expert moves via huge databases. Consequently, the intelligence of these programs mainly originates from the programmer rather than the system itself. Furthermore, these systems are hard-wired to one specific game and hence cannot directly be used to play different or even newly created games. In a world which requires increasing dynamics and flexibility, Artificial Intelligence therefore encounters a challenge with growing interest: the development of systems which are able to play and reason about *arbitrary* games. This is the idea of *General Game Playing.*

1.1 General Game Playing

The term General Game Playing (GGP) first appeared in connection with a program that was able, in principle, to play arbitrary chess-like board games [Pit68]. Although two further approaches followed some decades later [Pel93, KP97], wide interest in this research area first spread with the launch of an international competition for General Game Playing in 2005 [GLP05, Thi11b]. Following the style of the first competition for single-agent planning in 1998 [McD00], the special-purpose *Game Description Language* (GDL) [LHH$^+$06] has been developed to provide a unified framework for the description of complete-information games and hence promoted the development and exchange of new methods and ideas and their direct evaluation in a competitive environment. Its recent extension to incomplete-information games [Thi10] was shown to be capable of formulating arbitrary games in the sense of Game Theory [Thi11a]. The GDL provides a compact way to describe the rules of a game based on a structure similar to the declarative programming language Prolog, allowing to deduce game-specific information—such as the initial state, the legal moves, and the goal conditions—by the straightforward application of existing standard Prolog inference mechanisms.

Using these mechanisms, an agent which exhaustively evaluates the game tree to find the current best move can easily be implemented. However, the time-restricted setting of the competition allows exhaustive search in a few very simple games only and is not possible in more complex games like Chess at all. Successful players hence need to utilise other evaluation techniques as well. Monte Carlo Tree Search [KS06, Cou07] as a form of selective blind search has been successfully applied in this endeavour [FB11, KSS11, KE11, MC11, MSWS11]. As it estimates potentially valuable successor states by performing random playouts to terminal states, it does not rely on any structural analysis of the game. Recent developments however indicate that further progress in this direction is possible only with the incorporation of knowledge about the game at hand [FB11, KSS11]. A whole body of work confirms this trend, identifying structural game knowledge to be of major assistance to a general game player for tasks such as the following.

Game Classification The general structure of the GDL allows to describe a wide range of diverse problems, not all of which might effectively be tackled with the same technique. For example, Minimax search with α-β pruning [RN03] is only applicable to two-player zero-sum games with alternating moves. The classification of a game, as well as information concerning which of the players are teammates or opponents, enables a player to choose the most efficient algorithm [KDS06, Sch11].

Search Space Reduction As the search space grows exponentially with respect to search depth, exploiting game specific structures can lead to a major increase of efficiency. Detected symmetries allow to search structurally similar parts of the game tree only once [Clu07, Sch10]. Identified independent subgames can be considered separately [CSMG09, ZST09] in order to cut down the branching factor of the search tree. Player payoffs that monotonically increase during game play allow to neglect subtrees rooted in states whose payoffs are smaller than the currently best found.

Heuristic Construction Structural Knowledge can be exploited for the evaluation of states, the most common identified structures are mentioned in the following [Clu07, Kai07, KDS06, MS11, MT09, ST07]. Game boards, i.e. cells which are connected according to some adjacency relation, provide valuable information concerning pieces such as rooks, crosses, or counting tokens. These can in turn be utilised for distance estimates to desired (partial) goal states and estimates on the value of states by their current quantity or diversity of potential moves. Furthermore, knowledge about structural persistence, such as markers that remain placed until the end of the game, can be exploited to speed up and improve the state evaluation function [CSMG09, Sch11] during game play.

1.2 Automated Theorem Proving for GGP

As the GDL provides the bare rules of a game only, game-specific knowledge which helps to achieve better game play has to be extracted by an automated general game player fully automatically. To this end, successful general game-playing systems perform random game simulations to test the validity of certain properties, and rely on their informed guess in case no violation could be detected. Consequently, the players run the risk of serious drawbacks in case property violations occur in parts of the game tree that have not been visited. E.g., a wrong game classification may suggest an unsuited technique that considerably reduces performance or yields wrong results; a search space reduction may cut off an easy win or an avoidable loss from consideration; and a heuristic based on erroneous beliefs may result in the expansion of potentially useless states. A method which *formally* verifies game properties can hence be of valuable assistance to a general game-playing system. Even beyond this setting, a verification system can be utilised by the game designer to reliably obtain insights on a newly created game.

Structural Validity While the syntactic structure of a game description can easily be checked, the description may nevertheless deviate from the exact intended game semantically. This scenario occurred during the 2006 AAAI Competition, and caused quite some disturbance among both participants and organisers (we have a further look on this incident in Section 3.1). A verification system can assist to prove desired structural properties such as the uniqueness of cell content, the possibility of each player to perform a legal move in each non-terminal state, the termination of the game with consistent payoff information after finitely many steps, and the possibility to win for each of the players.

Player-Specific Knowledge The recent extension of the GDL to incomplete-information games poses further challenges to the game designer. For example, the information of players may not be sufficient to know which moves they can apply, whether the game has terminated in the current game state, or which payoffs apply in terminal states.

A system which is able to formally verify this kind of knowledge can additionally be used by the player to exploit information about what an opponent knows in several game states, for example by choosing actions that reveal as few hints about the current game state to the opponent as possible.

Problem Analysis Since each game in the general sense of Game Theory is expressible in GDL, a wide range of problems from various sciences can be described with this formalism. Once a newly designed game description for a specific problem matches all desired structural criteria, a formal verification system can provide valuable insights on complex hidden specifics of the problem and hence actively foster its understanding.

The first approach that addresses the formal verification of game properties in the field of General Game Playing has been given in [vdHRW07b]. It allows the specification of rich properties over the GDL for complete-information games. A recent follow-up considers player-specific knowledge in properties which involve exactly one state [RT11a]. Since property verification amounts to a full state space analysis in both approaches, they are however unsuited for players and designers in the General Game Playing setting alike, which mainly concentrates on games that are far beyond reach of reasonably-timed exhaustive search in practice.

A further approach has been able to circumvent this problem by applying the proof method of induction [ST09], achieving practicable timings for the verification of properties even in complex games. Properties do not concern player-specific knowledge and are restricted to consider exactly one state, and successfully verified properties amount to be state invariants which hold in each reachable state of the game.

In this thesis, we build on the suggested induction method for the verification of knowledge-free state invariants to establish the first comprehensive framework which allows to formulate, and efficiently verify in practice, a wide range of properties over the GDL. Verifiable properties are invariants of reachable game states and may involve

- *local temporal reference*, e.g. to formulate and verify how specific information in a current game state influences arbitrary finitely reachable future game states, and
- *player-specific knowledge*, e.g. to formulate and verify what is known to a player in certain game states in the setting of incomplete-information games, possibly interconnected with local temporal reference.

To this end, we proceed as follows.

1.3 Structure of the Thesis

- In Chapter 2, we introduce the foundations that we will need throughout the thesis. Especially, we introduce *Answer Set Programming* [GL88, Gel08] as an alternative semantic to Prolog programs and present some additional language constructs. We then give a short account on the formalisation of games in Game Theory, which is followed by the presentation of the syntax of the Game Description Language and a multiagent semantic based on state transition systems.
- In Chapter 3, we develop syntax and semantic of a simple yet expressive property description language to address knowledge-free properties with local temporal reference, combining elements from the GDL and Temporal Logic. We then provide a proof theory which is able to verify these properties with respect to a given game description using Answer Set Programming. We formally prove the soundness of our method, and show that completeness can be obtained under additional requirements. We further show that the method can be adjusted to completely solve single player games and develop an extension which further increases proof efficiency by considering multiple properties at once.
- In Chapter 4, we extend our language to address the formulation of player-specific knowledge. We provide a semantic based on possible worlds which satisfies the

S5 properties (see, e.g., [FHMV95]), and develop an alternative semantic based on interconnected state sequences which is provably equivalent with respect to properties that do not involve formulations about what players do *not* know. We then generalise our formal proof method and the respective soundness and completeness results from Chapter 3 to this extension. Chapters 3 and 4 constitute the main theoretical contributions of this thesis.

- In Chapter 5, we sketch our implementation of the proof method on top of the successful general game player Fluxplayer [ST07, Sch11], integrating tools from the state-of-the-art answer set solving collection Potassco [GKK$^+$11a]. We then report on a variety of experiments which show that our method is practically applicable even in complex games.

- In Chapter 6, we discuss related approaches to proving properties over the GDL, including all approaches that have been mentioned in Section 1.2.

- In Chapter 7, we summarise our main contributions and provide pointers to future work.

Chapter 2

Preliminaries

In this chapter we summarise notions and results which are known from the literature and will be needed throughout this work. It includes and extends passages of own published work (we give a detailed overview of included own published material in Section 7.1). The chapter is divided into two parts. In Section 2.1, we introduce logic programs, which form the basis for the formulation of games in the setting of General Game Playing. We provide additional language constructs from the field of Answer Set Programming that will be needed to establish our formal verification method for game properties. In Section 2.2, we first give a formal characterisation of games by means of Game Theory. We then introduce the Game Description Language as a representation formalism for games, provide a characterisation of several game properties, and sketch an architecture which allows automated agents to play arbitrary games.

2.1 Answer Set Programming

Logic Programs with negation are a common formalism to represent and reason about knowledge, and the advantages and drawbacks of several interpretations, including procedural semantics via SLDNF resolution and declarative semantics based on first-order logic, have extensively been studied in the literature (see, e.g., [ABW87, Llo87, Apt97] for a comprehensive introduction). Distinguished from these approaches, the *answer set semantics* introduced in [GL88] has received remarkable attention, as its generate-and-test strategy allows to circumvent some of the drawbacks of previously introduced semantics. Further work, including extensions to incorporate classical negation and disjunctive information [GL91], has led this line of research to identify as the new paradigm of *Answer Set Programming* [MT99, Nie99, LTT99]. It now provides a variety of concepts beyond classical logic programs, including weight constraints [NSS99] and nested logical connectives [LTT99], and has hence become popular as a convenient modelling formalism. Thorough introductions to this active field of research can be found in [Gel08] and [FL05a].

In the following we shortly present the syntax of logic programs as in [Llo87] and [Apt97] (Section 2.1.1) and restate the original answer set semantics introduced in [GL88] (Section 2.1.2). These concepts will be needed to define syntax and semantics of the Game Description Language. We then introduce a special class of weight constraints [NSS99] (Section 2.1.3) which is needed to establish our proof method.

2.1.1 Syntax of Logic Programs

For the definition of a logic program, we follow the style of [ST10].

Definition 2.1 (Logic Program).

- *A* term *is either a variable, or a function symbol with* n *terms as arguments. In case* $n = 0$*, a term is also called* constant*.*
- *An* atom *is a predicate symbol with terms as arguments.*
- *A* literal *is an atom* a*, also called* positive *literal, or its negation* not a*, also called* negative *literal.*
- *A* (simple) clause *is of the form* h `:-` $l_1, \ldots, l_n$. $(n \geq 0)$*, to be understood as implication of the form "if* l_1 *and* $\ldots$ *and* l_n*, then* h *", where*
  - h *is an atom, called* head *of the (simple) clause, and*
  - *each* l_i *is a literal,* $l_1, \ldots, l_n$ *is called* body *and each* l_i *is called* body literal *of the (simple) clause.*

  When $n = 0$*, a (simple) clause is written in the form* h . *and called* fact*.*
- *A* logic program *is a finite set of (simple) clauses.*

An expression *is a term, a literal, or a (simple) clause. An expression is called* ground *if it does not contain variables.* ■

Throughout this work we use the Prolog convention of denoting variables by uppercase letters while predicate and function symbols start with a lowercase letter. We sometimes abbreviate atoms $p(t_1, \ldots, t_k)$ by $p(\vec{t})$ and terms $f(t_1, \ldots, t_k)$ by $f(\vec{t})$.

A logic program G uniquely determines a (usually infinite) set of terms Σ_{var} as follows. By f/k we denote a function symbol f and its arity k, and define a set F such that $f/k \in F$ if and only if term $f(t_1, \ldots, t_k)$ occurs in G for some terms $t_1, \ldots, t_k$. The set of terms Σ_{var} over G is defined as the smallest set such that:

- If X is a variable, then $X \in \Sigma_{var}$.
- If $f/0 \in F$, then $f \in \Sigma_{var}$.
- If $f/k \in F$ and $t_1, \ldots, t_k \in \Sigma_{var}$ $(k \geq 1)$, then $f(t_1, \ldots, t_k) \in \Sigma_{var}$.

Finally, the set of ground terms Σ over G comprises all terms from Σ_{var} which do not contain variables. Similar to the set F of function symbols with arities, we define a set P of predicate symbols with arities which contains all elements p/k for which an atom $p(t_1, \ldots, t_k)$ occurs in G. The set of atoms over G is then given as the smallest set that contains $p(t_1, \ldots, t_k)$ if $p/k \in P$ and $t_1, \ldots, t_k \in \Sigma_{var}$. The set of all ground atoms over G comprises all elements of the previous set which do not contain variables.

A *substitution* is used to replace variables with terms in expressions. More formally, a substitution θ is a finite set $\{X_1/t_1, \ldots, X_n/t_n\}$ such that $X_1, \ldots, X_n$ are variables and $t_1, \ldots, t_n$ are (possibly non-ground) terms from Σ_{var}. An *instance of an expression* E (wrt. θ) is obtained by simultaneously replacing each occurrence of variable X_i in E by term t_i for all $1 \leq i \leq n$. The (possibly infinite) set of ground instances of clauses of a logic program G is obtained by applying all substitutions to clauses in G which replace variables with arbitrary ground terms from Σ. Two expressions E and E' *unify* if there is a substitution θ such that the instances of E and E' wrt. θ coincide. We will now introduce two subclasses of logic programs with interesting properties (mentioned below) that will be of importance later on. Their definition requires the notion of a set *partition*, which is defined as a collection of subsets $S_1, \ldots, S_n$ of a set S such that the S_i are pairwise disjoint and their union yields S. A partition will also be written as $S = S_1 \mathbin{\dot{\cup}} \ldots \mathbin{\dot{\cup}} S_n$ in the following.

Definition 2.2 (Stratified and Allowed Logic Program)**.**

- *A logic program G is called* stratified *[ABW87] if there is a partition $G = G_1 \mathbin{\dot{\cup}} \ldots \mathbin{\dot{\cup}} G_n$ such that, for each $1 \leq i \leq n$, the following are true:*
  - *For each positive body literal a which occurs in a clause in G_i, all clauses of G with head a are contained within $\bigcup_{j \leq i} G_j$.*
  - *For each negative body literal* not a *which occurs in a clause in G_i, all clauses of G with head a are contained within $\bigcup_{j<i} G_j$.*
- *A logic program is called* allowed *[LT86] if, in each of its clauses c, all variables occurring in c occur also in a positive body literal in c.* ■

The following definition, first given in [ABW87], is helpful for determining whether a logic program is stratified, and provides a notion for the structural dependency of program atoms. We follow the style of [ST10].

Definition 2.3 (Dependency Graph)**.** *The* dependency graph *for a logic program G is a directed, labelled graph whose nodes are the predicate symbols that occur in G and where there is a* positive *edge $p \xrightarrow{+} q$ if G contains a clause $p(\vec{s})$ `:-` $\ldots, q(\vec{t}), \ldots$ *.*, *and a* negative *edge $p \xrightarrow{-} q$ if G contains a clause $p(\vec{s})$ `:-` $\ldots$, not $q(\vec{t}), \ldots$ *. We say* p depends on q *in G if there is a path from p to q in the dependency graph of G.* ■

A logic program G is stratified if and only if there are no cycles involving a negative edge in the dependency graph of G [ABW87]. This classification is useful as stratified programs will play an important role in the definition of the Game Description Language. We close this section with an example.

Example 2.4 (Stratified and Allowed Logic Program)*.* Consider the logic program G:

```
a(1).
a(2).
b(2).
p(X) :- a(X), not b(X).
q :- r.
r :- q, not p(1).
```

G is easily seen to be allowed. Moreover, it is stratified since (among others) there is a partition $G = G_1 \mathbin{\dot{\cup}} G_2 \mathbin{\dot{\cup}} G_3$ that matches the requirements of Definition 2.2, namely such that $G_1 = \{$ `a(1).`, `a(2).`, `b(2).`$\}$, $G_2 = \{$ `p(X) :- a(X), not b(X).`$\}$, and where G_3 contains all remaining clauses. ■

2.1.2 Answer Set Semantics for Logic Programs

Logic programs can be understood as first-order logic formulas, treating variables universally quantified in front of each clause and conjunctively connecting the clauses. Hence they qualify for a declarative model-theoretic semantics. We will not specify this semantics here, but focus on a subclass of models widely known as *Herbrand Models* which will be important for the answer set semantics.

Definition 2.5 (Herbrand Model). *Let G be a logic program and let $\mathcal{A}$ be a set of ground atoms over G. Then $\mathcal{A}$ is a* model *for ground atom a iff $a \in \mathcal{A}$, and $\mathcal{A}$ is a model for ground negative literal* not *a iff $a \notin \mathcal{A}$. Finally, $\mathcal{A}$ is a* Herbrand Model *for G iff, for all ground instances c of clauses in G, $\mathcal{A}$ being a model for all body literals of c implies that $\mathcal{A}$ is a model for the head of c.* ■

Logic programs without negation are well known to provide a unique *minimal* Herbrand Model $\mathcal{A}$, meaning that dropping any subset of $\mathcal{A}$ results in a set which is not a Herbrand Model itself. In the case of negation, this is not generally true: a logic program can have several, a unique, or no minimal Herbrand Model. However, logic programs that are stratified (cf. Definition 2.2) are known to admit a unique *standard model* [ABW87], a specific Herbrand Model that is considered the "most intuitive" interpretation of a logic program. It is minimal (although there could be other minimal Herbrand Models as well) and *supported*, meaning that every atom in the standard model is an instance of the head of a clause of the logic program.

Having defined Herbrand Models, we are now prepared for the answer set semantics of logic programs as given in [GL88]. It provides a subclass of Herbrand Models with interesting properties that will be summarised below.

Definition 2.6 (Answer Set Semantics). *Given a logic program G and a set of ground atoms $\mathcal{A}$ over G, let $G^{\mathcal{A}}$ be the set of negation-free implications h* `:-` *$a_1, \ldots, a_k$*`.`*, obtained by taking all ground instances of clauses in G and*

- *deleting all clauses with a negative body literal* not *a such that $a \in \mathcal{A}$,*
- *deleting all negative body literals from the remaining clauses.*

Then $\mathcal{A}$ is an answer set *for G if and only if $\mathcal{A}$ is the unique minimal Herbrand Model for $G^{\mathcal{A}}$.* ■

Each answer set of a logic program is minimal [GL88] and supported [Lif96]. It is easy to see that for logic programs without negation, there is a unique answer set which coincides with the unique minimal Herbrand Model. Including negation, a similar correspondence can be established for some logic programs: if a logic program is stratified, then it admits a unique answer set which coincides with the unique standard model [GL88]. In the following, for a stratified logic program G, we write $G \vdash p$ to denote that ground atom p is contained in this unique answer set or, equivalently, that p is contained in the unique standard model for G.

Example 2.7 (Herbrand Model and Answer Set). Reconsider the program G from Example 2.4 together with the set of ground atoms $\mathcal{A} = \{$ `a(1)`, `a(2)`, `b(2)`, `p(1)`$\}$. Then $G^{\mathcal{A}}$ is

```
a(1).
a(2).
b(2).
p(1) :- a(1).
q :- r.
```

and $\mathcal{A}$ is the unique minimal Herbrand Model for $G^{\mathcal{A}}$. Hence, $\mathcal{A}$ is an answer set for G, and it is unique since G is stratified (as shown in Example 2.4). ■

2.1.3 Extension of Logic Programs by Weight Atoms

For the direct formulation of cardinalities and costs in combinatorial problems, the concept of weight constraints has been included, among others, to the syntax and answer set semantics of logic programs [NSS99]. They allow to provide a lower and an upper bound to the sum of previously assigned weights for literals which occur in an answer set. We will only make use of the special case of weight atoms[1], where the lower bound equals or is greater than zero, the mentioned literals are positive (and hence atoms), and their weights are 1. In the following, we formally include weight atoms to the syntax of logic programs and adapt the semantics from [NSS99] to our setting.

Definition 2.8 (Answer Set Program)**.**

- *A* weight atom *is of the form* $l\{a_1,\ldots,a_k\}u$*, where* $k,l\in\mathbb{N}$*,* $u\in\mathbb{N}\cup\{\infty\}$*,* $0\leq l\leq u$*, and* $a_1,\ldots,a_k$ *are ground atoms.*
- *A* clause *is of the form* h `:-` $l_1,\ldots,l_n$`.`*, where*
  - h *is either an atom or a weight atom, and*
  - *each* l_i *is either a literal or a weight atom.*
- *An* answer set program *is a finite set of clauses.*

If a weight atom $l\{a_1,\ldots,a_k\}u$ *occurs in the head of a clause, each* $a_i\in\{a_1,\ldots,a_k\}$ *is considered a clause head. The notions of fact, body, expression and groundness extend to answer set programs as expected. As a shorthand, we may refer to an answer set program as a program.* ■

For a weight atom $l\{a_1,\ldots,a_k\}u$, both l and u can be omitted, which is tacitly interpreted as $l=0$ and $u=\infty$, respectively. The semantics of this additional construct will be given as an extension of Definitions 2.5 and 2.6 in the following. Intuitively, the clause reduction from Definition 2.6 removes all clauses which contain some negative body literal which does not agree with the answer set candidate, and all negative body literals in the remaining clauses are omitted. Concerning weight atoms, the upper bound u can be considered as negative information, formulating that there are *no* $u+1$ atoms out of the specified set which hold in the answer set candidate (the lower bound l, at the contrary, states the positive information that there *are* l instances). Seen that way, the following definition is a natural extension

[1]Weight atoms are called "cardinality constraints" in [NSS99].

of Definition 2.6. It removes all clauses with unagreed negative information (that is, clauses that contain a weight atom with non-matching upper bound), and removes the negative part (the upper bound) of weight atoms in the remaining clauses. Weight atoms in the head of a clause, however, need a special treatment.

Definition 2.9 (Extension of the Answer Set Semantics). *Let G be an answer set program and $\mathcal{A}$ be a set of ground atoms. $\mathcal{A}$ is a model for weight atom $l\{a_1, \ldots, a_k\}u$ iff $\mathcal{A}$ contains at least l and at most u different elements of $\{a_1, \ldots, a_k\}$. $\mathcal{A}$ is a model for G iff, for all ground instances c of clauses in G, $\mathcal{A}$ being a model for all body literals and body weight atoms of c implies that $\mathcal{A}$ is a model for the head atom or head weight atom of c.*

Now let $G^{\mathcal{A}}$ be the set of negation-free implications h :- $l_1, \ldots, l_k$. obtained by performing the reductions from Definition 2.6 on all ground instances of clauses in G and, additionally,

- *deleting all clauses with a weight atom in the body such that answer set candidate $\mathcal{A}$ exceeds its upper bound,*
- *deleting all upper bounds from body weight atoms in the remaining clauses, and*
- *replacing each remaining clause of the form $l\{a_1, \ldots, a_k\}u$:- $l_1, \ldots, l_n$., by a set of clauses a_i :- $l_1, \ldots, l_n$., for each $a_i \in \{a_1, \ldots, a_k\} \cap \mathcal{A}$.*

The reduced set of clauses $G^{\mathcal{A}}$ admits a unique minimal model [NSS99], and $\mathcal{A}$ is an answer set for G if and only if it coincides with this unique minimal model and, additionally, is a model for G. ∎

Also in this extended setting, each answer set is known to be supported [Fer05]. However, it is not necessarily minimal. This is easily seen with the answer set program $\{0\{a\}1\,.\}$, which has the two answer sets $\{\}$ and $\{a\}$. The additional requirement that $\mathcal{A}$ is a model for G is necessary for the correct treatment of weight atoms in clause heads. E.g., consider the answer set program $G = \{0\{a\}0\,.\}$ and the set $\mathcal{A} = \{a\}$, which clearly should not be an answer set for G. However, $G^{\mathcal{A}} = \{a\,.\}$ and hence coincides with $\mathcal{A}$, which yields that $\mathcal{A}$ is an answer set when omitting the requirement.

To express that literals $l_1, \ldots, l_n$ should not be true together, we will further make use of the construct

$$:\!-\ l_1, \ldots, l_n\,.,$$

which we will call a *constraint*. Several semantics have been suggested, we decide in favour of the semantics from [NSS99] which considers constraints as abbreviation for $1\{\}$:- $l_1, \ldots, l_n$. Note that the answer sets of $G \cup \{$:- $l_1, \ldots, l_n\,.\}$ are exactly the answer sets of G *except* for those that do satisfy all of $l_1, \ldots, l_n$.

In [LT94], a Splitting Theorem for logic programs (including disjunction in clause heads) has been introduced. It has been extended to nested formulas in [EL04] and further been generalised in [Fer05]. We adapt the last-mentioned version to our restricted setting of weight atoms, using the fact that weight atoms can equivalently be expressed as nested formulas [FL05b]. The theorem will play an important role in the proofs of several properties of our verification method. It basically states that an answer set program can be split into a basic part and an additional part whenever the basic part does not involve atoms of the additional part, and that the splitting preserves (a reduced version of) the answer sets of the basic part.

Theorem 2.10 (Splitting Theorem)**.** *Let P and Q be two programs such that atoms in P are not unifiable with heads from clauses in Q. Then $\mathcal{A}_{P\cup Q}$ is an answer set for $P \cup Q$ iff the set $\mathcal{A}_P$, obtained from $\mathcal{A}_{P\cup Q}$ by restricting to instances of atoms from P, is an answer set for P and $\mathcal{A}_{P\cup Q}$ is an answer set for $(\bigcup_{a\in\mathcal{A}_P}\{a\,.\}) \cup Q$.* $\square$

Note that, for two programs P_1 and P_2 which relate to a program Q as P relates to Q in Theorem 2.10, the following is implied: if $\mathcal{A}_{P_1\cup Q}$ is an answer set for $P_1 \cup Q$ and its restriction $\mathcal{A}_{P_1}$ to instances of atoms from P_1 is also an answer set for P_2, then $\mathcal{A}_{P_1\cup Q}$ is an answer set for $P_2 \cup Q$ as well. This correspondence will prove valuable for the consideration of relations between answer set programs P_1 and P_2 in the presence of additional contexts Q. For a better understanding of the introduced notions, we give another example.

Example 2.11 (Answer Set and Splitting Theorem)*.* Consider the following program G:

```
2 {a(1), a(2), a(3)} 2.
b(2).
p(X) :- a(X), not b(X).
q :- 1 {r, q} 1.
r :- q, 0 {p(1), q} 0.
```

Further consider the set of ground atoms $\mathcal{A} = \{$ `a(1)`, `a(2)`, `b(2)`, `p(1)` $\}$. Then $G^{\mathcal{A}}$ is the following program.

```
a(1).
a(2).
b(2).
p(1) :- a(1).
p(3) :- a(3).
q :- 1 {r, q}.
```

$\mathcal{A}$ is the unique minimal model for $G^{\mathcal{A}}$. Since, additionally, $\mathcal{A}$ satisfies G, $\mathcal{A}$ is also an answer set for G. However, it is not unique: the sets $\mathcal{A}' = \{$ `a(2)`, `a(3)`, `b(2)`, `p(3)` $\}$ and $\mathcal{A}'' = \{$ `a(1)`, `a(3)`, `b(2)`, `p(1)`, `p(3)` $\}$ are answer sets for G as well.

One possibility to split G according to Theorem 2.10 is the following: Let P be the clauses from the first two lines of G, and let Q be the remaining clauses. The head atoms `a(1)`, `a(2)`, `a(3)`, `b(2)` of P are not unifiable with the head atoms `p(X)`, `q`, `r` from Q. Since the above-mentioned set $\mathcal{A} = \{$ `a(1)`, `a(2)`, `b(2)`, `p(1)` $\}$ is an answer set for $G = P \cup Q$, direction $\Rightarrow$ of Theorem 2.10 implies that the set $\mathcal{A}_P = \{$ `a(1)`, `a(2)`, `b(2)` $\}$ is an answer set for P, and that $\mathcal{A}$ is an answer set for

```
a(1).
a(2).
b(2).
p(X) :- a(X), not b(X).
q :- 1 {r, q} 1.
r :- q, 0 {p(1), q} 0.
```

$\mathcal{A}_P = \{$ `a(1)`, `a(2)`, `b(2)`$\}$ is also an answer set for the following program P'.

```
a(2).
a(1) :- a(2).
b(2) :- a(1), a(2).
```

Hence, by the remark following Theorem 2.10, $\mathcal{A}$ is also an answer set for the program $G' = P' \cup Q$. ■

2.2 General Game Playing

In this section, we introduce to the field of General Game Playing. To this end, we formally define a game by means of Game Theory. We then introduce a compact specification language for games, the Game Description Language, and give a transition semantics based on answer sets for logic programs. We conclude this section with a list of properties that are mentioned throughout this work and shortly summarise how communication is organised between a control instance and players during game play.

2.2.1 Formalisation of Games

In the field of Game Theory (see, e.g., [Ras07] and [Osb04] for an introduction), two equivalent formal notions have been established, the *strategic form* and the *extensive form* of a game. We restate the latter definition, following the style of [Thi11a] which is based on [Ras07].

Definition 2.12 (Game). *An* n-player game *($n \geq 1$) consists of:*

1. *a finite tree composed of* states*, called* game tree*, where the root is called* initial *state and the leaves are called* terminal *states;*

2. *a function which maps each non-terminal state to a player or the additional pseudo player "Nature", indicating that the state belongs to that player or Nature;*

3. *a function which maps each terminal state to a real-numbered payoff for each player excluding Nature;*

4. *a probability measure for each state S of the game tree which belongs to Nature, assigning a probability to each successor of S (the probability with which Nature "chooses" that successor);*

5. *for each player r (excluding Nature) and each set $\mathcal{H}_i$ of all nodes of the game tree at depth i, a partition called* information partition $\mathcal{H}_i = \mathcal{H}_{i,1} \dot\cup \ldots \dot\cup \mathcal{H}_{i,n}$ *into* information sets *such that:*

   (a) *all children of a node which belongs to r are in different information sets of $\mathcal{H}_{i+1}$; and*

   (b) *if $i > 0$, then for all information sets $\mathcal{H}_{i,j} \subseteq \mathcal{H}_i$ the predecessors of all nodes in $\mathcal{H}_{i,j}$ are in the same information set of $\mathcal{H}_{i-1}$.* ■

The (finite) set of *actions*[2] of each player r is implicitly given by the nodes which belong to r (item 2) in the game tree (item 1) and win, loss, as well as intermediate results can be specified with the payoff function (item 3). In addition to the n players, a pseudo player "Nature" is defined which, according to item 4, allows to model elements of chance like the outcome of rolling dice or tossing coins. Item 5 allows to specify incomplete information[3] of a player, e.g. when he is not informed about his opponents cards in a card game. This is achieved by an information partition for each player r and each stage of the game (corresponding to a depth i of the game tree) such that each information set exactly contains the states which are indistinguishable for r at that particular stage. Each partition has to be such that r knows his own moves (item 5a) and such that r can distinguish two states S and S' whenever he is able to distinguish their predecessors (item 5b).

According to Definition 2.12, players always have to take turns making their moves. However, simultaneous moves can be simulated by sequences of single-player moves, not letting the players know about the outcome of previously taken moves in that sequence by specifying appropriate information partitions [Ras07]. We conclude this section with an example that will be needed and further developed throughout the remainder of this work.

Example 2.13 (The Game Quarto). "Quarto" [Kis03] is played on a 4×4 game board and involves two players. It uses 16 different pieces, one for each combination of four characterising binary attributes (e.g. short/tall, black/white, etc.). Initially, the board is empty and the first player starts by selecting one of the pieces for placement by the second player. The players take turns repeating this procedure with yet unplaced pieces until either no more pieces are available (in which case the game ends in a draw) or one player wins by having completed a horizontal, vertical or diagonal line of four pieces with at least one shared attribute (e.g., they are all white).

Quarto can be modelled as a 2-player game according to Definition 2.12: The game tree is indicated in Figure 2.1, where the initial state (the root node) belongs to the first player, its successors to the second player, their successors again to the second player, etc. Additional information concerning the last-moving player can easily be added to each state, enabling the definition of a payoff function which correctly accounts for win, draw and loss of a player e.g. by assigning payoff 100, 50, and 0, respectively. Neither chance nor incomplete information are involved in Quarto, hence the probability measure is negligible, and each information partition of all states $\mathcal{H}_i$ at depth i divides into singleton sets $\mathcal{H}_{i,j} = \{S\}$ for all states S at depth i. ■

2.2.2 Syntax of the Game Description Language

Definition 2.12 provides a general and natural characterisation of a game, but most games have huge state spaces and therefore cannot be directly specified via this model in practice. This motivated the development of the general Game Description Language

[2]We will use the terms *action* and *move* interchangeably in this work.

[3]As pointed out in [Thi10], there is a clash of terminology: in Artificial Intelligence, an agent who is not fully aware of the current state of the environment is said to have *incomplete* information, whereas Game Theory uses the term *imperfect* information. We decide to adopt the terminology from [Thi10]. A more formal classification of when we understand a game to be of incomplete information is deferred to Section 2.2.4.

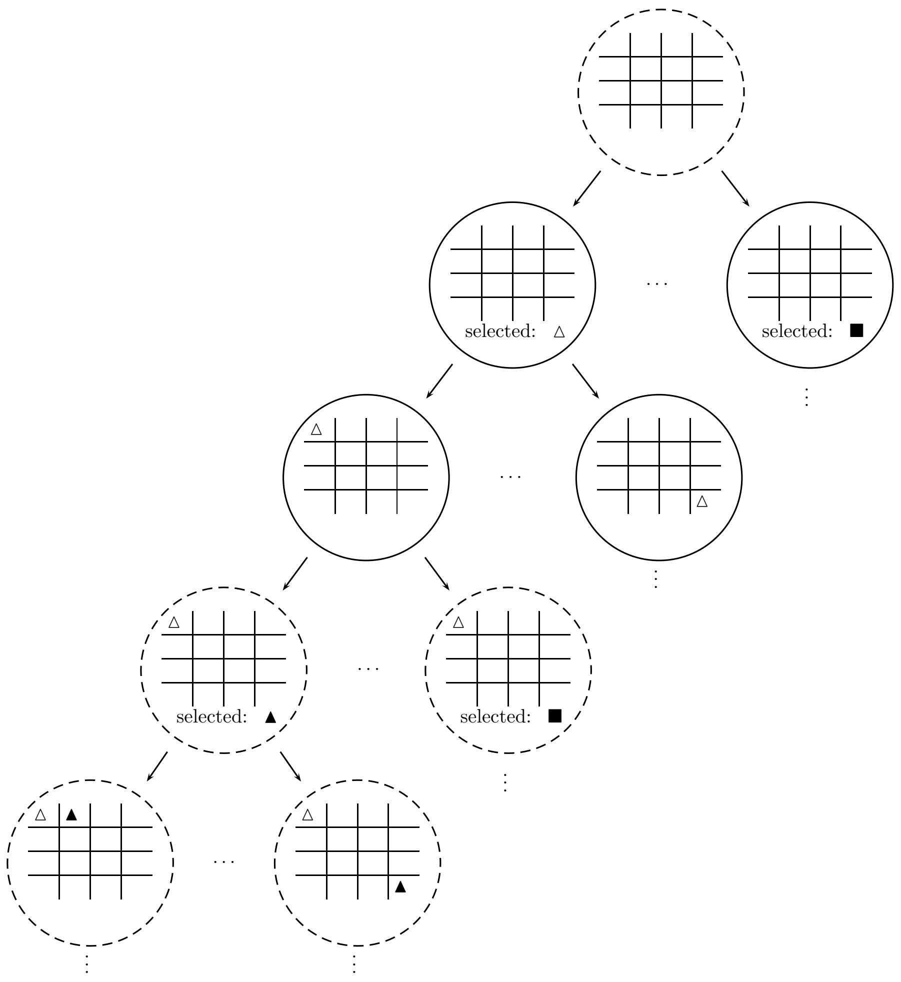

Figure 2.1: An excerpt of the game tree for the game Quarto (cf. Example 2.13). Dashed circles indicate nodes belonging to player 1 (hence, player 1 makes the first move), solid-circled nodes belong to player 2. Different attributes for the (schematic) pieces used here are black/white and square/triangle.

(GDL) [GLP05, LHH$^+$06], which can be used to provide a fully axiomatic, compact description of any n-player game. On the one hand the language is declarative and easy to understand and use by humans, and on the other hand it can be processed fully automatically by a general game-playing system.

In the following, we restate the syntax of the GDL in its recently extended version which incorporates incomplete information and elements of chance [Thi10]. As a language especially designed for game descriptions, the GDL uses pre-defined keywords. A complete list of the keywords together with their intended meaning is given in Table 2.1. Most of the keywords are predicate symbols. The only exception is the constant **random**, which represents the counter part of the pseudo player “Nature” in Definition 2.12 and can be used to model elements of chance and hidden information. A description of a game is defined as a logic program with some restrictions imposed on the use of these keywords.

role (r)	r is a player
init (f)	f holds in the initial position
distinct (t_1, t_2)	terms t_1 and t_2 are syntactically unequal
true (f)	f holds in the current position
legal (r, m)	player r can do move m in the current position
does (r, m)	player r does move m
next (f)	f holds in the next position
terminal	the current position is terminal
goal (r, n)	player r gets n points in the current position
sees (r, p)	player r perceives p in the next position
random	the random player

Table 2.1: The keywords of the GDL. **random** is a constant, all other keywords are predicate symbols.

Definition 2.14 (GDL Syntax). *A* GDL specification *is a logic program where*

- *if* **role** *appears as head of a clause, then this clause is a fact;*
- **init** *only appears as head of clauses and does not depend on any of* **true**, **legal**, **does**, **next**, **sees**, **terminal**, *or* **goal**;
- **distinct** *only appears in the body of clauses;*
- **true** *only appears in the body of clauses;*
- **does** *only appears in the body of clauses, and none of* **legal**, **terminal**, *or* **goal** *depends on* **does**;
- **next** *and* **sees** *only appear as head of clauses.*

Keyword **distinct** *is handled by tacitly assuming an additional (finite) stratified and allowed set of clauses encoding that* **distinct**(s, t) *holds exactly for each pair* $s, t \in \Sigma$ *of syntactically different ground terms.* ■

Sometimes, GDL specifications also contain disjunction in the bodies of clauses. These however can always be reformulated to match the structure of a logic program and are hence neglected here. In the following, we use the terms GDL specification, game description and GDL description interchangeably. As a convention, values n for **goal**(r, n) range over $\{0, 1, \ldots, 100\}$. We recall from [LHH$^+$06] the additional restrictions on game descriptions that ensure finiteness of the set of derivable instances for all relevant queries. Our definition again follows the style of [ST10].

Definition 2.15 (Valid GDL). *To constitute a* valid *GDL specification, a logic program* G *and its dependency graph* Γ *must satisfy the following.*

- G *must be stratified and allowed (cf. Section 2.1.1).*
- *If* p *and* q *occur in a cycle in* Γ *and* G *contains a clause*

$$p(s_1, \ldots, s_m) \text{ :- } l_1(\vec{t_1}), \ldots, q(v_1, \ldots, v_k), \ldots, l_n(\vec{t_n})$$

then for every $i \in \{1, \ldots, k\}$,

– *v_i is ground, or*

– *v_i is one of $s_1, \ldots, s_m$, or*

– *v_i occurs in some $\vec{t_j}$ $(1 \leq j \leq n)$ such that l_j does not occur in a cycle with p in Γ.*

The last condition imposes a restriction on the combination of function symbols and recursion to ensure finiteness and decidability in all cases. ■

In the Game Description Language, players are assumed to always perform simultaneous moves, and the union of all moves taken at a certain game stage is also called a *joint move*. Sequential moves can be represented using pseudo actions without effect for all the players which are currently not allowed to move. We conclude this section with a complete GDL specification for the game Quarto as introduced in Example 2.13.

Example 2.16 (GDL Specification for Quarto). A complete GDL specification for Quarto (cf. Example 2.13) is shown in Figure 2.2. The two players are called `r1`, `r2`, and the 16 pieces are represented by constants `p0000`, `p0001`, `p0010`, …, `p1111`, where each bit position stands for one of the four attributes. The actions are

- `select`(p): piece p gets selected for placement,
- `place`(p, x, y): piece p is placed on the free board cell with coordinates (x, y), and
- `noop`: an action without effect, performed by the player who currently has no control.

The game positions are represented using these state components, henceforth called *fluents*:

- `cell`(x, y, p): board cell (x, y) contains piece p (where $p =$ `b` for blank cells),
- `pool`(p): piece p is available for selection,
- `sctrl`(r): role r currently has control to select a piece,
- `pctrl`(r): role r currently has control to place a piece, and
- `selected`(p): the last action has been to select piece p.

Lines 1 and 2 in Figure 2.2 define the names of the two players and the initial state. A player can select or place a piece when he has control to do so (lines 5 and 6); otherwise he can only do `noop`, a move without effect (line 7). Lines 10 to 17 define the true fluent instances of successor states, including frame axioms for remaining pieces in the pool (line 10) and persistent marked cells (lines 14 and 15). According to line 20, the two players see each other's moves, which induces complete information. A state is terminal if either there is a line of pieces with a common attribute (line 23) or the board has no empty position (line 24). The player who completes a line wins the game with maximal payoff 100 (line 27) and leaves his opponent with minimal payoff 0 (line 29). Both players obtain payoff 50 in case of a completely filled board with no line (line 28). It is straightforward to verify that this logic program satisfies all requirements of a valid GDL description. ■

```
role(r1). role(r2). init(cell(1,1,b)).  ...  init(cell(4,4,b)).
init(sctrl(r1)).    init(pool(p0000)).  ...  init(pool(p1111)).


legal(R,select(P))    :- true(sctrl(R)), true(pool(P)).
legal(R,place(P,X,Y)) :- true(pctrl(R)), true(selected(P)), true(cell(X,Y,b)).
legal(R,noop)         :- role(R), not true(sctrl(R)), not true(pctrl(R)).


next(pool(P))     :- true(pool(P)), not does(r1,select(P)),
                                    not does(r2,select(P)).
next(selected(P)) :- does(R,select(P)).
next(cell(X,Y,P)) :- does(R,place(P,X,Y)).
next(cell(X,Y,S)) :- true(cell(X,Y,S)), does(R,select(P)).
next(cell(X,Y,S)) :- true(cell(X,Y,S)), does(R,place(P,X1,Y1)), !=(X,Y,X1,Y1).
next(sctrl(R))    :- true(pctrl(R)).
next(pctrl(R1))   :- true(sctrl(R2)), otherrole(R1,R2).


sees(R,move(R2,M)) :- otherrole(R,R2), does(R2,M).


terminal :- line.
terminal :- not boardopen.


goal(R,100) :- line, placedlast(R).
goal(R, 50) :- not line, not boardopen, role(R).
goal(R,  0) :- line, otherrole(R,R1), placedlast(R1).


placedlast(R) :- true(sctrl(R)).

boardopen :- true(cell(X,Y,b)).

line :- row.
line :- column.
line :- diagonal.

row      :- true(cell(1,Y,P1)), true(cell(2,Y,P2)),
            true(cell(3,Y,P3)), true(cell(4,Y,P4)), sameattr(P1,P2,P3,P4).
column   :- true(cell(X,1,P1)), true(cell(X,2,P2)),
            true(cell(X,3,P3)), true(cell(X,4,P4)), sameattr(P1,P2,P3,P4).
diagonal :- true(cell(1,1,P1)), true(cell(2,2,P2)),
            true(cell(3,3,P3)), true(cell(4,4,P4)), sameattr(P1,P2,P3,P4).
diagonal :- true(cell(1,4,P1)), true(cell(2,3,P2)),
            true(cell(3,2,P3)), true(cell(4,1,P4)), sameattr(P1,P2,P3,P4).

sameattr(P1,P2,P3,P4) :- nthbit(N,P1,Bit), nthbit(N,P2,Bit),
                         nthbit(N,P3,Bit), nthbit(N,P4,Bit).

!=(X1,Y1,X2,Y2) :- index(X1), index(Y1), index(X2), index(Y2), distinct(X1,X2).
!=(X1,Y1,X2,Y2) :- index(X1), index(Y1), index(X2), index(Y2), distinct(Y1,Y2).

nthbit(1,p0000,0).    index(1).    otherrole(r1,r2).
nthbit(2,p0000,0).    index(2).    otherrole(r2,r1).
...                   index(3).
nthbit(4,p1111,1).    index(4).
```

Figure 2.2: A GDL specification of the game Quarto.

2.2.3 Transition Semantics for the Game Description Language

While in Section 2.2.1, a state has been introduced as a node in the game tree, the GDL understands a state as a set of fluents. We will furtheron refer to a state S by means of the GDL setting. More precisely, we consider a state S to be a subset of the ground terms Σ over the game description. Interpreting a GDL specification requires to encode positions and joint moves as logic program facts. To this end, we introduce two abbreviations: $S^{\mathtt{true}}$, where $S = \{f_1, \dots, f_n\}$ is a finite subset of Σ (a *finite* state); and $A^{\mathtt{does}}$, where $A\colon \{r_1, \dots, r_k\} \mapsto \Sigma$ is an assignment of moves to players:

$$\begin{array}{lcl} S^{\mathtt{true}} & := & \{\mathbf{true}\,(f_1).\,,\ \dots,\ \mathbf{true}\,(f_n).\,\} \\ A^{\mathtt{does}} & := & \{\mathbf{does}\,(r_1, A(r_1)).\,,\ \dots,\ \mathbf{does}\,(r_k, A(r_k)).\,\} \end{array} \tag{2.1}$$

With these preliminary considerations, a game description G can informally be interpreted as follows [GLP05, LHH$^+$06, ST10]:

1. Each derivable instance of the form $\mathbf{role}(r)$ denotes a player with the name r. The overall number n of such instances r different from **random** classifies the game description to encode an n-player game.
2. Each derivable instance of the form $\mathbf{init}(f)$ denotes a fluent f which is true in the initial state, and all fluents which do not occur in any of these instances are false in the initial state.
3. The legal moves a of a player depend on the current state S and can be determined by all instances of head $\mathbf{legal}(r, a)$ which are derivable from G extended by $S^{\mathtt{true}}$.
4. Similarly, the clauses for **terminal** and $\mathbf{goal}(r, n)$ define terminal states and payoff n for player r with respect to a given state.
5. A direct successor state S' can exactly be determined relative to a given finite state S and the performed joint move A of all players by all the derivable instances of $\mathbf{next}(f)$ from G extended by $S^{\mathtt{true}}$ and $A^{\mathtt{does}}$.
6. Similarly, the derivable instances of $\mathbf{sees}(r, p)$ with respect to G extended by $S^{\mathtt{true}}$ and $A^{\mathtt{does}}$ describe all players' percepts in successor state S'.

This informal semantics is made precise with the following definition, which was first given in [ST10] and extended to the generalisation of the language to incomplete information in [Thi10]. It characterises derivable information via entailment $\vdash$ over the unique standard model for a logic program. This is possible since a valid GDL description G is stratified (cf. Definition 2.15), and hence, for each state S and joint action A, also $G \cup S^{\mathtt{true}}$ and $G \cup S^{\mathtt{true}} \cup A^{\mathtt{does}}$ are stratified.

Definition 2.17 (GDL Semantics). *The* semantics *of a valid GDL specification G is given by this state transition system $(R, S_{init}, T, l, u, \mathcal{I}, g)$:*

- *the* roles *or* players*:*

$$R = \{r : G \vdash \mathbf{role}(r)\}$$

- *the* initial *state:*
$$S_{init} = \{f : G \vdash \textbf{init}(f)\}$$

- *the* terminal *states:*
$$T = \{S : G \cup S^{\texttt{true}} \vdash \textbf{terminal}\}$$

- *the* legality *relation:*
$$l = \{(r, a, S) : G \cup S^{\texttt{true}} \vdash \textbf{legal}(r, a)\}$$

- *the* update *function:*
$$u(A, S) = \{f : G \cup A^{\texttt{does}} \cup S^{\texttt{true}} \vdash \textbf{next}(f)\}$$

- *the* information *relation:*
$$\mathcal{I}(A, S) = \{(r, p) : r \neq \textbf{random} \textit{ and } G \cup A^{\texttt{does}} \cup S^{\texttt{true}} \vdash \textbf{sees}(r, p)\}$$

- *the* goal *relation:*
$$g = \{(r, v, S) : r \neq \textbf{random} \textit{ and } G \cup S^{\texttt{true}} \vdash \textbf{goal}(r, v)\}$$

for all finite subsets $S \subseteq \Sigma$ *and assignments* $A \colon R \mapsto \Sigma$*, where* Σ *is the set of ground terms over* G*.* ■

Since **random** is a pseudo player whose sole purpose is the modelling of chance and incomplete information, it is not included in the information relation and the goal relation. Hence, sensing information and payoffs for the random player are not defined. We have omitted the additional probability distribution defined in [Thi10] which determines the behaviour of **random** when playing the game, as it will not be of importance in this work. The syntactic restrictions imposed on valid GDL specifications justify the restriction to finite sets $S^{\texttt{true}}$ and $A^{\texttt{does}}$, as the following proposition from [HST12] shows.

Proposition 2.18 (Finiteness of the GDL). *Suppose* G *is a valid GDL specification, then*

1. $\{r : G \vdash \textbf{role}(r)\}$ *is finite.*
2. $\{f : G \vdash \textbf{init}(f)\}$ *is finite.*
3. $\{f : G \cup A^{\texttt{does}} \cup S^{\texttt{true}} \vdash \textbf{next}(f)\}$ *is finite.* □

According to this proposition, only states that are finite can be reached from the initial state in a game described by a valid GDL specification. However, note that the set of reachable states can still be infinite [ST10] using clauses like

```
init(f(0)).
next(f(s(X))) :- true(f(X)).
```

Based on the transition semantics given in Definition 2.17, we introduce some further notation concerning successive state transitions from [Thi09, Thi10] which will be important throughout this work. Single state transitions are based on the formal semantics of the GDL given in Definition 2.17. There is a transition from state S to state S' if S is not terminal and can be updated to S' with respect to a move assignment A which comprises a legal move for each of the players. Successive state transitions are then composed of multiple single state transitions. This is formally stated as follows.

Definition 2.19 (State Transitions and State Sequences). *For the semantics* $(R, S_{init}, T, l, u, \mathcal{I}, g)$ *of a valid GDL specification and arbitrary finite states* $S, S' \subseteq \Sigma$, *we write* $S \xrightarrow{A} S'$, *to be read as "state* S *develops to state* S' *under joint action* A", *if the following holds:*

- $A : R \mapsto \Sigma$ *is such that* $(r, A(r), S) \in l$ *for each* $r \in R$,
- $S' = u(A, S)$, *and*
- $S \notin T$

We call $S_0 \xrightarrow{A_0} S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{m-1}} S_m$ *(where* $m \in \mathbb{N}$*) a* (state) sequence, *sometimes abbreviated as* $(S_0, S_1, \ldots, S_m)$ *when reference to* $A_0, A_1, \ldots, A_{m-1}$ *is not needed. When the first state of a sequence* σ *is the initial state* S_{init}, *we also call* σ *a* development, *and denote the set of all developments by* Δ_G. *Moreover, a state* S *is called* reachable *iff there is a development the last state of which is* S. ■

We denote sequences with σ and developments with δ, possibly with super- or subscripts. The length of a sequence $\sigma = (S_0, \ldots, S_m)$ is m, sometimes denoted by $|\sigma|$, and the last state S_m of σ is also referred to via the notion $last(\sigma)$. We say that two developments of the same length *differ* if the joint actions of at least one step are different in at least one action performed by one of the players. Two developments δ_1, δ_2 with the same length are *indistinguishable* for player r if r has the same information in $last(\delta_1)$ and $last(\delta_2)$ [FHMV95].[4] Again, we conclude the section by a continuation of our running example Quarto.

Example 2.20 (State Transitions). The clauses for Quarto in Figure 2.2 entail the initial state S_{init}:

$$\{sctrl(r1), pool(p0000), pool(p0001), \ldots, pool(p1111), cell(1,1,b), \ldots, cell(4,4,b)\}$$

Adding $S_{init}^{\mathtt{true}}$ to the set of clauses allows to derive the legal moves of both players in S_{init}:

$$\{(r1, select(p0000), S_{init}), \ldots, (r1, select(p1111), S_{init}), (r2, noop, S_{init})\} \subseteq l$$

Consider, say, $A = \{r1 \mapsto select(p0000), r2 \mapsto noop\}$, then further adding $A^{\mathtt{does}}$ to the logic program in Figure 2.2 allows to infer the updated state, $u(A, S_{init})$:

$$\{pctrl(r2), selected(p0000), pool(p0001), \ldots, pool(p1111), cell(1,1,b), \ldots, cell(4,4,b)\}$$

■

[4] We will provide a more formal characterisation of indistinguishable sequences in Section 4.2.2.

2.2.4 Game Properties

Game Theory defines a game via sequential moves of single players and models simultaneous moves via incomplete information concerning the current game state using information partitions. The Game Description Language follows the opposite approach: players always perform simultaneous moves and sequential moves are represented using pseudo actions. This discrepancy causes some subtleties concerning a formal classification for turn-taking and complete-information games. However, [Thi11a] has shown that any game in the sense of Definition 2.12 can be described in GDL. This renders GDL a universal formalism for the description of games, allowing to base the definitions of several game specific properties on GDL instead of the game-theoretic definition. We will nonetheless refer to the properties of a game description and the properties of a game interchangeably. In the following, we define game properties that we will refer to throughout the remainder of the thesis. As some definitions may differ from the usual intuition, we explain our choices immediately afterwards.

Definition 2.21 (Game Properties)**.** *Let G be a valid game description. A* strategy for player r *is a function which maps each reachable state S to a legal action for player r in S.*

- *G is called* single-player *if the set of all players (including* **random***) is of size one.*
- *G is called* zero-sum *if, in each reachable terminal state that admits payoffs for all players except* **random***, these payoffs add to* 100.
- *G is called* turn-taking *if, in each reachable non-terminal state, there is at most one non-***random** *player which has two or more legal moves.*
- *G is called to be of* complete information *if each player except* **random** *can distinguish each pair of different developments. Otherwise it is called to be of* incomplete information.
- *G is called* strongly winnable by player r *if there is a strategy for r that yields maximal payoff for r after finitely many moves, disregarding the strategies of all other players.*
- *G is called* weakly winnable by player r *if there is a development which ends in a terminal state that yields maximal payoff for r.*
- *G is called* playable *if, for each player r and each reachable non-terminal state S, there is at least one legal action for r in S.*
- *G is called* monotonic *if, for each player r different from* **random** *and each reachable state S, r has exactly one goal value in S, and goal values never decrease in the course of the development of the game.* ■

Although the random player is only used to model elements of chance, we decide to consider a game *not* to be single-player in case **random** is present, as this classification will be more useful in the context of this work. In Game Theory, a game is usually considered zero-sum if, as the name suggests, the sum of payoffs in all terminal states is

0, whereas an invariant sum unequal to 0 is considered constant-sum. However, due to the convention of payoffs ranging from 0 to 100 in General Game Playing [LHH+06], sum 100 exactly represents the game-theoretic meaning of property zero-sum and hence justifies the terminology we apply. At a first glance, property turn-taking may appear counterintuitive as well. However, as the GDL amounts to the formulation of simultaneous moves, the definition for turn-taking must incorporate pseudo actions of players which are not supposed to move. Concerning the properties complete and incomplete information, each game can easily be enriched by a clause

```
sees(R1,move(R2,M)) :- role(R1),  distinct(R1,random),
                       role(R2),  distinct(R1,R2),
                       does(R2,M).
```

This clause assures that each player except for **random** always perceives the moves of each other player, including **random**. This implies that he has complete state knowledge, as the complete game description is handed to each player prior to game play and contains a complete description of the initial state [Thi10]. Hence, this clause can be used whenever a game is intended to be of complete information. The remaining property definitions follow [LHH+06]. As a last remark to the game properties, note that each single-player game is weakly winnable if and only if it is strongly winnable.

2.2.5 Execution Model

We have now introduced all theoretical details of the Game Description Language that will be needed throughout this work. In this section, we want to make a short side trip to practice. I.e., we will summarise how general game-playing agents are organised to play arbitrary games against each other, for example in the setting of a competition. Although the material presented here will not directly be relevant in the remainder of this work, it provides the context this work is embedded in and is hence important for a more accurate picture.

The game management infrastructure has been introduced in [GLP05, LHH+06] and adapted to games with incomplete information in [Thi10]. Central part of the infrastructure is the *Game Manager*. Players communicate solely with the Game Manager using HTTP messages. The information flow is depicted in Figure 2.3. The Game Manager distributes the game description at the beginning, collects joint moves from the players during a *match* (i.e., a single run of a game), maintains a consistent, up-to-date game state and reports the end of a match. Moreover it has access to a database with information concerning game descriptions, players and matches and provides graphical output for spectators of a match. The following messages are used to communicate with the players:

START This message is sent by the Game Manager to each player once prior to the beginning of a match. It contains a unique match id, the role which is assigned to the player that gets this message, a valid game description, an integer called *start clock* which announces the time in seconds before the match starts, and an integer called *play clock* announcing the time in seconds between each of the moves.

Players are supposed to respond to this message with READY.

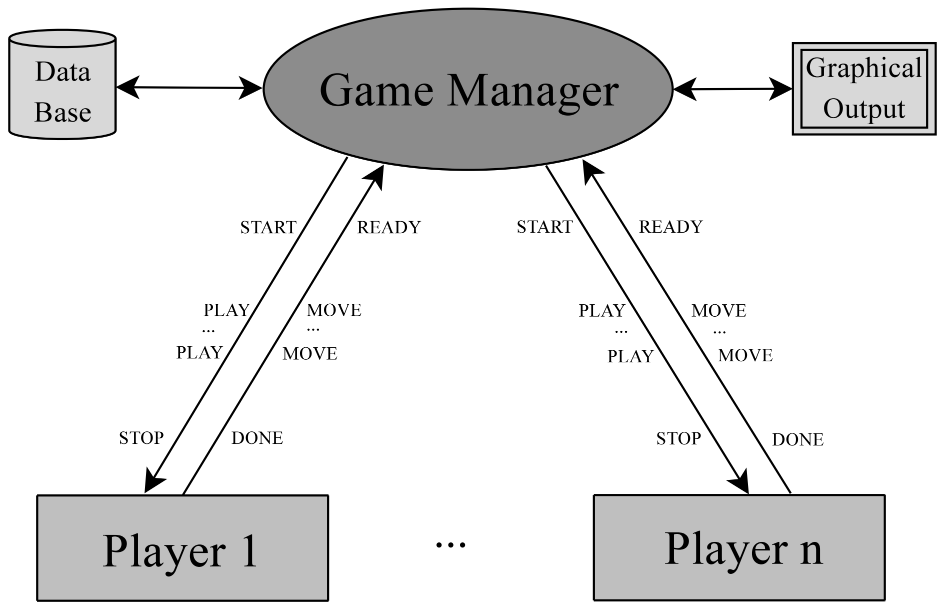

Figure 2.3: The General Game Playing Execution Model. The Game Manager controls the information flow, communicating with the players via HTTP messages.

PLAY This message is repeatedly sent by the Game Manager to each player during the match. It contains the match id and, in the original setting of complete information games as introduced in [GLP05, LHH$^+$06], the previously performed joint move. In the setting of [Thi10], instead of the joint move, players are informed about their percepts according to the information relation $\mathcal{I}$ from Definition 2.17. The first of these messages is issued when all players are ready, or the time specified in the start clock has passed after sending the START message.

Players are supposed to respond to this message with a chosen legal move. If the response time exceeds the limit set in the play clock, the Game Manager assigns an arbitrary legal move to that player.

STOP This message is issued once to communicate the end of the match. It again contains the match id and, depending on the setting, the previously taken joint move or percept information.

Players are supposed to respond to this message with DONE.

In the setting of incomplete-information games, the pseudo player **random** is not considered an actual player. Instead, its actions are chosen by the Game Manager randomly with uniform probability from its set of legal moves in the current state.

The General Game Playing Competition has first been issued in 2005 [GLP05] and since been taking place annually[5]. About 10 teams play newly designed and previously unknown games in a competitive setting with start clocks usually around 300 seconds and play clocks of about 30 seconds. Prior to sending a game description to the players,

[5]We refer to `games.stanford.edu` and `www.general-game-playing.de` for further information.

the game manager arbitrarily renames all used predicate and term symbols except for the keywords of the GDL. This ensures that game-playing agents do not rely on implicit non-structural information the game designer has put in, e.g. the reference on cells of a game board via the fluent `cell`(x, y).

2.3 Summary

We introduced logic programs, dependency graphs for logic programs, and the answer set semantics which defines answer sets as special models for logic programs which are minimal and supported. Stratified logic programs admit a unique answer set which corresponds to the standard model. We generalised both syntax and semantics to answer set programs by the addition of weight atoms and constraints, and restated a theorem which allows the splitting of some answer set programs under preservation of desired properties.

In the second part, we stated the formal definition of an n-player game by means of Game Theory, and showed how the game Quarto, our running example for the following chapter, can be expressed with this formalism. We defined the GDL as a subclass of logic programs, and gave a specification for Quarto in this formalism as well. We then provided a multiagent semantics for the GDL based on state transition systems, defined some game properties that will be important throughout this work, and shortly sketched the execution model for General Game Playing.

Chapter 3

Sequence Invariants

In this chapter, we develop a formal method which is able to efficiently and fully automatically prove game-specific properties which hold across all reachable game states. To this end, we first motivate a property class that incorporates (restricted) time reference and allows to be verified avoiding a full game tree search. We provide a language for the description of these properties, which we will call sequence invariants, and define their semantics with respect to a specific game description. We then develop an induction method which allows to prove that a specific sequence invariant holds across all reachable states of the game. Our method suggests encodings of the property and the given game description to answer set programs, which then allows to use an off-the-shelf answer set solver to automatically prove the truth of the property for all reachable states. We formally show the correctness of our method and develop extensions which allow to prove multiple properties at once and to additionally solve single player games. The chapter includes and extends passages of own published work (we again refer to Section 7.1 for a detailed listing of the included material).

3.1 The Importance of Sequence Invariants in GGP

Recall the clauses for Quarto in line 24 and 34, respectively, from the game description in Figure 2.2:

```
terminal  :- not boardopen.
boardopen :- true(cell(X,Y,b)).
```

Suppose these two clauses were replaced by

```
terminal :- not true(cell(X,Y,b)), index(X), index(Y).
```

At first glance, this seems not to alter the meaning, namely, that the game terminates if there is no blank cell. In fact, however, there is a crucial difference regarding the implicit quantification of the variables `X` and `Y`. While the original two clauses imply **terminal** if there do *not* exist `X` and `Y` such that **true**`(cell(X,Y,b))`, the alternative clause implies **terminal** if there do exist `X` and `Y` such that *not* **true**`(cell(X,Y,b))`. The first placement of a piece at *any* cell yields a state which satisfies the body of one ground instance of the alternative clause (as the marked cell is not blank anymore) and hence untruly renders this state terminal, whereas the original clauses imply termination only when *all* cells are marked. The organisers of the General Game Playing Competition in 2006 used a GDL specification for the game of Othello [IK94] with a similar defect, which caused quite some disturbance, first among the participants and then among the organisers themselves. A proof system that allows to formally verify game descriptions would have been of invaluable assistance to the game designers in order to prevent such mishaps. The bug that we just introduced in the Quarto game description, say, would be immediately detected when attempting to prove the following intended property:

If there is a blank cell and no completed line, then Quarto is not terminated. (3.1)

In addition to assisting the game design, a proof system can also help a general game-playing system to discover valuable information about a previously unknown game. This information can then be used, for example, to choose an appropriate algorithm to search the game or to construct a suitable, game-dependent heuristic. As an example, knowing that players take turns making their moves allows to apply a minimax algorithm with pruning, and knowing the truth of some fluents to be persistent can be used for a more accurate state evaluation. In the following we will motivate a class of game properties which allows an efficient verification and is expressive enough to comprise many interesting properties of a game description, including the previously mentioned ones.

To begin with, we consider the class of properties which make statements about single states of a game, which we will call *state invariants*. They are "local", which means that they can be verified for all reachable states by an analysis of the GDL clauses rather than by a complete search through the whole game tree. This covers many interesting properties, including (3.1). As another example, the Quarto property

Each cell contains at most one piece. (3.2)

allows a general game player to infer the existence of a board structure, which is valuable knowledge to construct a good heuristic for playing the game [KDS06, Clu07, ST07]. Also the general properties zero-sum, turn-taking, and playability (cf. Definition 2.21) belong to this class. However, many interesting properties cannot be expressed by referring to a single state. Consider, for example,

If no player can place a piece now, then in the next state one player can do so. (3.3)

This property is not a state invariant due to the inherent reference to subsequent game states. A similar argument applies to the requirement of never-decreasing goal values in the general property of monotonicity from Definition 2.21. However, both properties can be seen as state *sequence* invariants with degree 1, meaning that their formulation requires a "lookahead" of exactly one joint move.

3.2 Formalisation of Sequence Invariants

In the following we will first define a formal language over the syntax of GDL that allows the formulation of state sequence invariants (Section 3.2.1). We then discuss some intuitions for the interpretation of sequence invariants and provide a semantics which best matches these intuitions (Section 3.2.2). Finally, we discuss some properties of the semantics (Section 3.2.3).

3.2.1 Syntax

Our language for the formulation of state sequence invariants is restricted in that no infinite sequences and no quantification over sequences is allowed, which turns out to be a beneficial tradeoff between expressibility and efficient verifiability. A simple and elegant way to obtain such a language is by extending GDL with the unary operator "$\bigcirc$" borrowed from Temporal Logic (see, for example, [KM08]) to refer to successor game states.

Definition 3.1 (Sequence Invariants). *We define $\mathcal{P}$ to be the set of ground atoms $p(\vec{t})$ over a valid GDL specification G such that $p \notin \{\mathbf{init}, \mathbf{next}\}$ and p does not depend on* **does** *in G. Then the set $\mathcal{SIN}_G$ of* (state) sequence invariants over G *is the smallest set with*

- *$\mathcal{P} \subseteq \mathcal{SIN}_G$;*
- *Let $\varphi[\vec{X}]$ denote a formula obtained from a formula φ by replacing arbitrary ground terms with variables from $\vec{X} = (X_1, \ldots, X_k)$. Furthermore, let $D_{\vec{X}} = D_{X_1} \times \ldots \times D_{X_k}$ for* finite *sets $D_{X_1}, \ldots, D_{X_k}$ of ground terms from Σ, called* (variable) domains*.*

  If $\varphi, \varphi_1, \varphi_2 \in \mathcal{SIN}_G$, then also the following are in $\mathcal{SIN}_G$:

  – *$\neg\varphi$, $\varphi_1 \wedge \varphi_2$, and $\varphi_1 \vee \varphi_2$;*
  – *$(\exists \vec{X} : D_{\vec{X}})\, \varphi[\vec{X}]$, and $(\forall \vec{X} : D_{\vec{X}})\, \varphi[\vec{X}]$;*
  – *$(\exists_{l..u} \vec{X} : D_{\vec{X}})\, \varphi[\vec{X}]$, for each $l \in \mathbb{N}$ and $u \in \mathbb{N} \cup \{\infty\}$ s.t. $l \leq u$;*
  – *$\bigcirc\varphi$.*

Additionally, for variables $\vec{X} = (X_1, \ldots, X_k)$ and ground terms $\vec{t} = (t_1, \ldots, t_k)$ from Σ, by $\varphi[\vec{X}/\vec{t}]$ we denote the formula which is obtained from φ by replacing all variables in $\vec{X}$ with the respective ground terms in $\vec{t}$. ■

Since atoms over **init** and **next** are excluded, the unary predicate symbol **true** provides the only means for referring to fluents and thus to states, which keeps the language clear and simple. Predicate symbols that depend on **does** are excluded for technical reasons, as will become clear at the end of Section 3.3.1. We allow restricted quantification, by explicit specification of a *finite* domain for each variable; and we use counting quantifiers of the form $(\exists_{l..u} \vec{X} : D_{\vec{X}})\, \varphi[\vec{X}]$ to give a lower (l) and upper (u) bound for the number of ground instances $\vec{t}$ for a vector of variables $\vec{X}$ such that $\varphi[\vec{X}/\vec{t}]$ is true. If $u = \infty$ then there is no upper bound. Modality $\bigcirc\varphi$ states that φ holds at the (legal) successor of the current game state. We additionally define the binary connective $\supset$ for implication as the macro $\varphi_1 \supset \varphi_2 := \neg\varphi_1 \vee \varphi_2$, and use the terms "(state) sequence invariant" and "formula" interchangeably. In the remainder, φ, ψ, and ρ (possibly with subscripts) are always used to refer to state sequence invariants.

The following definition gives a formal classification for the maximal "nesting" of the modal operator $\bigcirc$ in a sequence invariant. It will be important to determine the necessary length of state sequences for the interpretation of sequence invariants.

Definition 3.2 (Degree of a Sequence Invariant). *For a valid GDL description G and sequence invariants $\varphi, \varphi_1, \varphi_2 \in \mathcal{SIN}_G$, we define the* degree of φ, *denoted* $\deg(\varphi)$, *recursively as follows:*

$$\begin{array}{lcll}
\deg(p) & := & 0 & \textit{for } p \in \mathcal{P} \\
\deg(\neg\varphi) & := & \deg(\varphi) & \\
\deg(\varphi_1 \wedge \varphi_2) & := & \max\{\deg(\varphi_1), \deg(\varphi_2)\} & \\
\deg(\varphi_1 \vee \varphi_2) & := & \max\{\deg(\varphi_1), \deg(\varphi_2)\} & \\
\deg((\exists \vec{X} : D_{\vec{X}})\, \varphi[\vec{X}]) & := & \deg(\varphi[\vec{X}/\vec{t}]) & \textit{for an arbitrary } \vec{t} \in D_{\vec{X}} \\
\deg((\forall \vec{X} : D_{\vec{X}})\, \varphi[\vec{X}]) & := & \deg(\varphi[\vec{X}/\vec{t}]) & \textit{for an arbitrary } \vec{t} \in D_{\vec{X}} \\
\deg((\exists_{l..u} \vec{X} : D_{\vec{X}})\, \varphi[\vec{X}]) & := & \deg(\varphi[\vec{X}/\vec{t}]) & \textit{for an arbitrary } \vec{t} \in D_{\vec{X}} \\
\deg(\bigcirc\varphi) & := & \deg(\varphi) + 1 &
\end{array}$$

■

Before we conclude this section with some examples, we formally classify the subformulas of a sequence invariant.

Definition 3.3 (Subformulas of a Sequence Invariant). *For a valid GDL description G and sequence invariants $\varphi, \varphi_1, \varphi_2, \psi \in \mathcal{SIN}_G$, ψ is said to be a* subformula *of φ if, and only if, $\psi \in sub(\varphi)$ according to the following definition:*

$$\begin{array}{lcll}
sub(p) & := & \{p\} & \textit{for } p \in \mathcal{P} \\
sub(\neg\varphi) & := & \{\neg\varphi\} \cup sub(\varphi) & \\
sub(\varphi_1 \wedge \varphi_2) & := & \{\varphi_1 \wedge \varphi_2\} \cup sub(\varphi_1) \cup sub(\varphi_2) & \\
sub(\varphi_1 \vee \varphi_2) & := & \{\varphi_1 \vee \varphi_2\} \cup sub(\varphi_1) \cup sub(\varphi_2) & \\
sub((\exists \vec{X} : D_{\vec{X}})\, \varphi[\vec{X}]) & := & \{(\exists \vec{X} : D_{\vec{X}})\, \varphi[\vec{X}]\} \cup \bigcup_{\vec{t} \in D_{\vec{X}}} sub(\varphi[\vec{X}/\vec{t}]) & \\
sub((\forall \vec{X} : D_{\vec{X}})\, \varphi[\vec{X}]) & := & \{(\forall \vec{X} : D_{\vec{X}})\, \varphi[\vec{X}]\} \cup \bigcup_{\vec{t} \in D_{\vec{X}}} sub(\varphi[\vec{X}/\vec{t}]) & \\
sub((\exists_{l..u} \vec{X} : D_{\vec{X}})\, \varphi[\vec{X}]) & := & \{(\exists_{l..u} \vec{X} : D_{\vec{X}})\, \varphi[\vec{X}]\} \cup \bigcup_{\vec{t} \in D_{\vec{X}}} sub(\varphi[\vec{X}/\vec{t}]) & \\
sub(\bigcirc\varphi) & := & \{\bigcirc\varphi\} \cup sub(\varphi) &
\end{array}$$

■

Example 3.4 (Sequence Invariants). Consider the previously mentioned property (3.1). Denoting the set of board indices by $I = \{1, 2, 3, 4\}$, it can be formulated via the following formula of degree 0:[1]

$$((\exists X, Y : I)\, \mathit{true}(\mathit{cell}(X, Y, b)) \wedge \neg \mathit{line}) \supset \neg \mathit{terminal}.$$

Property (3.2) can be formulated via a formula of degree 0, too, if we denote the set of pieces by $D_P = \{\mathit{p0000}, \mathit{p0001}, \ldots, \mathit{p1111}\}$:

$$(\forall X, Y : I)\, (\exists_{0..1} P \colon D_P)\ \mathit{true}(\mathit{cell}(X, Y, P)). \tag{3.4}$$

Property (3.3), however, refers to two consecutive states and hence requires a formula of degree 1:

$$\neg(\exists R \colon \{\mathit{r1}, \mathit{r2}\})\, \mathit{true}(\mathit{pctrl}(R)) \supset \bigcirc(\exists R \colon \{\mathit{r1}, \mathit{r2}\})\, \mathit{true}(\mathit{pctrl}(R)). \tag{3.5}$$

Let us refer to this formula with φ, and to its subformula $(\exists R \colon \{\mathit{r1}, \mathit{r2}\})\, \mathit{true}(\mathit{pctrl}(R))$ with ψ. Resolving the macro $\supset$, we have that $\varphi = \neg\neg\psi \vee \bigcirc\psi$. The set of subformulas of φ is $sub(\varphi) = \{\varphi, \neg\neg\psi, \bigcirc\psi, \neg\psi, \psi, \mathit{true}(\mathit{pctrl}(\mathit{r1})), \mathit{true}(\mathit{pctrl}(\mathit{r2}))\}$. ■

[1] In quantifiers, when two variables X and Y have the same domain D, we abbreviate $(X, Y) : D \times D$ by $X, Y : D$.

3.2.2 Semantics

Intuitively, a sequence invariant φ with degree n is true in a state S_0 if and only if all "relevant" sequences $(S_0, \dots, S_m)$ satisfy φ. Clearly, all sequences with $m = n$ are relevant, and sequences where $m > n$ are irrelevant since they provide no more information (regarding φ) than their respective initial subsequences of length n. By the same argument there is no need to consider sequences of *infinite* length, opposed to the semantics for various Temporal Logics (see, for example, [KM08]). Also irrelevant are sequences with $m < n$ that can be extended by a legal transition, as they are contained in sequences with greater length. However, two types of sequences with $m < n$ cannot be extended and thus need to be considered:

- *Terminated Sequences* (i.e. sequences that end in a terminal state). These are relevant for entailment, lest arbitrary formulas of the form $\psi \wedge \bigcirc\rho$ be considered true in any terminal state S_t regardless of the truth of ψ (since $\psi \wedge \bigcirc\rho$ is naturally true with respect to *all* sequences of length ≥ 1 in S_t when no such sequence exists).

- *Non-Playable Sequences* (i.e. sequences that end in a non-terminal state with no legal move for at least one player). Although they influence entailment, we neglect non-playable sequences for the moment and defer the discussion on this issue to Section 3.6.4.

Terminated sequences could in principle be extended by a pseudo joint action ϵ that defines a transition from each terminal state S_t into S_t itself, that is, $S_t \xrightarrow{\epsilon} S_t$. Every terminated sequence could thus be extended to length n, which would allow to give a semantics for invariants over sequences of length n only. However, this has unintended side effects. For example, implications of the shape $\neg\varphi \supset \bigcirc\varphi$ (like, e.g., formula (3.5) for Quarto from Section 3.2.1) would never be considered true in all reachable states in case there is a terminal state S_t that satisfies $\neg\varphi$, as the successor state of S_t (which is S_t again) cannot satisfy the converse formula φ. Similar considerations with other pseudo continuations of terminal states lead to equally non-verifiable albeit intuitively valid sequence invariants. The following definition of entailment takes into account all of our foregoing considerations. I.e., it takes into account all sequences that match the length of the formula degree, and additionally incorporates shorter terminated sequences by considering parts of the formula that are "uncovered" by a respective terminated sequence to be true.

Definition 3.5 (Semantics for Sequence Invariants). *Let G be a valid GDL specification. A sequence $(S_0, \dots, S_m)$ is called* n-max *if it is of length n, or if it is shorter and ends in a terminal state. Let S_0 be a state and φ be a formula such that* $\deg(\varphi) = n$. *We say that S_0* satisfies *φ (written $S_0 \vDash \varphi$) if for all n-max sequences $S_0 \xrightarrow{A_0} \dots \xrightarrow{A_{m-1}} S_m$ $(m \leq n)$ we have that $(S_0, \dots, S_m) \vDash \varphi$ as follows:*

$(S_0, \dots, S_m) \vDash p$	*iff*	$G \cup S_0^{\texttt{true}} \vdash p \qquad (p \in \mathcal{P})$
$(S_0, \dots, S_m) \vDash \neg\varphi$	*iff*	$(S_0, \dots, S_m) \nvDash \varphi$
$(S_0, \dots, S_m) \vDash \varphi_1 \wedge \varphi_2$	*iff*	$(S_0, \dots, S_m) \vDash \varphi_1$ *and* $(S_0, \dots, S_m) \vDash \varphi_2$
$(S_0, \dots, S_m) \vDash \varphi_1 \vee \varphi_2$	*iff*	$(S_0, \dots, S_m) \vDash \varphi_1$ *or* $(S_0, \dots, S_m) \vDash \varphi_2$
$(S_0, \dots, S_m) \vDash (\exists \vec{X} : D_{\vec{X}})\, \varphi[\vec{X}]$	*iff*	*there is a* $\vec{t} \in D_{\vec{X}}$ *s.t.* $(S_0, \dots, S_m) \vDash \varphi[\vec{X}/\vec{t}]$

$(S_0,\dots,S_m) \models (\forall \vec{X} : D_{\vec{X}})\, \varphi[\vec{X}]$	*iff*	*for all* $\vec{t} \in D_{\vec{X}}$*:* $(S_0,\dots,S_m) \models \varphi[\vec{X}/\vec{t}]$
$(S_0,\dots,S_m) \models (\exists_{l..u} \vec{X} : D_{\vec{X}})\, \varphi[\vec{X}]$	*iff*	*there are* $\geq l$ *and* $\leq u$ *different* $\vec{t} \in D_{\vec{X}}$ *s.t.* $(S_0,\dots,S_m) \models \varphi[\vec{X}/\vec{t}]$
$(S_0,\dots,S_m) \models \bigcirc\varphi$	*iff*	$m = 0$ *or* $(S_1,\dots,S_m) \models \varphi$

■

A crucial part here is $(S_0,\dots,S_m) \models \bigcirc\varphi$ for $m = 0$: in case we reach the end of a state sequence, every formula of the form $\bigcirc\varphi$ must be true. Together with the definition of an n-max sequence, this correctly grasps the intuition for terminated sequences of length smaller than n, so that, for example, formula (3.5) is clearly entailed in each terminal state. In general, $\bigcirc\varphi$ is considered true in every terminal state even if φ is inconsistent. In our setting this is perfectly acceptable as we are just interested in the truth of a formula in reachable states—all states beyond are irrelevant. It is worth mentioning that $\bigcirc\neg\varphi$ and $\neg \bigcirc \varphi$ are only equivalent for non-terminal states, whereas for every terminal state S_t we have that $(S_t) \models \bigcirc\neg\varphi$ but $(S_t) \not\models \neg \bigcirc \varphi$. In the following, we say a formula is *valid* if each reachable state satisfies the formula with respect to the given semantics.

3.2.3 Properties of the Semantics

The following proposition relates sequences that are longer than the degree n of the formula to be proved to n-max sequences. This generalises formula entailment to a context with additional formulas that can have a higher degree. It is conditioned on the standard restriction to *playable* GDL games according to Definition 2.21.

Proposition 3.6 (Sequence Extension)**.** *Let G be a valid GDL specification, φ be a sequence invariant of degree n, $(S_0,\dots,S_m)$ be an n-max sequence, and $\widehat{n} \geq n$ arbitrary.*

1. *Let G be playable and state S_0 reachable. Then $(S_0,\dots,S_m)$ can be extended to an $\widehat{n}$-max sequence $(S_0,\dots,S_m,\dots,S_{\widehat{m}})$.*
2. *For all $\widehat{n}$-max sequences $(S_0,\dots,S_m,\dots,S_{\widehat{m}})$ extended from $(S_0,\dots,S_m)$:*

$$(S_0,\dots,S_m) \models \varphi \textit{ iff } (S_0,\dots,S_m,\dots,S_{\widehat{m}}) \models \varphi.$$

Proof:

1. By induction on $\widehat{n}$. The base case $n = \widehat{n}$ is immediate. For the induction step, assume that $(S_0,\dots,S_m)$ can be extended to an $\widehat{n}$-max sequence $(S_0,\dots,S_m,\dots,S_{\widehat{m}})$. If $S_{\widehat{m}}$ is terminal, then $(S_0,\dots,S_{\widehat{m}})$ is also $\widehat{n}+1$-max. Otherwise, since $S_{\widehat{m}}$ is reachable and G playable, there are $A_{\widehat{m}}$ and $S_{\widehat{m}+1}$ such that $S_{\widehat{m}} \xrightarrow{A_{\widehat{m}}} S_{\widehat{m}+1}$. Then $(S_0,\dots,S_m,\dots,S_{\widehat{m}},S_{\widehat{m}+1})$ is $\widehat{n}+1$-max.
2. By induction on the structure of φ. For the base case, consider $\varphi = p$ for some ground atom $p \in \mathcal{P}$. Entailment for a ground atom only involves the first state of a sequence, which implies the claim. For the induction step, consider $\varphi = \bigcirc\psi$ and let $(S_0,\dots,S_m,\dots,S_{\widehat{m}})$ be an arbitrary $\widehat{n}$-max sequence extended from $(S_0,\dots,S_m)$. If S_0 is terminal, then $m = \widehat{m} = 0$, hence the two

sequences are identic. Otherwise, S_1 exists and we have $(S_0,\dots,S_m) \vDash \bigcirc\psi$ iff $(S_1,\dots,S_m) \vDash \psi$ iff (by the induction hypothesis) $(S_1,\dots,S_m,\dots,S_{\widehat{m}}) \vDash \psi$ iff $(S_0,\dots,S_m,\dots,S_{\widehat{m}}) \vDash \bigcirc\psi$. The remaining cases can be argued similarly. □

Since a playable GDL specification provides legal moves for every role only in states that are both non-terminal and reachable, the first item of Proposition 3.6 requires the assumption of S_0 being reachable. As an example, reconsider the GDL specification of Quarto depicted in Figure 2.2. Although this game is playable, there are (unreachable) states S which are non-terminal and non-playable, e.g. if *pctrl*(*r1*) $\in S$, *sctrl*(*r1*) $\notin S$, and *selected*(p) $\notin S$ for all pieces p. Then player *r1* has no legal move in S, and the 0-max sequence (S) cannot be extended to a 1-max sequence. This has the following consequence: even if an n-max formula φ is known to be true with respect to all $\widehat{n}$-max sequences starting at S for some $\widehat{n} \geq n$, φ is not necessarily true with respect to all n-max sequences starting at S, unless S is a *reachable* state. This explains the restriction to identical initial subsequences in the equivalence result in the second item of Proposition 3.6.

The mentioned consequence further yields that, for non-reachable states S, $S \vDash \varphi \wedge \psi$ does not necessarily imply $S \vDash \varphi$ even with a playable GDL specification. Since ψ could be of higher degree than φ, φ can still be false with respect to a shorter, non-extendable (and non-reachable) sequence. However, $S \vDash \varphi$ and $S \vDash \psi$ always implies $S \vDash \varphi \wedge \psi$, even for non-reachable states S and non-playable GDL specifications.

3.3 Prerequisites for the Verification Method

While in theory state sequence invariants can be verified by a complete search through the set of reachable states (provided the game is finite, of course), as investigated in [RvW09], our interest lies in finding a practical proof method that can be applied to games with far too large a state space to permit complete search. In the following, we will present such a method in three steps. First, we define the so-called temporal extension of a set of GDL clauses that allows us to compute a fixed number of state transitions within a single program (Section 3.3.1). Thereafter we show how this program can be extended by clauses that encode a given state sequence invariant (Section 3.3.2). These two steps comprise the content of this section. Finally, in the subsequent Section 3.4, we demonstrate how the combined program can be used to verify the encoded invariant against the game description.

3.3.1 Temporal GDL Extension

The Game Description Language (GDL) is based on an elementary time structure that consists of only two time points, "before" (encoded by **true**) and "after" (encoded by **next**). Without further additions, a game description can thus be used only for reasoning about a single state transition: given a complete, finite state and a joint move, standard entailment allows to determine a successor state according to Definition 2.17. This suffices to verify sequence invariants with degree 0, but invariants of higher degree require multiple successive state transitions and hence necessitate the introduction of additional time points in the clauses. This has been done, e.g., by [Thi09], and is adapted to our setting as follows.

Definition 3.7 (Temporal GDL Extension). *For a valid GDL specification G, we call $G_{\leq n}$ the* temporal extension of G of degree n, *where $G_{\leq n} := \bigcup_{0 \leq i \leq n} G_i$ and each G_i is constructed by*

- *omitting all clauses from G with head* **init***;*
- *replacing each occurrence of*
  - $\mathbf{next}(f)$ *by* $\mathbf{true}(f, i+1)$*,*
  - $\mathbf{sees}(r, p)$ *by* $\mathbf{sees}(r, p, i+1)$*, and*
  - $p(t_1, \ldots, t_n)$ *by* $p(t_1, \ldots, t_n, i)$*, for each predicate symbol* $p \notin \{\mathbf{next}, \mathbf{sees}\}$.

Furthermore, the timed variants *of the sets of unit clauses $S^{\mathtt{true}}$ and $A^{\mathtt{does}}$, defined as (2.1) on page 20, are*

$$\begin{array}{rcl} S^{\mathtt{true}}(i) & := & \{\mathbf{true}\,(f_1, i).\,,\ \ldots,\ \mathbf{true}\,(f_n, i).\,\} \\ A^{\mathtt{does}}(i) & := & \{\mathbf{does}\,(r_1, A(r_1), i).\,,\ \ldots,\ \mathbf{does}\,(r_k, A(r_k), i).\,\} \end{array}$$

for any $S = \{f_1, \ldots, f_n\} \subseteq \Sigma$; $A\colon \{r_1, \ldots, r_k\} \mapsto \Sigma$; and $i \geq 0$. ∎

Example 3.8 (Temporal GDL Extension). Let G be the GDL specification of Quarto depicted in Figure 2.2 and consider the clause in lines 10–11:

```
next(pool(P)) :- true(pool(P)),
                 not does(r1,select(P)),
                 not does(r2,select(P)).
```

The temporal extension $G_{\leq n}$ contains the following clause for each $0 \leq i \leq n$:

```
true(pool(P), i+1) :- true(pool(P), i),
                      not does(r1,select(P), i),
                      not does(r2,select(P), i).
```

∎

The resulting program can be made more efficient by omitting the time argument in any atom over a predicate symbol that is neither a GDL keyword nor depends on **true** or **does** in the original game description, further details on this issue can be found in Section 5.2.1. Note also that, strictly speaking, $G_{\leq n}$ may not be stratified even if G is; as a simple example consider the stratified clause `next(f) :- not true(g).`, whose temporal extension is `true(f,`$i+1$`) :- not true(g,`i`)`. However, the temporally extended program could be easily rewritten into an equivalent but stratified program: instead of simply adding a time argument to atoms p, time could be encoded into their respective predicate symbols, obtaining different predicate symbols p_i for each time step. Definition 3.7 is more readable, and we will nonetheless tacitly assume that $G_{\leq n}$ is always stratified.

The following result shows that a temporally extended GDL specification can be used to reason over the GDL via state sequences. It generalises a similar result from [Thi09].

Theorem 3.9 (Correctness of the Temporal GDL Extension)**.** *Let G be a valid GDL specification and $S_0 \xrightarrow{A_0} S_1 \dots \xrightarrow{A_{m-1}} S_m$ a sequence. Consider the program $P = S_0^{\mathtt{true}}(0) \cup G_{\leq m} \cup \bigcup_{i=0}^{m-1} A_i^{\mathtt{does}}(i)$, then for all $0 \leq i \leq m$ and predicate symbols $p \notin \{\,\mathbf{init}, \mathbf{next}\}$ that do not depend on* **does***, we have*

$$G \cup S_i^{\mathtt{true}} \vdash p(\vec{t}) \textit{ iff } P \vdash p(\vec{t}, i)$$

Proof: Let $P_0 = S_0^{\mathtt{true}}(0)$ and $P_m = G_{m-1} \cup A_{m-1}^{\mathtt{does}}(m-1) \cup P_{m-1}$ for $m > 0$. We first prove the intermediate result

$$S_m = \{f : P_m \vdash \mathbf{true}(f, m)\} \tag{3.6}$$

by induction on m. The base case $m = 0$ is immediate. Induction step: By Definition 2.17 (the GDL Semantics) we have

$$f \in S_{m+1} \text{ iff } f \in u(A_m, S_m) \text{ iff } G \cup A_m^{\mathtt{does}} \cup S_m^{\mathtt{true}} \vdash \mathbf{next}(f).$$

Using the induction hypothesis, this is equivalent to

$$G \cup A_m^{\mathtt{does}} \cup \{\mathbf{true}(f') : P_m \vdash \mathbf{true}(f', m)\} \vdash \mathbf{next}(f).$$

The clauses from G which solely contribute to the initial state encoding do not influence entailment of $\mathbf{next}(f)$, since their heads do not occur in the remaining clauses. Together with the construction of the temporal GDL extension from Definition 3.7, this yields equivalence to

$$G_m \cup A_m^{\mathtt{does}}(m) \cup \{\mathbf{true}(f', m) : P_m \vdash \mathbf{true}(f', m)\} \vdash \mathbf{true}(f, m+1).$$

Similarly, atoms from P_m other than $\mathbf{true}(f, m)$ do not influence entailment of $\mathbf{true}(f, m+1)$, hence we get equivalence to

$$G_m \cup A_m^{\mathtt{does}}(m) \cup \{p : P_m \vdash p\} \vdash \mathbf{true}(f, m+1).$$

Since P_m does not contain heads of $G_m \cup A_m^{\mathtt{does}}(m)$, we can apply the Splitting Theorem (Theorem 2.10) to establish equivalence to

$$G_m \cup A_m^{\mathtt{does}}(m) \cup P_m \vdash \mathbf{true}(f, m+1).$$

Since $P_{m+1} = G_m \cup A_m^{\mathtt{does}}(m) \cup P_m$, this completes the induction step and hence proves the intermediate result (3.6). For the remainder, it follows

$$G \cup S_i^{\mathtt{true}} \vdash p(\vec{t}) \text{ iff } G \cup \{\mathbf{true}(f') : P_i \vdash \mathbf{true}(f', i)\} \vdash p(\vec{t}),$$

which, by arguments similar to those for the intermediate result, is in turn equivalent to

$$G_i \cup P_i \vdash p(\vec{t}, i).$$

Case $i = m$ yields $P = G_i \cup P_i$. Case $i < m$: the unique answer sets for $G_i \cup P_i$ and $G_i \cup P_i \cup A_i^{\mathtt{does}}$ agree on the true instances of $p(\vec{t}, i)$, as $p(\vec{t}, i)$ does not depend on **does**. Since $G_i \cup P_i \cup A_i^{\mathtt{does}}$ does not contain clause heads from $P \setminus (G_i \cup P_i \cup A_i^{\mathtt{does}})$, entailment of $p(\vec{t}, i)$ is again not affected. Hence both cases $i = m$ and $i < m$ yield equivalence to

$$P \vdash p(\vec{t}, i).$$

□

The temporal GDL extension of degree $m - 1$ already incorporates clauses with head $\mathbf{true}(f, m)$ and is hence sufficient for reasoning about atoms $p(\vec{t})$ of the form $true(f)$ up to depth m. However, different atoms $p(\vec{t})$ require an extension up to degree m to include all relevant temporalised GDL clauses with head $p(\vec{t}, m)$, which motivates the occurrence of both m and $m - 1$ in program P of the theorem. Since predicate symbol **legal** never depends on **does** in a valid GDL specification and all moves in a sequence are legal, our theorem implies $P \vdash \mathbf{legal}(r, A_i(r), i)$ for all $r \in R$ and $0 \leq i \leq m - 1$, for any given sequence $S_0 \xrightarrow{A_0} S_1 \dots \xrightarrow{A_{m-1}} S_m$. Similarly, $P \vdash \mathbf{terminal}(m)$ holds if and only if S_m is a terminal state. The last step in the proof for this theorem requires **does**-independent predicate symbols p, which was the rationale behind the corresponding restriction to sequence invariants made in Definition 3.1.

3.3.2 Encoding Sequence Invariants

Next we show how game-specific knowledge in form of sequence invariants can be encoded as logic program clauses which, together with the temporal extension $G_{\leq n}$ of a valid GDL specification G, allows their formal verification against arbitrary n-max sequences. We first define the requirements for a suitable encoding and then provide an instance which satisfies these requirements.

To encode a formula to an answer set program, we need a previously unused atom with arity 0. Since encodings for severals formulas will occur in the same answer set program at a later point, these atoms are required to be unequal for syntactically different formulas. To express this, we assume a unary injective function η. For example, syntactically different formulas φ and ψ can be encoded such that $\eta(\varphi) =$ `phi` and $\eta(\psi) =$ `psi` (assuming that `phi` and `psi` do not occur elsewhere). However, in our example encoding, even syntactically identical formulas at different time levels will require different names, e.g. the two occurrences of subformula φ in formula $\varphi \supset \bigcirc\varphi$. Hence, with slight abuse of notation, we also use η to denote a binary injective function with an additional time level argument. For example, subformulas φ at different time levels can be encoded using atoms $\eta(\varphi, 0) =$ `phi0` and $\eta(\varphi, 1) =$ `phi1`. In Chapter 4, we will additionally use η with a third argument. Similarly to η, we use three versions of an injective function Enc to denote the encoding of a formula φ (by $Enc(\varphi)$), possibly with respect to a time level i (by $Enc(\varphi, i)$), and with a further argument in Chapter 4.

The following definition gives a formal classification of a formula encoding. It is based on single sequences, and requires that a formula φ is true with respect to a sequence if and only if the temporal GDL extension, together with an encoding of that sequence and an encoding of φ, yields a unique answer set which entails the unique atom $\eta(\varphi)$ corresponding to φ. Since additional encodings of formulas with possibly higher degree may occur in the same answer set program, the correspondence needs to respect a possibly higher degree of the temporalised GDL clauses and sequences.

Definition 3.10 (Encoding for Sequence Invariants). *Let $\eta(\varphi)$ be a 0-ary atom which represents a unique name for sequence invariant φ with degree n. An* encoding *of φ, denoted $Enc(\varphi)$, is a finite set of clauses whose heads do not occur elsewhere and such that, for each $\widehat{n} \geq n$ and $\widehat{n}$-max sequence $S_0 \xrightarrow{A_0} S_1 \dots \xrightarrow{A_{\widehat{m}-1}} S_{\widehat{m}}$ $(\widehat{m} \leq \widehat{n})$ of a valid GDL specification G, the program $P = S_0^{\mathtt{true}}(0) \cup G_{\leq \widehat{n}} \cup \bigcup_{i=0}^{\widehat{m}-1} A_i^{\mathtt{does}}(i) \cup Enc(\varphi)$ fulfils the following:*

- *P has exactly one answer set;*
- *$(S_0, \dots, S_{\widehat{m}}) \models \varphi$ iff $P \vdash \eta(\varphi)$.* ■

Note that Theorem 3.9 (Correctness of the Temporal GDL Extension) is also applicable to program P from Definition 3.10, as P only adds the clauses $\bigcup_{m<i\leq\widehat{n}} G_i \cup \bigcup_{i=m}^{\widehat{m}-1} A_i^{\mathtt{does}}(i) \cup Enc(\varphi)$, the heads of which do not occur in the logic program used in the theorem.

A Sample Encoding

Table 3.1 provides a recursive definition of how a sequence invariant can be encoded as a logic program: First, every atom $p(\vec{t})$ at time point i is translated to a clause which entails $\eta(p(\vec{t}), i)$ (a unique name atom for $p(\vec{t})$ at time point i) in case $p(\vec{t}, i)$ holds (case 1). Formulas with connectives different from "$\bigcirc$" recursively resolve to their correspondent subformulas (cases 2–7). Finally, $Enc(\bigcirc\psi, i)$ is constructed so as to entail $\eta(\bigcirc\psi, i)$ in case time point i is terminal or subformula ψ is true at time point $i+1$ (case 8).

Example 3.11 (Sample Encoding). Recall from page 30 the sequence invariant (3.5) for Quarto which states that if no player can place a piece now, then in the next state one player can do so. Rewriting "$\supset$" as a disjunction and removing double negation, we obtain the following equivalent formula:

$$(\exists R\colon\{r1, r2\})\, true(pctrl(R)) \vee \bigcirc(\exists R\colon\{r1, r2\})\, true(pctrl(R)).$$

Applying the recursive definition in Table 3.1 yields the following encoding, where atom `phi0` is the unique name $\eta(\varphi, 0)$ for the overall formula.

```
phi0 :- ex0.              ex0 :- a0.   a0 :- true(pctrl(r1),0).
phi0 :- nxt_ex1.          ex0 :- b0.   b0 :- true(pctrl(r2),0).
                                                                  (3.7)
nxt_ex1 :- terminal(0).   ex1 :- a1.   a1 :- true(pctrl(r1),1).
nxt_ex1 :- ex1.           ex1 :- b1.   b1 :- true(pctrl(r2),1).
```

■

It is easy to verify that the size of the encoding of a given formula is always linear in the size of the original formula. Together with the underlying temporally extended GDL specification the given encoding is correct wrt. the definition of formula entailment, as the following result shows.

Theorem 3.12 (Correctness of the Sample Encoding). *Let G be a valid GDL specification and φ be a sequence invariant. Then $Enc(\varphi) := Enc(\varphi, 0)$ with the unique name atom $\eta(\varphi) := \eta(\varphi, 0)$ for φ (cf. Table 3.1) is an encoding of φ.*

1. $Enc(p(\vec{t}), i)$	$=$	$\{\eta(p(\vec{t}), i) \texttt{ :- } p(\vec{t}, i)\,\texttt{.}\}$
2. $Enc(\neg\psi, i)$	$=$	$\{\eta(\neg\psi, i) \texttt{ :- } \textbf{not } \eta(\psi, i)\,\texttt{.}\}$ $\cup\, Enc(\psi, i)$
3. $Enc(\psi_1 \wedge \psi_2, i)$	$=$	$\{\eta(\psi_1 \wedge \psi_2, i) \texttt{ :- } \eta(\psi_1, i), \eta(\psi_2, i)\,\texttt{.}\}$ $\cup\, Enc(\psi_1, i) \,\cup\, Enc(\psi_2, i)$
4. $Enc(\psi_1 \vee \psi_2, i)$	$=$	$\{\eta(\psi_1 \vee \psi_2, i) \texttt{ :- } \eta(\psi_1, i)\,\texttt{.},$ $\eta(\psi_1 \vee \psi_2, i) \texttt{ :- } \eta(\psi_2, i)\,\texttt{.}\}$ $\cup\, Enc(\psi_1, i) \,\cup\, Enc(\psi_2, i)$
5. $Enc((\exists \vec{X} : D_{\vec{X}})\, \psi[\vec{X}], i)$	$=$	$\bigcup_{\vec{t} \in D_{\vec{X}}} \{\eta((\exists \vec{X} : D_{\vec{X}})\, \psi[\vec{X}], i) \texttt{ :- } \eta(\psi[\vec{X}/\vec{t}], i)\,\texttt{.}\}$ $\cup \bigcup_{\vec{t} \in D_{\vec{X}}} Enc(\psi[\vec{X}/\vec{t}], i)$
6. $Enc((\forall \vec{X} : D_{\vec{X}})\, \psi[\vec{X}], i)$	$=$	$\{\eta((\forall \vec{X} : D_{\vec{X}})\, \psi[\vec{X}], i) \texttt{ :- }$ $\eta(\psi[\vec{X}/\vec{t_1}], i), \ldots, \eta(\psi[\vec{X}/\vec{t_n}], i)\,\texttt{.}\}$ $\cup \bigcup_{\vec{t} \in D_{\vec{X}}} Enc(\psi[\vec{X}/\vec{t}], i)$, where $D_{\vec{X}} = \{\vec{t_1}, \ldots, \vec{t_n}\}$
7. $Enc((\exists_{l..u} \vec{X} : D_{\vec{X}})\ \psi[\vec{X}], i)$	$=$	$\{\eta((\exists_{l..u} \vec{X} : D_{\vec{X}})\ \psi[\vec{X}], i) \texttt{ :- }$ $l\{\eta(\psi[\vec{X}/\vec{t}], i) : \vec{t} \in D_{\vec{X}}\}u\,\texttt{.}\}$ $\cup \bigcup_{\vec{t} \in D_{\vec{X}}} Enc(\psi[\vec{X}/\vec{t}], i)$
8. $Enc(\bigcirc\, \psi, i)$	$=$	$\{\eta(\bigcirc\, \psi, i) \texttt{ :- } \textbf{terminal}\,(i)\,\texttt{.},$ $\eta(\bigcirc\, \psi, i) \texttt{ :- } \eta(\psi, i+1)\,\texttt{.}\}$ $\cup\, Enc(\psi, i+1)$

Table 3.1: Encoding an arbitrary formula φ as a logic program. By $\eta(\varphi, i)$ we denote a 0-ary atom providing a unique name for φ with respect to every time point i between 0 and the degree of φ.

Proof: Let $\widehat{n} \geq \deg(\varphi)$, $S_0 \xrightarrow{A_0} S_1 \ldots \xrightarrow{A_{\widehat{m}-1}} S_{\widehat{m}}$ an arbitrary $\widehat{n}$-max sequence and $P_\varphi = S_0^{\texttt{true}}(0) \cup G_{\leq \widehat{n}} \cup \bigcup_{i=0}^{\widehat{m}-1} A_i^{\texttt{does}}(i) \cup Enc(\varphi, 0)$. P_φ clearly admits a unique answer set. We prove $(S_0, \ldots, S_{\widehat{m}}) \vDash \varphi$ iff $P_\varphi \vdash \eta(\varphi, 0)$ via structural induction on φ. First note that the uniqueness of $\eta(\psi, i)$ for each formula ψ and each time step i implies $\eta(\psi, i)$ to be in the unique answer set $\mathcal{A}$ for P_φ if and only if there is a clause with head $\eta(\psi, i)$ in $Enc(\varphi, 0)$ such that its body is satisfied in $\mathcal{A}$.

Base Case $\varphi = p(\vec{t})$: $(S_0, \ldots, S_{\widehat{m}}) \vDash p(\vec{t})$ iff (by the semantics for sequence invariants in Definition 3.5) $G \cup S_0^{\texttt{true}} \vdash p(\vec{t})$ iff (by the established correctness of the temporal GDL extension in Theorem 3.9, cf. the remark following Definition 3.10) $P_\varphi \vdash p(\vec{t}, 0)$ iff $P_\varphi \vdash \eta(p(\vec{t}), 0)$.

Induction Step: The cases different from $\varphi = \bigcirc\psi$ follow by an argumentation similar to the base case, together with the induction hypothesis. Now consider formula $\varphi = \bigcirc\psi$ with degree $n + 1$.

- If S_0 is terminal: $(S_0) \vDash \bigcirc\psi$ follows by Definition 3.5, $P_\varphi \vdash \mathbf{terminal}(0)$ follows by Theorem 3.9 and yields $P_\varphi \vdash \eta(\varphi, 0)$.

- If S_0 is non-terminal then $(S_1, \ldots, S_{\widehat{m}}, S_{\widehat{m}+1})$ exists and is $\widehat{n}$-max, hence

$$(S_0, S_1, \ldots, S_{\widehat{m}}, S_{\widehat{m}+1}) \vDash \bigcirc\psi \text{ iff } (S_1, \ldots, S_{\widehat{m}}, S_{\widehat{m}+1}) \vDash \psi.$$

  Let $\cdot^{i \to i+1}$ be a renaming that replaces each time argument i by $i+1$ in timed GDL atoms and each occurrence of $\eta(\rho, i)$ by $\eta(\rho, i+1)$ for each formula ρ. Then, for program $P_\psi^{i\to i+1} = S_1^{\mathtt{true}}(1) \cup (G_{\leq \widehat{n}+1} \setminus G_0) \cup \bigcup_{i=1}^{\widehat{m}} A_i^{\mathtt{does}}(i) \cup \mathit{Enc}(\psi, 1)$, the induction hypothesis implies

$$(S_1, \ldots, S_{\widehat{m}}, S_{\widehat{m}+1}) \vDash \psi \text{ iff } P_\psi^{i\to i+1} \vdash \eta(\psi, 1).$$

  Since $S_1^{\mathtt{true}}(1) = \{\mathbf{true}(f, 1) : G_0 \cup A_0^{\mathtt{does}}(0) \cup S_0^{\mathtt{true}}(0) \vdash \mathbf{true}(f, 1)\}$ (by Definitions 2.17 and 3.7 concerning GDL semantics and temporal GDL extension) and because clause heads in $P_\psi^{i\to i+1}$ do not occur in $G_0 \cup A_0^{\mathtt{does}}(0) \cup S_0^{\mathtt{true}}(0)$ and clause heads in $G_0 \cup A_0^{\mathtt{does}}(0) \cup S_0^{\mathtt{true}}(0)$ different from atoms in $S_1^{\mathtt{true}}(1)$ do not occur in $P_\psi^{i\to i+1}$, we have that

$$P_\psi^{i\to i+1} \vdash \eta(\psi, 1) \text{ iff } S_0^{\mathtt{true}}(0) \cup G_{\leq \widehat{n}+1} \cup \bigcup_{i=0}^{\widehat{m}} A_i^{\mathtt{does}}(i) \cup \mathit{Enc}(\psi, 1) \vdash \eta(\psi, 1)$$

  This, in turn, is equivalent to $P_\varphi \vdash \eta(\varphi, 0)$ since $P_\varphi \nvdash \mathbf{terminal}$ by Theorem 3.9. □

In order to keep our framework general, in the following we abstract from our specific encoding and consider any $\mathit{Enc}(\varphi)$ that satisfies the requirements of the Encoding Definition 3.10.

3.4 Verification of Sequence Invariants

We proceed by showing how a temporally extended GDL description along with an encoding of a formula can be used to automate an induction proof for the validity of the formula. To prove that a state sequence invariant φ holds in each reachable state S (i.e., $S \vDash \varphi$), we will construct two answer set programs dependent on φ: a base case to show that φ is entailed in the initial state, and an induction step to show that, provided a state entails φ, each legal successor state will also entail φ. Together this implies that φ holds in all reachable states.

As we have seen in Section 2.2.3, fluents (i.e., state features) can grow indefinitely, hence the set which contains all ground fluents that occur in a reachable state (henceforth denoted by *FDom*) may be infinite. Consequently, also the set of all actions of a player r (henceforth denoted by $\mathit{ADom}(r)$) is potentially infinite, e.g. due to a clause like

```
legal(r,a(X)) :- true(X).
```

which defines a legal action for every fluent. In order to develop a decidable proof method for sequence invariants, we have to restrict our attention to GDL specifications that are *finite* in the sense that the associated set $FDom$ is finite. By the recursion restriction of a valid GDL description (cf. Definition 2.15) this suffices to guarantee that $ADom(r)$ is finite as well.[2]

3.4.1 Base Case

Action Generator

Based on the sets $ADom(r)$ of possible actions for player r, we follow [Thi09] to define an answer set program which encodes the fundamental requirement that each player has to perform a legal move in each non-terminal state up to time step n. Let $P^{legal}_{\leq n}$ consist of the following clauses P^{legal}_{i} for each time step $0 \leq i \leq n$ and role $r \in R$:[3]

$$\begin{array}{ll} (c_1) & \texttt{terminated}\,(i)\ \texttt{:-}\ \textbf{terminal}\,(i). \\ (c_2) & \texttt{terminated}\,(i)\ \texttt{:-}\ \texttt{terminated}\,(i-1). \\ (c_3) & 1\{\textbf{does}\,(r,a,i) : a \in ADom(r)\}1\ \texttt{:-}\ \textbf{not}\ \texttt{terminated}\,(i). \\ (c_4) & \texttt{:-}\ \textbf{does}\,(r,\texttt{A},i),\ \textbf{not}\ \textbf{legal}\,(r,\texttt{A},i). \end{array} \tag{3.8}$$

Clause (c_3) states that each player performs exactly one move in each non-terminal state, and clause (c_4) ensures that each of the performed moves is legal. Since the GDL possibly yields legal joint moves even in terminal states, and keyword **terminal** is not necessarily entailed in pseudo states that are obtained by performing these moves, terminal states at time step i cannot just be referred to via **terminal**(i) in (c_3). Instead, additional atoms `terminated`(i) are defined in clauses (c_1) and (c_2) to indicate all time steps up to n which match or exceed the time step of a terminal state. Subsequently, $P^{legal}_{\leq n}$ will also be called an *action generator*.

Base Case Program

For a game description G and a formula φ over G with degree n, the answer set program for the *base case* is defined as follows:

$$P^{bc}_{\varphi}(G) = S^{\texttt{true}}_{init}(0) \cup G_{\leq n} \cup P^{legal}_{\leq n-1} \cup Enc(\varphi) \cup \{\ \texttt{:-}\ \eta(\varphi)\,.\}$$

Put in words, $P^{bc}_{\varphi}(G)$ consists of an encoding for the initial state, $S^{\texttt{true}}_{init}(0)$; a temporal GDL specification up to time step n, $G_{\leq n}$; the necessary requirements concerning legal moves, $P^{legal}_{\leq n-1}$; an encoding for the formula φ, $Enc(\varphi)$; and the statement that φ should not be entailed in any answer set of $P^{bc}_{\varphi}(G)$, $\{\ \texttt{:-}\ \eta(\varphi)\,.\}$. In case $P^{bc}_{\varphi}(G)$ has no answer set, the last clause implies that there is no state sequence starting at S_{init} that makes φ false. Equivalently, each state sequence starting at S_{init} satisfies φ—which means that φ is entailed by S_{init}.

3.4.2 Induction Step

State Generator

For the induction step answer set program, the state encoding $S^{\texttt{true}}_{init}(0)$ needs to be substituted by a general "state generator" program, whose answer sets produce the

[2]A detailed discussion on how to reliably compute both $FDom$ and $ADom$ will follow in Section 5.1.

[3]We tacitly assume that predicate symbol `terminated` does not occur elsewhere.

reachable states of a GDL game. In general, the computation of the reachable states requires a full game tree traversal which is not feasible in interesting games (e.g., the game tree of chess is estimated with about 10^{45} states). This motivates the use of an easy approximation that may comprise unreachable states as well. The simplest such approximation is the program which solely consists of the weight atom

$$0 \; \{\, \mathbf{true}(f, 0) : \; f \in \mathit{FDom}\}.$$

which generates *all* combinations of fluents, whether reachable or not. In general, we define a state generator as a program which for each reachable state admits an answer set representing that state, and which may yield additional answer sets corresponding to some non-reachable states.

Definition 3.13 (State Generator)**.** *A* state generator *for a valid GDL specification G is an answer set program P^{gen} such that*

- *The only atoms in P^{gen} are of the form $\mathbf{true}(f, 0)$, where $f \in \Sigma$, or auxiliary atoms that do not occur elsewhere; and*
- *for every reachable state S of G, P^{gen} has an answer set $\mathcal{A}$ such that for all $f \in \Sigma$: $\mathbf{true}(f, 0) \in \mathcal{A}$ iff $f \in S$.* ■

The practical necessity for using a superset of the reachable states in the induction step has interesting consequences, which are best seen with an example. Suppose we want to prove the Quarto sequence invariant which states that each cell contains at most one piece, that is, $\varphi = (\forall X, Y : I)\,(\exists_{0..1} P : D_P)\; \mathit{true}(\mathit{cell}(X, Y, P))$ (cf. (3.4)). The (unreachable!) state

$$S = \{\mathit{cell}(1, 1, b), \mathit{selected}(p0000), \mathit{selected}(p0001), \mathit{pctrl}(r1), \mathit{pctrl}(r2)\}$$

satisfies φ. In S, players $r1$ and $r2$ both have the legal move of placing a selected piece at cell $(1, 1)$. Consider, then, the case where they choose to place different pieces. This results in the successor state

$$\{\mathit{cell}(1, 1, p0000), \mathit{cell}(1, 1, p0001), \mathit{sctrl}(r1), \mathit{sctrl}(r2)\}$$

which violates φ. Hence, there is an undesired counterexample for the induction step as long as S is considered potentially reachable. However, knowing that sequence invariant

$$(\exists_{1..1} C : \{\mathit{sctrl}(r1), \mathit{sctrl}(r2), \mathit{pctrl}(r1), \mathit{pctrl}(r2)\})\; \mathit{true}(C)$$

holds in all reachable states of Quarto allows to reject S and all similar states that contain more than one "control" fluent. As a consequence, none of the direct successors of the remaining φ-satisfying states violates φ—which establishes a successful proof of the induction step. This shows that the addition of all previously proved sequence invariants to a state generator can positively influence the outcome of a subsequent proof attempt. The following construction of the answer set program for the induction step of a proof accounts for this issue by the inclusion of formulas which are already known to be valid.

Induction Step Program

For a game description G, an arbitrary state generator P^{gen} over G, a set Ψ of valid sequence invariants that have at most degree n_Ψ, a formula φ with degree n_φ, and $\widehat{n} = \max(n_\Psi, n_\varphi + 1)$, the *induction step* answer set program is

$$\begin{aligned} P^{is}_{\varphi,\Psi}(G) \quad = \quad & P^{gen} \cup G_{\leq \widehat{n}} \cup P^{legal}_{\leq \widehat{n}-1} \cup Enc(\varphi \supset \bigcirc\varphi) \cup \{ \texttt{:-}\ \eta(\varphi \supset \bigcirc\varphi).\ \} \cup \\ & \textstyle\bigcup_{\psi\in\Psi}(Enc(\psi) \cup \{ \texttt{:-}\ \textbf{not}\ \eta(\psi).\ \}). \end{aligned}$$

Put in words, $P^{is}_{\varphi,\Psi}(G)$ differs from $P^{bc}_{\varphi}(G)$ in the following way. First, an arbitrary state generator P^{gen} is used instead of the initial-state encoding. Second, the time horizon has increased from n to $\widehat{n}$. Third, formulas are now encoded thus: $\bigcup_{\psi\in\Psi}(Enc(\psi) \cup \{ \texttt{:-}\ \textbf{not}\,\eta(\psi)\texttt{.}\})$, which ensures that $P^{is}_{\varphi,\Psi}(G)$ has only answer sets that, for all formulas $\psi \in \Psi$, contain $\eta(\psi)$ and hence represent $\widehat{n}$-max sequences which satisfy ψ. These sequences still include all reachable $\widehat{n}$-max sequences and are, by the clauses $Enc(\varphi \supset \bigcirc\varphi) \cup \{ \texttt{:-}\ \eta(\varphi \supset \bigcirc\varphi)\texttt{.}\}$, further restricted to those which satisfy $\neg(\varphi \supset \bigcirc\varphi)$. In case $P^{is}_{\varphi,\Psi}(G)$ is inconsistent, there is no reachable $\widehat{n}$-max sequence which satisfies $\neg(\varphi \supset \bigcirc\varphi)$. Equivalently, each reachable $\widehat{n}$-max sequence satisfies $(\varphi \supset \bigcirc\varphi)$—which implies that φ is satisfied in all direct successors of reachable states that themselves satisfy φ.

Remark on the Linear-Time Encoding of Formulas

The induction step proof for a formula φ with degree n requires an encoding of the induction hypothesis $S_0 \vDash \varphi$. In case the degree n of the formula is greater than 0, this condition refers to a treelike structure which cannot reliably be represented in the linear time structure we apply. More precisely, an accurate counter example for the induction step proof is a state S_0 such that

1. *all* n-max sequences starting at S_0 satisfy φ, and

2. S_0 admits one direct successor state S_1 such that one n-max sequence starting at S_1 violates φ.

In our case, however, the first condition is altered in that a counter example is a state S_0 such that a *single* n-max sequence starting at S_0 satisfies φ. This is also demonstrated in Figure 3.1. By considering only one instead of all such sequences, we apply a weaker induction hypothesis that does not neglect some of the states S_0 which violate condition 1 (and are hence no counter example). Hence, the linear time encoding of the induction hypothesis may introduce counter examples for actually valid formulas φ for non-reachable states S_0 [4], which is emphasised with the following small example:

```
init(f).
role(r).

next(f) :- does(r,a).
```

[4]Indeed, no introduced counter example can be based on a *reachable* state S_0, as the validity of φ implies $S_0 \vDash \varphi$, which in turn implies that S_0 satisfies condition 1.

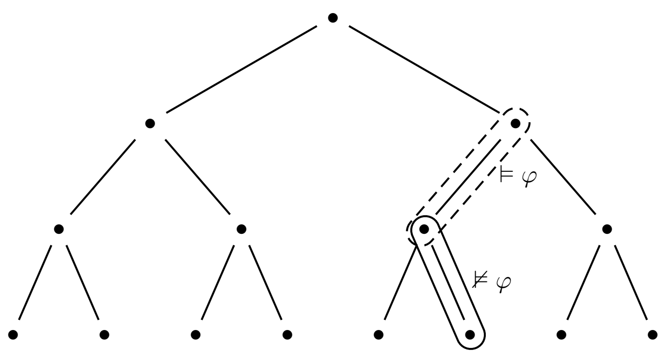

Figure 3.1: A graphical representation for a counter example of formula φ with degree 1 in a schematic game tree. The dashed line marks a 1-max sequence which satisfies φ (for the induction hypothesis), the solid line marks a 1-max sequence which violates φ. Put together, both sequences form a 2-max sequence σ such that $\sigma \models \varphi \wedge \neg \bigcirc \varphi$, and hence such that $\sigma \not\models \varphi \supset \bigcirc\varphi$.

```
next(g) :- true(g).

legal(r,a) :- true(f).
legal(r,b) :- true(g).
```

Since f holds initially and g does not hold initially, only action a is applicable for player r in the initial state, and yields exactly one possible successor state that equals the initial state. Consequently, formula $\varphi = \mathit{true}(f) \supset \bigcirc\mathit{true}(f)$ holds in all reachable states. However, lacking the information that state $\{f, g\}$ is non-reachable, the following is a counter example for the induction step in our setting:

$$\sigma = \{f, g\} \xrightarrow{\{r\,:\,a\}} \{f, g\} \xrightarrow{\{r\,:\,b\}} \{g\}$$

Now subsequence $\{f, g\} \xrightarrow{\{r\,:\,a\}} \{f, g\}$ of σ satisfies φ, representing our linearly encoded induction hypothesis. However, beginning at the first state $\{f, g\}$ of σ, there is another sequence which does not satisfy φ, namely $\{f, g\} \xrightarrow{\{(r,b)\}} \{g\}$. Hence, this counter example could be sorted out when using the stronger induction hypothesis which comprises *all* relevant sequences emerging from state $\{f, g\}$, and φ could be proved. In our linear setting, this counter example can be removed by previously proved formulas as motivated in Section 3.4.2, in this case a preceding proof of the valid formula $\neg\mathit{true}(g)$. Note that the linear-time encoding also influences formulas with a degree greater than 0 from the incorporated set of previously proved formulas Ψ.

3.4.3 Example

In this section we exemplarily demonstrate how a formula can be proved to hold in all reachable states. To this end, recall the example encoding (3.7) for the Quarto state invariant

$$\varphi = \neg(\exists R\colon\{r1, r2\})\,\mathit{true}(\mathit{pctrl}(R)) \supset \bigcirc(\exists R\colon\{r1, r2\})\,\mathit{true}(\mathit{pctrl}(R)).$$

Base Case. Let G be the clauses in Figure 2.2 at page 19. Since the initial state contains $\mathit{sctrl}(r1)$, the temporal extension of clause 17 in Figure 2.2 implies that the atom $\mathbf{true}(\texttt{pctrl(r2)}, 1)$ is contained in each answer set of $P^{bc}_{\varphi}(G)$. Consequently, the example encoding for φ implies that also $\eta(\varphi, 0) = \texttt{phi0}$ is contained. This however contradicts the constraint `:- phi0.` in $P^{bc}_{\varphi}(G)$, so $P^{bc}_{\varphi}(G)$ has no answer set and, thus, φ holds in the initial state of the game.

Induction Step. The induction step program $P^{is}_{\varphi,\Psi}(G)$ contains $\mathit{Enc}(\varphi \supset \bigcirc\varphi, 0) \cup \{\ \texttt{:-}\ \eta(\varphi \supset \bigcirc\varphi)\texttt{.}\}$, where $\mathit{Enc}(\varphi \supset \bigcirc\varphi, 0)$ can be specified such as to contain

- the encoding $\mathit{Enc}(\varphi, 0)$ for φ as given in (3.7), where $\eta(\varphi, 0) = \texttt{phi0}$;
- an additional set $\mathit{Enc}(\varphi, 1)$ that differs from $\mathit{Enc}(\varphi, 0)$ only in the used name atoms and time points increased by 1, where we specify $\eta(\varphi, 1) = \texttt{phi1}$; and
- the following additional clauses, where $\eta(\varphi \supset \bigcirc\varphi) = \texttt{phi\_imp\_nxt\_phi}$:

```
phi_imp_nxt_phi :- neg_phi.     neg_phi :- not phi0.
phi_imp_nxt_phi :- nxt_phi.

nxt_phi :- terminal(0).
nxt_phi :- phi1.
```

The specified encoding together with the constraint `:- phi_imp_nxt_phi.` implies that each of the answer sets $\mathcal{A}$ of $P^{is}_{\varphi,\Psi}(G)$ satisfies the following three conditions:

1. $\texttt{phi0} \in \mathcal{A}$,
2. $\mathbf{terminal}(0) \notin \mathcal{A}$, and
3. $\texttt{phi1} \notin \mathcal{A}$.

Now let r range over $\{\mathit{r1}, \mathit{r2}\}$. The body of one clause with head `phi0` of $\mathit{Enc}(\varphi, 0) \subseteq \mathit{Enc}(\varphi \supset \bigcirc\varphi, 0)$ must be true in $\mathcal{A}$ due to the first condition and cannot be satisfied by $\mathbf{terminal}(0)$ due to the second condition. Hence, for some r either $\mathbf{true}(\texttt{pctrl}(r), 0) \in \mathcal{A}$ or $\mathbf{true}(\texttt{pctrl}(r), 1) \in \mathcal{A}$. Furthermore, since $\mathit{Enc}(\varphi, 1) \subseteq \mathit{Enc}(\varphi \supset \bigcirc\varphi, 0)$, the third condition implies that $\mathbf{true}(\texttt{pctrl}(r), 1) \notin \mathcal{A}$ and $\mathbf{true}(\texttt{pctrl}(r), 2) \notin \mathcal{A}$ for each r (and that $\mathbf{terminal}(1) \notin \mathcal{A}$).

Hence, there must be some r such that $\mathbf{true}(\texttt{pctrl}(r), 0) \in \mathcal{A}$. By the temporal extension of clauses 16 and 17 in Figure 2.2, this results in existence of an r such that $\mathbf{true}(\texttt{pctrl}(r), 2) \in \mathcal{A}$, in contradiction to the previously mentioned implications of the third condition. Thus, $P^{is}_{\varphi,\Psi}(G)$ has no answer set, which implies that φ is satisfied in all direct successors of reachable states that themselves satisfy φ.

Note that for this example argumentation the applied state generator in $P^{is}_{\varphi,\Psi}(G)$ is completely unimportant. This implies that the example formula φ can be proved with *any* state generator, even if it is completely uninformed and needs to consider any finite set of fluents a possible state.

3.5 Properties of the Verification Method

The following result is a prerequisite for the soundness and completeness proofs of the verification method introduced in the previous section. It provides a one-to-one relation between answer set programs encoding a particular state sequence and those including an action generator.

Theorem 3.14 (Answer Set Correspondence). *Let G be a valid GDL specification and $\mathcal{A}$ be a subset of the ground atoms over G together with $\{\texttt{terminated}(i) : i \in \mathbb{N}\}$. The following two statements are equivalent:*

(1) $\mathcal{A}$ is an answer set for the program

$$P = S_0^{\texttt{true}}(0) \cup G_{\leq n} \cup P_{\leq n-1}^{legal}.$$

(2) There is an n-max sequence $\sigma = (S_0 \xrightarrow{A_0} S_1 \dots \xrightarrow{A_{m-1}} S_m)$ such that $\mathcal{A}$ is the unique answer set for the program

$$P^\sigma = S_0^{\texttt{true}}(0) \cup G_{\leq n} \cup P_{\leq n-1}^{c_1,c_2} \cup \bigcup_{i=0}^{m-1} A_i^{\texttt{does}}(i),$$

where $P_{\leq n-1}^{c_1,c_2} = \bigcup_{i=0}^{n-1} P_i^{c_1,c_2}$ and $P_i^{c_1,c_2}$ denotes all clauses of the shape (c_1) and (c_2) in the part P_i^{legal} of the action generator, defined as (3.8) at page 40.

Proof:

<u>(2) $\Rightarrow$ (1)</u>: First we show that $\mathcal{A}$ satisfies P. Since P differs from P^σ only by containing clauses of the shape (c_3) and (c_4) (defined as part of the action generator in (3.8) at page 40) instead of $\bigcup_{i=0}^{m-1} A_i^{\texttt{does}}(i)$, this follows if $\mathcal{A}$ satisfies all clauses (c_3) and (c_4) for $0 \leq i \leq n-1$.

- Time steps $0 \leq i < m$: for each player r there is exactly one action a such that $\mathbf{does}(r, a, i) \in \mathcal{A}$, namely $a = A_i(r)$; and $\mathbf{legal}(r, a, i) \in \mathcal{A}$ follows by definition of σ and Theorem 3.9 (Correctness of the Temporal GDL Extension). This satisfies the clauses (c_3) and (c_4) for $0 \leq i < m$.

- Time steps $m \leq i \leq n-1$: If one such i exists, then $m < n$ and hence S_m is terminal, which implies $\mathbf{terminal} \in \mathcal{A}$ (again by Theorem 3.9) and thus $\{\texttt{terminated}(j) : m \leq j \leq n-1\} \subseteq \mathcal{A}$. This satisfies the clauses (c_3) and (c_4) for $m \leq i \leq n-1$.

Now $\mathcal{A}$ is also an answer set for the program constructed from P by omitting constraints (c_4), since its reduct coincides with the reduct of P^σ. By the previous argumentation $\mathcal{A}$ satisfies all constraints (c_4) and hence is also an answer set for P.

<u>(1) $\Rightarrow$ (2)</u>: Let G_n^{dep} be the clauses from G_n whose heads depend on **does**, and let $G_n^{\overline{dep}}$ be all others. By induction on n, we prove that if

$$P_n = S_0^{\texttt{true}}(0) \cup G_{\leq n-1} \cup G_n^{\overline{dep}} \cup P_{\leq n-1}^{legal}$$

has answer set $\mathcal{A}_n$, then there is an n-max sequence $\sigma = (S_0 \xrightarrow{A_0} S_1 \ldots \xrightarrow{A_{m-1}} S_m)$ such that $\mathcal{A}_n$ is the unique answer set for

$$P_n^\sigma = S_0^{\texttt{true}}(0) \cup G_{\leq n-1} \cup \overline{G_n^{dep}} \cup P_{\leq n-1}^{c_1,c_2} \cup \bigcup_{i=0}^{m-1} A_i^{\texttt{does}}(i).$$

This implies the claim for $P = P_n \cup G_n^{dep}$ and $P^\sigma = P_n^\sigma \cup G_n^{dep}$ by the Splitting Theorem 2.10, since P_n and P_n^σ do not contain heads of G_n^{dep}. For the Base Case $n = 0$ the two programs coincide. Induction Step: Assume that P_{n+1} has answer set $\mathcal{A}_{n+1}$. Since $P_{n+1} = P_n \cup G_n^{dep} \cup \overline{G_{n+1}^{dep}} \cup P_n^{legal}$, P_n does not contain heads of $P_{n+1} \setminus P_n$, hence (by Theorem 2.10) P_n has an answer set $\mathcal{A}_n$ such that $\mathcal{A}_n \subseteq \mathcal{A}_{n+1}$. By the induction hypothesis there is an n-max sequence $\sigma = (S_0 \xrightarrow{A_0} S_1 \ldots \xrightarrow{A_{m-1}} S_m)$ such that $\mathcal{A}_n$ is the unique answer set for P_n^σ. We consider two cases:

- S_m terminal: σ is also $(n+1)$-max. We have $\mathbf{terminal}(m) \in \mathcal{A}_n$ (by Theorem 3.9) and hence $\{\texttt{terminated}(i) : m \leq i \leq n\} \subseteq \mathcal{A}_{n+1}$. This implies that $\mathcal{A}_{n+1}$ does not contain any instance $\mathbf{does}(r, a, n)$, hence $\mathcal{A}_{n+1}$ is also an answer set for P_{n+1}^σ.

- S_m non-terminal: Then $m = n$, hence $\mathbf{terminal}(i) \notin \mathcal{A}_n$ for all $0 \leq i \leq n$ (by Theorem 3.9) and hence $\texttt{terminated}(n) \notin \mathcal{A}_{n+1}$. By (c_3) and (c_4) there is a mapping A_n such that for each $r \in R$ there is exactly one a such that $\{\mathbf{legal}(r, a, n), \mathbf{does}(r, a, n)\} \subseteq \mathcal{A}_{n+1}$. All $\mathbf{legal}(r, a, n)$ must also be in $\mathcal{A}_n$ (as in $P_{n+1} \setminus P_n$ these heads do not exist) and hence (again by Theorem 3.9) we have $S_n \xrightarrow{A_n} S_{n+1}$ for some S_{n+1}. In this case $\sigma' = (S_0, \ldots, S_n, S_{n+1})$ is $(n+1)$-max. By construction, $\mathcal{A}_{n+1}$ is the unique answer set for $P_{n+1}^{\sigma'}$. $\square$

3.5.1 Soundness

The following theorem states the soundness of the verification method.

Theorem 3.15 (Soundness). *Let G be a playable and valid GDL specification whose initial state is S_{init}. Let Ψ be a set of sequence invariants over G which are satisfied in all reachable states, and let φ be a sequence invariant. If $P_\varphi^{bc}(G)$ and $P_{\varphi,\Psi}^{is}(G)$ are inconsistent, then for all finite developments $(S_{init}, S_1, \ldots, S_k)$ we have $S_k \models \varphi$.*

Proof: Let $\deg(\varphi) = n$. The proof is via induction on k. For the base case, we prove that $P_\varphi^{bc}(G)$ being inconsistent implies $S_{init} \models \varphi$. For the induction step, we prove that if there are S_k, A_k, and S_{k+1} such that $S_k \models \varphi$ and $S_k \xrightarrow{A_k} S_{k+1}$, then $P_{\varphi,\Psi}^{is}(G)$ being inconsistent implies $S_{k+1} \models \varphi$.

Base Case: We prove that if $S_{init} \not\models \varphi$, then $P_\varphi^{bc}(G)$ admits an answer set.

$S_{init} \not\models \varphi$ implies that there is an n-max development $\sigma = S_{init} \xrightarrow{A_0} S_1 \ldots \xrightarrow{A_{m-1}} S_m$ such that $(S_{init}, S_1, \ldots, S_m) \not\models \varphi$. Now let P^σ and P be as in the Answer Set Correspondence Theorem 3.14 (where $S_{init} = S_0$). $P^\sigma \cup \mathit{Enc}(\varphi)$ admits a unique answer set $\mathcal{A}$. By the Encoding Definition 3.10 we have $\eta(\varphi) \notin \mathcal{A}$, hence $\mathcal{A}$ is also the unique answer set for $P^\sigma \cup \mathit{Enc}(\varphi) \cup \{\texttt{ :- } \eta(\varphi)\texttt{.}\}$. P^σ and $P = P_\varphi^{bc}(G) \setminus (\mathit{Enc}(\varphi) \cup \{\texttt{ :- } \eta(\varphi)\texttt{.}\})$ do not contain heads of $\mathit{Enc}(\varphi) \cup \{\texttt{ :- } \eta(\varphi)\texttt{.}\}$, hence by the Splitting Theorem 2.10 and Theorem 3.14, $\mathcal{A}$ is also an answer set for $P_\varphi^{bc}(G)$.

Induction Step: Let $\widehat{n} = \max(n_\Psi, n+1)$ for the maximal degree n_Ψ of formulas in Ψ. Assume $S_k \xrightarrow{A_k} S_{k+1}$ for some A_k and S_{k+1}. We prove that if $S_{k+1} \not\models \varphi$, then $P^{is}_{\varphi,\Psi}(G)$ admits an answer set.

$S_{k+1} \not\models \varphi$ implies that there is an n-max sequence $S_{k+1} \xrightarrow{A_{k+1}} S_{k+2} \dots \xrightarrow{A_{k+m}} S_{k+m+1}$ (where $0 \leq m \leq n$) such that $(S_{k+1}, \dots, S_{k+m+1}) \not\models \varphi$. It follows that $\sigma_{n+1} = S_k \xrightarrow{A_k} S_{k+1} \xrightarrow{A_{k+1}} S_{k+2} \dots \xrightarrow{A_{k+m}} S_{k+m+1}$ is $(n+1)$-max and that $\sigma_{n+1} \not\models \bigcirc\varphi$. Furthermore, by the induction hypothesis we have $S_k \models \varphi$ and hence also $\sigma_{n+1} \models \varphi$ by the Sequence Extension Proposition 3.6. These arguments imply that $\sigma_{n+1} \not\models \varphi \supset \bigcirc\varphi$.

Since S_k is reachable and the GDL specification is playable, σ_{n+1} can be extended to an $\widehat{n}$-max sequence $\sigma_{\widehat{n}}$ by Proposition 3.6 such that $\sigma_{\widehat{n}} \not\models \varphi \supset \bigcirc\varphi$, and S_k satisfying each $\psi \in \Psi$ also implies $\sigma_{\widehat{n}} \models \psi$. An argumentation similar to the base case—considering $\varphi \supset \bigcirc\varphi$ instead of φ, $\widehat{n}$ instead of n, $\sigma_{\widehat{n}}$ instead of σ, and the additional subprogram $\bigcup_{\psi\in\Psi}(\mathit{Enc}(\psi) \cup \{\texttt{ :- } \mathbf{not}\, \eta(\psi)\,\texttt{.}\})$—implies existence of an answer set $\mathcal{A}$ for $(P^{is}_{\varphi,\Psi}(G) \setminus P^{gen}) \cup S_k^{\texttt{true}}(0)$. Now $P^{is}_{\varphi,\Psi}(G)$ is obtained by exchanging $S_k^{\texttt{true}}(0)$ with the state generator P^{gen}, which (by reachability of S_k, Definition 3.13 of a state generator, and the Splitting Theorem 2.10) in turn implies existence of an answer set. $\square$

Note that the playability assumption of the GDL specification in Theorem 3.15 can be omitted in case $n_\Psi \leq \deg(\varphi) + 1$ for the maximal degree n_Ψ of formulas in Ψ, since then the induction step proof does not require an extension of sequence σ_{n+1} according to Proposition 3.6.

3.5.2 Restricted Completeness

Since the set of reachable states is hard to compute in interesting games, our proof method allows to specify an easily obtainable superset of the reachable states in the induction step proof. This is realised with the notion of a state-generator program in Section 3.4.2, which is only assumed to provide a corresponding answer set for each reachable state, and hence might yield additional answer sets which correspond to non-reachable states. We have pointed out that this relaxation has the unintended side effect of introducing counter examples for formulas which are actually valid, rendering the proof method incomplete. In the following, we show that this is the *only* source for unintended counter examples. To this end, let us introduce a class of state generators which do not yield answer sets corresponding to non-reachable states.

Definition 3.16 (Accurate State Generator). *A state generator P^{gen} over a valid GDL specification is called* accurate *if, for every answer set $\mathcal{A}$ of P^{gen}, there is a reachable state S such that for all $f \in \Sigma$: $\mathbf{true}(f, 0) \in \mathcal{A}$ iff $f \in S$.* $\blacksquare$

Assuming an accurate state generator, the proof method can now be proved complete as follows.

Theorem 3.17 (Restricted Completeness). *Let G be a valid GDL specification whose initial state is S_{init}. Let Ψ be a set of sequence invariants over G which are satisfied in all reachable states, and let φ be a sequence invariant. Moreover, let $P^{is}_{\varphi,\Psi}(G)$ be constructed over an accurate state generator. If for all finite developments $(S_{init}, S_1, \dots, S_k)$ we have $S_k \models \varphi$, then $P^{bc}_{\varphi}(G)$ and $P^{is}_{\varphi,\Psi}(G)$ are inconsistent.*

Proof: We prove that if $P^{bc}_{\varphi}(G)$ or $P^{is}_{\varphi,\Psi}(G)$ admits an answer set, then there is a reachable state S such that $S \not\models \varphi$. Consider the following cases:

- If $P^{bc}_{\varphi}(G)$ admits an answer set $\mathcal{A}$, then $\mathcal{A}$ is also an answer set for $P^{bc}_{\varphi}(G) \setminus \{\ \texttt{:-}\ \eta(\varphi)\texttt{.}\}$, and $\mathcal{A}$ does not contain $\eta(\varphi)$. Since the heads of $\mathit{Enc}(\varphi)$ do not occur elsewhere, the Answer Set Correspondence Theorem 3.14 can be applied to $P^{bc}_{\varphi}(G) \setminus \{\ \texttt{:-}\ \eta(\varphi)\texttt{.}\}$ (using the Splitting Theorem 2.10) to conclude existence of a deg(φ)-max development $\sigma = (S_{init}, S_1, \ldots, S_m)$ such that $\mathcal{A}$ is also an answer set for the program $P^{\sigma} \cup \mathit{Enc}(\varphi)$, where P^{σ} is as in Theorem 3.14 (with $S_0 := S_{init}$, $n := \deg(\varphi)$, and where σ starts in S_{init}). By the definition of an encoding (Definition 3.10), we have that $\sigma \not\models \varphi$ and hence that $S_{init} \not\models \varphi$.
- If $P^{is}_{\varphi,\Psi}(G)$ admits an answer set, then the set of all fluents f for which $\mathbf{true}(f, 0)$ is contained in this answer set is a reachable state, as instances $\mathbf{true}(f, 0)$ can only result from the accurate state generator P^{gen}. Hence, replacing P^{gen} with the program $S_0^{\texttt{true}}(0)$ that also represents this state does not influence answer set existence by the Splitting Theorem 2.10, which implies that $(P^{is}_{\varphi,\Psi}(G) \setminus P^{gen}) \cup S_0^{\texttt{true}}$ admits an answer set $\mathcal{A}$. An argumentation similar to the previous item—considering $\varphi \supset \bigcirc\varphi$ instead of φ, $\widehat{n} = \max(n_{\Psi}, n+1)$ instead of deg(φ), S_0 instead of S_{init}, and the additional subprogram $\bigcup_{\psi\in\Psi}(\mathit{Enc}(\psi) \cup \{\ \texttt{:-}\ \mathbf{not}\, \eta(\psi)\texttt{.}\})$—implies existence of an $\widehat{n}$-max sequence $\sigma = (S_0, S_1, \ldots, S_{\widehat{m}})$ such that $\sigma \not\models \varphi \supset \bigcirc\varphi$. It follows that $(S_1, \ldots, S_{\widehat{m}}) \not\models \varphi$, and hence that $S_1 \not\models \varphi$ (by the Sequence Extension Proposition 3.6). □

It is easy to see that the accuracy of the proof method increases when state generators are restricted successively by removing some of their answer sets which correspond to non-reachable states, e.g. in case new information has been obtained by further game analysis or foregoing proofs of state sequence invariants. The completeness result provides an important implication in this setting: it shows that the verification method converges to perfect results and that it hence is reliable. In practice however, the strong assumption of an accurate state generator can hardly be met. We dedicate Section 5.4.1 to experiments which show that the proof method is nevertheless effective.

3.5.3 Sound and Complete Verification at Fixed Depth

The established soundness and completeness results yield the following interesting implication regarding the base case program $P^{bc}_{\varphi}(G)$. It states that the program construction for the base case can be used for the sound and complete verification of a formula with respect to all states at one given depth of the game tree. Now and in the following, we will use $\bigcirc^n$ to abbreviate n consecutive $\bigcirc$-operators.

Proposition 3.18 (Correctness on Single States)**.** *Let G be a valid GDL specification, φ be a sequence invariant over G, and let $t \in \mathbb{N}$.*

$$P^{bc}_{\bigcirc^t\varphi}(G) \textit{ is inconsistent } \quad \textit{iff} \quad \textit{for all developments } \delta \in \Delta_G \textit{ s.t.} |\delta| = t : \mathit{last}(\delta) \models \varphi$$

Proof: $\Rightarrow$: Suppose that there is a development $\delta = (S_{init}, S_1, \ldots, S_t)$ and a deg(φ)-max sequence $\sigma = (S_t, \ldots, S_{t+m})$ such that $\sigma \not\models \varphi$. Clearly, for the composed development $\delta' = (S_{init}, S_1, \ldots, S_t, \ldots, S_{t+m})$ we have that $\delta' \not\models \bigcirc^t\varphi$, and hence that

$S_{init} \not\models \bigcirc^t \varphi$ (since δ' is $\deg(\bigcirc^t\varphi)$-max). Using the argumentation of the base case in the proof of the Soundness Theorem 3.15, we obtain that $P^{bc}_{\bigcirc^t\varphi}(G)$ admits an answer set.

$\Leftarrow$: Suppose that $P^{bc}_{\bigcirc^t\varphi}(G)$ admits an answer set. By the argumentation concerning the base case program in the Completeness Theorem 3.17, there is a $\deg(\bigcirc^t\varphi)$-max development δ' such that $\delta' \not\models \bigcirc^t\varphi$. Case $|\delta'| < t$ yields a contradiction, as δ' being $\deg(\bigcirc^t\varphi)$-max then implies that δ' is terminated and hence that $\delta' \models \bigcirc^t\varphi$. Hence, $|\delta'| \geq t$, which yields that δ' has the form $\delta' = (S_{init}, S_1, \ldots, S_t, \ldots, S_{t+m})$ for some states $S_1, \ldots, S_t, \ldots, S_{t+m}$. Subsequence $(S_t, \ldots, S_{t+m})$ is $\deg(\varphi)$-max (since δ' is $\deg(\bigcirc^t\varphi)$-max) and such that $(S_t, \ldots, S_{t+m}) \not\models \varphi$ (since $\delta' \not\models \bigcirc^t\varphi$). □

Hence, the base case construction of our proof method can also be applied as a sound and complete check of whether a formula φ is entailed by all reachable states of a game in arbitrary depth t of the game tree. In Section 3.6.1 we show how this result can be applied to solve single-player games. In practice, this proposition is of course useful for small depths t of the game tree only. In order to solve $P^{bc}_{\bigcirc^t\varphi}(G)$, an answer set solver has to perform a complete search in the partial game tree up to depth $t + \deg(\varphi)$ in the worst case, as $P^{bc}_{\bigcirc^t\varphi}(G)$ does not involve any short cuts such as the local search in combination with an induction argument that we apply in our induction method. On the other side, the base case program does not need the strong assumption of an accurate state generator to be complete.

Note that Proposition 3.18 allows to conclude that a formula is *not* valid whenever the base case program for this formula admits an answer set. This conclusion cannot be drawn from an answer set for the induction step program (unless the state generator is known to be accurate), as the answer set could correspond to a non-reachable state and the formula hence nevertheless be valid.

3.6 Improvements

In this section, we discuss several modifications and extensions of our proof method which allow to solve single-player games (Section 3.6.1), to prove multiple properties using only two generated answer set programs (Section 3.6.2), to exhaustively prove properties from a given set (Section 3.6.3), and to handle non-playable games (Section 3.6.4).

3.6.1 Solving Single-Player Games

We will now show that our verification approach for state sequence invariants is capable of solving single-player games. Let G be a valid GDL specification with only one player r. Then the goal for r is to find a sequence of own legal moves which yields a terminal state with the maximal outcome of 100 points. More formally, the goal is to find a sequence $S_{init} \xrightarrow{A_0} S_1 \ldots \xrightarrow{A_{m-1}} S_k$ such that S_k satisfies the formula

$$\psi = \mathit{terminal} \wedge \mathit{goal}(r, 100).$$

Naturally, ψ is not valid unless the initial state S_{init} itself satisfies ψ (which happens only if the game is terminated in the initial state and would hence be trivial to solve), hence our induction proof method is not readily applicable to solve single-player games.

Algorithm 3.1 Prove Weak Winnability for player r

```
Input: $G$ - a valid GDL specification, $r$ - a player from $G$
  $\rho := \neg(terminal \wedge goal(r, 100))$
  while $true$ do
    if $P^{bc}_{\rho}(G)$ admits answer set $\mathcal{A}$ then
      return "the game is weakly winnable"
    else
      $\rho := \bigcirc\rho$
    end if
  end while
```

However, given a maximal time horizon t, we can state a formula φ which expresses that the goal ψ of r will *not* be achieved within t moves :

$$\varphi = (\bigcirc^0\neg\psi) \wedge (\bigcirc^1\neg\psi) \wedge \ldots \wedge (\bigcirc^t\neg\psi)$$

Now we construct the base case program $P^{bc}_{\varphi}(G)$ for φ. If $P^{bc}_{\varphi}(G)$ is inconsistent, then (by Proposition 3.18) $S_{init} \models \varphi$, which implies that the game cannot be solved within t moves. Otherwise, there is a t-max development $\delta = (S_{init} \xrightarrow{A_0} S_1 \ldots \xrightarrow{A_{t'}} S_{t'})$ with $t' \leq t$ such that $\delta \not\models \varphi$. This implies that $\delta \not\models \bigcirc^i\neg\psi$ for some $i \leq t$. However, this implication is not true for any i such that $t' < i \leq t$, as δ is too short and terminated in this case and does hence satisfy arbitrary formulas $\bigcirc^i\rho$. The implication is also not true for any i such that $i < t'$, as the respective state S_i in δ is non-terminal and hence does not satisfy ψ, implying that the sequence $(S_{init}, S_1, \ldots, S_i)$ and hence also δ satisfies $\bigcirc^i\neg\psi$. Hence i must be equal to t', we have $\delta \not\models \bigcirc^{t'}\neg\psi$ and thus $S_{t'} \models terminal \wedge goal(r, 100)$. The single player r can then win the game by performing action $A_0(r)$ in S_{init}, and by performing action $A_i(r)$ in each subsequent state S_i for $1 \leq i < t'$. This approach is not restricted to single-player games: in the case of $n \geq 2$ players, we can use it readily to prove weak winnability (cf. Definition 2.21) for an arbitrary player r.

Instead of attempting one single proof for φ, we can also attempt successive proofs on $\bigcirc^t\neg\psi$ for $t = 0, 1, 2, \ldots$ until an answer set for $P^{bc}_{\bigcirc^t\neg\psi}(G)$ is found. This procedure is summarised by Algorithm 3.1. It allows to drop the requirement of a given maximal time horizon t, and has the following properties:

Sound and Complete The algorithm terminates with answer "the game is weakly winnable" iff there is a $t \in \mathbb{N}$ such that $P^{bc}_{\bigcirc^t\neg\psi}(G)$ admits an answer set iff (by Proposition 3.18) there is a development $\delta = (S_0 \xrightarrow{A_0} S_1 \ldots \xrightarrow{A_t} S_t)$ such that $(S_t) \not\models \neg(terminal \wedge goal(r, 100))$ iff (by the formula semantics from Definition 3.5) there is a development δ which ends in a terminal state which yields maximal payoff for player r iff (by the definition of weak winnability in Definition 2.21) G is weakly winnable for player r.

Constructive and Minimal Assume the algorithm terminates with an answer set $\mathcal{A}$ for $P^{bc}_{\bigcirc^t\neg\psi}(G)$. It is easy to see that $\mathcal{A}$ uniquely corresponds to a t-max development δ and that this development, following Proposition 3.18, is of length t and such that $(last(\delta)) \not\models \neg\psi$. Hence, the algorithm can be used to effectively construct a sequence of joint actions that allows player r to win (if such a sequence exists). Moreover, for each $0 \leq t' < t$, $P^{bc}_{\bigcirc^{t'}\neg\psi}(G)$ is inconsistent, hence the constructed development δ is of minimal length.

The algorithm only terminates if the game is indeed weakly winnable for player r. It can hence *not* be applied to verify that a game is *not* weakly winnable, which would require a known upper bound for t that overestimates the maximal depth of the game tree. In Section 5.4.3 we report on experiments which show that weak winnability can effectively be shown in a variety of games.

3.6.2 Proving Multiple Properties At Once

Requiring a general game player to invoke an ASP system individually for each formula in a large set of candidate properties is not feasible for the practice of General Game Playing with a limited amount of time to analyse the rules of a hitherto unknown game. In the following we therefore develop a crucial extension of our method that enables a general game player to invoke the ASP system only once in order to determine precisely which of a whole set Φ of formulas is valid wrt. a given game description. We will show that for this purpose it suffices to construct only two answer set programs for Φ, one to establish all base case proofs and one for all induction steps. For any $\varphi \in \Phi$, then, if all answer sets for the base case program satisfy φ, we know that φ is entailed in the initial state. If additionally all answer sets of the induction step program satisfy $\varphi \supset \bigcirc\varphi$, we can conclude that φ is entailed in all reachable states. In practice, this results in a significantly more efficient proof method, especially when grouping structurally similar formulas which, for example, have the same degree or incorporate different instances of the same atoms. In Section 5.4.1 we will further motivate this intuition by an experiment setup that allows to prove various properties efficiently.

For a game description G and a finite set of state sequence invariants Φ with maximal degree $\widehat{n}_{bc}$, the generalised base case answer set program is defined as follows:

$$P^{bc}_{\Phi}(G) = S^{\texttt{true}}_{init}(0) \cup G_{\widehat{n}_{bc}} \cup P^{legal}_{\widehat{n}_{bc}-1} \cup \bigcup_{\varphi\in\Phi} Enc(\varphi)$$

Compared to $P^{bc}_{\varphi}(G)$, the constraint $\{$ `:-` $\eta(\varphi)$`.`$\}$ is no longer used, which results in a unique answer set for *each* of the $\widehat{n}_{bc}$-max sequences starting in S_{init} (as opposed to distinct answer sets for sequences that violate φ in $P^{bc}_{\varphi}(G)$ only). This is necessary to keep all relevant answer sets for formulas from Φ different from φ which do not satisfy `:-` $\eta(\varphi)$. Moreover, encodings are added for all the formulas in Φ, consequently raising the overall degree of the generated answer set program to the maximal formula degree $\widehat{n}_{bc}$.

Now let Ψ again be a finite set of valid sequence invariants, and let $\widehat{n}_{is}$ be the maximal degree of all formulas in $\Phi \cup \Psi \cup \{\varphi \supset \bigcirc\varphi\}$. Applying similar changes to $P^{is}_{\varphi,\Psi}(G)$, we define the generalised induction step answer set program as follows:

$$\begin{aligned} P^{is}_{\Phi,\Psi}(G) \quad = \quad & P^{gen} \cup G_{\widehat{n}_{is}} \cup P^{legal}_{\widehat{n}_{is}-1} \cup \textstyle\bigcup_{\varphi\in\Phi} Enc(\varphi \supset \bigcirc\varphi) \cup \\ & \textstyle\bigcup_{\psi\in\Psi}(Enc(\psi) \cup \{\ \texttt{:-}\ \textbf{not}\ \eta(\psi).\ \}). \end{aligned}$$

The generalised method can be proved sound, too.

Theorem 3.19 (Soundness on Multiple Properties). *Let G be a playable and valid GDL specification whose initial state is S_{init}. Moreover, let Φ and Ψ be sets of sequence invariants over G such that each $\psi \in \Psi$ is satisfied in all reachable states, and let $\varphi \in \Phi$. If every answer set for $P^{bc}_{\Phi}(G)$ contains $\eta(\varphi)$ and every answer set for $P^{is}_{\Phi,\Psi}(G)$ contains $\eta(\varphi \supset \bigcirc\varphi)$, then for all finite sequences $(S_{init}, S_1, \ldots, S_k)$ we have $S_k \models \varphi$.*

Proof: The proof is similar to the proof for the Soundness Theorem 3.15, with the following additional observations:

- Considering the base case, S_{init} is reachable, hence by the Sequence Extension Proposition 3.6 the n-max sequence σ that violates φ can be extended to an $\widehat{n}_{bc}$-max sequence that violates φ.
- Considering the induction step, S_k is reachable, hence by Proposition 3.6 the $\widehat{n}$-max sequence $\sigma_{\widehat{n}}$ that violates $\varphi \supset \bigcirc\varphi$ can be extended to an $\widehat{n}_{is}$-max sequence that violates $\varphi \supset \bigcirc\varphi$.
- The additional encodings $\bigcup_{\rho\in\Phi\setminus\{\varphi\}} Enc(\rho)$ in $P^{bc}_{\Phi}(G)$ ($\bigcup_{\rho\in\Phi\setminus\{\varphi\}} Enc(\rho \supset \bigcirc\rho)$ in $P^{is}_{\Phi,\Psi}(G)$, respectively) only result in additional unique name atoms in an obtained answer set, without falsifying any other atoms.
- The absence of constraints `:-` $\eta(\varphi)$. in $P^{bc}_{\Phi}(G)$ (and of `:-` $\eta(\varphi \supset \bigcirc\varphi)$. in $P^{is}_{\Phi,\Psi}(G)$, respectively) does not influence the existence of an answer set $\mathcal{A}$ which is such that $\eta(\varphi) \notin \mathcal{A}$ ($\eta(\varphi \supset \bigcirc\varphi) \notin \mathcal{A}$).

Now, $S_{init} \not\models \varphi$ ($S_k \not\models \varphi \supset \bigcirc\varphi$, respectively) results in an answer set for $P^{bc}_{\Phi}(G)$ ($P^{is}_{\Phi,\Psi}(G)$, respectively) which does not contain $\eta(\varphi)$ ($\eta(\varphi \supset \bigcirc\varphi)$, respectively), which proves the claim. $\square$

Note that, for each formula φ, name atom $\eta(\varphi)$ is contained in *all* answer sets for $P^{bc}_{\Phi}(G)$ if and only if it is contained in the *intersection* of all answer sets for $P^{bc}_{\Phi}(G)$ (similarly for $\eta(\varphi \supset \bigcirc\varphi)$ and $P^{is}_{\Phi,\Psi}(G)$). This fact will turn out useful in the practical implementation of the proof method which is presented in Chapter 5.

The following theorem shows that the generalisation succeeds in proving at least all the state sequence invariants that can be proved with the original method.

Theorem 3.20 (Provability). *Consider the same assumptions and naming conventions as in Theorem 3.19.*

(1) If $P^{bc}_{\varphi}(G)$ is inconsistent then $\eta(\varphi)$ is in all answer sets of $P^{bc}_{\Phi}(G)$.

(2) If $P^{is}_{\varphi,\Psi}(G)$ is inconsistent then $\eta(\varphi \supset \bigcirc\varphi)$ is in all answer sets of $P^{is}_{\Phi,\Psi}(G)$.

Proof:

(1) Let $P_{\widehat{n}_{bc}}$ be as P in the Answer Set Correspondence Theorem 3.14, replacing S_0 by S_{init} and n by $\widehat{n}_{bc}$. Assume that $P^{bc}_{\Phi}(G) = P_{\widehat{n}_{bc}} \cup \bigcup_{\rho\in\Phi} Enc(\rho)$ admits an answer set $\mathcal{A}$ such that $\eta(\varphi) \notin \mathcal{A}$. Then there is an $\widehat{n}_{bc}$-max sequence $\sigma_{\widehat{n}_{bc}}$ starting at S_{init} such that $\sigma_{\widehat{n}_{bc}} \not\models \varphi$ by Theorem 3.14 and the Encoding Definition 3.10. Then for the initial n-max fragment σ_n of $\sigma_{\widehat{n}_{bc}}$ we have $\sigma_n \not\models \varphi$ by the Sequence Extension Proposition 3.6. Thus, again by Theorem 3.14 and

Definition 3.10, $P_n \cup \mathit{Enc}(\varphi)$ (with P_n as P in Theorem 3.14, replacing S_0 by S_{init}) admits an answer set $\mathcal{A}'$ such that $\eta(\varphi) \notin \mathcal{A}'$, which is also an answer set for $P^{bc}_{\varphi}(G) = P_n \cup \mathit{Enc}(\varphi) \cup \{\texttt{ :- } \eta(\varphi)\texttt{.}\}$.

(2) Assume that $P^{is}_{\Phi,\Psi}(G)$ admits an answer set $\mathcal{A}$ such that $\eta(\varphi \supset \bigcirc\varphi) \notin \mathcal{A}$ and let $S^{\mathtt{true}}_0(0) \subseteq \mathcal{A}$ be the set of all atoms of the shape $\mathbf{true}(f, 0)$ in $\mathcal{A}$. Then $(P^{is}_{\Phi,\Psi}(G) \setminus P^{gen}) \cup S^{\mathtt{true}}_0(0)$ admits an answer set $\mathcal{A}'$ such that $\eta(\varphi \supset \bigcirc\varphi) \notin \mathcal{A}'$. The claim now follows by an argumentation similar to (1), where we use S_0 instead of S_{init}, $\widehat{n}_{is}$ instead of $\widehat{n}_{bc}$, and $\varphi \supset \bigcirc\varphi$ instead of φ. □

It should be stressed, however, that the converse of (2) in Theorem 3.20 does not hold: An answer set for $P^{is}_{\varphi,\Psi}(G)$ represents an established $\widehat{n}$-max sequence σ (cf. the Answer Set Correspondence Theorem 3.14) that violates $\varphi \supset \bigcirc\varphi$. σ however might not be extendable to an $\widehat{n}_{is}$-max sequence (cf. the remark following Proposition 3.6) that could serve as counterexample for $\varphi \supset \bigcirc\varphi$ in $P^{is}_{\Phi,\Psi}(G)$. Hence our efficiency improvement even strengthens the result, depending on the maximal degree $\widehat{n}$ of the given formula set Φ. For the same reason, adding proved formulas as evidence can strengthen the results of both the original method and its generalisation. Under the assumption of an accurate state generator, Theorem 3.20 implies completeness of the generalised method, as in this case also the original method is complete (cf. Theorem 3.17).

3.6.3 A General Scheme for Conjunctive Formula Proofs

In Section 3.4.2, we have argued that a successful proof for some formulas may have to assume further valid formulas in the induction step. Sometimes, however, proving some of these valid formulas in turn needs to assume the validity of the formula which is to be proved in the first place. E.g., in Quarto, both formulas

$$(\exists_{0..1} P\!:\!\{r_1, r_2\})\ \mathit{true}(\mathit{pctrl}(P)) \text{ and } (\exists_{0..1} P\!:\!\{r_1, r_2\})\ \mathit{true}(\mathit{sctrl}(P))$$

are valid, but can only be proved valid when assuming the other to be valid already. Put another way, the induction step proof for each of the formulas admits only non-reachable sequences as counter examples which do not satisfy the respective other formula. In cases like this, the proof of the *conjunction* of the formulas will be successful, as the induction hypothesis then also comprises the conjunction of both formulas and hence neglects the beforementioned counter examples. These considerations also apply to more than two formulas. In the following, we propose a general algorithm which takes a finite set of formulas Φ and a finite set of already proved formulas Ψ as input. It proves all formulas from Φ which are provable under the given evidence Ψ (with respect to an arbitrarily given fixed state generator), considering all possibilities of conjunctive provability as mentioned before.

The algorithm uses a set of conjunctive formulas $\mathit{Conjuncts}(\Phi, \mathit{Seqs})$ from which it chooses the formula which is to be attempted next. It depends on the set of formulas Φ which are currently neither proved to be valid nor disproved for some reachable state (hence, they can be *attempted*), and a set of sequences Seqs which are counter examples for previously attempted formulas. With these components, $\mathit{Conjuncts}(\Phi, \mathit{Seqs})$ is defined to be the set of all formulas $\varphi = \varphi_1 \wedge \ldots \wedge \varphi_k$ such that

Algorithm 3.2 Algorithm which proves all conjuncts from a given set of properties

```
Input:
    $G$ - playable and valid GDL specification
    $\Phi$ - finite set of sequence invariants which are to be proved
    $\Psi$ - finite set of valid sequence invariants
Initialise:
1  $Seqs := \emptyset$
2  $\Phi :=$ subset of $\{\varphi \in \Phi : S_{init} \models \varphi\}$ which contains all valid formulas from $\Phi$
3  while $Conjuncts(\Phi, Seqs) \neq \emptyset$ do
4    choose $\varphi = \varphi_1 \wedge \ldots \wedge \varphi_k \in Conjuncts(\Phi, Seqs)$ (for $\{\varphi_1, \ldots, \varphi_k\} \subseteq \Phi$)
5    if $P^{is}_{\varphi,\Psi}(G)$ admits an answer set $\mathcal{A}$ then
6      $\mathcal{A}$ corresponds to a $\deg(\varphi)$-max sequence $\sigma$ which violates $\varphi \supset \bigcirc\varphi$
7      $Seqs := Seqs \cup \{\sigma\}$
8    else
9      $Seqs := Seqs \setminus \{\sigma \in Seqs : \exists\psi \in \{\varphi_1, \ldots, \varphi_k\}$ s.t. $\sigma$ is $n$-max for some
                                                        $n \geq \deg(\psi)$ and $\sigma \not\models \psi\}$
10     $\Phi := \Phi \setminus \{\varphi_1, \ldots, \varphi_k\}$
11     $\Psi := \Psi \cup \{\varphi_1, \ldots, \varphi_k\}$
12   end if
13 end while
```

1. $\varphi_1, \ldots, \varphi_k$ are pairwise distinct and elements of Φ (hence, the conjunctive formula φ is built from currently attempted formulas), and

2. $\varphi \supset \bigcirc\varphi$ is satisfied by all sequences from *Seqs* which are at least $\deg(\bigcirc\varphi)$-max (hence, the conjunctive formula φ does not have a counter example yet).

The algorithm is given as Algorithm 3.2. The set Φ of formulas which is to be proved is first reduced such as to contain only formulas which are true initially, for example by one base case proof via the approach from Section 3.6.2 (line 2). Some initially-true formulas might be sorted out when information about their violation in some reachable state is available. The While-Loop (lines 3 to 13) is performed as long as some conjunctive formula φ can be chosen from $Conjuncts(\Phi, Seqs)$ (lines 3 and 4). An induction step proof is performed. In case it admits an answer set (line 5), the chosen formula φ could not be proved (line 6), and the obtained counter sequence is remembered (line 7). Otherwise (line 8), the conjunctive formula φ has been successfully proved valid. As a consequence, all sequences which violate any of the conjuncts of φ must necessarily start in a non-reachable state and can hence be omitted (line 9). Furthermore, the set of conjuncts that has formed φ is shifted from the set of attempted formulas (line 10) to the set of proved formulas (line 11).

Termination

The Algorithm terminates as soon as the finite set $Conjuncts(\Phi, Seqs)$ of attemptable formulas is empty. In the following we motivate that this happens after finitely many iterations of the While-Loop. For this matter, we first define an ordering $<_2 \subseteq \mathbb{N} \times \mathbb{N}$

such that

$$(p', q') <_2 (p, q) \text{ holds iff } p' < p, \text{ or } p' = p \text{ and } q' < q.$$

Each set $\mathit{Conjuncts}(\Phi, \mathit{Seqs})$ can exactly be associated with an element (p, q) of that ordering by setting p to be the number of formulas in Φ and q to be the number of conjunctive formulas in $\mathit{Conjuncts}(\Phi, \mathit{Seqs})$. I.e.,

$$(p, q) = (|\Phi|, |\mathit{Conjuncts}(\Phi, \mathit{Seqs})|).$$

We now argue that performing one iteration starting with the set $\mathit{Conjuncts}(\Phi, \mathit{Seqs})$ and its associated element (p, q) will produce a set $\mathit{Conjuncts}(\Phi', \mathit{Seqs}')$ whose associated element (p', q') is smaller than (p, q) with respect to $<_2$, which implies finite termination as there are only *finite* decreasing chains of elements in $<_2$. To this end, assume that $\varphi = \varphi_1 \wedge \ldots \wedge \varphi_k$ has currently been selected from $\mathit{Conjuncts}(\Phi, \mathit{Seqs})$. The following cases may arise:

- $P^{is}_{\varphi,\Psi}(G)$ admits an answer set (lines 6 and 7). Then $\Phi' = \Phi$ and $\mathit{Seqs}' = \mathit{Seqs} \cup \{\sigma\}$. Since Seqs' contains all sequences from Seqs, no additional formula can be contained in the updated set $\mathit{Conjuncts}(\Phi', \mathit{Seqs}')$. However, since the additional sequence σ does not satisfy $\varphi \supset \bigcirc\varphi$, φ is not contained in this set anymore. Hence,

  $$(p', q') = (p, q - l)$$

  for some $l \geq 1$ (as φ is omitted and some other formulas may have been omitted due to the additional sequence σ as well), which is smaller than (p, q) with respect to $<_2$.

- $P^{is}_{\varphi,\Psi}(G)$ is inconsistent (lines 9 to 11). Then $\Phi' = \Phi \setminus \{\varphi_1, \ldots, \varphi_k\}$ and hence (since $\{\varphi_1, \ldots, \varphi_k\} \subseteq \Phi$ and the φ_i are pairwise distinct)

  $$(p', q') = (p - k, q + l)$$

  for some $l \geq 0$ (since some other conjunctive formulas might have become available due to the newly omitted counter sequences), which is smaller than (p, q) with respect to $<_2$.

Correctness

- At each point of execution, the set Ψ contains only valid formulas, which can be argued by induction. The base case considers the starting point of the algorithm, and hence satisfies this condition since the set Ψ is assumed to contain only valid formulas. For the induction step, assume that Ψ contains only valid formulas prior to some iteration of the While-Loop. Ψ is only altered in case some formula $\varphi = \varphi_1 \wedge \ldots \wedge \varphi_k$ has been picked for which the generated induction step answer set program $P^{is}_{\varphi,\Psi}(G)$ does not admit an answer set. In that case, the formulas $\varphi_1, \ldots, \varphi_k$ are added to Ψ. It remains to show that the added formulas are indeed valid. Since φ is true in the initial state of the game (by the initialisation of the formula set Φ, applying the Sequence Extension Proposition 3.6), the unsatisfiability of $P^{is}_{\varphi,\Psi}(G)$ implies that φ is valid (the technical arguments are similar to those for the soundness result of the proof method). Consequently, also each of its conjuncts $\varphi_1, \ldots, \varphi_k$ is valid (again applying Proposition 3.6).

- After termination of the algorithm, the remaining set Φ only contains formulas which cannot be proved without further information. I.e., after termination, for each formula subset $F \subseteq \Phi$ and the formula $\varphi = \bigwedge_{\psi \in F} \psi$, there is a sequence σ which

  1. does not satisfy $\varphi \supset \bigcirc\varphi$ (and hence is a counter example for φ), and
  2. satisfies all the proved valid formulas from Ψ.

  The first condition is true since *Conjuncts*(Φ, Seqs) is empty after termination, which implies that each conjunctive formula φ violates item 2 in the definition of set *Conjuncts*(Φ, Seqs) and hence has a counter sequence in *Seqs*. The second condition is true since running the algorithm only produces counter sequences which satisfy all known valid formulas (as these formulas are included in the induction step proof), and since sequences in *Seqs* which do not satisfy newly proved formulas are immediately removed.

Discussion

The algorithm provides a general method for exhaustively attempting proofs for a given formula set (with respect to an arbitrarily given fixed state generator) under the incorporation of previously obtained information. It is, however, to be understood as a scheme rather than a concrete method, as the following details are not considered in this work (we will provide some hints on these issues in the Future Work Section 7.2).

- The method for disproving formulas (line 2) prior to starting the While-Loop is left open. This step provides the only possibility for reliably sorting out non-valid formulas, and a more restrictive set can greatly increase performance of the algorithm.

- No heuristic for the choice of a formula from the set *Conjuncts*(Φ, Seqs) is specified. However, also this is a crucial point for overall performance, as it influences the time consumption of the proof (more complex formulas need more complex proofs) and the "quality" of the counter examples (those which violate more attempted formulas provide more information on potentially unsuccessful proof attempts).

- No method for checking the entailment of formulas with respect to sequences is specified. As this check has to be performed repeatedly (however at most twice for each formula in Φ for each newly obtained counter sequence, as motivated below), it does influence performance.

Let us conclude this section with a remark concerning an implementation of the algorithm. In order to calculate the elements of the current set *Conjuncts*(Φ, Seqs), an entailment check for each of the exponentially many conjunctive formulas over Φ with respect to all sequences in *Seqs* is *not* necessary. Instead, for each newly obtained sequence σ and each formula $\psi \in \Phi$, the two entailment checks $\sigma \vDash \psi$ and $\sigma \vDash \neg\bigcirc\psi$ suffice, when their results are remembered in all following iterations (for the following considerations, we assume that σ does not entail a formula ψ in case σ is n-max for some $n < \deg(\psi)$). Suppose that $\{\varphi_1, \ldots, \varphi_k\}$ is any subset of the formulas from Φ

which are satisfied by a sequence σ, and let ψ be a formula which is such that $\sigma \vDash \psi$ and $\sigma \vDash \neg \bigcirc \psi$. Then the latter also implies that $\sigma \vDash (\neg \bigcirc \psi) \vee \neg \bigcirc (\varphi_1 \wedge \ldots \wedge \varphi_k)$ (since the first disjunct is already satisfied, the second disjunct is unimportant for the entailment of the disjunctive formula), which is equivalent to $\sigma \vDash \neg\bigcirc(\psi \wedge \varphi_1 \wedge \ldots \wedge \varphi_k)$. Since all formulas $\psi, \varphi_1, \ldots, \varphi_k$ are true wrt. σ, this further implies

$$\sigma \vDash (\psi \wedge \varphi_1 \wedge \ldots \wedge \varphi_k) \wedge \neg \bigcirc (\psi \wedge \varphi_1 \wedge \ldots \wedge \varphi_k).$$

This in turn yields that the conjunctive formula $\psi \wedge \varphi_1 \wedge \ldots \wedge \varphi_k$ is not an element of $\mathit{Conjuncts}(\Phi, \mathit{Seqs})$ and hence cannot be chosen for a proof attempt. As $\varphi_1, \ldots, \varphi_k$ were arbitrarily chosen from the formulas in Φ which are satisfied by the counter sequence σ, we can conclude the following: The two before-mentioned entailment checks, performed for each formula from Φ each time a newly obtained counter sequence σ has been found, suffice to know that *any* subset of formulas from Φ which are true wrt. σ is not available to form a conjunctive formula for a proof attempt in case this subset contains a formula ψ such that also $\neg \bigcirc \psi$ is satisfied by that sequence σ. Putting this information together for all obtained counter sequences yields all conjunctions which are not applicable, and hence allows to choose a different one for a proof attempt.

3.6.4 Non-Playable Sequences

Sequence invariant entailment $\vDash$ (cf. Definition 3.5), defined over n-max sequences for the degree n of the formula to be verified, does not account for sequences that are of length smaller than n and end in a non-terminal state that does not permit a move for one of the players (also called non-playable sequences). This has an interesting effect, as the following simple, non-playable game shows:

```
role(r).
init(f).
legal(r,a) :- true(f).
```

Consider the sequence invariant that axiomatises playability, that is,

$$\varphi = \neg \mathit{terminal} \supset (\forall R{:}D_R)(\exists M{:}D_M)\, \mathit{legal}(R, M)$$

with the domains $D_R = \{r\}$ and $D_M = \{a\}$ for the example game. Additionally, consider an arbitrary formula ψ with degree 1 that is known to be satisfied in the initial state, $S_{init} = \{f\}$. Then $\varphi \wedge \psi$ is satisfied in S_{init} since the only 1-max sequence $\{f\} \xrightarrow{\{r\,:\,a\}} \{\}$ satisfies $\varphi \wedge \psi$. Formula $\varphi \wedge \psi$ is also satisfied in state $\{\}$, as no 1-max sequence emerges from that state. Since these are the only reachable states, $\varphi \wedge \psi$ is considered true in each reachable state, contradicting our intuition that the game is non-playable and hence that φ should be false. The only counterexample, however, would be the sequence $(\{\})$ of length 0, which is non-playable with respect to length 1. But as non-playable sequences are not among the 1-max sequences, $(\{\})$ will never be considered in our setting.

On the one hand, playability is a standard requirement for General Game Playing Competitions and thus can be presupposed by a general game-playing system. A GDL game designer, on the other hand, might be particularly interested in proving whether a game she has designed is indeed playable, which motivates the following considerations. To begin with, observe that playability of a game has no influence on the outcome of a proof attempt for sequence invariant φ when tried together with a set Ψ of previously proved formulas of degree less or equal to 1, since:

Base Case amounts to verifying φ with respect to the only 0-max sequence starting in S_{init}, namely (S_{init}), which incorporates no state transition and hence is independent of the playability assumption; and

Induction Step amounts to verifying φ with respect to every 1-max sequence which starts in a state satisfying φ, which is again independent of the playability assumption since φ represents playability itself.

Hence, the proof method can be used to reliably prove the playability formula φ, relying on previously proved formulas of degree ≤ 1 only, in order to assume playability thereafter. If this proof attempt is not successful, the (indirect) playability assumption can be dropped by incorporating non-playable sequences into the proof method as follows:

- Altering the definition of an n-max sequence (cf. Definition 3.5) such that in case of length smaller than n the last state of the sequence might also be non-terminal and not permit a legal move for one of the players.
- Adding the following clauses to the game description G (cf. the GDL Syntax Definition 2.14):

```
has_no_legal(R) :- not has_legal(R), role(R).
has_legal(R)    :- legal(R,A).
```

- Adding to $Enc(\bigcirc\psi, i)$ (cf. Section 3.3.2) for each $r \in R$:

$$\eta(\bigcirc\psi, i) \texttt{ :- has\_no\_legal}(r, i).$$

- Adding to P^{legal}_{n-1} (cf. Section 3.4.1) for all $0 \leq i \leq n-1$ and $r \in R$:

$$\texttt{terminated}(i) \texttt{ :- has\_no\_legal}(r, i).$$

Besides increasing the complexity of the constructed Answer Set Programs $P^{bc}_{\varphi}(G)$ and $P^{is}_{\varphi,\Psi}(G)$, this modification weakens the proof method. As an example, suppose we extend the (non-playable) game from the beginning of this section by the following clauses:

```
next(f) :- true(f), not true(g).
next(g) :- true(g).
```

Note that the extended game is playable, as opposed to the original one. Now consider the formula $true(f)$, which holds in each reachable state. A proof attempt, however, yields the (unreachable) 1-max sequence $\sigma = (\{f, g\}, \{g\})$ as a counterexample for the induction step, since $\sigma \not\models true(f) \supset \bigcirc true(f)$. Assuming playability and the original setting, σ is rejected as soon as some formula ψ with degree 2 is known to be satisfied in each reachable state and added to $P^{is}_{\varphi,\Psi}(G)$, because σ cannot be extended to a 2-max sequence. This indeed allows to prove the induction step for $true(f)$. This is in contrast to the modified setting, where σ is also considered 2-max and might satisfy ψ as well. In conclusion, the presented modification reliably copes with games which are not known to be playable, but whenever this assumption can be made, the more efficient and stronger original proof method should be used instead.

3.7 Discussion

We will now discuss our choice to establish a proof method using the paradigm of Answer Set Programming (Section 3.7.1), and give an overview of the attributes of our property specification language which are essential for efficient property verification in practice (Section 3.7.2).

3.7.1 Choosing Answer Set Programming

We have decided to establish a proof method using the paradigm of Answer Set Programming for the following two main reasons:

- Answer Set Programming allows a straight forward encoding of game descriptions. Each program of the Game Description Language, possibly enriched with state and move information, is stratified and hence admits a unique answer set which corresponds to the well-known standard model of a logic program. Hence, the Answer Set Semantics provides a natural way for reasoning over game descriptions. Furthermore, Answer Set Programming provides helpful additional language constructs such as weight atoms and constraints which greatly ease the formulation of atom quantities. E.g., action generators need to state that each agent performs *exactly one* action in each non-terminal state, and game properties frequently include expressions such as "there are *at least* m and *at most* n instances of $p(\vec{t})$". The specification of conditions like these, say, with propositional logic is considerably more complex. Furthermore, Answer Set Programming can take advantage of their explicit representation by a special-purpose evaluation [GKKS09]. The Answer Set Semantics also allows for compact formula encodings, as answer sets are minimal. For example, a conjunctive formula $\varphi_1 \wedge \varphi_2$ can be encoded via the one clause $\eta(\varphi_1 \wedge \varphi_2)$ `:-` $\eta(\varphi_1), \eta(\varphi_2)$. (together with further encodings for the respective subformulas), without the need for additional constructs that deal with unintended models satisfying $\eta(\varphi_1 \wedge \varphi_2)$ but not both $\eta(\varphi_1)$ and $\eta(\varphi_2)$.
- Answer Set Programming has become a state-of-the-art reasoning technique. It has successfully been applied to a variety of problems, including areas such as product configuration, NASA shuttle controllers, and systems biology (pointers to a comprehensive list of applications can be found, e.g., in [GKKS11a]). The ASP system we use for our implementation (we dedicate Section 5.3 to a brief introduction) has achieved first ranks in several tracks of international competitions such as ASP (concerning Answer Set Programming), PB (concerning Pseudo-Boolean functions), and SAT (concerning the satisfiability of propositional formulas). Its success in these diverse research areas qualifies Answer Set Programming as the first choice for our verification problems.

3.7.2 Expressibility Versus Practical Useability

The design of a verification method which is applicable even in games far beyond reach of exhaustive search requires several restrictions. The following two items summarise the two main design choices for our property specification language that we have applied in order to achieve efficient property verification.

- As the first choice, we allow the reference to direct successor states *universally* only. I.e., we utilise the unary operator $\bigcirc$, and our semantics follows the style of *Linear Temporal Logic* (LTL, see e.g. [BK08]) to satisfy $S \vDash \bigcirc\varphi$ if and only if φ is satisfied in *each* direct successor of state S. The expression of *existential* quantification, on the contrary, is not possible in our framework. This restriction allows us to compactly represent each counter example of a formula via *one single* state sequence. In comparison, a modification of the interpretation of $\bigcirc$ to *existential* path quantification would require a partial game tree. Allowing both types of temporal operators in the style of *Compution Tree Logic* (CTL, see e.g. [BK08]) would necessitate an even more complex structure for counter examples which combines elements of sequences and trees. As a consequence, the temporal GDL extension, required for the verification of temporal properties using Answer Set Programming, would need a *branching time* structure, opposed to the linear time structure via natural numbers $0, 1, 2, \ldots$ that we apply.

  In principle, a branching time structure is achievable via the specification of terms in time arguments similar to the Situation Calculus [McC63], namely such that each term encodes all joint actions which have been performed so far. As an example in Quarto,

  $$do([noop, place(p0000,1,1)], do([select(p0000), noop], s_0))$$

  can be used to represent the state resulting from a state represented as s_0 when player r_2 places the piece $p0000$ at cell $(1, 1)$ which has previously been selected by player r_1. However, for a branching factor b of the game and a degree n of the property which is to be verified, this would result in an exponential blowup of the GDL specification to $\Sigma_{0 \leq i \leq n} b^i$ different time-extended GDL clause sets instead of the linear blowup to n clause sets we achieve with a linear time structure. As current state-of-the-art answer set solvers rely on a complete clause expansion prior to solving a problem, a branching-time structure cannot practicably be handled in most games even for properties with a small degree.

- Our second choice is the restriction to *bounded* time reference. I.e., we do not allow temporal operators to express "always", "eventually", or "until", which are common in temporal logics such as LTL and CTL. Our restriction allows to concentrate on the *local* scheme of a linearly temporalised GDL specification which only involves a *finitely bounded* number of time steps. This is possible since counter examples can be represented within this structure, and a property can also be verified with respect to all reachable states by proving the non-existence of a counter example, instead of proving its entailment in each individual state. The additional utilisation of an induction hypothesis further allows to efficiently cut parts of the search space.

The restrictions which are put via a linear time structure and finite time reference are not severe. In fact, in this chapter we have seen that many interesting properties can still be expressed. In Section 5.4.1, we will demonstrate the expressiveness with further interesting properties. Chapter 5 also provides reports on extensive experiments showing that our method is indeed practically applicable even in complex games.

As a further comment to practical useability, note that the initial state of the game is by no means interconnected with the rest of the constructed answer set programs, and hence can arbitrarily be replaced by any reachable state of the game. This allows to readily apply our method also during gameplay, which is interesting e.g. to discover properties which only hold, say, in the final stage of a game. Furthermore, as the induction step proof for a property does not depend on the specified initial state, it only has to be performed once per property. In case it is successful, a successful base case proof with respect to a later state S_0 of the game then suffices to establish the validity with respect to all reachable states that follow S_0.

3.8 Summary

In this chapter, we developed a sound theory to prove rich invariance properties for games formulated over the GDL. To this end, we first introduced a simple yet expressive property description language to address game properties which may involve arbitrary finite sequences of game states. Its syntax incorporates basic atoms of the GDL, propositional constructs, (restricted) quantification including counting quantifiers, and the unary operator $\bigcirc$ borrowed from Temporal Logic. Its semantics is based on linear time via *finite* state sequences. We then introduced an extension of the GDL by an additional linear time argument, the temporal GDL extension, and proved that it correctly generalises reasoning over the GDL from single state transitions to arbitrary long finite state sequences. We defined an encoding for formulas in our language and proved that, together with a temporal GDL extension and an encoding of a state sequence, it yields an answer set program which admits a unique answer set that contains a special formula-name token if and only if that state sequence satisfies the formula. Based on this correspondence, we developed a proof theory which establishes the validity of a formula using the principle of induction: first we constructed an answer set program whose unsatisfiability implies that the formula holds in the initial state (for the base case), second we constructed an answer set program whose unsatisfiability implies that the formula holds in each direct successor of each state that itself satisfies the formula (for the induction step). We formally proved the soundness of our method, and showed that completeness can be obtained when the set of reachable states is known. We further showed that the method can be adjusted to correctly verify properties with respect to arbitrary single reachable states, which also yields the possibility to solving single player games and proving weak winnability in multiplayer games. Finally, we developed an extension of our method to prove multiple formulas simultaneously and provided a general algorithm for exhaustive formula proofs. We concluded with a motivation of our choices regarding Answer Set Programming and our language for the specification of game properties.

Chapter 4

Epistemic Sequence Invariants

The proof method developed so far can prove state sequence invariants over the global world state in both complete-information and incomplete-information games. However, the underlying formula language is not capable of expressing different *perspectives* of players which are caused by their not necessarily complete percepts of the world. For a game designer, specifying a game raises the question whether it satisfies certain key features, e.g. the property

When the game has terminated, each player knows that it has terminated. (4.1)

This is no longer obvious when moving to incomplete-information games. As another example, consider the property

Each player knows his legal and illegal moves. (4.2)

Both properties do not refer to the global world state (in which termination and legal moves can exactly be determined at all times), but to what players *know* about the world state. Even though a player gets to see the entire description of an incomplete-information game (cf. the general introduction to the execution model in Section 2.2.5) and hence, e.g., can exactly figure out the initial state as well as the legal moves of all players or the terminal condition in each state, only one performed joint move in the initial state can leave him completely uncertain about which of the possible successor states of the initial state might be the actual one. Hence, again, an automated verification technique of properties which involve knowledge can come to the game designers rescue. In addition, it can provide valuable information to a general game-playing program concerning the perspectives of his opponents in certain stages of the game. This chapter is dedicated to a generalisation of the presented verification method which can handle the mentioned properties.

To this end, we proceed as follows. We first introduce an incomplete-information game that gives an intuition of the subtleties that arise in this setting and will serve as running example throughout this chapter (Section 4.1). We then extend the syntax and the semantics of our property language to account for knowledge of individual players, and show that this extension matches well-known properties that are desired when reasoning about knowledge as well as the GDL-specific property of complete knowledge in the initial state (Section 4.2). As a linear time structure provides several advantages for a practical verification method in the knowledge-free case (cf. the discussion in

Section 3.7.2), we decide to keep this structure also in the generalisation of the proof method which is about to be presented. In Section 4.3, we motivate that this design decision requires to exclude properties involving formulations of what players do *not* know, which then allows to provide an equivalent alternative semantics for all remaining properties that solely concentrates on linear time and will form the foundation for the correctness result of the generalised proof method. We generalise both the temporal GDL extension and the formula encoding from Chapter 3 to the knowledge setting (Section 4.4), and use these generalisations to specify two answer set programs which are then utilised to prove the validity of a knowledge formula via induction (Section 4.5). After an extensive example for this generalised proof method (Section 4.6), we show that it remains sound and (restricted) complete (Section 4.7), and provide several improvements which allow to prove properties more efficiently (Section 4.8).

4.1 The Game Krieg-Tictactoe

The 2-player game *Krieg-Tictactoe* is a small but interesting incomplete-information game which will serve as running example throughout this chapter. It has been introduced in [ST11] inspired from *Kriegspiel* (see, e.g., [Pri94, RW05]), a chess variant where the players are not informed about the piece positions and moves of their opponent.

Krieg-Tictactoe is played on a 3×3 game board. Initially, the board is empty, and the players take turns repeating the following procedure: the active player attempts to mark an arbitrary cell with his own distinguished marker. If it is already marked, his move is rejected and causes a subsequent attempt to mark a different cell. This process repeats until a chosen empty cell gets marked, passing control to the other player who is not informed about the coordinates of the cell that has been marked. A partial initial game tree of Krieg-Tictactoe, together with an indication of states that are indistinguishable for some player, is shown in Figure 4.1. The game ends if the game board is completely filled (causing a draw) or one player wins by having completed a horizontal, vertical or diagonal line of three of his own pieces.

To specify a GDL specification for Krieg-Tictactoe, we choose the following actions:

- `mark`(x, y): attempt to mark cell (x, y) on the game board, and
- `noop`: an action without effect, performed by the player who currently has no control.

The game positions will be represented with the following fluents:

- `cell`(x, y, p): board cell (x, y) contains piece p (where $p =$ `b` is for blank cells),
- `control`(r): role r currently has control to mark a cell, and
- `tried`(m, n): a failed mark attempt at coordinate (m, n) has previously been carried out by the active player.

Figure 4.2 contains a complete GDL specification for Krieg-Tictactoe. Lines 1 and 2 define the two players and the initial state, and lines 5 to 8 specify the legal moves dependent on the current state. Lines 11 to 14 define state update in case of a

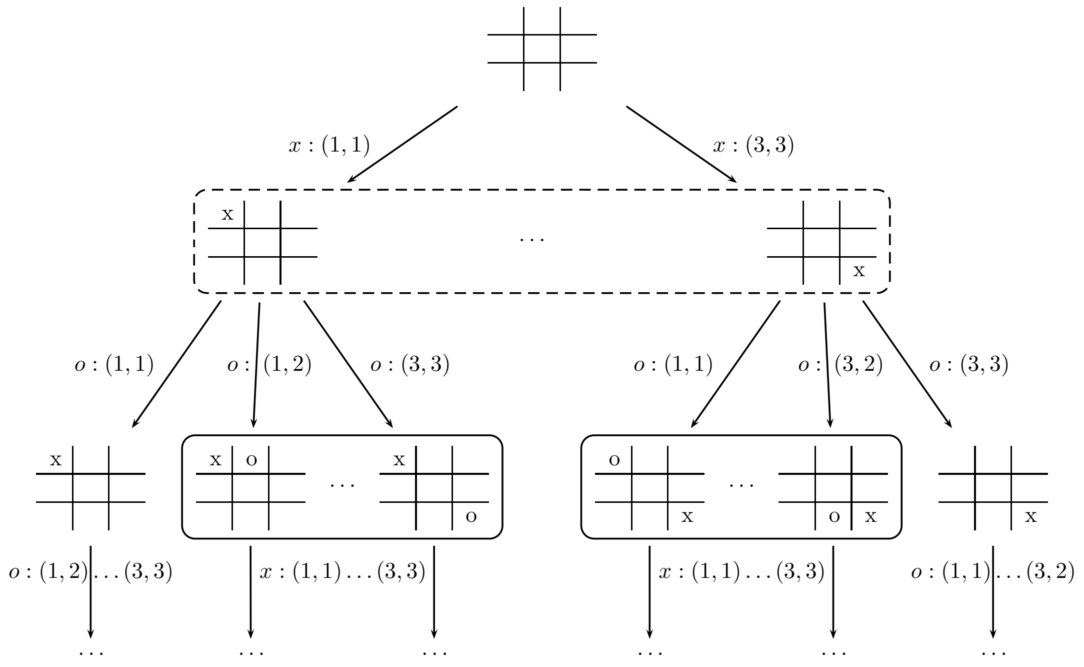

Figure 4.1: A partial game tree for Krieg-Tictactoe. Player x starts placing a marker on the empty game board. Player o does not witness the placement and hence subsequently considers all states within the dash-lined oval possible. His attempt to mark an arbitrary cell may fail or succeed. In the former case, o is allowed to try again, and has come to know the current game state. In the latter case, o still doesn't know the current game state (which is not represented here), and x cannot distinguish the states within each solid-lined oval. Player x proceeds attempting a further *arbitrary* placement (in our setting, we also allow him to try his previously marked cell again).

failed mark attempt by adding a new instance `tried`(m,n) (line 11) and keeping the present ones (line 12), and by keeping cell fluents (line 13) as well as control fluents (line 14) unchanged. A successful attempt is expressed with lines 16 to 23, adding a newly marked cell (lines 16 and 17), keeping all previously marked cells (lines 18 to 21), switching control (lines 22 and 23), and removing each instance `tried`(m,n) by not having clauses with head **next**(`tried`(m,n)) that apply in case of a valid move. After each performed joint action, the only information each player gets to see is a special constant `yourmove` in case it is currently his turn to place a piece (lines 28 to 30). A state is terminal in case of a completed line (lines 33 and 34) or a completely filled game board (line 35), and respective goal values are determined via lines 45 to 51.

In Krieg-Tictactoe, property (4.1) is not valid in all reachable states: consider, e.g., a non-terminal state where player x has already lined two markers, and where the missing cell as well as another cell are still empty and hence player x might mark one of these cells. Since player o is not informed about the chosen move, successful marking yields at least two possible successor states for player o, one of them being terminal and the other non-terminal. Hence, player o does *not* know whether the game is terminal after player x has successfully marked a cell in the mentioned state of the game. While this short explanation suffices to show that property (4.1) is not valid, a motivation for or against validity of property (4.2) is not that obvious, and (a simplified variant of) this property will serve as running example and is shown to be

```
role(x).   init(control(x)).
role(o).   init(cell(1,1,b)).  ...  init(cell(3,3,b)).


legal(R,mark(M,N))  :- true(control(R)), true(cell(M,N,Z)),
                              not true(tried(M,N)).
legal(x,noop)       :- true(control(o)).
legal(o,noop)       :- true(control(x)).


next(tried(M,N))  :- not validmove, does(R,mark(M,N)).
next(tried(M,N))  :- not validmove, true(tried(M,N)).
next(cell(M,N,Z)) :- not validmove, true(cell(M,N,Z)).
next(control(R))  :- not validmove, true(control(R)).

next(cell(M,N,x)) :-     validmove, does(x,mark(M,N)).
next(cell(M,N,o)) :-     validmove, does(o,mark(M,N)).
next(cell(M,N,Z)) :-     validmove, true(cell(M,N,Z)),
                             does(R,mark(I,J)), distinct(M,I).
next(cell(M,N,Z)) :-     validmove, true(cell(M,N,Z)),
                             does(R,mark(I,J)), distinct(N,J).
next(control(o))  :-     validmove, true(control(x)).
next(control(x))  :-     validmove, true(control(o)).

validmove :- does(R,mark(M,N)), true(cell(M,N,b)).


sees(R,yourmove) :- not validmove, true(control(R)).
sees(x,yourmove) :- validmove, true(control(o)).
sees(o,yourmove) :- validmove, true(control(x)).


terminal :- line(x).
terminal :- line(o).
terminal :- not open.

open :- true(cell(M,N,b)).

line(C) :- true(cell(M,1,C)), true(cell(M,2,C)), true(cell(M,3,C)).
line(C) :- true(cell(1,N,C)), true(cell(2,N,C)), true(cell(3,N,C)).
line(C) :- true(cell(1,1,C)), true(cell(2,2,C)), true(cell(3,3,C)).
line(C) :- true(cell(1,3,C)), true(cell(2,2,C)), true(cell(3,1,C)).


goal(x,100) :- line(x).
goal(x,50)  :- not line(x), not line(o).
goal(x,0)   :- line(o).

goal(o,100) :- line(o).
goal(o,50)  :- not line(x), not line(o).
goal(o,0)   :- line(x).
```

Figure 4.2: A GDL specification of the game Krieg-Tictactoe.

provable with our method later in this chapter.

4.2 Formalisation of Epistemic Sequence Invariants

In this section, we will define an extension of our language for state sequence invariants which enables to express knowledge of individual players (Section 4.2.1). We then extend its semantics accordingly (Section 4.2.2), prove that it satisfies properties which are desired in the setting of knowledge (Section 4.2.3), and show that agents always have complete knowledge about the initial game state (Section 4.2.4).

4.2.1 Syntax

In the following extension of the syntax of state sequence invariants we borrow operators K_r known from Modal Logic (see, e.g., [BdRV01]) which are also extensively studied in the standard text book [FHMV95], with the intention of expressing what an agent r knows about a finite sequence of successive game states.

Definition 4.1 (Epistemic Sequence Invariants)**.** *Let G be a valid GDL specification, R be the set of roles from the semantics of G, and recall the set $\mathcal{P}$ of ground atoms $p(\vec{t})$ over G such that $p \notin \{\mathbf{init}, \mathbf{next}\}$ and p does not depend on* **does** *in G. The set $\mathcal{ESIN}_G$ of* epistemic (state) sequence invariants over *G is the smallest set with*

- $\mathcal{P} \subseteq \mathcal{ESIN}_G$;
- *If $\varphi, \varphi_1, \varphi_2 \in \mathcal{ESIN}_G$, then also the following are in $\mathcal{ESIN}_G$:*
  - $\neg\varphi$, $\varphi_1 \wedge \varphi_2$, *and* $\varphi_1 \vee \varphi_2$;
  - $(\exists \vec{X} : D_{\vec{X}})\, \varphi[\vec{X}]$, *and* $(\forall \vec{X} : D_{\vec{X}})\, \varphi[\vec{X}]$;
  - $(\exists_{l..u} \vec{X} : D_{\vec{X}})\, \varphi[\vec{X}]$, *for each* $l \in \mathbb{N}$ *and* $u \in \mathbb{N} \cup \{\infty\}$ *s.t.* $l \leq u$;
  - $\bigcirc\varphi$;
  - $K_r\varphi$, *for each* $r \in R \setminus \{\mathbf{random}\}$.

The degree of *$\varphi \in \mathcal{ESIN}_G$ is defined in extension to Definition 3.2 (concerning the formula degree) by adding the following:*

$$\deg(K_r\varphi) := 0.$$

The notion of a subformula is extended as expected. ■

A formula $K_r\varphi$ states that agent r knows φ. Pseudo player **random** is excluded in this notion, as it is not considered by the information relation $\mathcal{I}(A, S)$ in the definition of the GDL semantics (cf. Definition 2.17) which will form the basis for the interpretation of formulas with knowledge operator K_r. Note that, even if φ has degree $\deg(\varphi) > 0$, the degree of $K_r\varphi$ is defined to be 0. The intention of this definition will become clear at the end of Section 4.2.2. Furtheron, we will use the notion "formula" also to refer to epistemic sequence invariants, and will additionally refer to formulas which do not contain any K_r as *knowledge-free* formulas.

Example 4.2 (Epistemic Sequence Invariants). Property (4.1) from page 63 (stating that when the game has terminated, each player knows that it has terminated) can be formulated with the set R of players in the respective game as follows.

$$\bigwedge_{r\in R}(\mathit{terminal} \supset K_r \mathit{terminal}) \tag{4.3}$$

Similarly, denoting by $\mathit{ADom}(r)$ the finite set of all actions of a player r, we can express property (4.2) (stating that each player knows his legal and illegal moves) via

$$\bigwedge_{r\in R}(\forall A\colon \mathit{ADom}(r))\,(K_r \mathit{legal}(r,A) \vee K_r \neg \mathit{legal}(r,A)) \tag{4.4}$$

■

4.2.2 Semantics

Intuitively, a player which has incomplete knowledge about a game state considers more than one game state possible. Consequently, a player knows a certain property in a game state if and only if that property is true in all game states he considers possible. The semantics of our property language will incorporate this intuition. To formally define it, we need a classification of the previously-mentioned possible game states which does not solely depend on the current game state. As an example to the contrary, consider the game state in Krieg-Tictactoe where cell $(1,1)$ is marked by x and cell $(1,2)$ is marked by o. Then the information of o depends on how that game state has been reached, as o may or may not have attempted to mark cell $(1,1)$ prior to marking cell $(1,2)$. While the resulting state is the same, performing this unsuccessful attempt implies that o knows the position of the previously placed marker x and hence only considers the actual state possible, whereas otherwise this fact is unknown to o and necessitates to consider possible all states with different positions of marker x (except for $(1,2)$) as well. To resolve this ambiguity, the notion of possible states has to incorporate the complete development which has led to the current game state.

In the following, we state a binary relation which defines classes of developments which are indistinguishable for a player and hence allow to determine all considered possible states with respect to a given game development (by taking the last states of all developments in the class of the given development). Put together for all players, the following definition provides an instance of a *Kripke frame* (see, e.g., [BdRV01] and [FHMV95]). It has been put in the context of the Game Description Language in [Thi10] and [RT11a].

Definition 4.3 (Accessibility Relation). *Let $(R, S_{init}, T, l, u, \mathcal{I}, g)$ be the semantics of a valid GDL description G, let r be a player from G different from* **random***, and let Δ_G be the set of developments over G. For two developments $\delta_1, \delta_2 \in \Delta_G$ such that $\delta_1 = (S_{init} \xrightarrow{A_0} S_1 \ldots \xrightarrow{A_{m-1}} S_m)$ and $\delta_2 = (S_{init} \xrightarrow{A'_0} S'_1 \ldots \xrightarrow{A'_{m-1}} S'_m)$, δ_2 is* accessible for player r at δ_1*, denoted $\delta_1 \sim_r \delta_2$, if, and only if, for each $0 \le i \le m-1$:*

- $\{p : (r,p) \in \mathcal{I}(A_i, S_i)\} = \{p : (r,p) \in \mathcal{I}(A'_i, S'_i)\}$ *(r's percepts are the same),*
- $A_i(r) = A'_i(r)$ *(r always takes the same action).*

$\sim_r$ *is called* accessibility relation for player r. *Furthermore, for the set* $\{r_1, \dots, r_n\}$ *of players besides* **random**, *the* accessibility relation of G, *denoted* $\sim$, *is defined as the Kripke frame*

$$\sim := (\Delta_G, \sim_{r_1}, \dots, \sim_{r_n}).$$

■

Note that each binary relation $\sim_r$ of the accessibility relation of a game is an *equivalence relation*, i.e., it is

- *reflexive*, namely such that for all developments $\delta \in \Delta_G$, we have that $\delta \sim_r \delta$;
- *symmetric*, namely such that for all developments $\delta_1, \delta_2 \in \Delta_G$, we have that if $\delta_1 \sim_r \delta_2$, then $\delta_2 \sim_r \delta_1$; and
- *transitive*, namely such that for all developments $\delta_1, \delta_2, \delta_3$, we have that if $\delta_1 \sim_r \delta_2$ and $\delta_2 \sim_r \delta_3$, then $\delta_1 \sim_r \delta_3$.

These notions will be needed to prove some well known properties of our semantics in Proposition 4.6. Note that reflexivity ensures that player r always considers the actual development of the game possible. This property distinguishes knowledge from possibly wrong *beliefs*, where the actual development is not necessarily considered possible.

The notion of accessible developments exactly corresponds to the notion of indistinguishable developments given in Section 2.2.3, which has been shown in [Thi10] and allows to use the terminology interchangeably in the remainder of this work.

Proposition 4.4 (Indistinguishability). *Let G be a valid GDL description and $(\Delta_G, \sim_{r_1}, \dots, \sim_{r_n})$ be the accessibility relation of G. Two developments $\delta_1, \delta_2 \in \Delta_G$ with the same length are indistinguishable for player r if, and only if, $\delta_1 \sim_r \delta_2$.* □

We are now ready to specify the semantics of epistemic sequence invariants as a direct extension of the semantics for state sequence invariants in Definition 2.17 (cf. page 20). Following [RT11a], it takes into account the foregoing development of the game in order to reliably evaluate formulas of the form $K_r\varphi$. Before we start, recall that we also denote the last state S_m of a sequence $\sigma = (S_0, \dots, S_m)$ by $last(\sigma)$, and its length m by $|\sigma|$. Furthermore, for two sequences $\sigma_1 = (S_0, \dots, S_m)$ and $\sigma_2 = (S_m, \dots, S_{m+k})$, we also denote their composition $(S_0, \dots, S_m, \dots, S_{m+k})$ as (σ_1, σ_2).

Definition 4.5 (Semantics for Epistemic Sequence Invariants). *Let G be a valid GDL specification, φ be a formula such that* $\deg(\varphi) = n$, *r be an arbitrary role different from* **random** *according to the semantics of G, and $\delta = (S_{init}, S_1, \dots, S_k)$ be a development. We say that S_k* satisfies *φ* wrt. *δ (written $S_k \models_\delta \varphi$) if for all n-max sequences $S_k \xrightarrow{A_k} \dots \xrightarrow{A_{k+m-1}} S_{k+m}$ $(m \leq n)$ we have that $(S_k, \dots, S_{k+m}) \models_\delta \varphi$ as follows:*

$(S_k, \dots, S_{k+m}) \models_\delta p$	*iff* $G \cup S_k^{\mathtt{true}} \vdash p \qquad (p \in \mathcal{P})$
$(S_k, \dots, S_{k+m}) \models_\delta \neg\psi$	*iff* $(S_k, \dots, S_{k+m}) \not\models_\delta \psi$
$(S_k, \dots, S_{k+m}) \models_\delta \psi_1 \wedge \psi_2$	*iff* $(S_k, \dots, S_{k+m}) \models_\delta \psi_1$ *and* $(S_k, \dots, S_{k+m}) \models_\delta \psi_2$
$(S_k, \dots, S_{k+m}) \models_\delta \psi_1 \vee \psi_2$	*iff* $(S_k, \dots, S_{k+m}) \models_\delta \psi_1$ *or* $(S_k, \dots, S_{k+m}) \models_\delta \psi_2$

$(S_k,\dots,S_{k+m}) \vDash_\delta (\exists \vec{X}\!:\!D_{\vec{X}})\,\psi[\vec{X}]$	*iff*	*there is an* $\vec{a} \in D_{\vec{X}}$ *s.t.* $(S_k,\dots,S_{k+m}) \vDash_\delta \psi[\vec{X}/\vec{a}]$
$(S_k,\dots,S_{k+m}) \vDash_\delta (\forall \vec{X}\!:\!D_{\vec{X}})\,\psi[\vec{X}]$	*iff*	*for all* $\vec{a} \in D_{\vec{X}}$*:* $(S_k,\dots,S_{k+m}) \vDash_\delta \psi[\vec{X}/\vec{a}]$
$(S_k,\dots,S_{k+m}) \vDash_\delta (\exists_{l..u} \vec{X}\!:\!D_{\vec{X}})\,\psi[\vec{X}]$	*iff*	*there are* $\geq l$ *and* $\leq u$ *different* $\vec{a} \in D_{\vec{X}}$ *s.t.* $(S_k,\dots,S_{k+m}) \vDash_\delta \psi[\vec{X}/\vec{a}]$
$(S_k,\dots,S_{k+m}) \vDash_\delta \bigcirc\psi$	*iff*	$m = 0$ *or* $(S_{k+1},\dots,S_{k+m}) \vDash_{\delta'} \psi$*, where* $\delta' = (\delta, (S_k, S_{k+1}))$
$(S_k,\dots,S_{k+m}) \vDash_\delta K_r\psi$	*iff*	*last*$(\delta') \vDash_{\delta'} \psi$ *for each* δ' *s.t.* $\delta \sim_r \delta'$

■

All cases different from $K_r\psi$ exactly correspond to their counterparts in the original formula semantics from Definition 3.5, with the addition of carrying a development δ as parameter which is only needed in the additional line addressing $K_r\psi$. Hence, for knowledge-free formulas, both semantics clearly coincide. To evaluate whether player r knows ψ in sequence $(S_k,\dots,S_{k+m})$, we need to consider the development that has led to S_k in order to find out which states S'_k are considered possible by player r in S_k. Formula ψ is then evaluated with respect to each of these possible states S'_k. Note that sequence $(S_k,\dots,S_{k+m})$ itself is not directly important for this evaluation. Instead, *all* deg(ψ)-max sequences emerging from S'_k have to be taken into account, which then of course indirectly includes $(S_k,\dots,S_{k+m})$ again due to the reflexivity of $\sim_r$. The beforementioned issue motivates that the degree of a formula of the form $K_r\psi$ is zero, as specified in Definition 4.1. Since the sole purpose of the deg-notion is the characterisation of all sequences which are relevant for the verification of a formula, and the verification of $K_r\psi$ requires no states beyond the current state (wrt. the currently considered state sequence), only sequences of length 0 need to be considered.

4.2.3 Satisfaction of the S5 Properties

Although the opinions concerning the interpretation of agent knowledge differ, the so-called *S5 properties* have widely been accepted as convenient conditions which should be satisfied by an appropriate knowledge reasoning formalism [FHMV95]. Propositional formulas enriched with modal operators K_r are known to satisfy these properties when interpreted with respect to *Kripke structures* (Kripke frames with an additional truth assignment for fluents in particular states) over equivalence relations. In the following, we show that the respective results (which are to be found, e.g., in [FHMV95]) can be adapted to our setting with formulas containing $\bigcirc$-operators and their semantics via finite state sequences, which implies that our semantics is well-defined.

Proposition 4.6 (Satisfaction of the S5 Properties)**.** *Let G be a playable and valid GDL specification and Δ_G be the set of developments over G. Then for all formulas $\varphi, \psi \in \mathcal{ESIN}_G$ the following properties, also known as the* S5 properties*, hold.*

1. *Distribution:*

$$\forall \delta \in \Delta_G : (last(\delta) \vDash_\delta K_r\varphi \wedge K_r(\varphi \supset \psi) \;\; implies \;\; last(\delta) \vDash_\delta K_r\psi)$$

2. *Knowledge Generalisation:*

$$(\forall \delta \in \Delta_G : last(\delta) \vDash_\delta \varphi) \;\; implies \;\; (\forall \delta \in \Delta_G : last(\delta) \vDash_\delta K_r\varphi)$$

3. *Knowledge:*

$$\forall \delta \in \Delta_G : (last(\delta) \models_\delta K_r\varphi \;\; implies \;\; last(\delta) \models_\delta \varphi)$$

4. *Positive Introspection:*

$$\forall \delta \in \Delta_G : (last(\delta) \models_\delta K_r\varphi \;\; implies \;\; last(\delta) \models_\delta K_rK_r\varphi)$$

5. *Negative Introspection:*

$$\forall \delta \in \Delta_G : (last(\delta) \models_\delta \neg K_r\varphi \;\; implies \;\; last(\delta) \models_\delta K_r\neg K_r\varphi)$$

Proof: First note that, since $last(\delta)$ is reachable for arbitrary developments $\delta \in \Delta_G$ and the GDL specification is playable, we can safely draw conclusions such as $last(\delta) \models_\delta \varphi \wedge \psi$ implying $last(\delta) \models_\delta \varphi$ and $last(\delta) \models_\delta \psi$ (cf. the discussion below the Sequence Extension Proposition 3.6 on page 32).

1. For an arbitrary development $\delta \in \Delta_G$, suppose $last(\delta) \models_\delta K_r\varphi \wedge K_r(\varphi \supset \psi)$. Then $last(\delta) \models_\delta K_r\varphi$ and $last(\delta) \models_\delta K_r(\varphi \supset \psi)$, which implies that, for all developments δ' s.t. $\delta \sim_r \delta'$, $last(\delta') \models_{\delta'} \varphi \wedge (\varphi \supset \psi)$ (by Proposition 3.6). This in turn yields $last(\delta') \models_{\delta'} \psi$ and hence $last(\delta) \models_\delta K_r\psi$.

2. Let $last(\delta) \models_\delta \varphi$ for all developments δ. Assume that there is a development δ' s.t. $last(\delta') \not\models_{\delta'} K_r\varphi$. Then there is a development δ'' s.t. $\delta' \sim_r \delta''$ and $last(\delta'') \not\models_{\delta''} \varphi$, in contradiction to $last(\delta) \models_\delta \varphi$ for all developments δ.

3. $last(\delta) \models_\delta K_r\varphi$ implies, for all developments δ' s.t. $\delta \sim_r \delta'$, $last(\delta') \models_{\delta'} \varphi$, and hence also $last(\delta) \models_\delta \varphi$ (by reflexivity of $\sim_r$).

4. Again suppose $last(\delta) \models_\delta K_r\varphi$. Assuming $last(\delta) \not\models_\delta K_rK_r\varphi$, there exist developments δ_1 and δ_2 with $\delta \sim_r \delta_1$ and $\delta_1 \sim_r \delta_2$ s.t. $last(\delta_2) \not\models_{\delta_2} \varphi$. Transitivity of $\sim_r$ yields $\delta \sim_r \delta_2$ and hence $last(\delta) \models_\delta \neg K_r\varphi$ in contradiction to $last(\delta) \models_\delta K_r\varphi$.

5. Suppose $last(\delta) \models_\delta \neg K_r\varphi$, then there is a development δ' s.t. $\delta \sim_r \delta'$ and $last(\delta') \not\models_{\delta'} \varphi$. Now for each development δ'' s.t. $\delta \sim_r \delta''$ we have $\delta'' \sim_r \delta$ by symmetry of $\sim_r$, and hence $\delta'' \sim_r \delta'$ by transitivity of $\sim_r$, which yields $last(\delta'') \models_{\delta''} \neg K_r\varphi$. This being true for *each* development δ'' s.t. $\delta \sim_r \delta''$ implies $last(\delta) \models_\delta K_r\neg K_r\varphi$. □

Note that, for epistemic sequence invariants φ such that $\deg(\varphi) = 0$,

$$last(\delta) \models_\delta \varphi \;\text{ implies }\; last(\delta) \models_\delta \psi$$

is equivalent to

$$last(\delta) \models_\delta \varphi \supset \psi,$$

since all state sequences σ starting in $last(\delta)$ collapse into the single state $last(\delta)$, resulting in the disappearance of quantification issues concerning these sequences which arise in the general case with arbitrary degree of φ. This correspondence allows, e.g., to equivalently view property *Distribution* from Proposition 4.6 to be a valid formula as follows:

$$\forall \delta \in \Delta_G : (\mathit{last}(\delta) \models_\delta (K_r\varphi \wedge K_r(\varphi \supset \psi)) \supset K_r\psi)$$

The same applies to the properties *Knowledge*, *Positive Introspection*, and *Negative Introspection*, but not to property *Knowledge Generalisation*.

Proposition 4.6 allows to draw some interesting conclusions regarding our goal to extend the verification method established in Section 3.4. For example, Property *Knowledge* allows to conclude that, if we already proved $K_r\varphi$ to be true in all reachable states, then also φ is true in all reachable states. Together with property *Knowledge Generalisation* this implies that, for a knowledge-free formula φ and arbitrary players $\{r_{i_1}, r_{i_2}, \ldots, r_{i_k}\}$, formula $\varphi_k = K_{r_{i_1}} K_{r_{i_2}} \ldots K_{r_{i_k}} \varphi$ holds in all reachable states if and only if φ holds in all reachable states and hence we can apply the original approach for knowledge-free formulas also to verify φ_k. However, note that property *Knowledge Generalisation* does *not* generalise to

$$\forall \delta \in \Delta_G : (\mathit{last}(\delta) \models_\delta \varphi \;\text{ implies }\; \mathit{last}(\delta) \models_\delta K_r\varphi)$$

Otherwise, for arbitrary reachable states S, player r would know every property which holds in S, implying that our specified semantics would not correctly grasp the intuition of knowledge properly. As an example to the contrary, consider the following.

Example 4.7 (Epistemic Sequence Invariant Semantics). Reconsider the formula

$$\varphi = \bigwedge_{r \in R} (\mathit{terminal} \supset K_r \mathit{terminal})$$

(it coincides with formula (4.3) on page 68) for property (4.1). It is not generally true in the game Krieg-Tictactoe, as argued at the end of Section 4.1. Hence, there must exist a development δ such that $\mathit{last}(\delta) \not\models_\delta \varphi$. Consider, for example, the development $\delta = S_{\mathit{init}} \xrightarrow{A_0} S_1 \xrightarrow{A_1} S_2 \xrightarrow{A_2} S_3 \xrightarrow{A_3} S_4 \xrightarrow{A_4} S_5$ with the following A_i:

$$\begin{array}{ll}
A_0 = \{x : \mathit{mark}(1,1), o : \mathit{noop}\} & A_1 = \{x : \mathit{noop}, o : \mathit{mark}(3,3)\} \\
A_2 = \{x : \mathit{mark}(1,2), o : \mathit{noop}\} & A_3 = \{x : \mathit{noop}, o : \mathit{mark}(3,2)\} \\
A_4 = \{x : \mathit{mark}(1,3), o : \mathit{noop}\} &
\end{array}$$

S_5 is terminal, hence we have $S_5 \models_\delta \mathit{terminal}$. Now consider the development $\delta' = S_{\mathit{init}} \xrightarrow{A_0} S_1 \xrightarrow{A_1} S_2 \xrightarrow{A_2} S_3 \xrightarrow{A_3} S_4 \xrightarrow{A_4'} S_5'$ with the following A_4':

$$A_4' = \{x : \mathit{mark}(2,1), o : \mathit{noop}\}$$

It deviates from δ only by the last move of player x (represented by joint-action A_4') where x marks cell $(2,1)$ instead of $(1,3)$, and by the resulting successor state S_5' which is non-terminal and hence such that $S_5' \not\models_{\delta'} \mathit{terminal}$. We have $\delta \sim_o \delta'$, as the actions and percepts of player o and the lengths of both developments coincide. Hence, $S_5 \not\models_\delta K_o \mathit{terminal}$, which implies that $S_5 \not\models_\delta \bigwedge_{r \in R}(\mathit{terminal} \supset K_r \mathit{terminal})$. ■

4.2.4 Complete Knowledge in the Initial State

Since a valid GDL specification contains a complete description of the initial state of the game, and since each player gets to know the complete specification prior to playing the game, each player knows all that is implied in the initial state. This is stated more formally with the following proposition.

Proposition 4.8 (Complete Knowledge in the Initial State). *Let φ be an epistemic sequence invariant over a valid GDL specification G, and let $\mathit{kf}_0(\varphi)$ be the formula obtained from φ by removing each occurrence of a knowledge operator which is not in the scope of any $\bigcirc$. Then*

$$S_{\mathit{init}} \models_{(S_{\mathit{init}})} \varphi \text{ iff } S_{\mathit{init}} \models_{(S_{\mathit{init}})} \mathit{kf}_0(\varphi)$$

Proof: Induction on the structure of φ.

Base Case: Consider an arbitrary epistemic sequence invariant φ such that $\varphi = \mathit{kf}_0(\varphi)$, then the claim follows immediately.

Induction Step: First note that any epistemic sequence invariant φ is composed of epistemic sequence invariants ψ which are such that $\psi = \mathit{kf}_0(\psi)$ (cf. the base case) via connectives different from $\bigcirc$. This allows to omit case $\varphi = \bigcirc\psi$ in the induction step proof, and all remaining connectives besides K_r yield the claim immediately by the induction hypothesis.

Now consider case $\varphi = K_r\psi$. $S_{\mathit{init}} \models_{(S_{\mathit{init}})} K_r\psi$ iff for all δ' such that $(S_{\mathit{init}}) \sim_r \delta'$, we have $\mathit{last}(\delta') \models_{\delta'} \psi$. Since the only δ' of that shape is $\delta' = (S_{\mathit{init}})$, the claim follows by the induction hypothesis. □

Note that applying the reduction $\mathit{kf}_0(\varphi)$ to the example formulas (4.3) and (4.4) (cf. page 68) yields the following two formulas:

$$\bigwedge_{r \in R} (\mathit{terminal} \supset \mathit{terminal})$$

$$\bigwedge_{r \in R} (\forall A : \mathit{ADom}(r))\,(\mathit{legal}(r, A) \vee \neg \mathit{legal}(r, A))$$

Both reduced formulas are true with respect to *arbitrary* initial game states in arbitrary games. By Proposition 4.8, this yields that also the original formulas (4.3) and (4.4) are true with respect to arbitrary initial game states in arbitrary games. This issue has some consequences on the generalised induction proof method we develop in this chapter, they will be addressed in Section 4.8.1.

4.3 Linear Time In The Setting Of Knowledge

A linear time structure has proved to be a beneficial tradeoff between expressibility and practical useability in our verification method for knowledge-free properties (cf. the discussion in Section 3.7.2). This motivates to keep this structure also in the generalisation of the proof method to knowledge formulas. In this section, we provide the necessary ingredients for establishing this proof method. To this end, we first define a class of formulas which is verifiable using a linear time structure in Section 4.3.1. Basically, it comprises all formulas which do not include formulations of what players do *not* know. Formulas of this class will hence be called positive-knowledge formulas. They share the attribute that each counter example can be represented as a collection of several (finite) state sequences, one of them being a real game development and one further sequence being associated with each occurrence of a knowledge operator of the considered formula. In Section 4.3.2, we introduce the notion of a view naming for a formula which allows to refer to each of these knowledge operators. Based on this notion, we then define sequence mappings as a formal structure to represent sequence collections in Section 4.3.3. Finally, we give an alternative semantics based on sequence mappings which will be needed to establish the soundness of our generalised proof method (Section 4.3.4), and prove that this semantics is equivalent to the original semantics for epistemic sequence invariants when considering positive-knowledge formulas (Section 4.3.5).

4.3.1 Positive-Knowledge Formulas

In the verification method for knowledge-free formulas φ developed in Chapter 3, we applied a linear time structure to represent counter examples, which amount to state sequences violating φ. This intuition can be generalised to the setting of knowledge for a subclass of epistemic sequence invariants. E.g., reconsider the formula $\varphi = \bigwedge_{r \in R}[\mathit{terminal} \supset K_r \mathit{terminal}]$ from (4.3) together with the argumentation against its validity from Example 4.7. Here, both given developments δ and δ' together can be considered a counter example for φ with respect to development δ. More generally, some formulas containing n occurrences of knowledge operators allow to consider each counter example as a collection of sequences of size $n+1$ which are related via the accessibility relation. Seen in the context of our verification approach, this allows to represent counter examples as answer sets using a linear time structure and hence enables to prove these formulas.

At the contrary, a counter example for the formula $\neg K_o \mathit{terminal}$ can *not* appropriately be characterised by a collection of single sequences: For an arbitrary development δ, we have $\mathit{last}(\delta) \not\models_\delta \neg K_o \mathit{terminal}$ iff $\mathit{last}(\delta) \models_\delta K_o \mathit{terminal}$. This is true iff, for *all* developments δ' such that $\delta \sim_o \delta'$, $\delta' \models_{\delta'} \mathit{terminal}$ holds. Intuitively, in addition to δ, a single counter example hence amounts to be a partial game tree, which is not representable with an answer set when using a linear time structure. Hence, in the following we restrict our attention to formulas that allow to faithfully characterise all potential counter examples as collection of finitely many state sequences. These will subsequently be called positive-knowledge formulas and are formally defined as follows.

Definition 4.9 (Positive- and Negative-Knowledge Formula)**.** *Let G be a valid GDL specification and $\varphi \in \mathcal{ESIN}_G$. φ is called* negative-knowledge *formula if $K_r\psi$ occurs in φ within the scope of some $\neg$, or within the scope of some $(\exists_{l..u} X : D_X)$ such that $u \neq \infty$. Otherwise, it is called* positive-knowledge *formula.* ■

For a positive-knowledge formula φ, $(\exists_{m..n} X : D_X)\, \varphi$ has to be treated as negative-knowledge formula for $n \neq \infty$, since, intuitively, the upper bound n involves a requirement of the form “not more than n instances a of X satisfy $\varphi[X/a]$” and hence incorporates negation as well. Positive-knowledge formulas do not characterise *all* formulas whose counter examples can be considered as collections of sequences. E.g., whenever φ is in that class, $\neg\neg\varphi$ should be in that class as well. We nevertheless decide to use this simpler characterisation for the benefit of a simpler notation.

Example 4.10 (Positive-Knowledge Formula). Consider a structurally simpler variant of formula (4.4) (cf. page 68), formulating that if it is legal for player x to mark cell $(1,1)$, then x knows about this:

$$\varphi = \neg \mathit{legal}(x, \mathit{mark}(\mathit{1},\mathit{1})) \vee K_x \mathit{legal}(x, \mathit{mark}(\mathit{1},\mathit{1})) \tag{4.5}$$

φ is a positive-knowledge formula which is valid in the game of Krieg-Tictactoe. It will serve as demonstration example for the subsequently developed extension of the verification method to positive-knowledge formulas. ■

4.3.2 View Namings

In Section 4.3.1 we have motivated that counter examples for positive-knowledge formulas can be represented by a collection of finite state sequences. In this section we provide the notion of a view naming which is needed to formally define these collections. Intuitively, a view naming assigns a different name to each part of a formula which has to be interpreted with respect to a possibly different development. A possibly different development (and hence a different name) is needed for each subformula ψ of a formula which is directly preceded by a knowledge operator K_r, as $K_r\psi$ is violated by a development δ if and only if ψ is violated by some possibly different development δ' which is such that $\delta \sim_r \delta'$. As the formula $K_r\varphi \vee K_r\neg\varphi$ shows, even different occurrences of the same knowledge operator may require different developments (and hence different names). In order to formally define a view naming, we have to introduce an auxiliary notion which allows to uniquely refer to subformulas of a formula φ in a way that distinguishes even syntactically equal subformulas occurring more than once.

Definition 4.11 (Position). *Let φ be an epistemic sequence invariant over a valid GDL description. For arbitrary subformulas ψ of φ, we define the* position of ψ in φ *recursively as follows.*

- *the position of φ in φ is ϵ;*
- *if the position of subformula $\neg\psi$ in φ is π, then the position of ψ in φ is $\pi\neg$ (similarly for $\bigcirc$ and K_r);*
- *if the position of subformula $\psi_1 \wedge \psi_2$ in φ is π, then the positions of ψ_1 and ψ_2 in φ are $\pi\wedge_1$ and $\pi\wedge_2$, respectively (similarly for $\psi_1 \vee \psi_2$); and*
- *if the position of subformula $(\exists\vec{X} : D_{\vec{X}})\,\psi[\vec{X}]$ in φ is π, then the position of $\psi[\vec{X}/\vec{t_i}]$ is $\pi\exists_i$, where for $D_{\vec{X}} = \{\vec{t_1}, \ldots, \vec{t_n}\}$ we arbitrarily fix an order such that $\vec{t_1} < \ldots < \vec{t_n}$ (similarly for $(\forall\vec{X} : D_{\vec{X}})\,\psi[\vec{X}]$ and $(\exists_{l..u}\vec{X} : D_{\vec{X}})\,\psi[\vec{X}]$).*

Furthermore, we write Pos_φ to denote the set of all positions of subformulas from φ. ■

Each subformula ψ of φ is in the scope of n knowledge operators $K_{r_1}, \ldots, K_{r_n}$ of φ for some $n \geq 0$. In case $n = 0$, ψ has to be evaluated with respect to a real game development δ_0. In case $n > 0$, ψ has to be evaluated with respect to a development δ_n which relates to a development δ_{n-1} such that player r_n considers δ_n possible in δ_{n-1}. In this case, ψ is evaluated with respect to a certain perspective, or *view*, which is exactly determined by the sequence of knowledge operators $K_{r_1}, \ldots, K_{r_n}$. The following definition of a view naming[1] characterises the view structure of φ by assigning unique names $v_0, v_1, \ldots$ to each of its subformulas such that two subformulas have the same name if and only if they are in the scope of the exact same occurrences of knowledge operators.

Definition 4.12 (View Naming). *Let G be a valid GDL specification and $\varphi \in \mathcal{ESIN}_G$ be a formula. A* view naming *for φ is a function $\mathcal{V}_\varphi : Pos_\varphi \to \{v_0, v_1, \ldots\}$ such that, for all positions $\pi_1, \pi_2 \in Pos_\varphi$ and their longest prefixes π'_1 and π'_2 which end in some K_{r_1} and K_{r_2} (where we assign $\pi'_i = \epsilon$ in case there is no such K_{r_i}):*

$$\mathcal{V}_\varphi(\pi_1) = \mathcal{V}_\varphi(\pi_2) \text{ iff } \pi'_1 = \pi'_2$$

We write $\mathcal{V}s_\varphi$ to denote the set $\{\mathcal{V}_\varphi(\pi) : \pi \in Pos_\varphi\}$ of all view names occurring in a view naming $\mathcal{V}_\varphi$. Furthermore, we define a function $\mathcal{L}_\varphi : Pos_\varphi \to \mathbb{N}$ such that $\mathcal{L}_\varphi(\pi)$ is the number of occurrences of $\bigcirc$ in π. For any subformula ψ of φ with position π, the natural number $\mathcal{L}_\varphi(\pi)$ will also be called the level of position π in φ*, or the* level of subformula ψ in φ*.* ■

Knowledge operators are referred to via positions, hence a formula of the form $K_r\varphi \vee K_r\psi$ results in different view names for the respective subformulas φ and ψ. This is necessary since a potential counter example does not necessarily consist of a unique sequence considered possible in a game development which violates both φ and ψ. In case $\psi = \neg\varphi$ there even is no such unique sequence, although $K_r\varphi \vee K_r\neg\varphi$ is still not necessarily valid. For similar reasons, subformulas are referred to via positions, yielding possibly different view names even for syntactically equal subformulas in case they do not occur in the scope of the same knowledge operators, which is also emphasised in our running example.

Example 4.13 (View Naming). Reconsider formula

$$\varphi = \neg \mathit{legal}(x, \mathit{mark}(1, 1)) \vee K_x \mathit{legal}(x, \mathit{mark}(1, 1))$$

(cf. (4.5) from Example 4.10). Subformula $\mathit{legal}(x, \mathit{mark}(1, 1))$ occurs twice, hence it has two positions $\pi_1 = \vee_1 \neg$ and $\pi_2 = \vee_2 K_x$. A view naming $\mathcal{V}_\varphi$ can be given for φ as follows:

- $\mathcal{V}_\varphi(\epsilon) = \mathcal{V}_\varphi(\vee_1) = \mathcal{V}_\varphi(\vee_2) = \mathcal{V}_\varphi(\vee_1 \neg) = v_0$
- $\mathcal{V}_\varphi(\vee_2 K_r) = v_1$

The level $\mathcal{L}_\varphi(\pi)$ of all positions $\pi \in Pos_\varphi$ is 0, as there is no occurrence of $\bigcirc$ in φ. ■

[1]Also the notion *world naming* is conceivable here. However, since a *possible world* is widely understood to refer to a state which is considered possible, but we are assigning names to subformulas instead of particular worlds, we decide to introduce a different notion.

4.3.3 Sequence Mappings

We now have all prerequisites we need for a formal characterisation of sequence collections for the representation of potential counter examples of positive-knowledge formulas. To motivate their necessity, we will now provide a short preview for the further development of this chapter. Sequence collections, which will be formalised via sequence mappings, can be seen as a natural extension of counter examples for knowledge-free formulas by a further dimension: while the temporal operator $\bigcirc$ requires the representation of counter examples as finite sequences and hence needed the inclusion of a time dimension in our verification process, the knowledge operator K_r will need a further dimension by requiring counter examples to consist of one finite state sequence *per view name*. Our proof method for knowledge-free formulas will be extended such that inconsistency of a base case program and an induction step program allows to conclude that a positive-knowledge formula is valid. This will be achieved as follows.

- The base case and induction step answer set programs from Section 3.4 will be extended by a further view dimension (in addition to the time dimension we introduced in Chapter 3). Each of its answer sets will then represent a sequence mapping for the encoded formula (this is not entirely correct for the induction step program, but we defer further details to a later section).
- An additional semantics for the interpretation of formulas with respect to sequence mappings is defined in Section 4.3.4, and its equivalence to the original semantics is established for positive-knowledge formulas in Section 4.3.5. This link then allows to conclude that an answer set for a positive-knowledge formula (which represents a sequence mapping) indeed represents a counter example for this formula with respect to the original semantics and hence allows to conclude that the formula is not valid.

The formal definition of a sequence mapping is given as follows. It assigns an appropriate development to each view name $v \in \mathcal{V}s_\varphi$ of a formula φ, obeying necessary relations of the assigned developments with respect to the accessibility relation. A detailed explanation of the definition can be found immediately thereafter.

Definition 4.14 (Sequence Mapping)**.** *Let G be a valid GDL specification. For a development $(S_0, \ldots, S_m)$, we define the k-prefix of $(S_0, \ldots, S_m)$ to be $(S_0, \ldots, S_{m'})$, where $m' = \min(k, m)$.*

Let φ be an epistemic sequence invariant over G, δ be a development, $\mathcal{V}_\varphi$ be a view naming for φ, $\mathcal{V}s_\varphi$ be the set of all view names of φ, and Δ_G the set of all developments over G. A sequence mapping for φ wrt. δ *is a function $\mathcal{M}_{\delta,\varphi} : \mathcal{V}s_\varphi \to \Delta_G$ such that:*

1. *for φ with position ϵ:*
$$\mathcal{M}_{\delta,\varphi}(\mathcal{V}_\varphi(\epsilon)) = (\delta, \sigma),$$
where σ is an arbitrary $\widehat{n}$-max sequence for some $\widehat{n} \geq \deg(\varphi)$;
2. *for each subformula $K_r\psi$ of φ with position π:*
$$\mathcal{M}_{\delta,\varphi}(\mathcal{V}_\varphi(\pi K_r)) = (\delta_\psi, \sigma_\psi),$$

where δ_ψ is an arbitrary development and σ_ψ is an arbitrary sequence starting at the last state of δ_ψ with the following conditions:

(a) For the $(|\delta| + \mathcal{L}_\varphi(\pi))$-prefix $\delta_{K_r\psi}$ of development $\mathcal{M}_{\delta,\varphi}(\mathcal{V}_\varphi(\pi))$, we have $\delta_{K_r\psi} \sim_r \delta_\psi$.

(b) Development $(\delta_\psi, \sigma_\psi)$ is $\widehat{n}$-max for some $\widehat{n} \geq (|\delta| + \mathcal{L}_\varphi(\pi) + \deg(\psi))$.

For a subformula ψ of φ at position π, we also write $\mathcal{M}_{\delta,\varphi}(\psi)$ instead of $\mathcal{M}_{\delta,\varphi}(\mathcal{V}_\varphi(\pi))$ when π is unimportant or clear from the context. Furthermore, for a sequence mapping $\mathcal{M}_{\delta,\varphi} : \mathcal{V}s_\varphi \to \Delta_G$ and a finite set of view names $\mathcal{V}s$, we write $\mathcal{M}_{\delta,\varphi}|_{\mathcal{V}s}$ to denote the sequence mapping with domain $\mathcal{V}s_\varphi \setminus \mathcal{V}s$ defined such that $\mathcal{M}_{\delta,\varphi}|_{\mathcal{V}s}(v) = \mathcal{M}_{\delta,\varphi}(v)$ for $v \in \mathcal{V}s_\varphi \setminus \mathcal{V}s$. ■

Sequence mappings range over view names from φ, hence all subformulas of φ within the scope of the same knowledge operators are assigned the same development, and are assigned separate developments otherwise. The top-level development (specified in item 1) includes a real game development δ. Each development for a subformula which is in the context of a knowledge operator (item 2) relates to a respective hierarchically higher development of the sequence mapping according to the accessibility relation.

We will shortly provide a more detailed explanation for the correspondence of sequence mappings to the semantics for epistemic sequence invariants given in Definition 4.5. The explanation requires a motivation for the following property of sequence mappings: For any view name v in the domain of a sequence mapping $\mathcal{M}_{\delta,\varphi}$, development $\mathcal{M}_{\delta,\varphi}(v)$ is at least of length $|\delta|$. This can be motivated by an inductive argument.

- For the base case we use item 1, hence the view name v equals $\mathcal{V}_\varphi(\epsilon)$. In this case, $\mathcal{M}_{\delta,\varphi}(v)$ incorporates δ itself as a prefix and is thus at least of length $|\delta|$.

- For the induction step we use item 2, hence there is a position πK_r such that view name v equals $\mathcal{V}_\varphi(\pi K_r)$. By the induction hypothesis there is a hierarchically higher development $\mathcal{M}_{\delta,\varphi}(\mathcal{V}_\varphi(\pi))$ of length $\geq |\delta|$. This implies that also the mentioned $(|\delta| + \mathcal{L}_\varphi(\pi))$-prefix $\delta_{K_r\psi}$ of development $\mathcal{M}_{\delta,\varphi}(\mathcal{V}_\varphi(\pi))$ is of length $\geq |\delta|$. Hence, the claimed relation $\delta_{K_r\psi} \sim_r \delta_\psi$ yields that also the prefix δ_ψ of development $\mathcal{M}_{\delta,\varphi}(\mathcal{V}_\varphi(\pi K_r))$ is of length $\geq |\delta|$, and thus that the development $\mathcal{M}_{\delta,\varphi}(\mathcal{V}_\varphi(\pi K_r))$ itself is of length $\geq |\delta|$.

We conclude that, for an arbitrary view name v from φ, its corresponding development $\mathcal{M}_{\delta,\varphi}(v)$ in a sequence mapping $\mathcal{M}_{\delta,\varphi}$ is of length $\geq |\delta|$ and can hence be written in the form $\mathcal{M}_{\delta,\varphi}(v) = (S_{init}, S_1, \ldots, S_{|\delta|}, \ldots, S_{|\delta|+m})$. This additionally implies that

$$(S_{|\delta|}, \ldots, S_{|\delta|+m}) \text{ is } \widehat{n}\text{-max for } \widehat{n} \geq \mathcal{L}_\varphi(\pi) + \deg(\psi). \tag{4.6}$$

Note, however, that suffix $(S_{|\delta|}, \ldots, S_{|\delta|+m})$ of $\mathcal{M}_{\delta,\varphi}(v)$ is not necessarily of length $\geq \mathcal{L}_\varphi(\pi)$, as it could be shorter and terminated.

Correspondence to the Formula Semantics

We can now have a closer look on the correspondence of sequence mappings from Definition 4.5 and the semantics for epistemic sequence invariants. To this end, the following considerations use the same namings and correspondent meanings as Definition 4.5.

- Item 1 defines the top-level development $\mathcal{M}_{\delta,\varphi}(\mathcal{V}_\varphi(\epsilon))$, which extends the given game development δ by an $\widehat{n}$-max sequence σ for some $\widehat{n} \geq \deg(\varphi)$ (note that the notation of a sequence composition (δ, σ) requires the last state of δ to be the first state of σ). Hence, development $\mathcal{M}_{\delta,\varphi}(\mathcal{V}_\varphi(\epsilon))$ is appropriate for the verification of φ with respect to the game development δ, i.e., for checking that $\sigma \not\models_\delta \varphi$.

- Item 2 considers a subformula $K_r\psi$ of φ with position π. We distinguish two cases for the development $\mathcal{M}_{\delta,\varphi}(\mathcal{V}_\varphi(\pi))$ which is of length $\geq |\delta|$ (cf. property (4.6)) and can hence be written as follows:

$$\mathcal{M}_{\delta,\varphi}(\mathcal{V}_\varphi(\pi)) = (S_{init}, S_1, \ldots, S_{|\delta|}, \ldots, S_{|\delta|+m})$$

  - $m \geq \mathcal{L}_\varphi(\pi)$: In this case, the length m of the suffix $\delta_\psi = (S_{|\delta|}, \ldots, S_{|\delta|+m})$ matches or exceeds the level $\mathcal{L}_\varphi(\pi)$ of formula $K_r\psi$ in φ. According to the semantics for epistemic sequence invariants, formula $K_r\psi$ does not hold with respect to the last state of prefix

$$\delta_{K_r\psi} = (S_{init}, S_1, \ldots, S_{|\delta|}, \ldots, S_{|\delta|+\mathcal{L}_\varphi(\pi)})$$

    of development $\mathcal{M}_{\delta,\varphi}(\mathcal{V}_\varphi(\pi))$ if and only if ψ does not hold with respect to the last state of some development δ_ψ such that $\delta_{K_r\psi} \sim_r \delta_\psi$. The definition of a sequence mapping emulates this semantics by relating the developments $\mathcal{M}_{\delta,\varphi}(\mathcal{V}_\varphi(\pi))$ and $\mathcal{M}_{\delta,\varphi}(\mathcal{V}_\varphi(\pi K_r))$ accordingly. By property (4.6) and since $m \geq \mathcal{L}_\varphi(\pi)$, the suffix σ_ψ of development $\mathcal{M}_{\delta,\varphi}(\mathcal{V}_\varphi(\pi K_r))$ starting at the last state of δ_ψ is $\widehat{n}$-max for some $\widehat{n} \geq \deg(\psi)$ and hence appropriate for the verification of ψ, i.e., for checking that $\sigma_\psi \not\models_{\delta_\psi} \psi$.
  - $m < \mathcal{L}_\varphi(\pi)$: In this case, the level of formula $K_r\psi$ in φ exceeds the length of development $\mathcal{M}_{\delta,\varphi}(\mathcal{V}_\varphi(\pi))$. Hence, $\mathcal{M}_{\delta,\varphi}(\mathcal{V}_\varphi(\pi))$ ends in a terminal state, as it is $\widehat{n}$-max for some $\widehat{n} \geq \mathcal{L}_\varphi(\pi) + \deg(K_r\psi)$ (by property (4.6)) and hence $\widehat{n}$-max for some $\widehat{n} \geq \mathcal{L}_\varphi(\pi)$ (since $\deg(K_r\psi) = 0$). In this case, the semantics for epistemic sequence invariants does not evaluate subformula $K_r\psi$, as $K_r\psi$ is a subformula of some formula $\bigcirc\rho$ which has been evaluated to true with respect to some terminated sequences (S_k) of length 0 (since $\mathcal{L}_\varphi(\pi) > m$). For technical reasons, we nevertheless relate the $(|\delta| + \mathcal{L}_\varphi(\pi))$-prefix $\delta_{K_r\psi}$ (which in this case equals the whole development $\mathcal{M}_{\delta,\varphi}(\mathcal{V}_\varphi(\pi))$) to δ_ψ such that $\delta_{K_r\psi} \sim_r \delta_\psi$, and require that $\mathcal{M}_{\delta,\varphi}(\mathcal{V}_\varphi(\pi K_r))$ is $\widehat{n}$-max for $\widehat{n} \geq (|\delta| + \mathcal{L}_\varphi(\pi) + \deg(\psi))$. This will be necessary to appropriately relate each answer set of a generated program in the generalised proof method to a sequence mapping.

Example 4.15 (Sequence Mapping). Consider a structurally simpler variant of formula (4.3) (cf. page 68), formulating that when the game has terminated, player o knows about this.

$$\varphi = \neg terminal \lor K_o terminal$$

A view naming $\mathcal{V}_\varphi$ for φ can be given similar to the one in Example 4.13 by assigning $\mathcal{V}_\varphi(\epsilon) = \mathcal{V}_\varphi(\vee_1) = \mathcal{V}_\varphi(\vee_2) = \mathcal{V}_\varphi(\vee_1 \neg) = v_0$ and assigning $\mathcal{V}_\varphi(\vee_2 K_o) = v_1$. Now reconsider the two developments δ and δ' from Example 4.7 on page 72:

$$\begin{aligned}\delta &= S_{init} \xrightarrow{A_0} S_1 \xrightarrow{A_1} S_2 \xrightarrow{A_2} S_3 \xrightarrow{A_3} S_4 \xrightarrow{A_4} S_5 \\ \delta' &= S_{init} \xrightarrow{A_0} S_1 \xrightarrow{A_1} S_2 \xrightarrow{A_2} S_3 \xrightarrow{A_3} S_4 \xrightarrow{A_4'} S_5'\end{aligned}$$

They deviate from each other in that player x marks a cell in joint action A_4 that yields a terminal state S_5, and marks another cell in joint action A_4' that yields a non-terminal state S_5'.

Based on these developments and the previously defined view naming, we can now specify a sequence mapping $\mathcal{M}_{\delta,\varphi}$ for φ as follows:

$$\begin{aligned}\mathcal{M}_{\delta,\varphi}(v_0) &= \delta \\ \mathcal{M}_{\delta,\varphi}(v_1) &= \delta'\end{aligned}$$

Since $\delta \sim_o \delta'$ (as motivated in Example 4.7), $\mathcal{M}_{\delta,\varphi}$ can easily be verified to match the requirements of a sequence mapping from Definition 4.14 and is hence well-defined. Subformula *terminal* with position $\vee_2 K_o$ is false with respect to the last state S_5' of development $\mathcal{M}_{\delta,\varphi}(v_1) = \delta'$, which yields (since $\delta \sim_o \delta'$) that formula $K_o\mathit{terminal}$ with position $\vee_2$ is false with respect to the last state S_5 of development $\mathcal{M}_{\delta,\varphi}(v_0) = \delta$. Since S_5 is terminal, this yields that $S_5 \not\models_\delta \varphi$. Hence, the two developments δ and δ' form a counter example for φ, and this counter example can formally be represented by the sequence mapping $\mathcal{M}_{\delta,\varphi}$. ∎

4.3.4 An Alternative Formula Semantics Over Sequence Mappings

The previous example shows at an intuitive level that a sequence mapping $\mathcal{M}_{\delta,\varphi}$ can be used to interpret formula φ by interpreting each subformula with respect to the sequence which is associated with its view name in $\mathcal{M}_{\delta,\varphi}$. In the following, we will formally grasp this intuition by defining an alternative semantics for epistemic sequence invariants over sequence mappings. As motivated in Section 4.3.3, this semantics will be needed to establish the link between an answer set for a program in our generalised proof method and the original semantics for formulas from Definition 4.5. More precisely, each answer set will represent a sequence mapping (with some additional subtleties for the induction step) which violates a positive-knowledge formula with respect to the alternative semantics, and the equivalence result we establish in Section 4.3.5 will allow to conclude that the respective formula is then also violated with respect to the original semantics.

Definition 4.16 (Semantics over Sequence Mappings). *Let G be a valid GDL specification, φ be an epistemic sequence invariant such that $\deg(\varphi) = n$, δ be a development, r be an arbitrary role according to the semantics of G, $\mathcal{P}$ be the set of ground atoms $p(\vec{t})$ over G such that $p \notin \{\mathbf{init}, \mathbf{next}\}$ and p does not depend on* **does** *in G, and $\mathcal{M}_{\delta,\varphi}$ be a sequence mapping for φ wrt. δ.*

We say that $\mathcal{M}_{\delta,\varphi}$ satisfies *φ, denoted $\mathcal{M}_{\delta,\varphi} \Vdash \varphi$, if $(\mathcal{M}_{\delta,\varphi}, |\delta|, \mathcal{V}_\varphi(\epsilon)) \Vdash \varphi$ holds according to the following definition:*

$(\mathcal{M}, k, v) \Vdash p$	*iff*	$G \cup S_k^{\mathtt{true}} \vdash p$, *where* $p \in \mathcal{P}$ *and* $\mathcal{M}(v) = (S_{init}, S_1, \ldots, S_k, \ldots, S_m)$
$(\mathcal{M}, k, v) \Vdash \neg\psi$	*iff*	$(\mathcal{M}, k, v) \nVdash \psi$
$(\mathcal{M}, k, v) \Vdash \psi_1 \wedge \psi_2$	*iff*	$(\mathcal{M}, k, v) \Vdash \psi_1$ *and* $(\mathcal{M}, k, v) \Vdash \psi_2$
$(\mathcal{M}, k, v) \Vdash \psi_1 \vee \psi_2$	*iff*	$(\mathcal{M}, k, v) \Vdash \psi_1$ *or* $(\mathcal{M}, k, v) \Vdash \psi_2$
$(\mathcal{M}, k, v) \Vdash (\exists \vec{X} : D_{\vec{X}})\, \psi[\vec{X}]$	*iff*	*there is an* $\vec{a} \in D_{\vec{X}}$ *s.t.* $(\mathcal{M}, k, v) \Vdash \psi[\vec{X}/\vec{a}]$
$(\mathcal{M}, k, v) \Vdash (\forall \vec{X} : D_{\vec{X}})\, \psi[\vec{X}]$	*iff*	*for all* $\vec{a} \in D_{\vec{X}}$*:* $(\mathcal{M}, k, v) \Vdash \psi[\vec{X}/\vec{a}]$
$(\mathcal{M}, k, v) \Vdash (\exists_{l..u} \vec{X} : D_{\vec{X}})\ \psi[\vec{X}]$	*iff*	*there are* $\geq l$ *and* $\leq u$ *different* $\vec{a} \in D_{\vec{X}}$ *s.t.* $(\mathcal{M}, k, v) \Vdash \psi[\vec{X}/\vec{a}]$
$(\mathcal{M}, k, v) \Vdash \bigcirc\psi$	*iff*	$k = m$ *or* $(\mathcal{M}, k+1, v) \Vdash \psi$, *where* $\mathcal{M}(v) = (S_{init}, S_1, \ldots, S_k, \ldots, S_m)$
$(\mathcal{M}, k, v) \Vdash K_r\psi$	*iff*	$(\mathcal{M}, k, \mathcal{V}_\varphi(\pi)) \Vdash \psi$, *where* π *is the position of* ψ *in* φ

■

All cases different from $K_r\psi$ exactly correspond to their counterparts in Definition 4.5: v refers to the view name of the currently considered subformula of φ and hence to the corresponding development $\mathcal{M}_{\delta,\varphi}(v) = (S_{init}, S_1, \ldots, S_k, \ldots, S_{k+m})$, and time step k determines the respective prefix $\delta = (S_{init}, S_1, \ldots, S_k)$.

Case $K_r\psi$, however, does not incorporate the accessibility relation anymore. Instead, it resolves $K_r\psi$ with respect to the development $\mathcal{M}_{\delta,\varphi}(K_r\psi)$[2] by considering ψ with respect to the appropriately related development $\mathcal{M}_{\delta,\varphi}(\psi)$. This reveals that $\mathcal{M}_{\delta,\varphi} \Vdash K_r\psi$ for a sequence mapping $\mathcal{M}_{\delta,\varphi}$ such that $\mathcal{M}_{\delta,\varphi}(K_r\psi) = (\delta, \sigma)$ does *not* generally imply the correspondent $\sigma \models_\delta K_r\psi$ in the original semantics: $\sigma \models_\delta K_r\psi$ requires that $last(\delta') \models_{\delta'} \psi$ holds for *all* developments δ' such that $\delta \sim_r \delta'$, whereas $\mathcal{M}_{\delta,\varphi} \Vdash K_r\psi$ only provides that correspondence for *one* such development δ'. However, following the spirit of a sequence mapping as structure for *counter examples* for a formula, it is possible to establish a correspondence between the existence of a sequence mapping $\mathcal{M}_{\delta,\varphi}$ such that $\mathcal{M}_{\delta,\varphi} \nVdash K_r\psi$ and the existence of a sequence σ such that $\sigma \not\models_\delta K_r\psi$, as $\sigma \not\models_\delta K_r\psi$ requires $last(\delta') \not\models_{\delta'} \psi$ to hold only for *one* development δ' such that $\delta \sim_r \delta'$.

A general result for the equivalence of both semantics with regard to positive-knowledge formulas will be established in the following section. It will need two propositions which provide useful correspondences between sequence mappings. The first proposition states that a sequence mapping over a formula φ can be altered to a sequence mapping over a subformula ψ of φ (by keeping all relevant developments and removing all remaining ones) and vice versa, such that the interpretation of ψ with respect to both sequence mappings yields the same result. It uses the notation $\mathcal{M}_{\delta,\varphi}|_{\mathcal{V}s_\psi}$ from the Sequence Mapping Definition 4.14, referring to a sequence mapping with reduced domain $\mathcal{V}s_\psi$ which is as $\mathcal{M}_{\delta,\varphi}$ on all view-name arguments from $\mathcal{V}s_\psi$.

Proposition 4.17 (Sequence-Mapping View Reduction). *Let G be a valid GDL specification, φ be an epistemic sequence invariant, ψ be a subformula of φ with the associated set of view names $\mathcal{V}s_\psi$, view name $\mathcal{V}_\varphi(\pi) = v$ for the position π of ψ,*

[2]Recall from the Sequence Mapping Definition 4.14 that for a subformula ρ of formula φ at position π, we also abbreviate $\mathcal{M}_{\delta,\varphi}(\mathcal{V}_\varphi(\pi))$ by $\mathcal{M}_{\delta,\varphi}(\rho)$.

$k \in \mathbb{N}$, δ be a development, and $\mathcal{M}_{\delta,\varphi}$ be a sequence mapping for φ wrt. δ. Then

$$(\mathcal{M}_{\delta,\varphi}, k, v) \Vdash \psi \quad \textit{iff} \quad (\mathcal{M}_{\delta,\varphi}|_{\mathcal{V}s_\psi}, k, v) \Vdash \psi.$$

Proof: Immediate, since the evaluation of a subformula ψ with respect to a structure $(\mathcal{M}_{\delta,\varphi}, k, v)$ only involves sequences $\mathcal{M}_{\delta,\varphi}(v)$ for view names v that correspond to subformulas from ψ. □

The second proposition is needed to establish a semantic correspondence between the interpretation of a formula $\bigcirc\varphi$ via a sequence mapping that is seen in connection with some development δ, and the interpretation of its respective subformula φ via the same sequence mapping when seen in connection with the development which prolongs δ by one state. It is divided into two parts which state the following.

1. A sequence mapping for $\bigcirc\varphi$ whose toplevel development (δ, σ) has length ≥ 1 can also be interpreted as a sequence mapping for φ and vice versa.
2. Given a structure which is a sequence mapping for both $\bigcirc\varphi$ and φ as specified in item 1, the interpretation (via this sequence mapping) of $\bigcirc\varphi$ at time level $|\delta|$ and of φ at time level $|\delta| + 1$ yields the same result.

Proposition 4.18 (Sequence-Mapping Correspondence). *Let G be a valid GDL specification, $\bigcirc\varphi$ be an epistemic sequence invariant, $\delta_{\bigcirc\varphi} = (S_{init}, \ldots, S_k)$ and $\delta_\varphi = (S_{init}, \ldots, S_k, S_{k+1})$ be arbitrary developments, and let $\sigma_{\bigcirc\varphi} = (S_k, \ldots, S_{k+m})$ and $\sigma_\varphi = (S_{k+1}, \ldots, S_{k+m})$ be arbitrary sequences. Furthermore, let $\mathcal{V}_{\bigcirc\varphi}$ and $\mathcal{V}_\varphi$ be view namings such that $\mathcal{V}_{\bigcirc\varphi}(\epsilon) = v$ and, for all $\pi \in Pos_\varphi$, $\mathcal{V}_{\bigcirc\varphi}(\bigcirc\pi) = \mathcal{V}_\varphi(\pi)$, and let $\mathcal{V}s$ denote the set of view names for $\mathcal{V}_{\bigcirc\varphi}$ and for $\mathcal{V}_\varphi$ (these two sets coincide).*

For each function $\mathcal{M} : \mathcal{V}s \to \Delta_G$ such that $\mathcal{M}(v) = (\delta_{\bigcirc\varphi}, \sigma_{\bigcirc\varphi}) = (\delta_\varphi, \sigma_\varphi)$, the following two statements hold:

1. *$\mathcal{M}$ is a sequence mapping for $\bigcirc\varphi$ wrt. $\delta_{\bigcirc\varphi}$ iff $\mathcal{M}$ is a sequence mapping for φ wrt. δ_φ.*
2. *If $\mathcal{M}$ is a sequence mapping for both $\bigcirc\varphi$ wrt. $\delta_{\bigcirc\varphi}$ and φ wrt. δ_φ, then $(\mathcal{M}, |\delta_{\bigcirc\varphi}|, \mathcal{V}_{\bigcirc\varphi}(\epsilon)) \Vdash \bigcirc\varphi$ iff $(\mathcal{M}, |\delta_\varphi|, \mathcal{V}_\varphi(\epsilon)) \Vdash \varphi$.*

Proof:

1. We show that the two conditions of the Sequence Mapping Definition 4.14 are satisfied for $\mathcal{M}$ with respect to $\bigcirc\varphi$ and $\delta_{\bigcirc\varphi}$ if and only if they are satisfied with respect to φ and δ_φ as follows:
   - Concerning item 1 of Definition 4.14: The top-level developments coincide, i.e., $\mathcal{M}(v) = (\delta_{\bigcirc\varphi}, \sigma_{\bigcirc\varphi}) = (\delta_\varphi, \sigma_\varphi)$). Furthermore, since $\deg(\bigcirc\varphi) = \deg(\varphi) + 1$ and $|\sigma_{\bigcirc\varphi}| = |\sigma_\varphi| + 1$, we have that $\sigma_{\bigcirc\varphi}$ is $n_{\bigcirc\varphi}$-max for some $n_{\bigcirc\varphi} \geq \deg(\bigcirc\varphi)$ iff σ_φ is n_φ-max for some $n_\varphi \geq \deg(\varphi)$.
   - Concerning item 2 of Definition 4.14: $K_r\psi$ is a subformula of $\bigcirc\varphi$ at level $l + 1$ iff $K_r\psi$ is subformula of φ at level l.
     - Concerning item 2 (a) of Definition 4.14: Since $(|\delta_{\bigcirc\varphi}| + (l + 1)) = ((|\delta_{\bigcirc\varphi}| + 1) + l) = (|\delta_\varphi| + l)$, the $(|\delta_{\bigcirc\varphi}| + (l + 1))$-prefix of $\mathcal{M}(K_r\psi)$ coincides with the $(|\delta_\varphi| + l)$-prefix of $\mathcal{M}(K_r\psi)$.

  - Concerning item 2 (b) of Definition 4.14: Similarly to item 2 (a), $(|\delta_{\bigcirc\varphi}| + (l+1) + \deg(\psi)) = (|\delta_\varphi| + l + \deg(\psi))$.

2. By the alternative formula semantics from Definition 4.16, we have that $(\mathcal{M}, |\delta_{\bigcirc\varphi}|, \mathcal{V}_{\bigcirc\varphi}(\epsilon)) \Vdash \bigcirc\varphi$ iff (since $\mathcal{M}(\mathcal{V}_{\bigcirc\varphi}(\epsilon))$ is $\widehat{n}$-max for some $\widehat{n} \geq |\delta| + \deg(\bigcirc\varphi)$ and S_{k+1} exists) $(\mathcal{M}, |\delta_{\bigcirc\varphi}| + 1, \mathcal{V}_{\bigcirc\varphi}(\epsilon)) \Vdash \varphi$ iff (since $\mathcal{V}_{\bigcirc\varphi}(\epsilon) = \mathcal{V}_{\bigcirc\varphi}(\bigcirc) = \mathcal{V}_\varphi(\epsilon)$ and $|\delta_{\bigcirc\varphi}| + 1 = |\delta_\varphi|$) $(\mathcal{M}, |\delta_\varphi|, \mathcal{V}_\varphi(\epsilon)) \Vdash \varphi$. □

4.3.5 Equivalence of the Two Formula Semantics

We are now ready to prove the equivalence of the original semantics given in Definition 4.5 (cf. page 69) and the alternative semantics on sequence mappings given in Definition 4.16 (cf. page 80) for positive-knowledge formulas. Informally, it states that a positive-knowledge formula is violated by the original semantics in the last state of a particular game development if and only if there is a sequence mapping for this formula with respect to the same game development that violates the formula in the alternative semantics. The result shows that the alternative semantics can be used to reliably interpret positive-knowledge formulas and will hence provide a correspondence between answer sets for generated answer set programs (which represent sequence mappings) and counter examples for formulas (with respect to the original semantics) in our generalised proof method.

Theorem 4.19 (Semantics Equivalence)**.** *Let G be a valid GDL specification, φ be a positive-knowledge formula over G, δ be a development, and σ be an $\widehat{n}$-max sequence starting at $last(\delta)$ and such that $\widehat{n} \geq \deg(\varphi)$. Then the following are equivalent.*

- $\sigma \not\models_\delta \varphi$
- *there is a sequence mapping $\mathcal{M}_{\delta,\varphi}$ such that $\mathcal{M}_{\delta,\varphi}(\varphi) = (\delta, \sigma)$ and $\mathcal{M}_{\delta,\varphi} \not\Vdash \varphi$*

Proof: <u>Base Case:</u> Consider an arbitrary knowledge-free formula φ such that $\mathcal{V}_\varphi(\epsilon) = v$. Note that, for the fixed sequence σ, there is exactly one sequence mapping $\mathcal{M}^\sigma_{\delta,\varphi}$ for φ wrt. δ such that $\mathcal{M}^\sigma_{\delta,\varphi}(v) = (\delta, \sigma)$. Hence, the claim follows by proving

$$\sigma \not\models_\delta \varphi \text{ iff } \mathcal{M}^\sigma_{\delta,\varphi} \not\Vdash \varphi$$

via a subsidiary induction on the structure of φ.

- Subsidiary Base Case $\varphi = p(\vec{t})$: $\sigma \not\models_\delta p(\vec{t})$ iff (Definition 4.5) $G \cup last(\delta) \nvdash p(\vec{t})$ iff (Definition 4.16) $(\mathcal{M}^\sigma_{\delta,\varphi}, |\delta|, v) \not\Vdash p(\vec{t})$ iff (Definition 4.16) $\mathcal{M}^\sigma_{\delta,\varphi} \not\Vdash p(\vec{t})$.
- Subsidiary Induction Step: Consider $\varphi = \bigcirc\psi$, and let $\sigma = (S_{|\delta|}, \ldots, S_{|\delta|+m})$. In case $S_{|\delta|}$ is terminal, we have $\sigma \models_\delta \varphi$ and $\mathcal{M}^\sigma_{\delta,\varphi} \Vdash \varphi$. Otherwise, $S_{|\delta|+1}$ exists, and $\sigma \not\models_\delta \varphi$ iff $(S_{|\delta|+1}, \ldots, S_{|\delta|+m}) \not\models_{(\delta,(S_{|\delta|},S_{|\delta|+1}))} \psi$ iff (by the Induction Hypothesis and the Sequence-Mapping Correspondence Proposition 4.18) $\mathcal{M}^\sigma_{\delta,\varphi} \not\Vdash \psi$ iff (again by Proposition 4.18) $\mathcal{M}^\sigma_{\delta,\varphi} \not\Vdash \varphi$. The other cases (including $\varphi = \neg\psi$ and $\varphi = (\exists_{l..u}\vec{X} : D_{\vec{X}})\ \psi$ for $u \neq \infty$) are argued similarly.

<u>Induction Step:</u> First note that cases $\varphi = \neg\psi$ and $\psi = (\exists_{l..u}\vec{X} : D_{\vec{X}})\ \psi$ for $u \neq \infty$ do not occur in the Induction Step due to the restriction of φ to a positive-knowledge formula. The remaining cases can be shown as follows.

- $\underline{\varphi = \psi_1 \wedge \psi_2}$: $\sigma \not\models_\delta \varphi$ implies $\sigma \not\models_\delta \psi_1$ or $\sigma \not\models_\delta \psi_2$. Assume $\sigma \not\models_\delta \psi_1$ (the other case is analogous). By the induction hypothesis (IH), there is a sequence mapping $\mathcal{M}_{\delta,\psi_1}$ such that $\mathcal{M}_{\delta,\psi_1}(\psi_1) = (\delta, \sigma)$ and $\mathcal{M}_{\delta,\psi_1} \not\Vdash \psi_1$. Now consider an arbitrary sequence mapping $\mathcal{M}_{\delta,\psi_2}$ for ψ_2 such that $\mathcal{M}_{\delta,\psi_2}(\psi_2) = \mathcal{M}_{\delta,\psi_1}(\psi_1)$ ($\mathcal{M}_{\delta,\psi_2}$ always exists since $\sim_r$ is reflexive). Without loss of generality, for the view namings $\mathcal{V}_{\psi_1}$ and $\mathcal{V}_{\psi_2}$, assume $\mathcal{V}_{\psi_1}(\epsilon) = \mathcal{V}_{\psi_2}(\epsilon)$ and $(\mathcal{V}s_{\psi_1} \setminus \{\mathcal{V}_{\psi_1}(\epsilon)\}) \cap (\mathcal{V}s_{\psi_2} \setminus \{\mathcal{V}_{\psi_2}(\epsilon)\}) = \emptyset$. We construct a sequence mapping $\mathcal{M}_{\delta,\varphi} : \mathcal{V}s_{\psi_1} \cup \mathcal{V}s_{\psi_2} \to \Delta_G$ such that
$$\mathcal{M}_{\delta,\varphi}(v) := \begin{cases} \mathcal{M}_{\delta,\psi_1}(v) & \text{if } v \in \mathcal{V}s_{\psi_1} \\ \mathcal{M}_{\delta,\psi_2}(v) & \text{else} \end{cases}$$
  $\mathcal{M}_{\delta,\psi_1} \not\Vdash \psi_1$ resolves to $(\mathcal{M}_{\delta,\psi_1}, |\delta|, \mathcal{V}_{\psi_1}(\epsilon)) \not\Vdash \psi_1$ which implies $(\mathcal{M}_{\delta,\varphi}, |\delta|, \mathcal{V}_\varphi(\epsilon)) \not\Vdash \psi_1$ (by the Sequence-Mapping View Reduction Proposition 4.17) and hence $\mathcal{M}_{\delta,\varphi} \not\Vdash \varphi$.

  For the opposite direction, let $\mathcal{M}_{\delta,\varphi}$ be a sequence mapping such that $\mathcal{M}_{\delta,\varphi}(\varphi) = (\delta, \sigma)$ and $\mathcal{M}_{\delta,\varphi} \not\Vdash \varphi$. Then $(\mathcal{M}_{\delta,\varphi}, |\delta|, \mathcal{V}_\varphi(\epsilon)) \not\Vdash \psi_1$ or $(\mathcal{M}_{\delta,\varphi}, |\delta|, \mathcal{V}_\varphi(\epsilon)) \not\Vdash \psi_2$ and hence (by the IH and Proposition 4.17) $\sigma \not\models_\delta \psi_1$ or $\sigma \not\models_\delta \psi_2$, which yields $\sigma \not\models_\delta \varphi$.

- $\underline{\varphi = \psi_1 \vee \psi_2}$: $\sigma \not\models_\delta \psi$ implies $\sigma \not\models_\delta \psi_1$ and $\sigma \not\models_\delta \psi_2$, hence by the IH there are sequence mappings $\mathcal{M}_{\delta,\psi_1}$ and $\mathcal{M}_{\delta,\psi_2}$ such that $\mathcal{M}_{\delta,\psi_1}(\psi_1) = \mathcal{M}_{\delta,\psi_2}(\psi_2) = (\delta, \sigma)$, $\mathcal{M}_{\delta,\psi_1} \not\Vdash \psi_1$, and $\mathcal{M}_{\delta,\psi_2} \not\Vdash \psi_2$. With these, we construct a sequence mapping $\mathcal{M}_{\delta,\varphi}$ as in the proof for case $\varphi = \psi_1 \wedge \psi_2$ which yields $\mathcal{M}_{\delta,\varphi} \not\Vdash \varphi$ by similar arguments.

  The opposite direction is argued similar to case $\varphi = \psi_1 \wedge \psi_2$.

- $\underline{\varphi = (\exists \vec{X} : D_{\vec{X}})\,\psi}$: Case $|D_X| = 0$ is immediate, and case $D_X = \{\vec{a}\}$ follows by choosing coinciding view namings for φ and $\psi[\vec{X}/\vec{a}]$. The remainder is similar to the proof of case $\varphi = \psi_1 \vee \psi_2$.

- $\underline{\varphi = (\forall \vec{X} : D_{\vec{X}})\,\psi}$: Similar to case $\varphi = (\exists \vec{X} : D_{\vec{X}})\,\psi$ (using arguments from case $\varphi = \psi_1 \wedge \psi_2$ instead of case $\varphi = \psi_1 \vee \psi_2$).

- $\underline{\varphi = (\exists_{l..\infty} \vec{X} : D_{\vec{X}})\ \psi}$: $\sigma \not\models_\delta \varphi$ implies that there are less than l different $\vec{a} \in D_{\vec{X}}$ s.t. $\sigma \models_\delta \psi[\vec{X}/\vec{a}]$, denote them with $\vec{A}$. Then for the remaining elements $\vec{b} \in D_{\vec{X}} \setminus \vec{A}$ we have $\sigma \not\models_\delta \psi[\vec{X}/\vec{b}]$ and hence, by IH, there are sequence mappings $\mathcal{M}_{\delta,\psi[\vec{X}/\vec{b}]}$ such that $\mathcal{M}_{\delta,\psi[\vec{X}/\vec{b}]}(\psi[\vec{X}/\vec{b}]) = (\delta, \sigma)$ and $\mathcal{M}_{\delta,\psi[\vec{X}/\vec{b}]} \not\Vdash \psi[\vec{X}/\vec{b}]$. Merging similar to the proof of case $\varphi = \psi_1 \vee \psi_2$ yields a sequence mapping $\mathcal{M}_{\delta,\varphi}$ such that, for all $\vec{b} \in D_{\vec{X}} \setminus \vec{A}$, we have $\mathcal{M}_{\delta,\varphi} \not\Vdash \psi[\vec{X}/\vec{b}]$. Hence there are less than l different $\vec{a} \in \vec{A}$ s.t. $\mathcal{M}_{\delta,\varphi} \Vdash \psi[\vec{X}/\vec{a}]$, which gives the claim.

  The opposite direction is argued similarly.

- $\underline{\varphi = \bigcirc\psi}$: Let $\sigma = (S_{|\delta|}, S_{|\delta|+1}, \ldots, S_{|\delta|+m})$. $\sigma \not\models_\delta \bigcirc\psi$ implies that $S_{|\delta|+1}$ exists, hence for $\sigma' = (S_{|\delta|+1}, \ldots, S_{|\delta|+m})$ and $\delta' = (S_{init}, \ldots, S_{|\delta|}, S_{|\delta|+1})$ we have $\sigma' \not\models_{\delta'} \psi$, which (by the IH) implies the existence of a sequence mapping $\mathcal{M}_{\delta',\psi}$ such that $\mathcal{M}_{\delta',\psi}(\psi) = (\delta', \sigma')$ and $\mathcal{M}_{\delta',\psi} \not\Vdash \psi$. Without loss of generality, let $\mathcal{V}_\varphi(\bigcirc\pi) = \mathcal{V}_\psi(\pi)$ for all $\pi \in Pos_\psi$ (and note that $\mathcal{V}_\varphi(\epsilon) = \mathcal{V}_\varphi(\bigcirc))$. Define $\mathcal{M}_{\delta,\varphi} : \mathcal{V}s_\varphi \to$

Δ_G by setting $\mathcal{M}_{\delta,\varphi}(v) := \mathcal{M}_{\delta',\psi}(v)$ for all $v \in \mathcal{V}s_\varphi = \mathcal{V}s_\psi$. By the Sequence-Mapping Correspondence Proposition 4.18, $\mathcal{M}_{\delta,\varphi}$ is also a sequence mapping for φ, and since $\mathcal{M}_{\delta',\psi} \nVdash \psi$, we also have $\mathcal{M}_{\delta,\varphi} \nVdash \varphi$.

For the opposite direction, let $\mathcal{M}_{\delta,\varphi}$ be a sequence mapping such that $\mathcal{M}_{\delta,\varphi}(\varphi) = (\delta,\sigma)$ and $\mathcal{M}_{\delta,\varphi} \nVdash \varphi$. Then $|\mathcal{M}_{\delta,\varphi}(\varphi)| > |\delta|$ (otherwise $\mathcal{M}_{\delta,\varphi}$ would satisfy φ according to Definition 4.16) and hence $(\delta,\sigma) = (\delta',\sigma')$ for some δ' and σ' such that $|\delta'| = |\delta|+1$ and $|\sigma'| = |\sigma|-1$. Moreover, $(\mathcal{M}_{\delta,\varphi}, |\delta|+1, \mathcal{V}_\varphi(\epsilon)) \nVdash \psi$ (again by Definition 4.16) which (by the IH and since $\mathcal{M}_{\delta,\varphi}$ is also a sequence mapping for ψ with respect to δ' by Proposition 4.18) yields $\sigma' \nvDash_{\delta'} \psi$ and hence $\sigma \nvDash_\delta \varphi$.

- $\underline{\varphi = K_r\psi}$: $\sigma \nvDash_\delta K_r\psi$ implies that there is a development $\delta' = (S_{init}, S'_1, \ldots, S'_{|\delta|})$ such that $\delta \sim_r \delta'$ and $S'_{|\delta|} \nvDash_{\delta'} \psi$, and hence that there exists a $\deg(\psi)$-max sequence σ' starting in $S'_{|\delta|}$ which is such that $\sigma' \nvDash_{\delta'} \psi$. By the IH, there is a sequence mapping $\mathcal{M}_{\delta',\psi}$ such that $\mathcal{M}_{\delta',\psi} = (\delta',\sigma')$ and $\mathcal{M}_{\delta',\psi} \nVdash \psi$. Without loss of generality, let $\mathcal{V}_\varphi(K_r\pi) = \mathcal{V}_\psi(\pi)$ for all $\pi \in Pos_\psi$ (and note that $\mathcal{V}_\varphi(\epsilon) \neq \mathcal{V}_\varphi(K_r)$). Define $\mathcal{M}_{\delta,\varphi} : \mathcal{V}s_\varphi \to \Delta_G$ by setting $\mathcal{M}_{\delta,\varphi}(\mathcal{V}_\varphi(\epsilon)) := (\delta,\sigma)$ and $\mathcal{M}_{\delta,\varphi}(v) := \mathcal{M}_{\delta',\psi}(v)$ for all $v \in (\mathcal{V}s_\varphi \setminus \{\mathcal{V}_\varphi(\epsilon)\}) = \mathcal{V}s_\psi$. We argue that $\mathcal{M}_{\delta,\varphi}$ is a sequence mapping for φ wrt. δ according to Definition 4.14 as follows.

  1. The toplevel development $\mathcal{M}_{\delta,\varphi}(\mathcal{V}_\varphi(\epsilon)) = (\delta,\sigma)$ clearly satisfies the requirements of item 1 in Definition 4.14.
  2. Consider subformula $K_r\psi$ of φ which equals φ and hence has position $\pi = \epsilon$. Then $\mathcal{M}_{\delta,\varphi}(\mathcal{V}_\varphi(K_r)) = (\delta',\sigma')$, which clearly satisfies the requirements (a) and (b) of item 2 in Definition 4.14. For the remaining subformulas, item 2 is satisfied as $\mathcal{M}_{\delta,\varphi}$ includes the subsequent sequence mapping $\mathcal{M}_{\delta',\psi}$.

  Hence $\mathcal{M}_{\delta',\psi} \nVdash \psi$ implies $\mathcal{M}_{\delta,\varphi} \nVdash \varphi$ (using the Sequence-Mapping View Reduction Proposition 4.17).

  For the opposite direction, let $\mathcal{M}_{\delta,\varphi}$ be a sequence mapping such that $\mathcal{M}_{\delta,\varphi}(\varphi) = (\delta,\sigma)$ and $\mathcal{M}_{\delta,\varphi} \nVdash \varphi$. Then $\mathcal{M}_{\delta,\varphi}|_{\mathcal{V}s_\psi} \nVdash \psi$ (by Proposition 4.17). By Definition 4.14 and property (4.6), for some development δ' and sequence σ' such that $\delta \sim_r \delta'$ and σ' is $\widehat{n}$-max for $\widehat{n} \geq \deg(\psi)$, we have $\mathcal{M}_{\delta,\varphi}|_{\mathcal{V}s_\psi}(\psi) = (\delta',\sigma')$. The IH implies $\sigma' \nvDash_{\delta'} \psi$, which in turn gives $\sigma \nvDash_\delta \varphi$. □

Let us emphasise that Theorem 4.19 indeed does not hold for negative-knowledge formulas. For this purpose, reconsider Example 4.7 (cf. page 72) regarding Krieg-Tictactoe with the terminal development δ, and the non-terminal development δ' considered possible by player o which is such that $\delta \sim_o \delta'$. Player o does not know whether the game has terminated in δ (as motivated in Example 4.7), i.e. we have

$$last(\delta) \vDash_\delta \neg K_o terminal,$$

and $\varphi = \neg K_o terminal$ is a negative-knowledge formula.

Now assume for a moment that Theorem 4.19 holds for φ. Then for all sequence mappings $\mathcal{M}_{\delta,\varphi}$ such that $\mathcal{M}_{\delta,\varphi}(\varphi) = \delta$ we must have that $\mathcal{M}_{\delta,\varphi} \Vdash \varphi$. However, there is a sequence mapping which violates this condition: consider $\mathcal{M}_{\delta,\varphi}$ for φ with respect to δ that maps each view name of φ to the development δ. Then $\mathcal{M}_{\delta,\varphi} \Vdash K_o terminal$

and hence $\mathcal{M}_{\delta,\varphi} \nVdash \varphi$, which implies that our assumption is wrong. The rationale behind this issue has been given with the discussion following the definition of the alternative semantics in Section 4.3.4.

In terms of the proof for Theorem 4.19, the issue arises for $\neg\psi$ in the induction step, where the direction from the second to the first item cannot be established: assume an arbitrary sequence mapping $\mathcal{M}_{\delta,\varphi}$ such that $\mathcal{M}_{\delta,\varphi}(\neg\psi) = (\delta, \sigma)$ and $\mathcal{M}_{\delta,\varphi} \nVdash \neg\psi$. Then $\mathcal{M}_{\delta,\varphi} \Vdash \psi$ is only known to hold for one particular sequence mapping $\mathcal{M}_{\delta,\varphi}$. In order to apply the induction hypothesis, however, this property is required for *all* sequence mappings $\mathcal{M}_{\delta,\varphi}$ such that $\mathcal{M}_{\delta,\varphi}(\psi) = (\delta, \sigma)$, which would then yield $\sigma \models_\delta \psi$ and hence $\sigma \not\models_\delta \neg\psi$. For similar reasons, $(\mathcal{M}_{\delta,\varphi}, i) \nVdash (\exists_{l..u}\vec{X} : D_{\vec{x}})\ \psi$ for $u \neq \infty$ does *not* generally imply $(\delta, \sigma) \not\models_\delta (\exists_{l..u}\vec{X} : D_{\vec{x}})\ \psi$.

4.4 Prerequisites for the Generalised Verification Method

We will now lift the preliminary notions we have established for our proof method in Section 3.3 to the knowledge setting. To this end, we first provide an additional dimension to the temporal GDL extension by introducing a further view argument (Section 4.4.1), which enables reasoning about the GDL via sequence mappings. Thereafter, we also enrich formula encodings by this view argument and provide an extended sample encoding (Section 4.4.2), which together with the view-extended GDL clauses then allows to verify formulas with respect to sequence mappings using answer set programs. These prerequisites will form the basis for the generalised proof method (to be introduced in Section 4.5) which constructs programs whose answer sets represent counter examples of formulas in the form of sequence mappings.

4.4.1 Epistemic Temporal GDL Extension

In Section 3.3.1, we introduced the temporal GDL extension $G_{\leq n}$. It allows to consider n successive state transitions in an answer set encoding of a GDL specification G and hence enables to automatically search for counter examples (which amount to be state sequences) of sequence invariants with degree n. In this section, we generalise this extension to account for epistemic sequence invariants, hence extending the structure of counter examples from single state sequences to finite collections of state sequences (represented by sequence mappings). To this end, the temporal GDL clauses will be extended by a further view argument. While the appropriate time level for the GDL extension in the knowledge-free setting is given by a natural number n, we will now need such a number for each of the views of a formula. The following notion provides this information.

Definition 4.20 (Formula Signature). *Let φ be an epistemic sequence invariant over a valid GDL specification. A* φ-signature *is a function $Sig : \mathcal{V}s_\varphi \to \mathbb{N}$ such that*

- *$Sig(\mathcal{V}_\varphi(\epsilon)) \geq \deg(\varphi)$; and*
- *for each subformula $K_r\psi$ of φ at position π, $Sig(\mathcal{V}_\varphi(\pi)) \geq \mathcal{L}_\varphi(\pi) + \deg(\psi)$.*

A φ-signature Sig is minimal *if, for all φ-signatures Sig' and for all $v \in \mathcal{V}s_\varphi$, we have $Sig(v) \leq Sig'(v)$. Moreover, a sequence mapping $\mathcal{M}_{\delta,\varphi}$* has φ-signature *Sig if, for each $v \in \mathcal{V}s_\varphi$, $\mathcal{M}_{\delta,\varphi}(v)$ is $|\delta| + Sig(v)$-max.* ■

Sequence mappings can have more than one φ-signature in case one of its developments is terminal. Moreover, if a φ-signature Sig' is greater than another φ-signature Sig with respect to each component, and a sequence mapping $\mathcal{M}'_{\delta,\varphi}$ over that greater signature Sig' incorporates all developments from a sequence mapping $\mathcal{M}_{\delta,\varphi}$ over Sig, then $\mathcal{M}_{\delta,\varphi}$ and $\mathcal{M}'_{\delta,\varphi}$ coincide with respect to entailment of φ. This is stated more precisely with the following proposition which generalises item 2 in the Sequence Extension Proposition 3.6 from page 32.

Proposition 4.21 (Sequence-Mapping Length Extension). *Let G be a valid GDL specification, φ be an epistemic sequence invariant, δ be a development, and let Sig and Sig' be φ-signatures such that $Sig(v) \leq Sig'(v)$ for all $v \in \mathcal{V}s_\varphi$. Furthermore, let $\mathcal{M}_{\delta,\varphi}$ and $\mathcal{M}'_{\delta,\varphi}$ be two sequence mappings for φ with respect to δ such that $\mathcal{M}_{\delta,\varphi}$ has signature Sig and $\mathcal{M}'_{\delta,\varphi}$ has signature Sig' and, for all $v \in \mathcal{V}s_\varphi$, $\mathcal{M}_{\delta,\varphi}(v)$ is the $(|\delta| + Sig(v))$-prefix of $\mathcal{M}'_{\delta,\varphi}(v)$. Then*

$$\mathcal{M}_{\delta,\varphi} \Vdash \varphi \ \textit{ iff } \ \mathcal{M}'_{\delta,\varphi} \Vdash \varphi.$$

Proof: The claim follows by induction on the structure of φ, where the base case considers φ arbitrarily knowledge-free and uses the Sequence Extension Proposition 3.6. □

Note that, for each sequence mapping $\mathcal{M}'_{\delta,\varphi}$ of the form mentioned in Proposition 4.21, the corresponding reduced mapping $\mathcal{M}_{\delta,\varphi}$ always satisfies the conditions of a sequence mapping from Definition 4.14: the suffixes of developments in $\mathcal{M}'_{\delta,\varphi}$ which are removed in $\mathcal{M}_{\delta,\varphi}$ always exceed the degree of the respective subformula and are hence not involved in any dependency with respect to the accessibility relation. Conversely, this structural independency implies that each sequence mapping $\mathcal{M}_{\delta,\varphi}$ of the form mentioned in Proposition 4.21 can be extended to a respective sequence mapping $\mathcal{M}'_{\delta,\varphi}$ in case the GDL specification is playable (using the Sequence Extension Proposition 3.6).

For an arbitrary GDL specification G and $n \in \mathbb{N}$, recall the notation $G_{\leq n}$ for the temporal extension of G of degree n (cf. Definition 3.7 on page 34). It will form the basis for the following *epistemic* temporal extension of G. The epistemic extension will provide a set of temporal GDL clauses for each formula view, where the maximal time level of each set is given via a formula signature. In addition, the following definition provides an encoding for sequence mappings $\mathcal{M}$ over a development δ. The encoding will only incorporate suffixes of sequences from $\mathcal{M}$ starting at depth $|\delta|$, the reason for this speciality is given below.

Definition 4.22 (Epistemic Temporal GDL Extension). *For any set of clauses C and any view name v, By $C[+v]$ we denote the set of clauses obtained from C by extending each occurring atom $p(\vec{t})$ to $p(\vec{t}, v)$.*

Let G be a valid GDL specification, φ be a formula, and Sig be a φ-signature. The epistemic temporal extension of G wrt. Sig*, denoted G_{Sig}, is defined as follows:*

$$G_{Sig} := \bigcup_{v \in \mathcal{V}s_\varphi} G_{\leq Sig(v)}[+v]$$

Let δ be a development and $\mathcal{M}_{\delta,\varphi}$ be a sequence mapping for φ wrt. δ. Based on the temporal state encodings $S^{\texttt{true}}(i)$ and the temporal action encodings $A^{\texttt{does}}(i)$ for sequences (cf. Definition 3.7), we define their generalisations to sequence mappings.

- *For an arbitrary view name v, let S_v denote the last state of the $|\delta|$-prefix of development $\mathcal{M}_{\delta,\varphi}(v)$. Then the state encoding for $\mathcal{M}_{\delta,\varphi}$ is defined as*

$$\mathcal{M}^{\mathtt{true}}_{\delta,\varphi} = \bigcup_{v \in \mathcal{V}s_\varphi} S^{\mathtt{true}}_v(0)[+v].$$

- *For an arbitrary view name v, let the length of development $\mathcal{M}_{\delta,\varphi}(v)$ be $|\delta|+m_v$. Furthermore, for arbitrary $i < m_v$, let $A_{v,i}$ denote the joint action which is performed in $\mathcal{M}_{\delta,\varphi}(v)$ at the last state of the $(|\delta|+i)$-prefix of $\mathcal{M}_{\delta,\varphi}(v)$. Then the action encoding for $\mathcal{M}_{\delta,\varphi}$ is defined as*

$$\mathcal{M}^{\mathtt{does}}_{\delta,\varphi} = \bigcup_{v \in \mathcal{V}s_\varphi} \Big(\bigcup_{0 \le i < m_v} A^{\mathtt{does}}_{v,i}(i)[+v]\Big).$$

■

As for the temporal GDL extension, we choose to simply add a view-name argument to atoms p instead of encoding the view name into their respective predicate symbols, and will nevertheless tacitly assume each epistemic temporal GDL extension to be stratified (cf. the remark below Example 3.8 on page 34). Furthermore, for each view name $v \in \mathcal{V}s_\varphi$, the encoding $\mathcal{M}^{\mathtt{true}}_{\delta,\varphi} \cup \mathcal{M}^{\mathtt{does}}_{\delta,\varphi}$ of a sequence mapping contains the v-extended encoding of the suffix starting at $S_{|\delta|}$ of $\mathcal{M}_{\delta,\varphi}(v) = (S_{init}, S_1, \ldots, S_{|\delta|}, \ldots, S_{|\delta|+m})$, i.e., all prefixes of length $|\delta|$ are omitted. This is always possible, as each sequence in $\mathcal{M}_{\delta,\varphi}$ is at least of length $|\delta|$ (cf. the argumentation below the Sequence Mapping Definition 4.14 on page 77). The necessity for the omitted prefixes will become clear in the (yet to be established) induction-step part of the proof method in Section 4.5.2. To provide a short glimpse, the induction step has to abstract from a game development as it considers *arbitrary* states whose respective developments cannot be encoded into a single program. The set of answer sets for the induction step program will hence correspond to (a superset of) partial sequence mappings which start at some depth $|\delta|$. Let us now consider an example which shows the principle of the previous definition.

Example 4.23 (Epistemic Temporal GDL Extension). Reconsider formula

$$\varphi = \neg legal(x, mark(1,1)) \vee K_x legal(x, mark(1,1))$$

and the view naming $\mathcal{V}_\varphi$ such that $\mathcal{V}_\varphi(\epsilon) = v_0$ and $\mathcal{V}_\varphi(\vee_2 K_r) = v_1$ (cf. Example 4.13 from page 76). Then $Sig : \{v_0, v_1\} \to \mathbb{N}$, defined such that $Sig(v_0) = Sig(v_1) = 0$, is a φ-signature. Let G be the game description for Krieg-Tictactoe from Figure 4.2 and consider the clause in line 28:

```
sees(R,yourmove) :- not validmove, true(control(R)).
```

The epistemic temporal extension G_{Sig} contains the following clause for each $v \in \{v_0, v_1\} = \mathcal{V}s_\varphi$:

```
sees(R,yourmove,1,v) :- not validmove(0,v), true(control(R),0,v).
```

■

We will now provide a theorem which shows that an epistemically and temporally extended GDL specification can be used to reason over the GDL via sequence mappings. It is a natural extension of the correctness result for the temporal GDL extension (cf. Theorem 3.9 at page 35).

Theorem 4.24 (Correctness of the Epistemic Temporal GDL Extension)**.** *Let G be a valid GDL specification, φ be an epistemic sequence invariant, δ a development, $\mathcal{M}_{\delta,\varphi}$ a sequence mapping for φ wrt. δ, and Sig be a φ-signature for $\mathcal{M}_{\delta,\varphi}$. Consider the program $P = \mathcal{M}^{\texttt{true}}_{\delta,\varphi} \cup G_{Sig} \cup \mathcal{M}^{\texttt{does}}_{\delta,\varphi}$, the view name $v \in \mathcal{V}s_\varphi$, and let $\mathcal{M}_{\delta,\varphi}(v) = (S_{init}, S_1, \ldots, S_{|\delta|}, \ldots, S_{|\delta|+m})$.*

1. *For all $0 \leq i \leq m$ and predicate symbols $p \notin \{\textbf{init}, \textbf{next}\}$ that do not depend on* **does***:*
$$G \cup S^{\texttt{true}}_{|\delta|+i} \vdash p(\vec{t}) \textit{ iff } P \vdash p(\vec{t}, i, v).$$

2. *For all $0 \leq i \leq m-1$:*
$$G \cup S^{\texttt{true}}_{|\delta|+i} \cup A^{\texttt{does}}_{|\delta|+i} \vdash \textbf{sees}(r,p) \textit{ iff } P \vdash \textbf{sees}(r,p,i+1,v).$$

Proof:

1. There is a partition $P = \dot{\bigcup}_{v \in \mathcal{V}s_\varphi} P_v$ of program P such that each P_v contains only clauses concerning atoms with view argument v. Each P_v is stratified and hence admits a unique answer set $\mathcal{A}_v$. The answer sets are pairwise disjoint and hence form a partition of an answer set $\mathcal{A}$, i.e. such that $\mathcal{A} = \dot{\bigcup}_{v \in \mathcal{V}s_\varphi} \mathcal{A}_v$ (by the Splitting Theorem 2.10). Let P' be as program P from Theorem 3.9. Since the maximal time horizon of clauses concerning v in G_{Sig} is $Sig(v)$, and the respective sequence $\mathcal{M}_{\delta,\varphi}(v)$ is $Sig(v)$-max (cf. property (4.6) on page 78), we have that program P_v is equal to $P'[+v] \cup (G_{\leq Sig(v)} \setminus G_{\leq m})[+v]$ for each $v \in \mathcal{V}s_\varphi$. We can apply Theorem 3.9 to obtain the claim, as heads of $(G_{\leq Sig(v)} \setminus G_{\leq m})[+v]$ do not occur in $P'[+v]$ (again by the Splitting Theorem).

2. Let $0 \leq i \leq m-1$. By an argumentation similar to the proof of the intermediate result $S_i = \{f : P_i \vdash \textbf{true}(f,i)\}$ for Theorem 3.9, using $\textbf{sees}(r,p)$ instead of $\textbf{next}(f)$, the following two statements are equivalent:

   - $G \cup S^{\texttt{true}}_{|\delta|+i} \cup A^{\texttt{does}}_{|\delta|+i} \vdash \textbf{sees}(r,p)$
   - $G_{\leq i} \cup S^{\texttt{true}}_{|\delta|}(0) \cup \bigcup_{j=0}^{i} A^{\texttt{does}}_{|\delta|+j}(j) \vdash \textbf{sees}(r,p,i+1)$

   The remainder follows by arguments concerning partitions of P and $\mathcal{A}$ similar to the proof of item 1. □

Note that program P from Theorem 4.24 does not incorporate the game development δ, as the encoding $\mathcal{M}^{\texttt{true}}_{\delta,\varphi} \cup \mathcal{M}^{\texttt{does}}_{\delta,\varphi}$ of sequence mapping $\mathcal{M}_{\delta,\varphi}$ only considers suffixes of its sequences which omit the first $|\delta|$ states (as pointed out below the epistemic temporal GDL extension Definition 4.22). However, the restrictions put by development δ as well as other developments which are related to δ via the accessibility relation, are still met by the unaltered suffixes of $\mathcal{M}_{\delta,\varphi}$. Again, this abstraction is necessary for our induction step encoding which will have to abstract from a given game development.

4.4.2 Encoding Positive-Knowledge Formulas

Corresponding to the generalisation of the temporal GDL extension to incorporate epistemic sequence invariants, we will now generalise the notion of a formula encoding. The original encoding definition requires that a formula is true with respect to a sequence if and only if its encoding together with a sequence encoding and a temporal GDL extension yields a unique answer set which contains a special formula name atom. Instead of a sequence, the generalised encoding definition requires a sequence mapping for that formula, and incorporates the alternative semantics over sequence mappings instead of the original semantics.

Definition 4.25 (Encoding for Epistemic Sequence Invariants). *Let G be a valid GDL specification, φ be an epistemic sequence invariant, and $\eta(\varphi)$ be a 0-ary atom which represents a unique name for φ. An* encoding of *φ, denoted $Enc(\varphi)$, is a finite set of clauses whose heads do not occur elsewhere and such that, for each sequence mapping $\mathcal{M}_{\delta,\varphi}$ and for each φ-signature Sig for $\mathcal{M}_{\delta,\varphi}$, the program $P = \mathcal{M}^{\mathtt{true}}_{\delta,\varphi} \cup G_{Sig} \cup \mathcal{M}^{\mathtt{does}}_{\delta,\varphi} \cup Enc(\varphi)$ fulfils the following:*

- *P has exactly one answer set;*
- *$\mathcal{M}_{\delta,\varphi} \Vdash \varphi$ iff $P \vdash \eta(\varphi)$.* ■

The example encoding given in Table 3.1 on page 38 can now be extended to epistemic sequence invariants such that it satisfies the requirements of Definition 4.25, which is shown with the following theorem.

Theorem 4.26 (Correctness of the Generalised Sample Encoding). *Let G be a valid GDL specification, $\varphi \in \mathcal{ESIN}_G$, and let $\eta(\varphi, i, v)$ be a 0-ary atom, denoting a unique name for φ with respect to every time point i and every view v. We generalise the example encoding given in Table 3.1 on page 38 as follows.*

- *We add a view argument v to cases 1–8. E.g., case 1 is adapted as follows (the other cases are adapted similarly):*

  1. $Enc(p(\vec{t}), i, v) = \{\eta(p(\vec{t}), i, v) \texttt{ :- } p(\vec{t}, i, v)\,\texttt{.}\}$; *and*

- *We add the following case 9, where π is the position of subformula $K_r\psi$ in φ:*

$$9.\quad Enc(K_r\,\psi, i, v) \;=\; \begin{array}{l}\{\eta(K_r\,\psi, i, v) \texttt{ :- } \eta(\psi, i, \mathcal{V}_\varphi(\pi K_r))\,\texttt{.}\}\\ \cup\, Enc(\psi, i, \mathcal{V}_\varphi(\pi K_r)).\end{array}$$

Then $Enc(\varphi) := Enc(\varphi, 0, \mathcal{V}_\varphi(\epsilon))$ with the unique name atom $\eta(\varphi) := \eta(\varphi, 0, \mathcal{V}_\varphi(\epsilon))$ for φ is an encoding of φ.

Proof: Let $\mathcal{M}_{\delta,\varphi}$ be an arbitrary sequence mapping and Sig be an arbitrary φ-signature for $\mathcal{M}_{\delta,\varphi}$. Program $P_\varphi = \mathcal{M}^{\mathtt{true}}_{\delta,\varphi} \cup G_{Sig} \cup \mathcal{M}^{\mathtt{does}}_{\delta,\varphi} \cup Enc(\varphi)$ clearly admits a unique answer set. The remainder is by induction on the structure of φ.

Base Case: Consider an arbitrary knowledge-free formula φ, let $v = \mathcal{V}_\varphi(\epsilon)$, and let $\mathcal{M}_{\delta,\varphi}(v) = (\delta, \sigma)$. Since $\mathcal{M}_{\delta,\varphi}$ is only defined on v, $\mathcal{M}_{\delta,\varphi}$ is the *only* sequence mapping such that $\mathcal{M}_{\delta,\varphi}(v) = (\delta, \sigma)$. This allows to apply the Semantics Equivalence Theorem 4.19 to obtain equivalence to $\sigma \models_\delta \varphi$. By the Encoding-Theorem for knowledge-free formulas (cf. Theorem 3.12 on page 37) we obtain equivalence to $P_\varphi \vdash \eta(\varphi, 0, v)$.

Induction Step: We consider the cases $\varphi = K_r\psi$ and $\varphi = \bigcirc\psi$, all remaining cases are argued similar to case $\varphi = K_r\psi$. Without loss of generality, let $\mathcal{V}_\varphi(K_r\pi) = \mathcal{V}_\psi(\pi)$ for all $\pi \in Pos_\psi$. Furthermore, let the sequence mapping $\mathcal{M}_{\delta,\psi}$ be such that $\mathcal{M}_{\delta,\psi} = \mathcal{M}_{\delta,\varphi}|_{\mathcal{V}s_\psi}$, and let program P_ψ be constructed similar to P_φ based on $\mathcal{M}_{\delta,\psi}$ instead of $\mathcal{M}_{\delta,\varphi}$.

- $\varphi = K_r\psi$: $\mathcal{M}_{\delta,\varphi} \Vdash \varphi$ iff (by the semantics over sequence mappings, cf. Definition 4.16) $(\mathcal{M}_{\delta,\varphi}, |\delta|, \mathcal{V}_\varphi(\epsilon)) \Vdash \varphi$ iff (by Definition 4.16) $(\mathcal{M}_{\delta,\varphi}, |\delta|, \mathcal{V}_\varphi(K_r)) \Vdash \psi$ iff (by the Sequence-Mapping View Reduction Proposition 4.17) $(\mathcal{M}_{\delta,\psi}, |\delta|, \mathcal{V}_\psi(\epsilon)) \Vdash \psi$ iff (by Definition 4.16) $\mathcal{M}_{\delta,\psi} \Vdash \psi$ iff (by the Induction Hypothesis) $P_\psi \vdash \eta(\psi, 0, \mathcal{V}_\psi(\epsilon))$ iff (by the Splitting Theorem 2.10 and the sample encoding definition case 9) $P_\varphi \vdash \eta(\varphi, 0, \mathcal{V}_\varphi(\epsilon))$.

- $\varphi = \bigcirc\psi$: Let $\mathcal{M}_{\delta,\varphi}(\mathcal{V}_\varphi(\epsilon)) = (\delta, (S_{|\delta|}, \ldots, S_{|\delta|+m}))$ and note that its suffix $(S_{|\delta|}, \ldots, S_{|\delta|+m})$ is $\widehat{n}$-max for some $\widehat{n} \geq \deg(\bigcirc\psi)$. We consider two cases:

  - $S_{|\delta|}$ is terminal: then $\mathcal{M}_{\delta,\varphi} \Vdash \varphi$ follows by Definition 4.16, and $P_\varphi \vdash$ **terminal**$(0, \mathcal{V}_\varphi(\epsilon))$ follows by Theorem 4.24 (Correctness of the Epistemic Temporal GDL Extension) and yields $P_\varphi \vdash \eta(\varphi, 0, \mathcal{V}_\varphi(\epsilon))$.
  - $S_{|\delta|}$ is non-terminal: then $\delta' = (\delta, (S_{|\delta|}, S_{|\delta|+1}))$ exists. By the Sequence-Mapping Correspondence Proposition 4.18, function $\mathcal{M}_{\delta',\psi} : \mathcal{V}s_\psi \to \Delta_G$ such that $\mathcal{M}_{\delta',\psi}(v) = \mathcal{M}_{\delta,\varphi}(v)$ for all $v \in \mathcal{V}s_\varphi$ is a sequence mapping for ψ with respect to δ', and we have

    $$\mathcal{M}_{\delta,\varphi} \Vdash \varphi \text{ iff } \mathcal{M}_{\delta',\psi} \Vdash \psi.$$

    Note that, although $\mathcal{M}_{\delta,\varphi}$ and $\mathcal{M}_{\delta',\psi}$ coincide, ψ is interpreted at time step $|\delta'|$, whereas φ is interpreted at time step $|\delta| = |\delta'| - 1$. In analogy to Theorem 3.12 (Correctness of the Sample Encoding), let $\cdot^{i \to i+1}$ be a renaming that replaces each time argument i by $i+1$ in timed and view-argumented GDL atoms and replaces each occurrence of $\eta(\rho, v, i)$ by $\eta(\rho, v, i+1)$ for each formula ρ and each view name v. The induction hypothesis then implies that

    $$\mathcal{M}_{\delta',\psi} \Vdash \psi \text{ iff } P_\psi^{i \to i+1} \vdash \eta(\psi, 1, \mathcal{V}_\psi(\epsilon))$$

    This, in turn, is equivalent to $P_\varphi \vdash \eta(\varphi, 0, \mathcal{V}_\varphi(\epsilon))$ by arguments similar to those following the introduction of $\cdot^{i \to i+1}$ in Theorem 3.12. □

An example for the encoding given in Theorem 4.26 is deferred to Section 4.6 (cf. page 96). This section provides a comprehensive demonstration of the generalised proof method which will be developed in the following. For the sake of generality, however, all theoretic considerations again abstract from the specific encoding and consider any $Enc(\varphi)$ that satisfies the requirements of the Encoding Definition 4.25.

4.5 Verification of Positive-Knowledge Formulas

In Section 3.4 we have shown how the encoding of a game property (i.e., a temporal formula), together with the temporal extension of a given set of game clauses, can be fed into an answer set solver in order to establish whether the clauses entail the property. We will now extend the method in order to prove positive-knowledge formulas φ via induction. As the original method, it is based on the construction of two answer set programs. One program is needed to establish the base case of the induction proof. It shows that, for the unique development $\delta = (S_{init})$ of length 0, we have that $S_{init} \vDash_\delta \varphi$; this program will be specified in Section 4.5.1. The other program is needed to establish the induction step. It shows that, for arbitrary states S which are reachable via some development δ and which are such that $S \vDash_\delta \varphi$, each of their direct successor states S' which is reachable via the development $\delta' = (\delta, (S, S'))$ is such that $S' \vDash_{\delta'} \varphi$; this program will be specified in Section 4.5.2. Base case and induction step put together then imply that, for all developments δ, we have that $last(\delta) \vDash_\delta \varphi$.

4.5.1 Base Case

Action-Set Generator

In Section 3.4.1, we have specified an action generator, i.e. a program which encodes that each player has to perform exactly one legal move in each non-terminal state of the game. We will now extend this program to account for the epistemic extension of the GDL clauses by ensuring the mentioned property with respect to each of the considered views of a formula. We incorporate additional restrictions to connect epistemically and temporally extended GDL clauses with different views according to the accessibility relation. For an epistemic sequence invariant φ, the constructed program will be based on a φ-signature Sig. It is denoted with P^{legal}_{Sig} and consists of the following clauses.

- Recall the finite set $ADom(r)$ of all possible actions for player r from Section 3.4 at page 39. For each view v, the requirement that each player has to perform a legal move in each non-terminal state of development $\mathcal{M}_{\delta,\varphi}(v)$ up to time step $Sig(v) - 1$ is encoded by v-extended clauses of the action generator $P^{legal}_{\leq Sig(v)-1}$ (cf. the clauses (3.8) at page 40). That is, for each $v \in \mathcal{V}s_\varphi$, $0 \leq i < Sig(v)$, and $r \in R$, we have the following:

$$\begin{array}{ll} (c_1) & \texttt{terminated}(i, v) \texttt{ :- } \textbf{terminal}(i, v). \\ (c_2) & \texttt{terminated}(i, v) \texttt{ :- } \texttt{terminated}(i-1, v). \\ (c_3) & 1\{\textbf{does}(r, a, i, v) : a \in ADom(r)\}1 \texttt{ :- } \textbf{not}\, \texttt{terminated}(i, v). \\ (c_4) & \texttt{ :- } \textbf{does}(r, \texttt{A}, i, v), \textbf{not legal}(r, \texttt{A}, i, v). \end{array} \qquad (4.7)$$

- The requirement that sequences of different views are related according to the requirements of the sequence mapping from Definition 4.14 is encoded as follows. Let $K_r\psi$ be a subformula of φ at position π, where $\mathcal{V}_\varphi(\pi K_r) = v$ and $\mathcal{V}_\varphi(\pi) = v'$, and for the level $\mathcal{L}_\varphi(\pi)$ of $K_r\psi$, let $0 \leq i < \mathcal{L}_\varphi(\pi)$.
  - The sequence for v is at least as long as the $\mathcal{L}_\varphi(\pi)$-prefix of the sequence for v':

$$\texttt{:- } \textbf{not}\, \texttt{terminated}(i, v'), \texttt{terminated}(i, v). \qquad (4.8)$$

– Player r performs the same actions wrt. views v and v':

$$\begin{array}{l} \texttt{:-}\ \textbf{not}\,\texttt{terminated}\,(i, v'\,), \textbf{does}\,(r, \texttt{A}\,, i, v\,), \textbf{not does}\,(r, \texttt{A}\,, i, v'). \\ \texttt{:-}\ \textbf{not}\,\texttt{terminated}\,(i, v'\,), \textbf{does}\,(r, \texttt{A}\,, i, v'), \textbf{not does}\,(r, \texttt{A}\,, i, v\,). \end{array} \tag{4.9}$$

– Player r has the same percepts wrt. views v and v':

$$\begin{array}{l} \texttt{:-}\ \textbf{not}\,\texttt{terminated}\,(i, v'\,), \textbf{sees}\,(r, \texttt{S}\,, i+1, v\,), \textbf{not sees}\,(r, \texttt{S}\,, i+1, v'). \\ \texttt{:-}\ \textbf{not}\,\texttt{terminated}\,(i, v'\,), \textbf{sees}\,(r, \texttt{S}\,, i+1, v'), \textbf{not sees}\,(r, \texttt{S}\,, i+1, v\,). \end{array} \tag{4.10}$$

Subsequently, P_{Sig}^{legal} will also be called an *action-set generator for Sig*.

Base Case Program

For a GDL description G, a formula φ, and the *minimal* φ-signature Sig (cf. the Formula Signature Definition 4.20), the answer set program for the *base case* is defined as follows.

$$\begin{array}{rcl} P_{\varphi}^{bc}(G) & = & (\bigcup_{v \in \mathcal{V}s_{\varphi}} S_{init}^{\texttt{true}}(0)[+v]) \cup G_{Sig} \cup P_{Sig}^{legal} \cup \\ & & Enc(\varphi) \cup \{\ \texttt{:-}\ \ \eta(\varphi).\ \} \end{array}$$

Put in words, $P_{\varphi}^{bc}(G)$ consists of an encoding of the initial state for each of the views of φ, $\bigcup_{v \in \mathcal{V}s_{\varphi}} S_{init}^{\texttt{true}}(0)[+v]$; an epistemic temporal GDL description corresponding to the minimal φ-signature Sig, G_{Sig}; the necessary requirements concerning legal moves for each of the respective sequences as well as their relation with respect to the accessibility relation, P_{Sig}^{legal}; and an encoding for φ together with the statement that φ should not be entailed in any answer set of $P_{\varphi}^{bc}(G)$, $Enc(\varphi) \cup \{\ \texttt{:-}\ \eta(\varphi)\texttt{.}\}$. In case $P_{\varphi}^{bc}(G)$ has no answer set, the last clause implies that there is no sequence mapping $\mathcal{M}_{\delta,\varphi}$ with the minimal φ-signature Sig that makes φ false—which means that each sequence mapping with signature Sig satisfies φ. This in turn implies that $S_{init} \models_{(S_{init})} \varphi$ (by the Semantics Equivalence Theorem 4.19).

4.5.2 Induction Step

State-Set Generator

For the induction step answer set program, the base case program $(\bigcup_{v \in \mathcal{V}s_{\varphi}} S_{init}^{\texttt{true}}(0)[+v])$ needs to be substituted by a program which, for each sequence mapping $\mathcal{M}_{\delta,\bigcirc\varphi}$, generates an encoding $\mathcal{M}_{\delta,\bigcirc\varphi}^{\texttt{true}}$. It corresponds to all states $S_{|\delta|}$ at time step $|\delta|$ of the respective developments $\mathcal{M}_{\delta,\bigcirc\varphi}(v)$ for all views $v \in \mathcal{V}s_{\bigcirc\varphi}$. These states are related according to the accessibility relation and hence incorporate their foregoing developments. Let us first formally characterise state collections with these properties.

Definition 4.27 (Development Mapping). *Let G be a valid GDL specification, $\varphi \in \mathcal{ESINN}_G$, and $\delta \in \Delta_G$. A function $\mathcal{D}_{\delta,\varphi} : \mathcal{V}s_{\varphi} \to \Delta_G$ is called* development mapping *for φ* wrt. *δ if*

- *$\mathcal{D}_{\delta,\varphi}(\mathcal{V}_{\varphi}(\epsilon)) = \delta$; and*

- *for each subformula $K_r\psi$ of φ with position π:*

$$\mathcal{D}_{\delta,\varphi}(\mathcal{V}_\varphi(\pi)) \sim_r \mathcal{D}_{\delta,\varphi}(\mathcal{V}_\varphi(\pi K_r)).$$

■

An ideal generator program for the induction step now exactly admits all answer sets which correspond to one particular development mapping by encoding the last states of all developments from that mapping. However, recall from the induction step program for knowledge-free formulas that the reachable states cannot be computed efficiently in general and hence have to be approximated by a set of states which, in addition to all reachable states, contains some non-reachable states as well (cf. Section 3.4.2). This implies that also foregoing developments for reachable states cannot be computed (otherwise the states themselves would also be known), and for non-reachable states they do not even exist. Hence, an ideal generator program cannot generally be constructed for practical verification, which again calls for a suitable abstraction. Similar to the knowledge-free case, this is possible by *overestimation*: the generator is allowed to provide additional answer sets which do not correspond to development mappings. E.g., for an arbitrary state generator P^{gen}, the simplest state-set generator provides an answer set for *every* combination of the states which are generated by P^{gen}:

$$\bigcup_{v \in \mathcal{V}s_\varphi} P^{gen}$$

This instance of a state-set generator completely ignores the restrictions put by development-mappings. In the following we define a state-set generator such that, for each development mapping, it yields an answer set representing the last states of all developments from that mapping, and may yield additional answer sets which do not correspond to development mappings.

Definition 4.28 (State-Set Generator)**.** *Let G be a valid GDL specification and φ be an epistemic sequence invariant. A* state-set generator *is an answer set program P^{setgen}_φ such that*

- *The only atoms in P^{setgen}_φ are of the form $\mathbf{true}(f,0,v)$, where $f \in \Sigma$ and $v \in \mathcal{V}s_\varphi$, or auxiliary atoms that do not occur elsewhere; and*
- *for every development mapping $\mathcal{D}_{\delta,\varphi}$ for φ wrt. development δ, P^{setgen}_φ has an answer set $\mathcal{A}$ such that for all $f \in \Sigma$ and for all $v \in \mathcal{V}s_\varphi$: $\mathbf{true}(f,0,v) \in \mathcal{A}$ iff $f \in last(\mathcal{D}_{\delta,\varphi}(v))$.* ■

Below Definition 4.22 (Epistemic Temporal GDL Extension) we pointed out that the encoding of a sequence mapping which is with respect to a game development δ omits all development prefixes of length $|\delta|$. This is due to the above-mentioned overestimation of the reachable states. The state-set generator fills this gap by providing state-set encodings which hypothetically relate to some unknown development δ, and may overestimate this relation by providing additional state-set encodings.

In the knowledge-free case, we pointed out that the overestimation of reachable states in the induction step causes unintended counter examples (cf. Section 3.4.2). This speciality carries over to the generalised setting. Hence it is generally helpful to add positive-knowledge formulas that have previously been proved to a state-set generator. For the sake of clarity, we refrain from the formal details and give a discussion on this matter in Section 4.8.2 instead.

Induction Step Program

For a GDL description G, a formula φ, and the minimal $\bigcirc\varphi$-signature Sig, the answer set program for the *induction step* is defined as

$$\begin{array}{rcl} P^{is}_{\varphi}(G) & = & P^{setgen}_{\bigcirc\varphi} \cup G_{Sig} \cup P^{legal}_{Sig} \cup \\ & & Enc(\varphi) \cup \{ \texttt{:-}\ \textbf{not}\ \eta(\varphi).\ \} \cup Enc(\bigcirc\varphi) \cup \{ \texttt{:-}\ \ \eta(\bigcirc\varphi).\ \}, \end{array}$$

where the view names of φ and $\bigcirc\varphi$ coincide, i.e., where $\mathcal{V}s_{\varphi} = \mathcal{V}s_{\bigcirc\varphi}$ and the view namings $\mathcal{V}_{\varphi}$ and $\mathcal{V}_{\bigcirc\varphi}$ are such that, for all $\pi \in Pos_{\varphi}$, we have $\mathcal{V}_{\varphi}(\pi) = \mathcal{V}_{\bigcirc\varphi}(\bigcirc\pi)$.

Put in words, $P^{is}_{\varphi}(G)$ deviates from $P^{bc}_{\varphi}(G)$ in the following way. First, an arbitrary state-set generator $P^{setgen}_{\bigcirc\varphi}$ is used instead of the initial-state encoding. Second, the clauses are constructed over a minimal $\bigcirc\varphi$-signature instead of a φ-signature, hence the time horizon has increased by 1 (since both signatures are minimal). Third, the clauses $Enc(\varphi) \cup \{ \texttt{:-}\ \textbf{not}\,\eta(\varphi)\texttt{.}\}$ ensure that each answer set represents a sequence mapping $\mathcal{M}_{\delta,\bigcirc\varphi}$ over Sig which satisfies φ (note that $\mathcal{M}_{\delta,\bigcirc\varphi}$ is also a sequence mapping for φ with respect to δ, as the view namings $\mathcal{V}_{\varphi}$ and $\mathcal{V}_{\bigcirc\varphi}$ coincide), and the clauses $Enc(\bigcirc\varphi) \cup \{ \texttt{:-}\ \eta(\bigcirc\varphi)\texttt{.}\}$ ensure that $\mathcal{M}_{\delta,\bigcirc\varphi}$ additionally violates $\bigcirc\varphi$. In case $P^{is}_{\varphi}(G)$ is inconsistent, there is no sequence mapping $\mathcal{M}_{\delta,\bigcirc\varphi}$ which satisfies these requirements—which implies that $\bigcirc\varphi$ is satisfied with respect to all sequence mappings with signature Sig that also satisfy φ. This in turn implies that $S_{|\delta|+1} \vDash_{(\delta,(S_{|\delta|},S_{|\delta|+1}))} \varphi$ is satisfied in all direct successors $S_{|\delta|+1}$ of reachable states $S_{|\delta|}$ that are such that $S_{|\delta|} \vDash_{\delta} \varphi$ (by the Sequence-Mapping Correspondence Proposition 4.18, the Semantics Equivalence Theorem 4.19, and the Sequence-Mapping Length Extension Proposition 4.21).

Remark on the Linear-Time Encoding of Positive-Knowledge Formulas

In Section 3.4.2 we have mentioned that, in a linear-time setting and for knowledge-free formulas φ such that $\deg(\varphi) \geq 1$, the induction hypothesis $S_0 \vDash \varphi$ is applied in a weaker linear-time variant, namely with respect to *single* sequences that start in S_0. I.e., a counter example for φ in the induction step is a $\deg(\bigcirc\varphi)$-max sequence $\sigma = (S_0, \dots, S_m)$ such that $\sigma \vDash \varphi \wedge \neg \bigcirc \varphi$, but not *all* $\deg(\varphi)$-max sequences $\sigma' = (S_0, \dots, S'_{m'})$ are necessarily such that $\sigma' \vDash \varphi$, although this is implied by the actual induction hypothesis $S_0 \vDash \varphi$.

To the generalised setting of positive-knowledge formulas, this explanation translates as follows. The induction hypothesis $last(\delta) \vDash_{\delta} \varphi$ states that φ is true with respect to *all* $\deg(\varphi)$-max sequences which start in $last(\delta)$. For each subformula $K_r\psi$ of φ, it states in turn that ψ is true with respect to *all* “appropriately related” sequences. To match the linear time structure, we again implicitly weaken the induction hypothesis: φ is assumed to be true with respect to *one* $\deg(\varphi)$-max sequence which starts in $last(\delta)$, and for each subformula $K_r\psi$ of φ, ψ is assumed to be true with respect to *one* “appropriately related” sequence. This is always possible, as each component $\sim_r$ of the accessibility relation is reflexive. A counter example for φ in the induction step then amounts to a collection of sequences, represented by the notion of a sequence mapping, which satisfies the weaker induction hypothesis for φ in the current time step as described above, and violates φ in the next time step.

In the following section, we will demonstrate that the weaker linear-time encoding of the induction hypothesis still suffices for proving an example formula in the game of Krieg-Tictactoe. This can be achieved since both the induction hypothesis and the encoding for the violation in the next time step use the exact same view namings of the respective formula, hence providing a sufficient compensation for the necessary relaxations in the induction step program. In Section 5.4.2, we provide experiments on additional games and properties which further demonstrate the effectiveness of our method.

4.6 An Example Proof

Before we establish a soundness result for the generalised verification method, we give a comprehensive example which demonstrates that the validity can reliably be established for formula

$$\varphi = \neg \mathit{legal}(x, \mathit{mark}(\mathit{1}, \mathit{1})) \vee K_x \mathit{legal}(x, \mathit{mark}(\mathit{1}, \mathit{1})),$$

from Example 4.10 (cf. page 75) in the game of Krieg-Tictactoe. Recall the clauses of Krieg-Tictactoe which have been given with the GDL specification G in Figure 4.2, and the view naming $\mathcal{V}_\varphi$ of φ from Example 4.13 (cf. page 76) such that

- $\mathcal{V}_\varphi(\epsilon) = \mathcal{V}_\varphi(\vee_1) = \mathcal{V}_\varphi(\vee_2) = \mathcal{V}_\varphi(\vee_1 \neg) = v_0$; and
- $\mathcal{V}_\varphi(\vee_2 K_r) = v_1$,

and note that the level $\mathcal{L}_\varphi(\pi)$ of all positions $\pi \in \mathit{Pos}_\varphi$ is 0. In the following, we show that both the base case answer set program $P^{bc}_\varphi(G)$ and the induction step answer set program $P^{is}_\varphi(G)$ for φ are inconsistent, which implies that φ is valid with respect to all reachable states.

4.6.1 Base Case

To show that $S_{\mathit{init}} \models_{S_{\mathit{init}}} \varphi$, we have to show $\mathcal{M}_{\delta,\varphi} \Vdash \varphi$ for all sequence mappings $\mathcal{M}_{\delta,\varphi}$ for φ with respect to $\delta = (S_{\mathit{init}})$ which are over the minimal φ-signature $\mathit{Sig} : \{v_0, v_1\} \to \mathbb{N}$ of φ. Sig is such that $\mathit{Sig}(v_0) = \mathit{Sig}(v_1) = 0$, hence the sequences $\mathcal{M}_{\delta,\varphi}(v_0)$ and $\mathcal{M}_{\delta,\varphi}(v_1)$ are 0-max and both reduce to the initial state, which implies that there is only one such sequence mapping $\mathcal{M}_{\delta,\varphi}$. According to the definition of a sequence mapping, we have $S_{\mathit{init}} \sim_x S_{\mathit{init}}$.

Sequence Mappings Correspond To Answer Sets

For a program which reliably encodes $\mathcal{M}_{\delta,\varphi}$, condition $S_{\mathit{init}} \sim_x S_{\mathit{init}}$ does not cause any restrictions. Consequently, the part $P^{\mathit{legal}}_{\mathit{Sig}}$ of the base case program $P^{bc}_\varphi(G)$ which is used to restrict answer sets to those that model sequence mappings reduces to the empty set (as $\mathit{Sig}(v_0) = 0$, $\mathit{Sig}(v_1) = 0$, and $\mathcal{L}_\varphi(\pi) = 0$ for all positions $\pi \in \mathit{Pos}_\varphi$). Recall the temporal extension $G_{\leq 0}$ of the game specification G of degree 0 (cf.

Example 4.23). The program which is obtained from $P^{bc}_{\varphi}(G)$ by omitting the encoding for φ,

$$\bigcup_{v \in \{v_0, v_1\}} \big(S^{\texttt{true}}_{init}(0) \cup G_{\leq 0}\big)[+v],$$

admits a unique answer set which contains a faithful encoding of $\mathcal{M}_{\delta,\varphi}$, since the contained instances of **true**(F,0,V) coincide with $\mathcal{M}^{\texttt{true}}_{\delta,\varphi}$.

Inconsistency

The encoding $Enc(\varphi)$ for φ can be obtained by applying the recursive definition given in Theorem 4.26. It yields the following clauses, where atom phi is the unique name $\eta(\varphi, 0, v_0)$ for formula φ.

```
                     phi :- neg_a0.
                     phi :- knows_a1.
                                                        (4.11)
neg_a0 :- not a0.    a0 :- legal(x,mark(1,1),0,v0).
knows_a1 :- a1.      a1 :- legal(x,mark(1,1),0,v1).
```

Finally, the base case program is the composition of the two mentioned programs together with the constraint :- phi. which expresses that φ should not be true in the initial state:

$$\bigcup_{v \in \{v_0, v_1\}} \big(S^{\texttt{true}}_{init}(0) \cup G_{\leq 0}\big)[+v] \cup \{(4.11)\} \cup \{\ \texttt{:- phi.}\}.$$

Now assume that $P^{bc}_{\varphi}(G)$ admits an answer set $\mathcal{A}$. Then phi $\notin \mathcal{A}$ due to the constraint :- phi. By the encoding (4.11) of φ we have **legal**(x,mark(1,1),0,v0) $\in \mathcal{A}$ and **legal**(x,mark(1,1),0,v1) $\notin \mathcal{A}$. This is a contradiction, since the clauses $\big(S^{\texttt{true}}_{init}(0) \cup G_{\leq 0}\big)[+v_0]$ and $\big(S^{\texttt{true}}_{init}(0) \cup G_{\leq 0}\big)[+v_1]$ coincide except for the name of the view argument. Hence, $P^{bc}_{\varphi}(G)$ is inconsistent, which implies that $S_{init} \models_{(S_{init})} \varphi$ (cf. Section 4.5.1).

4.6.2 Induction Step

The minimal $\bigcirc\varphi$-signature $Sig : \{v_0, v_1\} \to \mathbb{N}$ of $\bigcirc\varphi$ is such that $Sig(v_0) = Sig(v_1) = 1$. Consider an arbitrary sequence mapping $\mathcal{M}_{\delta,\bigcirc\varphi}$ for $\bigcirc\varphi$ with respect to an arbitrary development δ which is over the minimal $\bigcirc\varphi$-signature Sig. Then, for some development δ' such that $|\delta'| = |\delta|$ and some 1-max sequences σ and σ', we have $\mathcal{M}_{\delta,\varphi}(v_0) = (\delta, \sigma)$ and $\mathcal{M}_{\delta,\varphi}(v_1) = (\delta', \sigma')$. In case δ is terminal and hence $\sigma = (last(\delta))$, these sequences are related such that $\delta \sim_x \delta'$. Otherwise, there is a direct successor $S_{|\delta|+1}$ of $S_{|\delta|} = last(\delta)$ such that $\sigma = (S_{|\delta|}, S_{|\delta|+1})$, and $\mathcal{M}_{\delta,\varphi}(v_0) \sim_x \mathcal{M}_{\delta,\varphi}(v_1)$.

Sequence Mappings Correspond To Answer Sets

For the sake of simplicity, we assume a state-set generator $P^{setgen}_{\bigcirc\varphi}$ which does only provide answer sets based on states which contain exactly one instance of $control(r)$

and which contain exactly one instance of $cell(x, y, c)$ for each pair (x, y) such that $x, y \in \{1, 2, 3\}$. As we will see in the experiment Section 5.4.1, this information can easily be obtained automatically by preliminary proofs of the knowledge-free formulas $(\exists_{1..1}\{x, o\} \colon true)\ (control(C))$ and $(\forall X, Y \colon \{1, 2, 3\})\,(\exists_{1..1}P \colon D_P)\ true(cell(X, Y, P))$. Furthermore, the encoded states of each answer set are not restricted to be related according to the accessibility relation.

We first argue that the program obtained from $P^{is}_{\varphi}(G)$ by omitting the encoding for φ,

$$P = P^{setgen}_{\bigcirc\varphi} \cup \bigcup_{v \in \{v_0, v_1\}} G_{\leq 1}[+v] \cup P^{legal}_{Sig},$$

provides an answer set $\mathcal{A}$ which contains a faithful encoding of the (arbitrarily chosen) sequence mapping $\mathcal{M}_{\delta,\bigcirc\varphi}$ in that the contained instances of **true**`(F,0,V)` and **does**`(R,A,0,V)` coincide with $\mathcal{M}^{\mathtt{true}}_{\delta,\bigcirc\varphi}$ and $\mathcal{M}^{\mathtt{does}}_{\delta,\bigcirc\varphi}$.

For $\mathcal{M}_{\delta,\bigcirc\varphi}$, there is a development mapping $\mathcal{D}_{\delta,\varphi} : \{v_0, v_1\} \to \Delta_G$ such that $\mathcal{D}_{\delta,\varphi}(v_0) = \delta$ and $\mathcal{D}_{\delta,\varphi}(v_0) \sim_x \mathcal{D}_{\delta,\varphi}(v_1)$. Hence, $\mathcal{M}^{\mathtt{true}}_{\delta,\bigcirc\varphi}$ clearly satisfies the clauses of the state-set generator $P^{setgen}_{\bigcirc\varphi}$ (cf. Definition 4.28). Since $Sig(v_0) = Sig(v_1) = 1$ and the level $\mathcal{L}_{\varphi}(\bigcirc\pi) = 1$ for all $\pi \in Pos_{\varphi}$, the action-set generator P^{legal}_{Sig} (cf. clauses (4.7) to (4.10) in Section 4.5.1) consists of the following clauses:

- Each player $r \in \{\,\mathtt{x},\ \mathtt{o}\}$ performs exactly one legal move in each non-terminal state of time step 0 with respect to each of the view names $v \in \{v_0, v_1\}$.

$$\begin{array}{l} \mathtt{terminated}\,(0, v)\ \mathtt{:-}\ \ \mathbf{terminal}\,(0, v). \\ \mathtt{terminated}\,(0, v)\ \mathtt{:-}\ \ \mathtt{terminated}\,(-1, v). \\ 1\{\mathbf{does}\,(r, a, 0, v) : a \in ADom(r)\}1\ \mathtt{:-}\ \ \mathbf{not}\ \mathtt{terminated}\,(0, v). \\ \ \mathtt{:-}\ \ \mathbf{does}\,(r, \mathtt{A}, 0, v), \mathbf{not\ legal}\,(r, \mathtt{A}, 0, v). \end{array} \qquad (4.12)$$

  Sequence mapping $\mathcal{M}_{\delta,\varphi}$ contains the respective developments $\mathcal{M}_{\delta,\varphi}(v_0) = (\delta, \sigma)$ and $\mathcal{M}_{\delta,\varphi}(v_1) = (\delta', \sigma')$. Now either $last(\delta)$ is terminal, or it incorporates a legal move for each of the players. The same argumentation applies to δ'. Hence, the encoding $\mathcal{M}^{\mathtt{true}}_{\delta,\bigcirc\varphi} \cup \mathcal{M}^{\mathtt{does}}_{\delta,\bigcirc\varphi}$ of sequence mapping $\mathcal{M}_{\delta,\varphi}$ (which encodes the previously mentioned states at time step 0) together with $G_{\leq 1}[+v]$ is consistent with the clauses (4.12).

- If the state for view name v_0 at time step 0 is non-terminal, then also the state for view name v_1 at time step 0 is non-terminal.

```
:- not terminated(0,v0), terminated(0,v1).
```
(4.13)

  In case $last(\delta)$ is terminal, the constraint is satisfied. Otherwise, there is a direct successor of $last(\delta)$ in $\mathcal{M}_{\delta,\varphi}(v_0)$, and the sequences $\mathcal{M}_{\delta,\varphi}(v_0)$ and $\mathcal{M}_{\delta,\varphi}(v_1)$ are related such that $\mathcal{M}_{\delta,\varphi}(v_0) \sim_x \mathcal{M}_{\delta,\varphi}(v_1)$. This implies that also $last(\delta')$ is non-terminal, which again satisfies the constraint.

- If the state for view name v_0 at time step 0 is non-terminal, then player x performs the same actions with respect to both view names v_0 and v_1.

```
:- not terminated(0,v0), does(x,A,0,v1), not does(x,A,0,v0).
:- not terminated(0,v0), does(x,A,0,v0), not does(x,A,0,v1).
```
(4.14)

In case $last(\delta)$ is terminal, the constraints are satisfied. Otherwise, the suffix sequences σ and σ' of the respective developments $\mathcal{M}_{\delta,\varphi}(v_0) = (\delta, \sigma)$ and $\mathcal{M}_{\delta,\varphi}(v_1) = (\delta', \sigma')$ are of length 1 and hence each contain a single state transition such that the actions of player x coincide, which again satisfies the constraint.

- If the state for view name v_0 at time step 0 is non-terminal, then player x has the same percepts with respect to both view names v_0 and v_1.

```
:- not terminated(0,v0), sees(x,S,1,v1), not sees(x,S,1,v0).
:- not terminated(0,v0), sees(x,S,1,v0), not sees(x,S,1,v1).
```
(4.15)

Similar to the previous case, a terminal state $last(\delta)$ satisfies the constraints, and a non-terminal state $last(\delta)$ implies $\mathcal{M}_{\delta,\varphi}(v_0) \sim_x \mathcal{M}_{\delta,\varphi}(v_1)$ and hence the same percepts in $last(\sigma)$ and $last(\sigma')$.

Note that, while this argumentation shows that each sequence mapping $\mathcal{M}_{\delta,\varphi}$ of the above-mentioned shape corresponds to an answer set of the program P, the converse is not true: P might yield answer sets which do not correspond to sequence mappings due to the structure of state-set generators (cf. the explanation in Section 4.5.2).

Inconsistency

The induction step program needs the following additional encoding $Enc(\bigcirc\varphi)$ for $\bigcirc\varphi$, where atom `nxt_phi` is the unique name $\eta(\bigcirc\varphi, 0, v_0)$ for formula $\bigcirc\varphi$. It is important that the view namings $\mathcal{V}_\varphi$ and $\mathcal{V}_{\bigcirc\varphi}$ coincide, i.e. that $\mathcal{V}_\varphi(\pi) = \mathcal{V}_{\bigcirc\varphi}(\bigcirc\pi)$ (and hence $\mathcal{V}_{\bigcirc\varphi}(\epsilon) = \mathcal{V}_{\bigcirc\varphi}(\bigcirc)$ for all $\pi \in Pos_\varphi$. Then, each subformula of φ is encoded in $Enc(\varphi)$ with respect to the exact same view name as in $Enc(\bigcirc\varphi)$.

```
nxt_phi :- terminal(0,v0).      phi_1 :- neg_a0_1.
nxt_phi :- phi_1.               phi_1 :- knows_a1_1.

neg_a0_1 :- not a0_1.           a0_1 :- legal(x,mark(1,1),1,v0).
knows_a1_1 :- a1_1.             a1_1 :- legal(x,mark(1,1),1,v1).
```
(4.16)

Finally, the induction step program is given as

$$P \cup \{(4.11)\} \cup \{\texttt{:- } \mathbf{not}\ \texttt{phi.}\} \cup \{(4.16)\} \cup \{\texttt{:- nxt\_phi.}\}.$$

We argued above that P provides an answer set for each sequence mapping $\mathcal{M}_{\delta,\bigcirc\varphi}$ for $\bigcirc\varphi$ with respect to δ which is over the minimal $\bigcirc\varphi$-signature Sig. In the following we show that adding $\{(4.11)\} \cup \{\texttt{:- } \mathbf{not}\ \texttt{phi.}\} \cup \{(4.16)\} \cup \{\texttt{:- nxt\_phi.}\}$ causes inconsistency, implying that φ is satisfied in all direct successors of states reachable via development δ which itself satisfy φ (cf. Section 4.5.2).

Assume that $P^{is}_\varphi(G)$ admits an answer set $\mathcal{A}$. By encoding (4.16), together with constraint `:- nxt_phi.`, we have

$$\mathbf{terminal}\texttt{(0,v0)} \notin \mathcal{A}, \tag{4.17}$$

$$\mathbf{legal}\texttt{(x,mark(1,1),1,v0)} \in \mathcal{A}, \text{ and } \mathbf{legal}\texttt{(x,mark(1,1),1,v1)} \notin \mathcal{A}. \tag{4.18}$$

By (4.15), the instances of **sees** for player x coincide with respect to both views v_0 and v_1 at time step 1. In case we have **sees**`(x,yourmove,1,`v`)` $\in \mathcal{A}$ for all $v \in \{$ `v0`, `v1`$\}$, there are two possibilities by (the temporal epistemic extension of) clauses 28 and 29 in Figure 4.2:

- `validmove(0,`v`)` $\notin \mathcal{A}$ and **true**`(control(x),0,`v`)` $\in \mathcal{A}$: Then, by clause 14, we have **true**`(control(x),1,`v`)` $\in \mathcal{A}$ (and hence **true**`(control(o),1,`v`)` $\notin \mathcal{A}$, since we assume a unique `control`-instance of **true**).
- `validmove(0,`v`)` $\in \mathcal{A}$ and **true**`(control(o),0,`v`)` $\in \mathcal{A}$: Then, by clause 23 in Figure 4.2, we have that **true**`(control(x),1,`v`)` $\in \mathcal{A}$.

The opposite case, **sees**`(x,yourmove,1,`v`)` $\notin \mathcal{A}$ for all $v \in \{$`v0`, `v1`$\}$, can be argued similarly and yields that **true**`(control(o),1,`v`)` $\in \mathcal{A}$. Hence, also the `control`-instances of **true** for player x coincide at time step 1. This together with (4.18) and clause 5 in Figure 4.2 implies

$$\{\textbf{true}\texttt{(control(x),1,v0)}, \textbf{true}\texttt{(control(x),1,v1)}\} \subseteq \mathcal{A}, \tag{4.19}$$

$$\textbf{true}\texttt{(tried(1,1),1,v0)} \notin \mathcal{A}, \text{ and } \textbf{true}\texttt{(tried(1,1),1,v1)} \in \mathcal{A}. \tag{4.20}$$

Since **terminal**`(0,v0)` $\notin \mathcal{A}$ (cf. (4.17)), (4.12) and (4.13) imply **terminal**`(0,v1)` $\notin \mathcal{A}$, hence by (4.12) and (4.14) there is exactly one a such that **does**`(x,`a`, 0,`v`)` $\in \mathcal{A}$ for each $v \in \{$ `v0`, `v1`$\}$. Again by (4.12), this implies **legal**`(x,`a`, 0,`v`)` $\in \mathcal{A}$ for the same a and hence (by clauses 5-8) that the `control`-instances of **true** coincide in both views also at time step 0. Assume **true**`(control(o),0,v1)` $\in \mathcal{A}$, then clause 23 and (4.19) imply `validmove(0,v1)` $\in \mathcal{A}$ in contradiction to the clauses 11-12 and (4.20). Hence

$$\{\textbf{true}\texttt{(control(x),0,v0)}, \textbf{true}\texttt{(control(x),0,v1)}\} \subseteq \mathcal{A} \tag{4.21}$$

which, together with clause 14 and (4.19), yields

$$\texttt{validmove(0,v0)} \notin \mathcal{A} \text{ and } \texttt{validmove(0,v1)} \notin \mathcal{A}$$

By the clauses 11-12 and (4.20), this implies **does**`(x,mark(1,1),0,v0)` $\notin \mathcal{A}$, **true**`(tried(1,1),0,v0)` $\notin \mathcal{A}$, and (since the instances of **does** coincide in both views due to (4.14)) **true**`(tried(1,1),0,v1)` $\in \mathcal{A}$. This together with clause 5 and (4.21) implies

$$\textbf{legal}\texttt{(x,mark(1,1),0,v0)} \in \mathcal{A} \text{ and } \textbf{legal}\texttt{(x,mark(1,1),0,v1)} \notin \mathcal{A}$$

The encoding $\{(4.11)\} \cup \{$`:-` **not** `phi.`$\}$ of the induction hypothesis however implies that **legal**`(x,mark(1,1),0,v0)` $\notin \mathcal{A}$, or that **legal**`(x,mark(1,1),0,v1)` $\in \mathcal{A}$. This yields a contradiction, hence $P^{is}_{\varphi}(G)$ admits no answer set.

The last step of the argumentation needs the encoding (4.11) of the induction hypothesis, and the contradiction can only be established since the exact same view namings are used in the encoding (4.16). Indeed, specifying the latter encoding with a new view name v_2 instead of v_1 results in an answer set for $P^{is}_{\varphi}(G)$ and hence would not allow to prove φ valid with our method, as the modified induction hypothesis would become to weak to be of use.

4.7 Properties of the Generalised Verification Method

In analogy to Theorem 3.14, we will now establish a one-to-one relation between answer set programs encoding a particular sequence mapping and those including an action-set generator, which forms a prerequisite for the soundness and completeness proofs of the generalised verification method.

Theorem 4.29 (Generalised Answer Set Correspondence). *Let G be a valid GDL specification, $\varphi \in \mathcal{ESIN}_G$, $\delta \in \Delta_G$, Sig be a φ-signature, and $\mathcal{A}$ be a subset of the ground atoms over G together with $\{\mathtt{terminated}(i, v) : i \in \mathbb{N}$ and $v \in \mathcal{V}s_\varphi\}$. Moreover, let $\mathcal{D} : \mathcal{V}s_\varphi \to \Delta_G$ be a development mapping for φ wrt. δ. The following two statements are equivalent:*

(1) $\mathcal{A}$ is an answer set for

$$P = \mathcal{D}^{\mathtt{true}} \cup G_{Sig} \cup P^{legal}_{Sig},$$

where $\mathcal{D}^{\mathtt{true}}$ is defined in analogy to the state encoding for a sequence mapping from Definition 4.22.

(2) There is a sequence mapping $\mathcal{M}$ for φ wrt. δ over Sig and such that $\mathcal{M}(v) = (\mathcal{D}(v), \sigma)$ for all $v \in \mathcal{V}s_\varphi$, and $\mathcal{A}$ is the unique answer set for

$$P^{\mathcal{M}} = \mathcal{M}^{\mathtt{true}} \cup G_{Sig} \cup P^{c_1,c_2}_{Sig} \cup \mathcal{M}^{\mathtt{does}},$$

where P^{c_1,c_2}_{Sig} denotes all clauses of the shape (c_1) and (c_2) in the action-set generator P^{legal}_{Sig}, defined as (4.7) on page 92.

Proof: Program partitions for each $v \in \mathcal{V}s_\varphi$ as in the proof of item 1 in Theorem 4.24 (Correctness of the Epistemic Temporal GDL extension) allow to apply Theorem 3.14 to obtain equivalence of the following two statements:

(1') $\mathcal{A}$ is an answer set for the program obtained from P by omitting all clauses from the action-set generator P^{legal}_{Sig} which concern the accessibility relation, i.e. by omitting all clauses of the shape (4.8), (4.9), and (4.10).

(2') There is a function $\widehat{\mathcal{M}} : \mathcal{V}s_\varphi \to \Delta_G$ such that for all $v \in \mathcal{V}s_\varphi$ there is a $Sig(v)$-max sequence σ such that $\widehat{\mathcal{M}}(v) = (\mathcal{D}(v), \sigma)$ (note that $\widehat{\mathcal{M}}$ is not necessarily a sequence mapping, since the respective sequences σ need not be related appropriately), and $\mathcal{A}$ is the unique answer set for

$$P^{\widehat{\mathcal{M}}} = \widehat{\mathcal{M}}^{\mathtt{true}} \cup G_{Sig} \cup P^{c_1,c_2}_{Sig} \cup \widehat{\mathcal{M}}^{\mathtt{does}},$$

where $\widehat{\mathcal{M}}^{\mathtt{true}}$ and $\widehat{\mathcal{M}}^{\mathtt{does}}$ are defined in analogy to the state encoding and action encoding for a sequence mapping from Definition 4.22.

$\underline{(2) \Rightarrow (1)}$: Sequence mapping $\mathcal{M}$ satisfies the requirements of function $\widehat{\mathcal{M}}$ in (2'). Since (2') $\Rightarrow$ (1'), it remains to show that answer set $\mathcal{A}$ for $P^{\mathcal{M}}$ satisfies all clauses of the shape (4.8), (4.9), and (4.10).

To this end, let $K_r\psi$ be an arbitrary subformula of φ at position π with level $l = \mathcal{L}_\varphi(\pi)$ such that $\mathcal{V}_\varphi(\pi K_r) = v$ and $\mathcal{V}_\varphi(\pi) = v'$, and let development $\mathcal{M}(v')$ be such that $\mathcal{M}(v') = (\mathcal{D}(v'), (S'_{|\delta|}, S'_{|\delta|+1}, \ldots, S'_{|\delta|+m'}))$. We show that the mentioned clauses are satisfied for each time step i such that $0 \leq i < l$ as follows.

- Time steps $0 \leq i < \min(l, m')$: Let development $\delta_{K_r\psi}$ be the $(|\delta| + l)$-prefix of $\mathcal{M}(v')$. It has length greater $(|\delta| + i)$ (since $i < \min(l, m')$ and thus $i < l$), hence there is a development δ_ψ and a sequence σ_ψ such that $\mathcal{M}(v) = (\delta_\psi, \sigma_\psi)$ and $\delta_{K_r\psi} \sim_r \delta_\psi$ (cf. Sequence Mapping Definition 4.14). By Theorem 4.24, this implies, for each $0 \leq i < \min(l, m')$, action term $a \in ADom(r)$, and term $t \in \Sigma$:
  - $\mathbf{terminal}(i, v) \notin \mathcal{A}$ and hence $\texttt{terminated}(i, v) \notin \mathcal{A}$, which satisfies (4.8);
  - $\mathbf{does}(r, a, i, v) \in \mathcal{A}$ iff $\mathbf{does}(r, a, i, v') \in \mathcal{A}$, which satisfies (4.9);
  - $\mathbf{sees}(r, t, i + 1, v) \in \mathcal{A}$ iff $\mathbf{sees}(r, t, i + 1, v') \in \mathcal{A}$, which satisfies (4.10).
- Time steps $\min(l, m') \leq i < l$: If one such i exists, then $\min(l, m') = m'$. Since development $\mathcal{M}(v')$ is n-max for some $n \geq |\delta| + l$, it must be too short and terminated in this case. I.e., $S_{|\delta|+m'}$ is terminal, which implies $\mathbf{terminal}(m', v') \in \mathcal{A}$ (again by Theorem 4.24) and thus $\{\texttt{terminated}(i, v') : m' \leq i < l\} \subseteq \mathcal{A}$. This satisfies the clauses (4.8), (4.9), and (4.10) for $m' = \min(l, m') \leq i < l$.

<u>(1) ⇒ (2)</u>: Let $\mathcal{A}$ be an answer set for P. Then $\mathcal{A}$ satisfies all clauses of the shape (4.8), (4.9), and (4.10), and $\mathcal{A}$ is an answer set for the program in item (1'). Since (1') ⇒ (2'), it remains to show that function $\widehat{\mathcal{M}}$ from (2') is a sequence mapping for φ wrt. δ.

To this end, again consider an arbitrary subformula $K_r\psi$ of φ at position π with level $l = \mathcal{L}_\varphi(\pi)$ such that $\mathcal{V}_\varphi(\pi K_r) = v$ and $\mathcal{V}_\varphi(\pi) = v'$, and let development $\widehat{\mathcal{M}}(v')$ be such that $\widehat{\mathcal{M}}(v') = (\mathcal{D}(v'), (S'_{|\delta|}, S'_{|\delta|+1}, \ldots, S'_{|\delta|+m'}))$. The clauses of the shape (4.8), (4.9), and (4.10) in P imply the following for answer set $\mathcal{A}$ for each action term $a \in ADom(r)$ and term $t \in \Sigma$:

(a) If $\texttt{terminated}(i, v') \notin \mathcal{A}$, then $\texttt{terminated}(i, v) \notin \mathcal{A}$, for all $0 \leq i < l$ (by (4.8));

(b) If $\texttt{terminated}(i, v') \notin \mathcal{A}$, then $\mathbf{does}(r, a, i, v) \in \mathcal{A}$ iff $\mathbf{does}(r, a, i, v') \in \mathcal{A}$, for all $0 \leq i < l$ (by (4.9));

(c) If $\texttt{terminated}(i, v') \notin \mathcal{A}$, then $\mathbf{sees}(r, t, i + 1, v) \in \mathcal{A}$ iff $\mathbf{sees}(r, t, i + 1, v') \in \mathcal{A}$, for all $0 \leq i < l$ (by (4.10)).

Since $\widehat{\mathcal{M}}(v') = (\mathcal{D}(v'), (S'_{|\delta|}, S'_{|\delta|+1}, \ldots, S'_{|\delta|+m'}))$, we have $\mathbf{terminal}(i, v') \notin \mathcal{A}$ for all $0 \leq i < m'$ (by Theorem 4.24, Correctness of the Epistemic Temporal GDL Extension) and hence $\texttt{terminated}(i, v') \notin \mathcal{A}$ for all $0 \leq i < m'$. Together with items (a) to (c) (which range over $0 \leq i < l$), this implies

(a') $\texttt{terminated}(i, v) \notin \mathcal{A}$ for all $0 \leq i < \min(l, m')$;

(b') $\mathbf{does}(r, a, i, v) \in \mathcal{A}$ iff $\mathbf{does}(r, a, i, v') \in \mathcal{A}$ for all $0 \leq i < \min(l, m')$;

(c') $\mathbf{sees}(r, t, i, v) \in \mathcal{A}$ iff $\mathbf{sees}(r, t, i, v') \in \mathcal{A}$ for all $1 \leq i \leq \min(l, m')$.

Now let development $\widehat{\mathcal{M}}(v)$ be such that $\widehat{\mathcal{M}}(v) = (\mathcal{D}(v), (S_{|\delta|}, S_{|\delta|+1}, \ldots, S_{|\delta|+m}))$. Item (a') yields $\mathbf{terminal}(i, v) \notin \mathcal{A}$ for all $0 \leq i < \min(l, m')$, hence we have $m \geq \min(l, m')$ (again by Theorem 4.24, and since each player performs exactly one legal move in each non-terminal state due to the clauses (4.7) of the action-set generator). This implies that the prefix

$$\delta_\psi = (\mathcal{D}(v), (S_{|\delta|}, S_{|\delta|+1}, \ldots, S_{|\delta|+\min(l,m')}))$$

of $\widehat{\mathcal{M}}(v)$ exists. Furthermore, the $(|\delta| + l)$-prefix $\delta_{K_r\psi}$ of $\widehat{\mathcal{M}}(v')$ can be written as

$$\delta_{K_r\psi} = (\mathcal{D}(v'), (S'_{|\delta|}, S'_{|\delta|+1}, \ldots, S'_{|\delta|+\min(l,m')}))$$

Items (b') and (c') together with the requirements of $\mathcal{D}$ concerning $\sim_r$ imply that $\delta_{K_r\psi} \sim_r \delta_\psi$ (again by Theorem 4.24), hence $\widehat{\mathcal{M}}$ satisfies item 2 (a) of the Sequence Mapping Definition 4.14. The remaining items are satisfied due to the structure of $\mathcal{D}$ which provides the prefixes of length $|\delta|$ for $\widehat{\mathcal{M}}$. Hence, $\widehat{\mathcal{M}}$ is a sequence mapping for φ wrt. δ. □

4.7.1 Soundness

The following theorem states the soundness of the proof method for epistemic sequence invariants.

Theorem 4.30 (Generalised Soundness)**.** *Let G be a valid GDL specification whose initial state is S_{init}, and let φ be a positive-knowledge formula over G. If $P^{bc}_\varphi(G)$ and $P^{is}_\varphi(G)$ are inconsistent, then for all finite developments $\delta = S_{init} \xrightarrow{A_0} S_1 \ldots \xrightarrow{A_{k-1}} S_k$ we have $S_k \models_\delta \varphi$.*

Proof: The proof is via induction on k.

Base Case: Let $\delta = (S_{init})$. We prove that if $S_{init} \not\models_\delta \varphi$, then $P^{bc}_\varphi(G)$ admits an answer set.

$S_{init} \not\models_\delta \varphi$ implies that there is a $\deg(\varphi)$-max sequence σ such that $\sigma \not\models_\delta \varphi$ and hence (by the Semantics Equivalence Theorem 4.19) that there is a sequence mapping $\mathcal{M}'$ for φ wrt. δ such that $\mathcal{M}'(\varphi) = (\delta, \sigma)$ and $\mathcal{M}' \not\Vdash \varphi$. $\mathcal{M}'$ can be reduced to a sequence mapping $\mathcal{M}$ over the minimal φ-signature such that $\mathcal{M}(\varphi) = (\delta, \sigma)$ and $\mathcal{M} \not\Vdash \varphi$ (by the Sequence-Mapping Length Extension Proposition 4.21 and its subsequent remark). Now let $P^{\mathcal{M}}$ and P be as in the Generalised Answer Set Correspondence Theorem 4.29. $P^{\mathcal{M}} \cup \mathit{Enc}(\varphi)$ admits a unique answer set $\mathcal{A}$. By the Encoding Definition 4.25 we have $\eta(\varphi) \notin \mathcal{A}$ (since $\mathcal{M} \not\Vdash \varphi$), hence $\mathcal{A}$ is also the unique answer set for $P^{\mathcal{M}} \cup \mathit{Enc}(\varphi) \cup \{\ \texttt{:-}\ \eta(\varphi)\texttt{.}\}$. Since the related history of $\mathcal{M}$ is just the initial state, $\mathcal{M}^{\texttt{true}}$ and $\bigcup_{v \in \mathcal{V}s_\varphi} S^{\texttt{true}}_{init}(0)[+v]$ denote the same set, hence $P^{bc}_\varphi(G) = P \cup \mathit{Enc}(\varphi) \cup \{\ \texttt{:-}\ \eta(\varphi)\texttt{.}\}$. $P^{\mathcal{M}}$ and P do not contain heads of $\mathit{Enc}(\varphi) \cup \{\ \texttt{:-}\ \eta(\varphi)\texttt{.}\}$, hence by the Splitting Theorem 2.10 and Theorem 4.29 (where the development mapping $\mathcal{D}_{\delta,\varphi}$ is such that $\mathcal{D}_{\delta,\varphi}(v) = (S_{init})$ for all $v \in \mathcal{V}s_\varphi$), $\mathcal{A}$ is also an answer set for $P^{bc}_\varphi(G)$.

Induction Step: Let $\delta = (S_{init}, \ldots, S_k)$ and $\deg(\varphi) = n$. Assume $S_k \xrightarrow{A_k} S_{k+1}$ for some A_k and S_{k+1} and let $\delta' = (\delta, (S_k, S_{k+1}))$. We prove that if $S_{k+1} \not\models_{\delta'} \varphi$, then $P^{is}_\varphi(G)$ admits an answer set.

$S_{k+1} \not\models_{\delta'} \varphi$ implies that there is an n-max sequence $\sigma' = (S_{k+1}, \ldots, S_{k+m+1})$ such that $\sigma' \not\models_{\delta'} \varphi$. It follows that $\sigma = (S_k, S_{k+1}, \ldots, S_{k+m+1})$ is an $(n+1)$-max sequence such that $\sigma \not\models_\delta \bigcirc\varphi$. By an argumentation similar to the base case—considering

$\bigcirc\varphi$ instead of φ and an arbitrary development δ instead of the initial development (S_{init})—there is a sequence mapping $\mathcal{M}$ over the minimal $\bigcirc\varphi$-signature Sig such that $\mathcal{M}(\bigcirc\varphi) = (\delta, \sigma)$ and $\mathcal{M} \not\Vdash \bigcirc\varphi$, and the program

$$P = \mathcal{M}^{\mathsf{true}} \cup G_{Sig} \cup P^{c_1,c_2}_{Sig} \cup \mathcal{M}^{\mathsf{does}} \cup Enc(\bigcirc\varphi) \cup \{\ \texttt{:-}\ \eta(\bigcirc\varphi)\,\texttt{.}\}$$

(where P^{c_1,c_2}_{Sig} is as in Theorem 4.29) yields a unique answer set. Now program $P^{is}_{\varphi}(G)$ can be constructed by

1. adding $Enc(\varphi) \cup \{\ \texttt{:-}\ \mathbf{not}\,\eta(\varphi)\,\texttt{.}\}$ to P,
2. replacing $P^{c_1,c_2}_{Sig} \cup \mathcal{M}^{\mathsf{does}}$ with P^{legal}_{Sig} in the program resulting from item 1, and
3. replacing $\mathcal{M}^{\mathsf{true}}$ with $P^{setgen}_{\bigcirc\varphi}$ in the program resulting from item 2.

In the following we argue that these changes retain the satisfiability of the respectively resulting programs, which shows that $P^{is}_{\varphi}(G)$ admits an answer set and hence completes the proof of the induction step.

1. Without loss of generality, assume $\mathcal{V}_{\varphi}(\pi) = \mathcal{V}_{\bigcirc\varphi}(\bigcirc\pi)$ for all $\pi \in Pos_{\varphi}$ (and hence, $\mathcal{V}_{\bigcirc\varphi}(\epsilon) = \mathcal{V}_{\bigcirc\varphi}(\bigcirc)$). Then $\mathcal{M}$ is also a sequence mapping for φ with respect to δ (by the Sequence-Mapping Correspondence Proposition 4.18). By the induction hypothesis we have $S_k \models_{\delta} \varphi$. Hence, considering sequence $\sigma = (S_k, \ldots, S_{k+m}, S_{k+m+1})$ as above, we also have $(S_k, \ldots, S_{k+m}) \models_{\delta} \varphi$ (by the formula semantics from Definition 4.5) and thus $\sigma \models_{\delta} \varphi$ (by the Sequence Extension Proposition 3.6). Now the Semantics Equivalence Theorem 4.19 implies that each sequence mapping $\mathcal{M}'$ for φ with respect to δ which is such that $\mathcal{M}'(\varphi) = (\delta, \sigma)$ satisfies $\mathcal{M}' \Vdash \varphi$. Hence, especially $\mathcal{M}$ (which is such that $\mathcal{M}(\varphi) = (\delta, \sigma)$) satisfies $\mathcal{M} \Vdash \varphi$. This implies that $P \cup Enc(\varphi) \cup \{\ \texttt{:-}\ \mathbf{not}\,\eta(\varphi)\,\texttt{.}\}$ admits a unique answer set (by the Encoding Definition 4.25 and the Splitting Theorem 2.10).
2. Answer set existence for the program resulting from the replacement mentioned in item 2 follows by the application of the Generalised Answer Set Correspondence Theorem 4.29 and the Splitting Theorem 2.10.
3. First note that the sequence mapping $\mathcal{M}$ determines a unique development mapping which ranges over $\mathcal{V}s_{\varphi}$ and maps each view v from $\mathcal{V}s_{\varphi}$ to the $|\delta|$-prefix of $\mathcal{M}(v)$, as this prefix is always of length $|\delta|$ according to the motivation below the Sequence Mapping Definition 4.14. Hence, the state-set generator $P^{setgen}_{\bigcirc\varphi}$ admits an answer set which corresponds to $\mathcal{M}^{\mathsf{true}}$ according to Definition 4.28, which implies the existence of an answer set for the program resulting from the replacement mentioned in item 3 (again by the Splitting Theorem 2.10). □

4.7.2 Restricted Completeness

In Section 3.5.2, we established the completeness for knowledge-free formulas under the assumption of an *accurate* state generator (cf. Definition 3.16) which restricted generated states to be reachable. In the case of epistemic state sequence invariants, we will need the same assumption for the generated states of each view. Additionally, their respective developments need to be related according to the accessibility relation. This is stated with the following definition.

Definition 4.31 (Accurate State-Set Generator). *Let G be a valid GDL specification, $\varphi \in \mathcal{ESIN}_G$, and let P_{φ}^{setgen} be a state-set generator. P_{φ}^{setgen} is called* accurate *if, for every answer set $\mathcal{A}$ of P_{φ}^{setgen}, there is a development mapping $\mathcal{D}_{\delta,\varphi} : \mathcal{V}s_{\varphi} \to \Delta_G$ for φ wrt. δ such that for all $f \in \Sigma$ and for all $v \in \mathcal{V}s_{\varphi}$: $\mathbf{true}(f, 0, v) \in \mathcal{A}$ iff $f \in last(\mathcal{D}_{\delta,\varphi}(v))$.* ■

Based on the given definition, the completeness can be established as follows.

Theorem 4.32 (Generalised Restricted Completeness). *Let G be a valid GDL specification whose initial state is S_{init}, and let φ be a positive-knowledge formula over G. Moreover, let $P_{\varphi}^{is}(G)$ be constructed over an accurate state-set generator. If for all finite developments $\delta = S_{init} \xrightarrow{A_0} S_1 \ldots \xrightarrow{A_{k-1}} S_k$ we have $S_k \vDash_{\delta} \varphi$, then $P_{\varphi}^{bc}(G)$ and $P_{\varphi}^{is}(G)$ are inconsistent.*

Proof: We prove that if $P_{\varphi}^{bc}(G)$ or $P_{\varphi}^{is}(G)$ admits an answer set, then there is a development δ such that $last(\delta) \nvDash_{\delta} \varphi$. Consider the following cases:

- If $P_{\varphi}^{bc}(G)$ admits an answer set $\mathcal{A}$, then $\mathcal{A}$ is also an answer set for $P_{\varphi}^{bc}(G) \setminus \{$ `:-` $\eta(\varphi)$`.`$\}$, and $\mathcal{A}$ does not contain $\eta(\varphi)$. Let $\delta = (S_{init})$ and let $\mathcal{D}$ be the development mapping for φ wrt. δ which is such that $\mathcal{D}(v) = (S_{init})$ for all $v \in \mathcal{V}s_{\varphi}$. Since the heads of $Enc(\varphi)$ do not occur elsewhere, the Splitting Theorem 2.10 and the Generalised Answer Set Correspondence Theorem 4.29 imply existence of a sequence mapping $\mathcal{M}$ for φ wrt. δ over the minimal φ-signature such that $\mathcal{A}$ is the unique answer set for program $P^{\mathcal{M}}$, where $P^{\mathcal{M}}$ is defined as in Theorem 4.29. Since $\eta(\varphi) \notin \mathcal{A}$, we have $\mathcal{M} \nVdash \varphi$ by the Encoding Definition 4.25, which implies $S_{init} \nvDash_{\delta} \varphi$ by the Semantics Equivalence Theorem 4.19.
- Since the state-set generator $P_{\bigcirc\varphi}^{setgen}$ is accurate, existence of an answer set for $P_{\varphi}^{is}(G)$ implies (by Theorem 2.10) that there is a development mapping $\mathcal{D}$: $\mathcal{V}s_{\bigcirc\varphi} \to \Delta_G$ for $\bigcirc\varphi$ wrt. some development δ such that

$$P = (P_{\varphi}^{is}(G) \setminus P_{\bigcirc\varphi}^{setgen}) \cup \mathcal{D}^{\mathtt{true}}$$

  admits an answer set $\mathcal{A}$, where $\mathcal{D}^{\mathtt{true}}$ is again defined in analogy to the state encoding for a sequence mapping from Definition 4.22. Similar to the proof of the base case, Definition 4.25, Theorem 2.10, and Theorem 4.29 imply existence of a sequence mapping $\mathcal{M}$ for $\bigcirc\varphi$ wrt. δ over the minimal $\bigcirc\varphi$-signature such that $\mathcal{M} \nVdash \bigcirc\varphi$ and hence $(\mathcal{M}, |\delta|, \mathcal{V}_{\bigcirc\varphi}(\epsilon)) \nVdash \bigcirc\varphi$. This implies that development $\mathcal{M}(\mathcal{V}_{\bigcirc\varphi}(\epsilon))$ is of the form $(\delta, (S_{|\delta|}, S_{|\delta|+1}, \ldots, S_{|\delta|+m}))$ for some $m \geq 1$ and some states $S_{|\delta|+1}, \ldots, S_{|\delta|+m}$. By the Sequence-Mapping Correspondence Proposition 4.18 (and since the view names of φ and $\bigcirc\varphi$ coincide), $\mathcal{M}$ is also a sequence mapping for φ wrt. the development $\delta' = (\delta, (S_{|\delta|}, S_{|\delta|+1}))$, and such that $(\mathcal{M}, |\delta'|, \mathcal{V}_{\varphi}(\epsilon)) \nVdash \varphi$. This yields $last(\delta') \nvDash_{\delta'} \varphi$ by the Semantics Equivalence Theorem 4.19. □

Similar to Section 3.5.2, the main implication of the completeness result is that the proof method converges to perfect results when state-set generators are improving, and that the method is hence reliable. With an extensive example in Section 4.6 we have already shown that the method is effective in Krieg-Tictactoe even when using a state-set generator which incorporates non-reachable states and does not relate corresponding developments with respect to the accessibility relation. In Section 5.4.2, we will provide further experiments which show that the method is effective in other incomplete-information games as well.

4.7.3 Sound and Complete Verification at Fixed Depth

In Section 3.5.3 we have shown that the program construction for the base case allows the sound and complete verification of knowledge-free formulas with respect to all states at one given depth of the game tree. In the following, we will generalise this result to positive-knowledge formulas.

Proposition 4.33 (Generalised Correctness on Single States)**.** *Let G be a valid GDL specification, φ be a positive-knowledge formula over G, and let $t \in \mathbb{N}$.*

$$P^{bc}_{\bigcirc^t\varphi}(G) \textit{ is inconsistent} \quad \textit{iff} \quad \textit{for all developments } \delta \in \Delta_G \textit{ s.t.} |\delta| = t : \textit{last}(\delta) \vDash_\delta \varphi$$

Proof: The proof follows the argumentation of the proof for the corresponding result on correctness for single states (Proposition 3.18), using Theorem 4.30 instead of Theorem 3.15 (concerning Soundness) and Theorem 4.32 instead of Theorem 3.17 (concerning Completeness). □

The result does not involve a state-set generator and hence does not need the strong requirement of an accurate state-set generator to be complete. However, as its counterpart from Section 3.5.3, it does not apply any abstraction of the search space and is hence practically applicable to small depths t of the game tree only. We have already seen how the corresponding result for knowledge-free formulas can be used to solve single-player games. Its generalisation will prove useful in Section 4.8.1 and in the implementation of the proof method in Section 5.4.2.

4.8 Improvements

In this section, we discuss some possible extensions to our induction proof method for positive-knowledge formulas. First, we show how to obtain more significant results in the base case proof (Section 4.8.1). Then, we demonstrate how previously proved formulas can be incorporated in our method (Section 4.8.2) and how it can be extended in order to prove multiple formulas simultaneously (Section 4.8.3).

4.8.1 Strengthening the Base Case Proof

With Proposition 4.8 we have shown that players have complete knowledge of the initial state. More formally, we have shown that the initial state entails an epistemic sequence invariant φ if and only if it entails the formula $\mathit{kf}_0(\varphi)$ obtained from φ by removing all knowledge operators which are not in the scope of $\bigcirc$. This has two major consequences:

1. The answer set program for the base case can be constructed on $\mathit{kf}_0(\varphi)$ instead of φ without changing the result of the base case proof attempt, as this attempt correctly verifies φ with respect to the initial state (cf. Proposition 4.33 for $t = 0$).

2. In case the reduction $\mathit{kf}_0(\varphi)$ of a formula φ is true with respect to arbitrary initial game states, this also translates to φ itself. E.g., this is the case for the example formulas (4.3) and (4.4) from page 68 and for the formula

$$\neg \mathit{legal}(x, \mathit{mark}(1, 1)) \vee K_x \mathit{legal}(x, \mathit{mark}(1, 1))$$

from our running example.

While item 1 can be exploited to obtain a more efficient base-case verification in a proof attempt due to the reduced structural complexity of the answer set program which results from $\mathit{kf}_0(\varphi)$, item 2 has a negative consequence: The base case proof for a formula φ of the mentioned shape is always successful, even if φ is not valid in all reachable state. Hence, the base case proof does not provide any information for our induction method. More specifically, φ can never be proved to be *not* valid in our setting, as only a counter example for the base case allows to draw this conclusion due to the incompleteness of the method for non-accurate state-set generators in the induction step.

However, the generalised correctness on single states (Proposition 4.33) can be used to overcome this drawback: instead of attempting a proof for φ (or, equivalently, for $\mathit{kf}_0(\varphi)$) in the initial state, we can attempt proofs on $\bigcirc^t\varphi$ for arbitrary $t \geq 1$. In case of an answer set for some $P^{bc}_{\bigcirc^t\varphi}(G)$, the proposition allows to conclude that φ is not entailed in at least one reachable state, and hence that φ is not valid. This process can be iterated for increasing t as shown for Weak Winnability in Algorithm 3.1 (cf. Section 3.6.1 on page 49), yielding a method for the refutation of a formula which is increasingly informative over time. In addition to the formulas mentioned in item 2, this technique can be applied to strengthen the base case proof of any formula that is satisfied in the initial state of a game. In Section 5.4.2, we will use this approach to effectively obtain refutations for positive-knowledge formulas.

4.8.2 Adding Previously Proved Positive-Knowledge Formulas

A set of formulas which have previously been proved can help to eliminate unintended counter examples in the induction step which may occur whenever the underlying state-set generator is not accurate (cf. Section 4.5.2 and Section 4.7.2). Similar to the knowledge-free setting, this can be achieved by incorporating encodings for the additional formulas in the induction step program. Recall the induction step program $P^{is}_\varphi(G)$ for a positive-knowledge formula φ from Section 4.5.2. It is constructed over the minimal $\bigcirc\varphi$-signature Sig and assumes that the view names of φ and $\bigcirc\varphi$ coincide.

$$\begin{aligned} P^{is}_\varphi(G) \quad = \quad & P^{setgen}_{\bigcirc\varphi} \cup G_{\mathit{Sig}} \cup P^{legal}_{\mathit{Sig}} \cup \\ & \mathit{Enc}(\varphi) \cup \{\texttt{:-}\ \textbf{not}\ \eta(\varphi).\ \} \cup \mathit{Enc}(\bigcirc\varphi) \cup \{\ \texttt{:-}\ \ \eta(\bigcirc\varphi).\ \}. \end{aligned}$$

Let Ψ be a set of previously proved positive-knowledge formulas, i.e., such that for all $\psi \in \Psi$ and for all developments $\delta \in \Delta_G$ we have $\mathit{last}(\delta) \vDash_\delta \psi$. To incorporate this information in $P^{is}_\varphi(G)$, we need to add an encoding $\mathit{Enc}(\psi)$ for each $\psi \in \Psi$ together with the constraint $\{\ \texttt{:-}\ \textbf{not}\,\eta(\psi)\,\texttt{.}\}$ which states that ψ is true. Each of these encodings requires an epistemic temporal GDL extension together with an action-set generator. These required clauses have to respect the view-name structure of ψ

which is possibly different to the view-name structures of other formulas in $\{\bigcirc\varphi\} \cup \Psi$. Hence, we have to construct a joint structure which is compatible with each of the individual ones. This is achieved with the formula $\rho = (\bigcirc\varphi) \wedge \bigwedge_{\psi\in\Psi} \psi$, using the minimal ρ-signature Sig together with a view naming $\mathcal{V}_\rho$ of ρ. In slight deviation from the definition of a formula encoding, formulas $\psi \in (\{\bigcirc\varphi\} \cup \Psi)$ are then encoded over the view naming $\mathcal{V}_\rho$. The respective top-level views $\mathcal{V}_\rho(\psi)$ coincide, hence an interdependency is created which allows to restrict the search space. All subsequent view names for ψ, however, are completely disjoint from those for different formulas ψ', which is necessary since both view-name structures possibly differ.

$P^{is}_{\varphi,\Psi}(G)$ is given as follows. As for $P^{is}_{\varphi}(G)$, the view names of φ and $\bigcirc\varphi$ are assumed to coincide. Additionally, all applied encodings $Enc(\psi)$ are assumed over the view naming $\mathcal{V}_\rho$.

$$\begin{array}{rcl} P^{is}_{\varphi,\Psi}(G) & = & P^{setgen}_{\rho} \cup G_{Sig} \cup P^{legal}_{Sig} \cup \\ & & Enc(\varphi) \cup \{ \texttt{:-}\ \textbf{not}\ \eta(\varphi).\ \} \cup Enc(\bigcirc\varphi) \cup \{\ \texttt{:-}\ \ \eta(\bigcirc\varphi).\ \} \cup \\ & & \bigcup_{\psi\in\Psi}(Enc(\psi) \cup \{ \texttt{:-}\ \textbf{not}\ \eta(\psi).\ \}) \end{array}$$

Soundness and Completeness

For arbitrary formulas $\psi \in \Psi$ and developments $\delta \in \Delta_G$, each sequence mapping $\mathcal{M}_{\delta,\rho}$ for ρ with respect to δ corresponds to a sequence mapping $\mathcal{M}_{\delta,\psi} := \mathcal{M}_{\delta,\rho}|_{\mathcal{V}s_\psi}$ for ψ with respect to δ, and by the Sequence-Mapping View Reduction Proposition 4.17 we have

$$\mathcal{M}_{\delta,\rho} \Vdash \psi \text{ iff } \mathcal{M}_{\delta,\psi} \Vdash \psi.$$

By the Sequence-Mapping Length Extension Proposition 4.21 and its subsequent remark, sequence mapping $\mathcal{M}_{\delta,\psi}$ in turn corresponds to a sequence mapping $\mathcal{M}^{min}_{\delta,\psi}$ over the minimal ψ-signature such that

$$\mathcal{M}_{\delta,\psi} \Vdash \psi \text{ iff } \mathcal{M}^{min}_{\delta,\psi} \Vdash \psi.$$

The given arguments are sufficient to generalise the soundness (cf. Theorem 4.30) and completeness (cf. Theorem 4.32) of the proof method to an additionally considered set of previously proved formulas Ψ in the induction step. Concerning the soundness result, the GDL specification additionally needs to be playable in case the maximal degree n_Ψ of formulas in Ψ is such that $n_\Psi > \deg(\varphi) + 1$. This is due to a then necessary extension of a sequence mapping in the argumentation of the proof similar to the extension of a sequence in the argumentation of the proof for the Soundness Theorem 3.15. The completeness result does not need this restriction, since the respective reduction of a sequence mapping is always possible.

4.8.3 Proving Multiple Properties At Once

In Section 3.6.2 we have established an extension of the proof method for knowledge-free formulas which allows to verify multiple properties at once, constructing only one answer set program to establish all base case proofs, and one for all induction steps. This extension can be lifted to the proof method for positive-knowledge formulas. Let Φ be a finite set of positive-knowledge formulas which are to be proved. For the base case, we consider the formula $\rho^{bc} = \bigwedge_{\varphi\in\Phi} \varphi$, the respective minimal ρ^{bc}-signature Sig, a view naming $\mathcal{V}$ for ρ^{bc}, and the respective set $\mathcal{V}s$ of view names from $\mathcal{V}$. Assuming that all encodings are over $\mathcal{V}$, the base case program for multiple formulas is given as follows.

$$\begin{array}{rcl} P_{\Phi}^{bc}(G) & = & (\bigcup_{v \in \mathcal{V}s} S_{init}^{\texttt{true}}(0)[+v]) \cup G_{Sig} \cup P_{Sig}^{legal} \cup \\ & & \bigcup_{\varphi \in \Phi} Enc(\varphi) \end{array}$$

As in the knowledge-free case, constraints of the form $\texttt{:-}\ \eta(\varphi)$ are omitted in order to keep all relevant answer sets for formulas from Φ which do not satisfy one of the constraints.

For the induction step, we additionally consider a finite set Ψ of previously proved formulas. We consider the formula $\rho^{is} = (\bigcirc \bigwedge_{\varphi \in \Phi} \varphi) \wedge \bigwedge_{\psi \in \Psi} \psi$, the respective minimal ρ^{is}-signature Sig, a view naming $\mathcal{V}$ for ρ^{is}, and the respective set $\mathcal{V}s$ of view names from $\mathcal{V}$. Assuming that the view names of φ and $\bigcirc\varphi$ coincide and that the encodings are over $\mathcal{V}$, the induction case program for multiple formulas is given as follows.

$$\begin{array}{rcl} P_{\Phi,\Psi}^{is}(G) & = & P_{\rho^{is}}^{setgen} \cup G_{Sig} \cup P_{Sig}^{legal} \cup \\ & & \bigcup_{\varphi \in \Phi}(Enc(\varphi) \cup Enc(\bigcirc\varphi)) \cup \\ & & \bigcup_{\psi \in \Psi}(Enc(\psi) \cup \{\texttt{:-}\ \textbf{not}\ \eta(\psi).\ \}). \end{array}$$

We can prove this generalisation sound when requiring the GDL specification to be playable, using similar arguments to those in the proof of the corresponding Theorem 3.19 for knowledge-free formulas together with the additional arguments given in Section 4.8.2. Similarly, we can apply arguments from the proof of Theorem 3.20 to show that this generalisation succeeds in proving at least all the state sequence invariants that can be proved with the verification method for single positive-knowledge formulas, when the latter method is enriched by the same set of previously proved formulas Ψ as sketched in Section 4.8.2.

4.9 Discussion

In Section 3.7.2, we have argued that the success of our verification method for knowledge-free properties is majorly based on the compact representation of counter examples. This has been achieved by restrictions of our property specification language to

1. universal path quantification, in order to be able to represent counter examples in a *linear* time structure, and
2. bounded time reference, in order to be able to represent counter examples within a *finite* time structure.

Item 1 allows to restrict the blowup of the temporal GDL extension to be linear compared to the original GDL clauses, whereas it would be exponential when employing a branching-time structure. Since the latter cannot be dealt with efficiently even in simple games, this restriction is inherent for a successful verification method. The goal of carrying the efficiency of a linear time structure over to the verification of player-specific knowledge causes another inherent restriction: Finding a counter example for some formula φ that is *not* known to a player r requires to find a state where r

actually knows φ. This in turn amounts to a *universal* path quantification that is not representable within a linear time structure and hence disqualifies this kind of knowledge for verification.

Nevertheless, in this chapter we have seen that our language can express important game properties. Their validity is by no means seen easily, we refer to Section 4.6 which required a quite complex argumentation for the validity of $\neg \mathit{legal}(x, \mathit{mark}(1, 1)) \lor K_x \mathit{legal}(x, \mathit{mark}(1, 1))$. Hence, our method can provide valuable verification assistance. Experiments on several properties in different games with our practical implementation will follow in Section 5.4.2, showing that our method can successfully be applied even within tight time constraints.

4.10 Summary

In this chapter, we showed how our induction proof method for knowledge-free state sequence invariants can be extended to prove positive player-specific knowledge. To this end, we incorporated a unary knowledge operator K_r for each player r of the game to our formula syntax, and provided a semantics which defines possible worlds via developments that are indistinguishable for player r. We proved that our semantics satisfies the S5 properties. We then developed an alternative semantics based on multiple related state sequences, which we referred to as sequence mappings, and showed the equivalence of both semantics with respect to formulas which only involve positive knowledge. We provided a further extension of the temporal GDL extension by a view argument, called the epistemic temporal GDL extension, to generalise correct GDL reasoning over state sequences to sequence mappings. Its union with an extension of the formula encoding that accounts for knowledge operators and an encoding of a specific sequence mapping yields a unique answer set which contains a special formula-name token if and only if that sequence mapping satisfies the formula. Based on this correspondence, we extended the induction proof method established in Chapter 3 to correctly verify positive-knowledge formulas. We generalised the soundness result of the original method, and obtained completeness in case each reachable state and its respective set of indistinguishable developments for each player of the game are known. We further generalised the result for the correct verification of properties with respect to arbitrary single reachable states. Finally, we shortly sketched how to obtain stronger results with additional base case proofs, how to incorporate previously proved formulas, and how to prove multiple properties at once. We concluded with a discussion of the implications that arise using a linear time structure for property verification in the knowledge setting.

Chapter 5

Implementation

In this chapter, we focus on the implementation of the proof method we have developed for sequence invariants in Chapter 3 and extended to epistemic sequence invariants in Chapter 4, which demonstrates that the method is practically applicable even in time-restricted settings. The general picture of the implementation is as follows. The generation of all answer set programs which are needed to prove formulas, including some optimisations on the answer set programs which will be mentioned below, have been implemented in Prolog on top of the general game player Fluxplayer [ST07, Sch11], using a variety of helpful pre-implemented structures and algorithms. Generated answer set programs are passed to programs from the state-of-the-art answer set solving collection Potassco [GKK$^+$11a] in order to be solved. The obtained results are then again processed on top of Fluxplayer code. The parts of this chapter which do not concern experimental results for epistemic properties and single player games include passages of own published work (once more, we refer to Section 7.1 for a detailed listing of the included material).

We proceed in four stages. In Section 5.1, we show how the set of all fluents $FDom$ and the set of all actions $ADom(r)$ for each player r, which are needed to generate answer set programs, can be calculated efficiently from the implicit information given with the GDL specification. Section 5.2 then lists several optimisations we applied to reduce the size of the generated answer set programs and hence to increase the performance of the answer set solver. In Section 5.3, we shortly present the tools from the answer set solving collection Potassco that we use to process generated answer set programs. Finally, in Section 5.4, we report on a variety of experiments we conducted in order to prove knowledge-free properties, epistemic properties, and weak winnability in a variety of games from previous General Game Playing Competitions.

5.1 Domain Calculation

For a practical implementation of the developed proof method, we need a reliable way to compute the set of all potential actions $ADom(r)$ for each player r in the game in order to construct action generators (cf. Section 3.4.1) and action-set generators (cf. Section 4.5.1); furthermore, the set $FDom$ of all ground fluents is needed for state generators (cf. Section 3.4.2) and state-set generators (cf. Section 4.5.2). In principle, the minimal set $ADom(r)$ is the union of all actions for r which are legal in some reachable state of the game. Similarly, the minimal set *FDom* represents the union of all reachable states. However, these sets are not directly given in the game description, and their computation needs a full traversal of the game tree in general, which is not feasible in interesting games. This motivates an approximate computation which depends on the size of the game description rather than the actual state space of the game. In order to retain the earlier established soundness and completeness results, the only requirement to the approximation is to compute supersets of the above-mentioned minimal sets *ADom*(r) and *FDom*, as additional actions are constrained not to be considered in case they are illegal, and additional ground fluents only cause additional non-reachable states in state generators and state-set generators.

A general method which meets these requirements has formally been introduced in [Sch11] and implemented in Fluxplayer by the author of the cited work. The method will briefly be reflected in the remainder of this section. To begin with, consider as an example the following subset of the Quarto clauses in Figure 2.2 from page 19:

```
init(sctrl(r1)).
next(sctrl(R))   :- true(pctrl(R)).
next(pctrl(R1))  :- true(sctrl(R2)), otherrole(R1,R2).

otherrole(r1,r2).
otherrole(r2,r1).
```

The first clause implies that set *FDom* must contain *sctrl*(*r1*). Moreover, the second clause suggests that, whenever *pctrl*(r) is in *FDom* for some r, then also *sctrl*(r) must be in *FDom* due to the shared variable R. Similarly, the third clause yields that every occurrence of *sctrl*(r_2) in *FDom* such that *otherrole*(r_1, r_2) is true implies existence of *pctrl*(r_1) in *FDom*. This intuitive formulation for the given example clauses can be formalised to a general structure of domain dependencies for an arbitrary GDL specification as follows.

Definition 5.1 (Domain Graph). *Let G be a valid GDL specification, and let G' be G together with the following three clauses:*

```
true(F)     :-  init(F).
true(F)     :-  next(F).
does(R,M)   :-  legal(R,M).
```

A domain graph *for G is the smallest directed graph $D = (V, E)$ with vertices V and edges $E \subseteq V \times V$ such that:*

- *If a term $f(t_1, \ldots, t_m)$ $(m \geq 0)$ occurs as i-th argument of an n-ary predicate or function symbol p in the head of a clause in G', then $f/m \in V$, $(p/n, i) \in V$, and $f/m \to (p/n, i) \in E$.*
- *If a variable occurs as i-th argument of an n-ary predicate or function symbol p in the head of a clause $c \in G'$ and as j-th argument of an m-ary predicate or function symbol q in a positive literal in the body of c, then $(q/m, j) \in V$, $(p/n, i) \in V$, and $(q/m, j) \to (p/n, i) \in E$.* ■

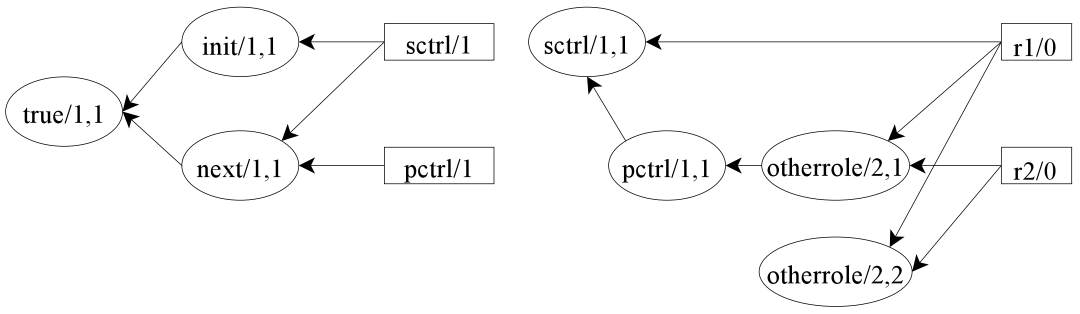

Figure 5.1: An example domain graph for calculating argument domains and ground instances of atoms and functions. Ellipses mark nodes which represent argument domains, while squares mark nodes representing ground instances.

Intuitively, for predicate or function symbols p with arity n, a node of the form p/n in the domain graph for G represents the set of all ground instances $p(t_1, \dots, t_n)$ that may be of relevance in the game, and each node of the form $(p/n, i)$ represents the domain of the i-th argument of p. The three additional clauses model the intuition from the Quarto example: terms which are encoded in the initial state via **init** carry over to predicate symbol **true**, similarly those from **true** connect to **next**, and performed actions via **does** relate to potential legal moves over **legal**. An example domain graph for the subset of the Quarto clauses given at the beginning of this section is shown in Figure 5.1.

Based on Definition 5.1, the set of all ground instances for arbitrary predicate and function symbols as well as the domain for each of their arguments can be obtained from the domain graph as follows.

Definition 5.2 (Argument Domains and Instances). *Let $D^+ = (V, E^+)$ be the transitive closure of the domain graph for a valid GDL specification G, let $0 \leq i \leq n$ and let p be an n-ary predicate or function symbol.*

- $dom(p/n, i) = \{c : c/0 \to (p/n, i) \in E^+\} \cup \bigcup_{q/m \to (p/n,i) \in E^+} inst(q/m)$
- $inst(p/n) = \{p(x_1, \dots, x_n) : x_1 \in dom(p/n, 1), \dots, x_n \in dom(p/n, n)\}$ ■

To ensure that each set $dom(p/n, i)$ and $inst(p/n)$ is finite, the *extended* GDL specification G' from Definition 5.1 has to obey the recursion restriction (Definition 2.15, second item). In case only G but not G' satisfies the recursion restriction, the set of reachable states of the game and thus also the set of ground fluents might be infinite (cf. Section 2.2.3).

With Definition 5.2, the sets $FDom$ and $ADom(r)$ (for any player r) can be given as $FDom := dom(\mathbf{true}/1, 1)$ and $ADom(r) := dom(\mathbf{does}/2, 2)$. However, instead of calculating the respective domains and listing all fluents $f \in FDom$ in the state generator, we apply Definition 5.2 to generate a finite set of negation-free clauses such that, for a new predicate symbol p of arity 1, their unique answer set entails $p(f)$ if and only if $f \in dom(\mathbf{true}/1, 1)$. In principle, the set $ADom(r)$ could be represented similarly. However, this often yields to many actions which are never legal for any player, motivating to further restrict the represented set $ADom(r)$ by adding state-independent information concerning the legal moves of player r to the finite set of negation-free clauses which encode $ADom(r)$. We skip the details of the clause-generation process and refer to [Sch11] for a thorough presentation.

5.2 Optimisations

Current answer set solvers perform a grounding step on the input answer set program prior to computing solutions. Grounding an ASP increases its size exponentially, in the worst case, and can hence easily dominate the step that actually solves the ASP both in memory and run-time requirements. Thus, for an efficient implementation, it is essential to reduce the size of the program as much as possible before grounding it; we will present the techniques we apply to reduce program size in Section 5.2.1. Since, on the other hand, a grounding method of an answer set solver is usually highly optimised, we further generate more compact formula encodings using variables, hence letting the ground instances be generated by the grounder; the applied encoding is presented in Section 5.2.2.

Since some clauses are omitted or newly introduced in the presented optimisations, the answer sets may differ from those for the original programs. However, this does not influence the soundness and completeness results which have been presented in this thesis, as all transformations respect the *existence* of an answer set with the desired properties, which can be motivated by means of the Splitting Theorem 2.10. Hence, applying these techniques prior to invoking the answer set solver has no influence on the outcome of the later presented experiments (besides reduced running times).

5.2.1 Reducing the Number of Clauses

Some of the reduction techniques mentioned in this section have already been utilised to optimise the implementation described in [ST09] and hence have been devised and implemented by the authors of the cited work. As they are also important for our implementation, we restate them using passages from a joint publication [HST12].

No Time Argument For Static Atoms In Section 3.3.1, we introduced the temporal extension of a GDL specification, which has further been extended to account for epistemic formulas in Section 4.4.1. Essentially, we added a time argument and a view argument to every atom of the specification. However, some of the predicate symbols of a game description are static, that is, do not depend on the state of the game or the moves of the players. Thus, they are equivalent in every time step and with respect to any view name. For example, in Quarto (Figure 2.2) all of the predicate symbols **role**, **distinct**, `sameattr`, `nthbit`, `!=`, `otherrole`, and `index` are static.

Formally, predicate symbol p is static if, and only if, there is no path from p to **true** or **does** in the dependency graph (cf. Definition 2.14) for the GDL description. We reduce the size of the generated answer set program by not including a time and a view argument upon generation of extended GDL clauses according to Definition 4.22 if a predicate symbol is static. Thus, clauses for atoms over those predicate symbols are only added once to the answer set program independent of the number of time steps and view names.

Cutting Encoding Indirections The encoding presented in Table 3.1 at page 38 as well as its epistemic extension from Theorem 4.26 (page 90) produces a unique 0-ary name atom for every subformula of a formula. For atomic formulas (1.), negations (2.), conjunctions (3.), all quantifiers (6.), counting quantifiers (7.), and knowledge operators (9.), each of these atoms occurs as the head of one clause only. Furthermore, unless the encoded formula contains the same sub-formula several times in the same time level and within the scope of the same knowledge operators, all of the generated atoms additionally occur only once in the body of some clause.

We reduce the number of clauses and the number of 0-ary name atoms in an encoding P by removing each clause ($r = (\mathit{head}$ `:-` $\mathit{body}))$ from P and replacing *head* by *body* in all remaining clauses of P if r is such that

- *head* does not occur as head of any other clause in P,
- *head* is not a name atom $\eta(\varphi)$ for any formula φ in a proof attempt with multiple formulas according to Sections 3.6.2 and 4.8.3, and
- if *body* is not a single atom, then *head* does neither occur negated nor in a weight atom in any other clause of P.

Separating Body Variables The following is the only of our applied optimisations which does not reduce but increase the size of the ASP in order to reduce the size of its ground version. It is based on the observation that grounded program size is strongly influenced by the number of variables in a clause, which can often be reduced by introducing new atoms and clauses. Consider, e.g., the clause

```
p(X,Z) :- q(X,Y), r(Y), s(Z).
```

where `Y` does solely occur in the body. If we replace this clause by

```
p(X,Z) :- qr(X), s(Z).
qr(X)  :- q(X,Y), r(Y).
```

where `qr` is a new predicate symbol, we obtain two clauses with two variables each instead of one clause with three variables. This changes the number of ground clauses from $|\mathit{dom}(X)| * |\mathit{dom}(Y)| * |\mathit{dom}(Z)|$ to $|\mathit{dom}(X)| * |\mathit{dom}(Z)| + |\mathit{dom}(X)| * |\mathit{dom}(Y)|$, which amounts to a considerable reduction if the variable domains comprise more than two elements.

We apply the following transformation to all clauses in an answer set program P until a fixed point is reached: Consider a clause $(\mathit{head}$ `:-` $\mathit{body}.) \in P$ containing a variable V which does not occur in *head*. Let body^+ be all literals from *body* that contain V, body^- be all remaining literals from *body*, and $\{V_1, \ldots, V_n\}$ be the set of variables in body^+ that also occur in body^- or in the head of the clause. Furthermore, let $\{V_1^-, \ldots, V_m^-\}$ be the variables in body^+ that only occur in negative literals in body^+. If body^- contains a variable that is not contained in $\{V_1, \ldots, V_n\}$, then *head* `:-` *body.* is replaced by the clauses

$$\begin{aligned} \mathit{head} \ \texttt{:-} & \ p(V_1, \ldots, V_n), \mathit{body}^-. \\ p(V_1, \ldots, V_n) \ \texttt{:-} & \ \mathit{body}^+, \mathit{dom}_1(V_1^-), \ldots, \mathit{dom}_m(V_m^-). \end{aligned}$$

where p and $dom_1, \ldots, dom_m$ are predicate symbols that do not occur anywhere else in P. For each $i \in \{1, \ldots, m\}$, $dom_i(V_i^-)$ is used to define the domain of variable V_i^-. This domain is obtained from an arbitrary occurrence of V_i^- in a positive literal in *body*, and for its declaration in the program we add a finite set of stratified clauses according to Definition 5.2 (Argument Domains and Instances) and the comment to clause generation below this definition. The additional atoms $dom_i(V_i^-)$ are necessary to keep the program allowed, i.e., to ensure that each variable in a clause occurs in at least one positive literal (cf. Definition 2.2 at page 9).

Removing Unnecessary Clauses Depending on the formula that is to be proved, some of the clauses in the generated ASP might not be necessary. For example, the temporal GDL extension (which forms the basis for the epistemic temporal GDL extension) contains clauses with head $\mathbf{true}(f, i+1)$ for all time steps i obtained from the next clauses of the game. However, the remaining clauses of the ASP do not contain any instance of $\mathbf{true}(f, n+1)$ for the last time step n. Hence, removing those clauses does not influence the existence of an answer set for the program. The same applies to $\mathbf{legal}(r, a, n)$: Legality of moves is irrelevant in the last time step (n), unless the formula to be proved depends on atom $\mathbf{legal}(r, a)$. Removing unnecessary clauses from an ASP reduces the size of the grounded program and thus the cost for both grounding and solving the answer set program.

To capture the notion of necessary vs. unnecessary clauses, we first extend the definition of dependency graphs (cf. Definition 2.14) to ground atoms. An extended dependency graph differs from the standard graph in that it has ground atoms as nodes instead of predicate symbols, and there is an edge $p(\vec{t_p}) \to q(\vec{t_q})$ in the graph whenever there is a ground instance of a clause with head $p(\vec{t_p})$ and $q(\vec{t_q})$ in the body. Now let *Names* be the set of all 0-ary name atoms $\eta(\varphi)$ such that φ is either a formula to be proved or to be included in the answer set program as already proved before. Furthermore, let *GeneratorAtoms* be the set containing each epistemic temporal extension of **terminal**, $\mathbf{legal}(r, m)$, and $\mathbf{sees}(r, p)$ which is needed in the constructed action-set generator (cf. Section 4.5.1). Then the set of necessary atoms for an answer set program P is

$$Atoms = \{q \ : \ \text{there is a } p \in Names \cup GeneratorAtoms \text{ such that } p \to^* q\}$$

where $p \to^* q$ denotes the existence of a (possibly 0-length) path from p to q in the extended dependency graph of P. To conclude, we remove all clauses from P whose head is an atom that does not unify with any atom in *Atoms*.

5.2.2 Formula Encoding With Variables

According to the formula encoding in Table 3.1 (cf. page 38) and its extension by view arguments sketched in Theorem 4.26 (cf. page 90), formulas with variables yield an encoding which consists of multiple sets of structural similar clauses for single instances of the formula variables. E.g., reconsider the encoding for $(\exists \vec{X} : D_{\vec{X}})\, \psi[\vec{X}]$:

$$\begin{aligned} Enc((\exists \vec{X} : D_{\vec{X}})\, \psi[\vec{X}], i, v) \quad = \quad & \textstyle\bigcup_{\vec{t} \in D_{\vec{X}}} \{\eta((\exists \vec{X} : D_{\vec{X}})\, \psi[\vec{X}], i, v) \text{ :- } \ \eta(\psi[\vec{X}/\vec{t}], i, v)\,.\} \\ & \cup \textstyle\bigcup_{\vec{t} \in D_{\vec{X}}} Enc(\psi[\vec{X}/\vec{t}], i, v) \end{aligned}$$

For each instance $\vec{t} \in D_{\vec{X}}$, a different clause set is generated, and these sets only differ with respect to the used name atoms and the considered variable instances on atom level. A more compact encoding Enc' can be obtained by carrying along the set of previously used variables $\vec{Y}$, extending the head name atom by arguments ranging over $\vec{Y}$, and furthermore adding the current quantifier variables to the variables of the body name atom as well as the subsequent encoding. Let $\vec{X} = (X_1, \ldots, X_k)$, $\vec{Y} = (Y_1, \ldots, Y_l)$, let $\mathit{doms}((\vec{X}, \vec{Y}))$ be an abbreviation for the list of atoms $\mathit{dom}_1(X_1), \ldots, \mathit{dom}_k(X_k), \mathit{dom}_1(Y_1), \ldots, \mathit{dom}_k(Y_l)$, and let $\mathit{Doms}(D_{\vec{X}})$ be a finite set of stratified clauses which encodes the domains of $\vec{X} = (X_1, \ldots, X_k)$ using the atoms $\mathit{dom}_1(X_1), \ldots, \mathit{dom}_k(X_k)$ (similar clauses for $\mathit{dom}_1(Y_1), \ldots, \mathit{dom}_k(Y_l)$ are assumed to be included at a higher level). Then Enc' can be specified as follows.

$$\begin{aligned} &\mathit{Enc}'((\exists \vec{X} : D_{\vec{X}})\, \psi[\vec{X}], i, v, \vec{Y}) = \\ &\qquad \{\eta((\exists \vec{X} : D_{\vec{X}})\, \psi[\vec{X}], i, v)(\vec{Y}) \texttt{ :- } \eta(\psi, i, v)((\vec{X}, \vec{Y})), \mathit{doms}((\vec{X}, \vec{Y}))\,.\} \\ &\qquad \cup\, \mathit{Enc}'(\psi[\vec{X}], i, v, (\vec{X}, \vec{Y})) \,\cup\, \mathit{Doms}(D_{\vec{X}}) \end{aligned}$$

Assuming that the variable names in different quantifier occurrences are pairwise distinct, the encoding for all formula structures besides $K_r\varphi$ can be altered similar to the example above, which yields a compact encoding where each clause with some head $p(\vec{Y})$ stands for $|D_{\vec{Y}}|$ instances. When encountering knowledge formulas, each of these instances requires a different view, which can be encoded (under slight abuse of view naming $\mathcal{V}_\varphi$ in the encoding for $K_r\varphi$) as extension of the view arguments by inherited variable arguments on atom level:

$$\mathit{Enc}(p(\vec{t}), i, v, \vec{Y}) = \{\eta(p(\vec{t}), i, v)(\vec{Y}) \texttt{ :- } p(\vec{t}, i, v(\vec{Y})), \mathit{doms}(\vec{Y})\,.\}$$

Correspondingly, the epistemic temporal GDL extension then has to be given over the variable-extended view names $v(\vec{Y})$ (also enriched with domain atoms $\mathit{doms}(\vec{Y})$) which occur in the respective formula encoding. Besides its compactness, this altered formula encoding has the advantage of shifting the process of ground clause generation to the highly optimised grounder of an answer set solver, which further increases performance.

Compact Encoding for Multiple Formulas The same considerations can be applied to obtain more compact encodings when proving multiple formulas at once (cf. Sections 3.6.2 and 4.8.3). E.g., consider the set of all formulas of the shape

$$\varphi_{f(\vec{t})} = \mathit{true}(f(\vec{t})) \supset \bigcirc \mathit{true}(f(\vec{t}))$$

where $f(\vec{t})$ is any element of the set $\mathit{inst}(\mathit{true}/1)$ (cf. Definition 5.2 at page 113) of all ground fluent instances in the currently considered game. This formula set will reappear in Section 5.4.1 as part of an experiment with multiple formulas. Instead of creating an encoding for each $\varphi_{f(\vec{t})}$, we create the *single* encoding

$$\mathit{Enc}'(\mathit{true}(f(X)) \supset \bigcirc \mathit{true}(f(X)), 0, v, X) \,\cup\, \mathit{Doms}(\mathit{inst}(\mathit{true}/1)),$$

where $\mathit{Doms}(\mathit{inst}(\mathit{true}/1))$ is again a finite set of stratified clauses encoding the elements of the set $\mathit{inst}(\mathit{true}/1)$. Hence, we encounter 1-ary name atoms $\mathit{name}(f(\vec{t}))$ instead of the original 0-ary name atoms $\eta(\varphi_{f(\vec{t})})$ in each answer set which satisfies $\varphi_{f(\vec{t})}$.

5.3 The Answer Set Solving Collection Potassco

The *Potsdam Answer Set Solving Collection* [GKK$^+$11a], abbreviated *Potassco*, provides a multitude of state-of-the art tools for solving answer set programs that have scored top ranks at several international competitions over the past years. This ongoing success motivates to use Potassco for the implementation of our proof method. In this section, we give a brief overview on the tools of Potassco which are relevant in this work. We have utilised them in version 3.0.1 with standard parameter settings.

5.3.1 Clingo

Clingo is an answer set solver which operates on non-ground input programs and will be used for the experiments in Sections 5.4.1 (concerning sequence invariants) and 5.4.2 (concerning epistemic sequence invariants). It is a composition of the grounder Gringo and the actual solver Clasp, which we both present in the following.

Gringo The grounder Gringo provides a rich input language for the specification of answer set programs which (among others) includes all constructs that are needed for the construction of base case and induction step answer set programs in this thesis. It creates all ground instances from clauses of a given answer set program, which enables to compute answer sets according to their semantics given in Definition 2.9 in a subsequent step. The input program is only required to be allowed (cf. Definition 2.2). This does not suffice to ensure finiteness of the ground program, e.g. for

```
p(a).
p(f(X)) :- p(X).
```

but has the advantage to simplify program declaration, allowing to apply the grounder to our constructed programs without further addition of any domain atoms (as any valid GDL specification is allowed by Definition 2.15 and all additional program constructs are allowed as well). In our case, the grounding will be finite as long as the extended GDL specification given in Definition 5.1 obeys the recursion restriction (cf. the discussion below Definition 5.2). The grounding algorithm is based on methods originating in database evaluation [GKKS11b].

Clasp The solver Clasp takes as input a ground answer set program provided by Gringo. It is first subject to a variety of simplifying equivalence-preserving transformations. Thereafter, a solving algorithm based on conflict-driven ASP solving [GKNS07] is applied: atoms are non-deterministically assigned true or false until either all atoms are assigned (and hence an answer set has been found) or a conflict is detected. In case the conflict arises with an atom that has not been chosen non-deterministically but inferred from the program, the answer set program is unsatisfiable. Otherwise, a *backjumping* process is triggered, releasing some assignments and remembering the conflict by an added constraint.

Clasp provides a mode for "cautious reasoning", which outputs the intersection of all answer sets. Due to an efficient incremental technique, the amount of computed answer sets which is needed to obtain the intersection is restricted by the number of atoms of the input program. We will use this mode in Section 5.4.1.

5.3.2 IClingo

IClingo [GKK$^+$08] is a modified version of Clingo which provides a built-in incremental approach to solving a problem, allowing to efficiently address problems which rely on a parameter k that corresponds to solution size. The parameter ranges over natural numbers and, starting with $k = 0$, is successively increased until a solution with respect to the current instance of k and hence of minimal complexity is found (if it exists). The input of IClingo are three interconnected answer set programs B, I, and Q, where

- B does not refer to k, it contains the part of the problem specification which does not depend on parameter k;
- I is specified using parameter k, it contains the part of the problem specification which is to be cumulated with respect to increasing instances of parameter k; and
- Q is specified using parameter k, it can be understood as a query which has to be evaluated with respect to the current instance of parameter k.

More formally, given answer set programs B, I, and Q as above, IClingo attempts to find a minimal integer $t \geq 0$ such that

$$P = B \cup \bigcup_{0 \leq i \leq t} I[k/i] \cup Q[k/t]$$

admits an answer set. We will use IClingo to compute Weak Winnability of games in Section 5.4.3.

5.4 Experimental Results

We will now report on several experiments we conducted with our system using a wide range of complete-information games from the past AAAI General Game Playing Competitions, all of which are available at the online game repositories at Dresden[1] and Stanford[2]. We further include some incomplete-information games from recent publications as well as all games from the first incomplete-information track that has been realised in a competition (see Section 5.4.2 for further details). We structure our experiments into three parts, each of which provides a figure with a summary of the results. Section 5.4.1 deals with a wide range of knowledge-free properties in multi-player games of complete and incomplete information (Figure 5.2). Section 5.4.2 then considers three common categories of positive-knowledge properties in several incomplete-information games (Figure 5.3). Section 5.4.3 concludes with experiments concerning the weak winnability of many single-player and multi-player games (Figure 5.4).

All experiments were run on an Intel Core 2 Duo CPU with 3.16 GHz, and the summarised proof results for formulas in all of the mentioned figures are denoted according to the following scheme:

[1]`ggpserver.general-game-playing.de/public/show_games.jsp`
[2]`games.stanford.edu/resources/resources.html`

"y" (yes)	"n" (no)	"?" (unknown)	"–" (aborted)
proved to be valid (true in each reachable state)	proved not to be valid (counter example for some reachable state)	result unknown (counter example for some not necessarily reachable state)	proof attempt aborted after 100 seconds or the consumption of 3 GB memory

5.4.1 Sequence Invariants

The first series of experiments is concerned with knowledge-free game properties which are attempted to be proved on a variety of multi-player games. Most of the games involve two non-**random** players, additionally there are games with three (3pttc and tttcc4, cf. Figure 5.2 at page 123), five (smallest) and six (chinesecheckers6) non-**random** players. We also include Krieg-Tictactoe and several further incomplete-information games (some of them only involve one player and **random**), including a card game, a game of dice, and the famous Monty Hall. We refer to Section 5.4.2 for further details on these games.

Property Categories

For each game the following sets of formulas were generated:

Functionals In Quarto, each cell contains at most one piece (cf. property (3.2) at page 28). For example, fluent $cell(1,1,p1111)$ means that cell $(1,1)$ houses piece $p1111$. More generally speaking, for every pair of cell coordinates (x,y) there is always at most one p such that $cell(x,y,p)$ is true. Similar properties hold in many games, and in order to detect these, we generate all formulas of the form

$$(\forall \vec{X} : D_{\vec{X}})\,(\exists_{l..1} \vec{Y} : D_{\vec{Y}})\ true(f(\vec{Z}))$$

for each fluent symbol f, each $l \in \{0,1\}$, and each non-empty subsequence $\vec{Y}$ of the variables in $f(\vec{Z})$, denoting with $\vec{X}$ the (possibly empty) sequence of all variables in $\vec{Z}$ that are not in $\vec{Y}$. In addition, we identify as *control fluents* those that have one argument that ranges over the roles (e.g., $sctrl(r1)$ and $pctrl(r2)$ in Figure 2.2). If the set F_c of all ground instances of these fluents incorporates two or more distinct fluent symbols, we also attempt to prove that exactly one of them holds at any time via the formula $(\exists_{1..1} F : F_c)\ true(F)$.

Legals We include the state sequence invariant for Playability (cf. Section 3.6.4),

$$\neg terminal \supset (\forall R : D_R)\,(\exists M : D_M)\, legal(R,M), \tag{5.1}$$

where D_R is the set of roles and D_M the (finite) domain of moves. In addition, we attempt to prove the property Turn-Taking. Recall from Definition 2.21 at page 23 that we consider a game to be turn-taking if at most one player besides **random** has two or more legal moves in each reachable game state. With D_R and D_M as above, this can be expressed via

$$(\exists_{0..1} R : (D_R \setminus \{random\}))\ (\exists_{2..\infty} M : D_M)\ legal(R,M). \tag{5.2}$$

Note that this formula makes no reference to the actually used name of a noop action. The name is hence completely independent from the proof result, and does not even have to be the same in each game state.

Goal A game is to be considered zero-sum if the goal values of all players besides the random player, in case they exist, add up to 100 in each reachable terminal state (cf. Definition 2.21). This we formulate via the state sequence invariant

$$\mathit{terminal} \supset \bigwedge_{\substack{g_1,\dots,g_n \in GV \\ g_1+\dots+g_n \neq 100}} (\neg \mathit{goal}(r_1, g_1) \vee \ldots \vee \neg \mathit{goal}(r_n, g_n)), \tag{5.3}$$

where $D'_R = \{r_1, \ldots, r_n\}$ is the set of roles different from *random* and GV is the (finite) set of goal values that occur in the game description. Using the same identifiers, we furthermore include a formula which expresses that the goal values of all players are unique in each terminal state,

$$(\forall R : D'_R)\,(\mathit{terminal} \supset (\exists_{1..1} V : GV)\ \mathit{goal}(R, V)). \tag{5.4}$$

To formulate that the game is monotonic (cf. Definition 2.21), we include a third formula

$$(\forall R : D'_R)\,(\varphi_1 \wedge \varphi_2), \text{ where} \tag{5.5}$$

- φ_1 expresses that the goal value for r is unique in each reachable state:

$$\varphi_1 = (\exists_{1..1} V : GV)\ \mathit{goal}(R, V), \text{ and}$$

- φ_2 formulates that each goal value for r in a state is not higher than any goal value for r in any of its direct successor states:

$$\varphi_2 = \neg \mathit{terminal} \supset \bigwedge_{\substack{v_1, v_2 \in GV \\ v_1 > v_2}} \neg(\mathit{goal}(R, v_1) \wedge \bigcirc \mathit{goal}(R, v_2)).$$

Persistence In Quarto, once a piece is placed on the board, it remains in this cell for the rest of the game. Likewise, pieces are always permanently removed from the pool. Similar properties occur in a variety of games, and in order to detect these, we generate the set of all formulas of the form

$$\mathit{true}(f(\vec{t})) \supset \bigcirc \mathit{true}(f(\vec{t})) \quad \text{and} \quad \neg \mathit{true}(f(\vec{t})) \supset \bigcirc \neg \mathit{true}(f(\vec{t})), \tag{5.6}$$

where $f(\vec{t})$ is any ground fluent instance in the game in question.

Experiment Setup

Proving a multitude of similar properties in one run spares the solver from repeating the same tasks over and over again, such as grounding, indexing, and several clause optimisations. This significantly lowers overall time consumption. However, proving *all* of the aforementioned formulas together does not yield optimal results, because different kinds of formulas tend to require different clauses from the game description for verification and to allow fewer clause optimisations when attempted jointly. This motivated the following setup for our experiments.

Functionals are simple-structured and provide valuable state space restrictions, hence they are the first to be tested and then included in all subsequent proofs. We perform a second run, which is likely to produce further successfully proved functionals since these are often interdependent—recall that in order to prove that each cell contains at most one piece (property (3.2)) in Quarto we first needed to discover that always exactly one instance of a control fluent holds (the respective argumentation can be found in Section 3.4.2). Proof attempts for legals were run separately since, unlike in the case of functionals, their induction step requires clauses with head **legal** for time step 1. Due to their complex structure (their encoding refers to *all* legal moves of the game) their addition to the established facts tends to slow down subsequent proof attempts, which is why they are not considered in any further proofs. The property class Goal contains the only properties we considered that refer to predicate symbol **goal** and the defining clauses; hence they are attempted in a further distinct run and the results are also not included subsequently. Persistence formulas have a higher degree and thus require a copy of the GDL clauses with an additional time step, which is why they are proved together in yet another separate run.

Recall that, in order to prove multiple properties in one run, we need to find out whether the respective name atoms are contained in *every* possible answer set of the generated two programs for the base case and the induction step (cf. Section 3.6.2). Equivalently, we may consider whether the respective name atoms are contained in the *intersection* of all answer sets. The option "cautious reasoning" which is provided for Clingo (cf. Section 5.3.1) does exactly that and hence allows us to spare a time-consuming search through all answer sets.

Results

In Quarto, the prover successfully shows that in all reachable game states there is at most one selected piece; that exactly one player has control over either selecting or placing a piece; and that each cell has exactly one value (that is, a piece or *empty*). Moreover, it proves that a piece which is placed in a cell remains in this cell; that a piece which is removed from the pool stays removed; and that a non-empty cell never becomes empty again.

In Krieg-Tictactoe, we obtain that exactly one player has control to place a piece; that each cell has exactly one value $v \in \{x, o, b\}$; that an x or o which is placed in a cell remains in this cell, and that a cell which is not blank (b) will never be blank again.

Results for the other formula sets are summarised in Figure 5.2, together with results for a variety of other games. Times in column "Functionals" indicate two proof attempts (each attempt including one ASP proof for the base case and one ASP proof for the induction step), the other times indicate one attempt. The times include both generation and grounding of the respective answer set programs (the latter is done by Clingo). Additional time in the range of a few seconds is needed for the initialisation of Fluxplayer (which includes the calculation of the domain graph) once for each newly considered game. In general, many instances of the four property classes can be proved within at most a few seconds, which demonstrates that our proof method

game	Functionals		Legals (*pl*,*tt*)		Goal (*z*,*u*,*m*)		Persistence	
3pttc	0.6	(4/10/18)	0.7	(y,y)	0.3	(?,y,y)	0.9	(77/354/362)
backgammon	4.6	(7/9/24)	0.8	(y,y)	0.5	(y,y,y)	11.7	(7/1903/1920)
bidding-tictactoe	0.2	(1/10/27)	0.1	(?,n)	0.2	(?,?,?)	0.2	(9/89/108)
breakthrough	1.0	(3/3/16)	1.5	(y,y)	1.6	(y,y,y)	1.1	(32/242/260)
capture_the_king	–	(3/8/21)	3.1	(?,y)	3.4	(y,y,n)	66.4	(7/1710/1744)
cardgame	0.1	(7/10/20)	0.0	(y,?)	0.2	(y,?,n)	0.1	(1/82/100)
catcha_mouse	0.3	(4/5/18)	0.2	(?,y)	0.2	(?,y,y)	1.2	(359/896/998)
CephalopodMicro	1.7	(5/17/32)	0.6	(y,y)	1.4	(y,y,y)	1.6	(18/209/220)
checkers	–	(3/8/24)	–		9.8	(?,?,?)	–	(0/1078/1098)
chinesecheckers6	–	(4/6/10)	1.2	(y,y)	44.9	(?,?,y)	29.1	(80/634/650)
chinookDisj	0.6	(4/4/16)	39.2	(y,y)	0.4	(y,y,?)	0.8	(32/354/388)
chomp	0.1	(3/4/11)	0.1	(?,y)	0.1	(y,y,y)	0.1	(58/61/120)
connect4	0.2	(2/3/16)	0.2	(?,y)	0.3	(y,y,?)	0.3	(294/492/508)
connectFourSim	0.6	(5/19/36)	0.1	(?,n)	0.4	(y,y,y)	0.4	(192/372/392)
dots_and_boxes	4.0	(3/26/52)	0.4	(y,y)	0.2	(?,y,?)	0.8	(2608/2782/2844)
endgame	–	(4/6/18)	3.3	(?,y)	1.2	(y,y,?)	7.1	(2/511/546)
knightfight	0.5	(3/10/19)	2.2	(?,y)	0.7	(?,?,n)	1.5	(100/602/608)
kriegtictactoe	0.1	(4/7/22)	0.1	(?,y)	0.2	(y,y,?)	0.2	(27/56/76)
kriegTTT_5x5	0.2	(4/11/30)	0.1	(y,n)	0.2	(?,?,?)	0.2	(77/183/310)
mastermind	0.9	(17/27/52)	0.2	(y,y)	0.6	(?,?,n)	3.3	(690/1410/1428)
meier	1.0	(7/17/29)	0.1	(?,?)	0.3	(y,?,n)	0.4	(101/313/336)
montyhall	0.1	(2/4/8)	0.0	(?,y)	0.1	(?,?,n)	0.1	(8/18/26)
9BoardTicTacToe	1.5	(6/36/70)	0.1	(y,y)	0.2	(y,y,y)	0.4	(162/254/346)
othello-comp2007	5.0	(3/5/16)	1.5	(y,y)	–		3.5	(8/250/260)
pawn_whopping	0.2	(3/5/16)	1.5	(y,y)	0.2	(y,y,n)	0.4	(32/234/260)
quarto	11.8	(6/7/23)	8.9	(?,y)	–		28.3	(288/582/616)
qyshinsu	1.0	(12/17/33)	0.2	(y,y)	0.4	(?,y,y)	2.2	(5/674/788)
smallest	1.1	(4/4/8)	0.1	(y,n)	15.5	(?,y,y)	0.5	(12/148/160)
tictactoe	0.1	(4/4/16)	0.1	(y,y)	0.2	(y,y,n)	0.1	(27/38/58)
transit	0.7	(8/11/18)	0.1	(y,y)	0.3	(y,y,n)	0.6	(37/312/320)
tttcc4	10.5	(4/8/18)	17.8	(y,y)	1.5	(?,y,y)	15.0	(311/1228/1244)
vis_pacman3p	1.0	(4/7/20)	0.6	(?,?)	1.8	(?,?,?)	2.6	(69/907/918)

Figure 5.2: Proof times in seconds for a selection of knowledge-free sequence invariants in a variety of multi-player games. Information in parentheses: $(m/n/l)$—m formulas proved true out of n formulas from the respective set which are true in the initial state, and l is the total number of formulas in the respective set (hence, $l - n$ formulas have been proved false); (pl, tt)—pl is the result for playability (cf. (5.1)) and tt the result for turn-taking (cf. (5.2)); (z, u, m)—z is the result for zero-sum (cf. (5.3)), u the result for uniqueness of goal values (cf. (5.4)), and m the result for monotonically increasing goal values (cf. (5.5)).

is applicable even in the highly time-constrained setting of a General Game Playing Competition. Proving multiple properties at once is especially effective with persistence properties, which usually requires to check several hundred instances per game. Also more complex games like the chess variants "endgame" and "capture_the_king" yield practicable results. Checkers, on the other hand, cannot be handled efficiently due to its inherent vast amount of legal moves comprising simple piece moves, double jumps, and triple jumps.

5.4.2 Epistemic Sequence Invariants

In Chapter 4 we have developed an extension of the proof method for sequence invariants which addresses the formulation and verification of game properties involving the knowledge of its players. To show its practical applicability, we will now report on experiments which are closely related to the examples that have been given in the introduction of Chapter 4. We have already introduced the incomplete-information game Krieg-Tictactoe together with its GDL specification in Figure 4.2 on page 66. We also include the following three incomplete-information games:

- The Card Game has been given as a first specification along with the introduction of the Game Description Language for incomplete-information games [Thi10]. The game starts with the random player dealing one of 8 cards to each of two participating players. In the following betting round, the players decide whether to fold or to present their card to the other, which determines their outcome in dependency of their decision and the value of their cards.

- The game Meier (also known as Mia) involves two players rolling dice in turn (modelled by the random player), betting on their concealed outcome, which has to be higher than the previous bet of the opposite player. Before rolling, a player may decide to end the game by mistrusting the previous bet of the opposite player. He wins if the opposite player was indeed lying, and loses otherwise.

- The famous Monty Hall problem [Ros09] has been formalised in [Thi11a]. Exactly one of three doors hides a price, and a candidate initially chooses one of them. Afterwards, the game master (modelled by the random player) opens one of the remaining doors that do not contain the price, and then allows the candidate to make a final choice between keeping his previous selection or switching to the remaining closed door.

The 1st German Open in General Game Playing[3] has been the first to include a track on incomplete-information games, and we furthermore include all games from this track in our experiments (further information on these games can be found at the mentioned web page).

Property Categories

To each of the aforementioned incomplete-information games, we will attempt proofs for all properties from the following three categories of positive-knowledge formulas.

[3]www.tzi.de/~kissmann/ggp/go-ggp

Knows Terminal With the positive-knowledge formula (4.3) (cf. page 68) we have formulated that when the game has terminated, each player knows that it has terminated. We consider this property separately for each of the players and additionally incorporate the knowledge of a game being *not* terminated. More precisely, for an arbitrary player r of the game besides the random player, we formulate that r knows whether the game has terminated with the positive-knowledge formula

$$K_r \mathit{terminal} \vee K_r \neg \mathit{terminal}.$$

Knows Legals Similar to the previous category, for each of the players r beside the random player, we formulate whether r knows which of his moves are currently legal. Additionally, we include formulas which express whether r knows the legal moves of other players as well. More precisely, for all players r and r' besides the random player, we consider the positive-knowledge formula

$$(\forall A \!:\! \mathit{ADom}(r))\,(K_r \mathit{legal}(r', A) \vee K_r \neg \mathit{legal}(r', A)).$$

Knows Goals As a last category, for all players r and r' beside the random player and the set GV of goal values that occur in the game description, we formulate whether r knows the goal values of r' in all terminal states with the positive-knowledge formula

$$\mathit{terminal} \supset ((\forall G \!:\! GV)\,(K_r \mathit{goal}(r', G) \vee K_r \neg \mathit{goal}(r', G))).$$

Proof Procedure

For each newly considered game, we first attempt to obtain a restrictive state-set generator (cf. Definition 4.28, page 94) by reusing the described procedure on "Functionals" from Section 5.4.1 as a first initialisation step. The respective results for the incomplete-information games that we consider here can all be found in Figure 5.2 at page 123, and the set of proved formulas Ψ from this category will be added to the induction step proof. We then attempt separate proofs on all formulas from the three introduced categories Knows Terminal, Knows Legals, and Knows Goals. Each proof attempt is done according to the following scheme, the proof results can be found in Figure 5.3. Let φ be any formula of the mentioned categories which is to be proved.

Phase 1 Note that φ does not contain any $\bigcirc$, hence according to Proposition 4.8 (Complete Knowledge in the Initial State, cf. page 73), omitting all knowledge operators from φ yields a formula $\mathit{kf}_0(\varphi)$ which is equivalent to φ with respect to entailment in the initial state of the game. Moreover, $\mathit{kf}_0(\varphi)$ is true with respect to any initial game state in any game, which implies that the Base-Case Program $P^{bc}_{\mathit{kf}_0(\varphi)}(G)$ will always be inconsistent (cf. the argumentation below Proposition 4.8). Hence, we can skip this part and directly attempt the induction-step proof with program $P^{is}_{\varphi}(G)$. If it is successful (i.e., if $P^{is}_{\varphi}(G)$ is inconsistent), then the considered formula is valid, and we skip Phase 2. If, on the other hand, $P^{is}_{\varphi,\Psi}(G)$ admits an answer set, the validity of φ is still unknown. In that case, we move to Phase 2.

Phase 2 Following the argumentation in Section 4.8.1, we possibly obtain a stronger result for the Base Case by attempting to find a consistent answer set program $P^{bc}_{\bigcirc^t\varphi}(G)$ for some $t \in \mathbb{N}$. This then allows to conclude that there is a development δ of length t such that $last(\delta) \not\models_\delta \varphi$ by Proposition 4.33 (Generalised Correctness on Single States), and hence that φ is violated by the *reachable* state $last(\delta)$, implying that φ is invalid. We attempt successive Base-Case Proofs on $\bigcirc^t\varphi$ for $t = 1, 2, \ldots$, until one of the following cases arises:

- A time limit of 20 seconds is reached. We stop the process, the validity of the formula is still unknown. For some $t \geq 1$, the last proof attempt for the base case has been on $\bigcirc^t\varphi$, and we indicate this t in Figure 5.3.
- We obtain an answer set for $P^{bc}_{\bigcirc^t\varphi}(G)$ for some $t \geq 1$. Then φ is invalid, and t is again indicated in Figure 5.3.

Results

The proof results are presented in Figure 5.3. We will now motivate some of the results that are obtained for the game Krieg-Tictactoe.

- Formula $\varphi = \bigwedge_{r\in R}(terminal \supset K_r terminal)$ is argued to be invalid in Example 4.7 (cf. page 72) by providing a development δ of length 5 such that $last(\delta) \not\models_\delta \varphi$. For the same development, we have $last(\delta) \not\models K_o terminal \vee K_o\neg terminal$. Figure 5.3 reports that the latter formula can first be disproved in depth $t = 5$, which implies that δ is a violating development of minimal length. Intuitively, this is due to the passed turn information via `sees(R, yourmove)` in the GDL specification of Krieg-Tictactoe in Figure 4.2, which causes each player to know the amount of currently placed pieces after each development of the game. Hence, player o first considers both a non-terminal and a terminal state possible after at least 5 joint moves. This happens for player x after at least 6 moves.

- Formula $(\forall A : ADom(r))\ (K_o legal(x, A) \vee K_o\neg legal(x, A))$ is first disproved in depth $t = 3$, which can be motivated as follows: Since player o has complete information about the initial state ($t = 0$: o knows that x can attempt placing at an arbitrary cell), he also knows that the placing-attempt of x will be successful ($t = 1$: o knows that x can only do noop). Afterwards, the placement of o can either fail ($t = 2$: o knows that x can still only do noop), or succeed ($t = 2$: o knows that x can attempt placing at an arbitrary cell again). Now assume the latter case, then x has two possibilities to a failed placing-attempt, each of which yields a state which allows x to attempt placing at one of the remaining eight cells. Player o does not know which cell has been attempted ($t = 3$: o does *not* know which of the cells can *not* be attempted by x). Hence, there is a reachable state where o does not know the legal moves of x, rendering the respective formula invalid.

- Formula $terminal \supset ((\forall G : GV)\ (K_o goal(x, G) \vee K_o\neg goal(x, G)))$ can only be disproved by a terminal state, which can first arise in depth $t = 5$. This is also the least depth where player o may consider both a non-terminal and a

game	Knows Terminal			Knows Legals			Knows Goals		
backgammon	r	20.2	(?,9)	r of r:	86.5	(?,1)	r of r:	21.7	(?,6)
				r of b:	86.8	(?,1)	r of b:	21.8	(?,6)
	b	20.3	(?,9)	b of r:	86.4	(?,1)	b of r:	21.6	(?,6)
				b of b:	86.6	(?,1)	b of b:	21.0	(?,6)
cardgame	j	0.0	(y)	j of j:	0.5	(y)	j of j:	19.6	(?,25)
				j of r:	0.4	(y)	j of r:	19.6	(?,25)
	r	0.0	(y)	r of j:	0.4	(y)	r of j:	19.6	(?,25)
				r of r:	0.4	(y)	r of r:	19.4	(?,25)
kriegtictactoe	x	1.5	(n,6)	x of x:	0.2	(y)	x of x:	2.7	(n,6)
				x of o:	1.6	(n,4)	x of o:	2.4	(n,6)
	o	1.2	(n,5)	o of x:	0.9	(n,3)	o of x:	1.7	(n,5)
				o of o:	0.2	(y)	o of o:	1.9	(n,5)
kriegTTT_5x5	x	1.4	(n,4)	x of x:	20.3	(?,5)	x of x:	2.2	(n,4)
				x of o:	1.4	(n,1)	x of o:	2.3	(n,4)
	o	1.3	(n,4)	o of x:	1.6	(n,1)	o of x:	1.9	(n,4)
				o of o:	20.3	(?,5)	o of o:	1.9	(n,4)
mastermind	p	19.7	(?,17)	p of p:	120.0	(?,1)	p of p:	21.3	(?,4)
meier	1	0.1	(y)	1 of 1:	24.4	(?,4)	1 of 1:	2.3	(n,6)
				1 of 2:	23.6	(?,4)	1 of 2:	2.3	(n,6)
	2	0.1	(y)	2 of 1:	24.2	(?,4)	2 of 1:	1.1	(n,3)
				2 of 2:	23.6	(?,4)	2 of 2:	1.0	(n,3)
montyhall	c	0.0	(y)	c of c:	0.1	(y)	c of c:	0.3	(n,3)
transit	t	19.5	(?,20)	t of t:	20.1	(?,15)	t of t:	19.5	(?,18)
				t of p:	1.0	(n,2)	t of p:	20.3	(?,19)
	p	20.1	(?,21)	p of t:	1.4	(n,3)	p of t:	19.5	(?,16)
				p of p:	19.5	(?,14)	p of p:	20.1	(?,16)
vis_pacman3p	p	20.8	(?,8)	p of p:	21.6	(?,7)	p of p:	9.7	(y)
				p of b:	7.2	(n,3)	p of b:	32.6	(?,4)
				p of i:	7.2	(n,3)	p of i:	32.7	(?,4)
	b	10.6	(n,5)	b of p:	5.2	(n,2)	b of p:	9.7	(y)
				b of b:	22.3	(?,7)	b of b:	31.9	(?,4)
				b of i:	13.7	(n,5)	b of i:	32.1	(?,4)
	i	13.7	(n,6)	i of p:	5.2	(n,2)	i of p:	9.5	(y)
				i of b:	14.9	(n,5)	i of b:	32.4	(?,4)
				i of i:	24.1	(?,7)	i of i:	39.5	(?,4)

Figure 5.3: Proof times in seconds for a selection of positive-knowledge formulas in several incomplete-information games. The names of players have been shortened to one letter. E.g., “j of r” in cardgame stands for “Jane knows the legal moves (goal values, resp.) of Rick”. The time for result (y) indicates one induction-step proof (Phase 1). The time for results (n, t) and $(?, t)$ indicates the overall time which has been needed for one induction-step proof (Phase 1) and t base-case proofs (Phase 2).

terminal state possible (cf. the argumentation in the first item). In the terminal state, player x finished a line, but this line is not finished in the non-terminal state which is additionally considered true by o. Hence, the goal values for x are necessarily disjoint in these two states, implying that o does not know the actual goal value.

In the version of the Monty Hall problem we apply, the candidate is not informed about the result of his final choice (this has been changed in the competition version). Consequently, he does not know his goal value in each terminal state, and our proof method disproves the respective formula from category Knows Goals. It also reveals that, in the Card Game, each player always knows about terminal states and legal moves of all players. However, this information can only be obtained when adding an additional valid formula stating that betting and dealing cannot take place simultaneously, which can easily be proved with our method beforehand. In this game, the formulas from category Knows Goals are valid, but cannot be proved due to missing information in the state-set generator. Consequently, the stronger base case proof described as Phase 2 in our proof procedure does not succeed in disproving the respective formulas at any depth and hence reaches the time limit.

Note that, according to Section 4.3.2 (View Namings), each formula φ_t in category Knows Terminal has an associated set of views $\mathcal{V}s_{\varphi_t}$ of size 3, whereas each formula φ_l in category Knows Legals is such that $|\mathcal{V}s_{\varphi_l}| = 2 * |ADom(r)| + 1$. This implies that the size of an answer set program which is generated for φ_l exceeds the size of the corresponding answer set program for φ_t by factor $|ADom(r)|$. A similar correspondence with factor $|GV|$ (for the goal values GV) is obtained for each formula φ_g in category Knows Goals. Accordingly, in Figure 5.3, times for proof attempts on formulas φ_l and φ_g are generally higher than those for φ_t. For example, in the game Backgammon, nine iterations on formulas in category Knows Terminal are possible within the time limit, whereas only one can be performed for category Knows Legals (and six for category Knows Goals). The induction step proofs mostly take time in the range of a few seconds only (in Figure 5.3, their time is given directly in case the proof attempt was successful, and can be obtained by subtracting the time limit 20 seconds due to Phase 2 from the given time in case the proof attempt could neither achieve success nor failure), and hence show the efficiency of our method. However, some invariants cannot be proved due to our rather uninformed state-set generator, and further methods to its restriction (e.g. by inclusion of previously-proved positive-knowledge formulas) are required.

5.4.3 Weak Winnability

We will now readdress Section 3.6.1 which presented a method to prove that a player r can weakly win the game at hand, i.e., that there is a development which ends in a terminal state where r obtains 100 points (cf. Definition 2.21). Algorithm 3.1 from page 50 provides a procedure which, if the game is weakly winnable, computes a development with the mentioned properties that is of minimal length. For its implementation, we use the solver IClingo (cf. Section 5.3.2).

Using IClingo to Prove Weak Winnability

In the following we present how to instantiate the programs B, I, and Q from IClingo in order to prove weak winnability. To this end, for a GDL specification G, let $G_0[k]$ be obtained from the temporal extension G_0 of G (cf. Definition 3.7) by replacing each occurrence 0 in the time argument of each atom by the lowercase letter k, and each occurrence of 1 by $k+1$.

- B consists of the initial state encoding $S^{\mathtt{true}}_{init}(0)$ together with the static (and hence not time-extended) clauses from $G_0[k]$ with respect to the optimisation for static predicate symbols introduced in Section 5.2.1.
- I consists of all clauses from $G_0[k]$ which are time-extended and hence not contained in B, together with the following variant of the Action Generator (cf. the clauses (3.8) in Section 3.4.1) for all roles r in the game:

$$\begin{array}{l} 1\{\textbf{does}\,(r, a, \mathtt{k})\,:\, a \in \mathit{ADom}(r)\}1\;\mathtt{:-}\;\;\textbf{not terminal}\,(\mathtt{k}\,).\\ \mathtt{:-}\;\;\textbf{does}\,(r, \mathtt{A}\,, \mathtt{k}\,),\; \textbf{not legal}\,(r, \mathtt{A}\,, \mathtt{k}\,).\\ \mathtt{:-}\;\;\textbf{terminal}\,(\mathtt{k\text{-}1}\,). \end{array} \tag{5.7}$$

- Q encodes that formula $\varphi = \neg(\,\textbf{terminal} \wedge\, \textbf{goal}(r, 100))$ should be false at k (and hence that the current state is terminal and yields 100 points for player r):

$$\begin{array}{rl} & \mathtt{:-\ phi.}\\ \mathtt{phi} & \mathtt{:-}\ \textbf{not}\ \mathtt{win.}\\ \mathtt{win} & \mathtt{:-}\ \textbf{terminal}\mathtt{(k),}\ \textbf{goal}\mathtt{(}r\mathtt{,100,k).} \end{array} \tag{5.8}$$

Correctness Observe that, for any fixed integer t and $\varphi = \neg(\mathit{terminal} \wedge \mathit{goal}(r, 100))$, program $P = B \cup \bigcup_{0 \le i \le t} I[k/i] \cup Q[k/t]$ from IClingo and program $P^{bc}_{\bigcirc^t\varphi}(G)$ are associated in the following way.

- The clauses which are obtained from (5.7) by replacing k with i for each $0 \le i \le t$ only deviate from the action generator (3.8) in that, instead of arbitrarily many previous states, only the currently considered state at depth t is allowed to be terminal. In $P^{bc}_{\bigcirc^t\varphi}(G)$, the clauses $\mathit{Enc}(\bigcirc^t\varphi) \cup \{\ \mathtt{:-}\ \eta(\bigcirc^t\varphi)\,\mathtt{.}\}$ ensure the same property.
- The clauses of $\mathit{Enc}(\bigcirc^t\varphi) \cup \{\ \mathtt{:-}\ \eta(\bigcirc^t\varphi)\,\mathtt{.}\}$ which do not relate to the previously mentioned effect on terminal states exactly correspond to the encoding (5.8).
- The remaining differences are only due to clause optimisations for static predicate symbols with respect to Section 5.2.1 and hence have no influence on satisfiability.

The given arguments motivate that program P admits an answer set if and only if program $P^{bc}_{\bigcirc^t\varphi}(G)$ admits an answer set, and hence that invoking IClingo with the programs B, I, and Q as above provides a faithful implementation of the Weak-Winnability Algorithm 3.1.

Single-Player Game	Solved		Multi-Player Game	Weak Win.	
8puzzle	25.0	(30)	3pttc	0.4	(10)
aipsrovers01	0.9	(10)	backgammon	–	
asteroidsparallel	0.1	(10)	bidding-tictactoe	0.1	(6)
blocksworldparallel	0.1	(3)	breakthrough	2.0	(11)
brain_teaser_extended	0.4	(10)	capture_the_king	8.9	(5)
buttons	0.0	(6)	cardgame	0.1	(2)
chinesecheckers1	35.1	(11)	catcha_mouse	0.2	(12)
circlesolitaire	0.2	(8)	CephalopodMicro	–	
coins	0.1	(4)	checkers	–	
firefighter	0.0	(49)	chinesecheckers6	–	
frogs_and_toads	–		chinookDisj	0.3	(7)
god	25.3	(4)	chomp	0.0	(2)
hanoi	–		connect4	0.1	(7)
hitori	14.7	(7)	connectFourSim	0.3	(7)
incredible	2.0	(13)	dots_and_boxes	–	
knightmove	–		endgame	–	
knightstour	5.4	(30)	knightfight	0.6	(7)
lightsout	57.9	(9)	kriegtictactoe	0.1	(5)
max_knights	–		kriegTTT_5x5	0.4	(4)
maze	0.0	(6)	mastermind	0.2	(2)
mimikry	–		meier	0.1	(3)
oisters_farm	0.0	(6)	montyhall	0.0	(3)
pancakes	0.2	(7)	9BoardTicTacToe	0.1	(5)
peg	–		othello-comp2007	–	
queens	4.3	(10)	pawn_whopping	0.2	(9)
slidingpieces	–		quarto	–	
snake_2008	79.7	(22)	qyshinsu	0.6	(5)
sudoku_simple	–		smallest	0.2	(10)
tpeg	–		tictactoe	0.0	(5)
twisty-passages	2.5	(199)	transit	0.1	(14)
uf20-01.cnf.SAT	0.2	(20)	tttcc4	1.7	(7)
wargame01	23.7	(14)	vis_pacman3p	–	

Figure 5.4: Times in seconds needed to solve a variety of single-player games (left-hand side) and to prove weak winnability for one of the players different from **random** in a variety of multi-player games (right-hand side). Numbers in parentheses denote the length of a (shortest) found development representing a solution.

Experiments

The proof results for weak winnability are shown in Figure 5.4. Since the outcome for one player is often similar among all players in a multi-player game, we exemplarily show the proof result only for the respective first role different from **random** which has been encountered in the GDL specification at hand. In Krieg-Tictactoe, the prover yields a development $S_{init} \xrightarrow{A_0} S_1 \xrightarrow{A_1} S_2 \xrightarrow{A_2} S_3 \xrightarrow{A_3} S_4 \xrightarrow{A_4} S_5$ such that S_5 is terminal and yields 100 points for player x, for example the development δ which has been given in Example 4.7 (cf. page 72). However, the prover does not yield a result for Quarto, as it is to complex to be solved within 100 seconds.

Figure 5.4 shows that our implementation is able to completely solve single-player games and to prove weak winnability in a majority of the presented games. As finding a solution involves a full tree search in the worst case (which is not possible in interesting games), a positive result can of course not generally be obtained, and hence especially causes timeouts for more complex games like Checkers.

5.5 Summary

In this chapter, we showed that our proof method can effectively be applied by general game players and game designers even in a time-restricted setting. We first presented a method to reliably compute overestimations of the set of all actions of a player and the set of all fluents which are needed for the construction of answer set programs in our proof method. Then we reported on several optimisations which can be applied to the generated answer set programs in order to reduce time consumption in the subsequent processing step. We shortly introduced the tools of the answer set solving collection Potassco that we have used to process the generated answer set programs. Finally, we provided results of rich experiments that we conducted for a variety of games, including several properties for knowledge-free formulas and positive-knowledge formulas as well as weak winnability.

Outlook While the implementation of our method makes use of several optimisations to reduce the size of the generated answer set programs (cf. Section 5.2), additional tools from the answer set solving collection Potassco offer potential for further running-time improvements which have not (yet) been utilised. First of all, we have not concentrated on the optimisation of input parameters which are accessible for adjustment in Clasp. The tool *Claspfolio* [GKK$^+$11b] automatically detects useful parameter settings and should certainly be tested in our setting. Another promising tool is *OClingo* [GGKS11]. Based on IClingo (cf. Section 5.3.2), it provides an additional concept for the specification of new information during the solving process. It is worth investigating how this approach can be exploited to add freshly proved formulas to a running proof attempt. Finally, it is promising to experiment with parallelisation, utilising the tool *Claspar* [EGG$^+$09].

We believe that further improvement in efficiency could be achieved by a combination of the grounding process with the solving process of an answer set program, with the goal of achieving a “lazy” grounding which is only done when necessary. For the verification of a property, often only the *existence* of some structure is important rather than certain instances. These instances however may involve many clauses which are unnecessarily grounded beforehand. To our knowledge, no successful Answer Set Solver currently follows this approach.

Chapter 6

Related Work

General Game Playing provides only one out of several formalisms which enable to reason about properties of environments with multiple agents, prominent further examples are the well-known formalisms of Knowledge Representation and Reasoning (a comprehensive survey of several action formalisms can be found in [MM11]), Game Theory (see, e.g., [Ras07, Osb04]), Model Checking (e.g., [BK08]), and Multi-Agent Systems (e.g., [SLB09]). In the following discussion of related work we solely concentrate on verification approaches which have been developed in connection with the Game Description Language, with the exception of Section 6.1.1 on Automated Planning. The chapter is divided into two parts: Section 6.1 is concerned with the discussion of related approaches that are exclusively concentrating on solving single-player games. Section 6.2 then presents related approaches which have been suggested to prove game-specific properties.

6.1 Solving Single-Player Games

We have shown that our approach is able to prove weak winnability and hence to completely solve (smaller) single-player games (cf. Section 5.4.3). In this section, we want to discuss further approaches which solve single-player games. We shortly visit the field of Automated Planning in Section 6.1.1, as it is strongly connected to solving single-player games in General Game Playing and contains some of the roots of our approach. In Section 6.1.2, we summarise approaches which are based on Answer Set Programming.

6.1.1 Automated Planning

The research area of *Automated Planning* is concerned with the development of computer systems which are able to automatically solve single-agent problems in arbitrary domains. The *Planning Domain Definition Language* (PDDL) [McD98, GL05] has become the standard for formalising problem instances, and by now there are about 50 different systems which compete in the International Planning Competition which has first been launched in 1998 [McD00].

General Game Playing can be understood as a generalisation of Automated Planning from single-agent to multi-agent problems. However, the language PDDL is based on the specification of positive and negative effect of actions, whereas negative effects are only implicitly given in the GDL by the negation-as-failure principle (assuming fluents to be false which are not explicitly stated to be true). This poses difficulties on the translation from one language to the other, limiting the direct mutual benefit from new ideas in both research areas. Nevertheless, some results from the planning community could be transferred to General Game Playing. We refer to [Sch11] and [Thi11b] for comprehensive summaries.

Domain Control Knowledge Our language for the specification of game properties has been inspired by work on the domain-independent Planner TLplan [BK00] which has remarkably influenced the planning community. It utilises previously gathered knowledge about the domain at hand to effectively guide search. Knowledge can be formulated using Linear-Time Temporal Logic (see, e.g., [BK08]), where the fluents of a given domain form the basic propositions. Formulas are progressed during search, and a state is neglected as soon as the progressed formula is false. Domain Control Knowledge is usually specified by the designer of a domain or the developer of a planner, but a first approach to *automatically* learn LTL formulas via training examples has recently been introduced [dlRM11]. An overview to further recent developments in this growing field, which now can handle the formulation of rich user preferences among plans, is provided in [BFM11].

Automated Discovery of State Invariants There is a main difference between formulas that express domain control knowledge and state sequence invariants in our setting: the former are used to distinguish "good" from "bad" reachable states and hence are *not* true in all reachable states (otherwise no search restriction could be applied), the latter are utilised to formulate properties that possibly amount to be valid and hence are true in all reachable states. Nevertheless, valid formulas provide valuable information in Planning as well, and several approaches for the automated discovery of state invariants (i.e., knowledge-free properties concerning a single state which hold across all reachable states) have been developed, including [GS00], [Lin04], and [Rin08].

Although the unary operators $\bigcirc$ and K_r are not considered in these approaches, it would be interesting to see whether some ideas can be incorporated in our setting, and to what extend our method can be adapted to discover state invariants for Planning. We mentioned before that the differences of PDDL and GDL do not allow to directly transport results from one to the other language, hence research in this direction is suggested for future work.

6.1.2 Via Answer Set Programming

The first approach that suggests to solve single-player games given in GDL via Answer Set Programming has been given in [Thi09]. Its proposed techniques have provided some of the ground work for the incorporation of the unary operator $\bigcirc$ to the verification method presented in this thesis: an extension of the game description by a time argument (which motivated the temporal GDL extension from Definition 3.7); a program which encodes that each player performs a legal move in each non-terminal state (which motivated the action generator (3.8)); and a result which proves that the extension correctly represents sequences of states (Theorem 3.9 is a generalisation of this result). Its implementation uses an earlier version of Clingo (cf. Section 5.3.1) from 2008 and relies on a given maximal time horizon within which a solution is to be found.

The mentioned approach has also been implemented in the general game player Centurio [MSWS11] as an alternative to Monte Carlo Tree Search [KS06, Cou07] in the case of single-player games. Two variants are described, one which uses estimated upper time bounds with Clingo and a second which uses the incremental solver IClingo (cf. Section 5.3.2). Single-player games have also been solved with Clingo in [GKKS11a]. Here, an unoptimised version is compared to a version which, prior to computing answer sets for a given answer set program, uses a clause optimisation technique similar to the separation of body variables which has been presented in Section 5.2.1.

6.2 Verification of Game Properties

The Game Description Language has soundly been embedded into several formalisms, including the action language $\mathcal{C}+$ ([GLL$^+$04]) in [Thi11c] for the subset of complete-information games, a variant of the Situation Calculus ([McC63]) in [ST11], and Game Theory (e.g., [Ras07, Osb04]) in [Thi11a]. This allows, in principle, to employ implemented reasoning techniques which have been developed for these formalisms to prove certain game properties, when additionally providing an implementation for the automated translation of game descriptions and game properties to the formalism at hand. As the main focus in this work is the engineering of a sound and practically applicable proof method for general game playing rather than a pure feasibility study, we subsequently draw our attention to the discussion of related approaches which

- provide a formal language to formulate a specific class of game properties over the Game Description Language,
- develop a sound method for the formal verification of these properties, and
- report on an implementation that allows to practically verify these properties with respect to an arbitrary given game description.

The following discussion comprises three parts: Section 6.2.1 presents an approach which allows to verify ATL formulas with a model checker. Section 6.2.2 discusses an approach for the verification of state invariants formulated via propositional formulas which uses Answer Set Programming. Finally, Section 6.2.3 provides insight to another model-checking approach which concentrates on game properties over standard epistemic logic.

6.2.1 ATL Formulas via Model Checking

The first approach that addresses the formal property verification over games formulated in the Game Description Language has been given in [vdHRW07b] and extended in [RvW09] and [Rua09]. It allows the specification of properties over the rich language of *Alternating-Time Temporal Logic* (ATL) [AHK02].

Syntax and Semantics With respect to a GDL specification G, ATL formulas φ can be characterised via the grammar

$$\varphi \quad ::= \quad p \quad | \quad \neg\varphi \quad | \quad \varphi \wedge \varphi \quad | \quad \langle\langle B\rangle\rangle \bigcirc \varphi \quad | \quad \langle\langle B\rangle\rangle \Box \varphi \quad | \quad \langle\langle B\rangle\rangle \, \varphi \, \mathcal{U} \, \varphi,$$

where $p \in \mathcal{P}$ is a ground atom of G according to the Sequence Invariants Definition 3.1 and B is a subset of the players of G, also referred to as *coalition*. Ground atoms p are also allowed to be dependent on **does** in this formalism, which however is not important for our considerations and hence neglected here. Coalitions B allow the quantification of paths in the game tree. E.g., $\langle\langle B\rangle\rangle \bigcirc \varphi$ states that coalition B can cooperate to achieve φ in the next state. More formally, formula $\langle\langle B\rangle\rangle \bigcirc \varphi$ is true in a state S if, and only if, there exists a strategy *strat* (cf. Definition 2.21) for each of the players in coalition B such that performing legal action $\mathit{strat}(S)$ yields a successor state which satisfies φ, no matter which legal actions are performed by the players that are not in B. Similarly, $\langle\langle B\rangle\rangle \Box \varphi$ states that coalition B can cooperate to achieve φ in the current state and each of its (possibly non-direct) successors, and $\langle\langle B\rangle\rangle \, \varphi_1 \, \mathcal{U} \, \varphi_2$ formulates that coalition B can cooperate to achieve φ_1 until φ_2 is true. Note that choosing B to be the set of all roles models an overall existential path quantification, and that choosing $B = \emptyset$ models an overall universal path quantification. For example, weak winnability of a game (cf. Definition 2.21) for a player r from the set of all roles R can be expressed as $\langle\langle R\rangle\rangle \, \top \, \mathcal{U} \, (\mathit{terminal} \wedge \mathit{goal}(r, 100))$ (where $\top$ stands for an arbitrary formula that is always true), and playability can be expressed as $\langle\langle\emptyset\rangle\rangle \Box (\neg\mathit{terminal} \supset \bigwedge_{r\in R}(\bigvee_{m\in ADom(r)} \mathit{legal}(r, m)))$.

The formal semantics of ATL formulas is given via Action-based Alternating Transition Systems [vdHRW07a]. A 1-to-1 correspondence between GDL and ATL is established both on the syntactic and the semantic level, which provides the link for the interpretation of ATL formulas over the GDL.

Implementation The method is implemented using the model checker MOCHA [AHM$^+$98]. It understands the modular input language Reactive Modules [AH99] which allows to provide a compact encoding of the game description. The model checker can then be used, in principle, to automatically verify arbitrary game properties formulated in ATL with respect to the initial state of the game. The provided game description is used to create all relevant state sequences which are needed in order to verify a given formula. E.g., the formula for playability, $\langle\langle\emptyset\rangle\rangle \Box (\neg\mathit{terminal} \supset \bigwedge_{r\in R}(\bigvee_{m\in ADom(r)} \mathit{legal}(r, m)))$, is verified by traversing each reachable non-terminal state of the game in order to check whether each player has a legal move.

Comparison

Expressivity The language of ATL allows to formulate a variety of knowledge-free game properties. We have already seen that ATL can express the before-mentioned property of playability. In general, each knowledge-free state sequence invariant φ according to Definition 3.1 can be expressed as $\langle\langle\emptyset\rangle\rangle \Box \varphi'$, where φ' represents φ without additional syntactic constructs such as (finitely-domained) quantifiers. Then $\langle\langle\emptyset\rangle\rangle \Box \varphi'$ holds with respect to the initial state if and only if φ is valid in our formalism. ATL further includes properties which are not expressible via state sequence invariants. E.g., the strong winnability of a player r (cf. Definition 2.21) can be formulated as $\langle\langle\{r\}\rangle\rangle \top \mathcal{U} \ (\mathit{terminal} \wedge \mathit{goal}(r, 100))$.

Not expressible, however, are properties which involve the different perspectives of players in incomplete-information games, e.g. the knowledge of a player concerning his legal and illegal moves (cf. formula (4.4)), which is accounted for in our formalism via the unary knowledge operator K_r.

Completeness Compared to our verification method, the main advantage of the model-checking approach is its completeness: in case a property is true, evoking the model checker with it will eventually report its truth. The model-checking approach does hence not require additional techniques which are necessary with our approach, such as the repetition of certain proof attempts with newly obtained information.

Time Consumption The advantage of completeness, however, also induces the main drawback of the approach: the proof for properties such as strong winnability and all state sequence invariants requires to check the *whole* game tree. An approach which does not abstract from this requirement can hence prove these properties in completely searchable games only. While sufficient time might be available for a game designer who wants to check simpler game descriptions, many of the games from previous general game playing competitions are not completely searchable in reasonable time, and especially in the time-restricted setting of the competition this approach is hence not applicable.

This is also reflected in experiments which have been conducted in the game of Tictactoe in [RvW09] (Tictactoe is completely searchable and hence manageable with the approach). The results for playability, compared to our approach, only moderately reflect the computational overhead which arises due to the full game-tree search. However, the encoding is based on the assumption that properties from the class "Functionals" for Tictactoe (which we introduced in Section 5.4.1) are known in advance, allowing to construct a more compact and hence more efficient encoding of the game description for the model checker. Functionals are in turn state sequence invariants, and a general game player would have to perform a full game-tree analysis in order to reliably find functionals first. Applying the model-checking approach to a broader range of games is hence likely to cause further inefficiencies.

Interestingly, proving weak winnability is significantly slower than with our approach, although induction is of no help in that case. This suggests that there is still space for optimisations which increase overall performance.

6.2.2 State Invariants via Answer Set Programming

The drawbacks of the method presented in Section 6.2.1 motivated an alternative approach, which has been introduced in [ST09] and enriched with further details in [Sch11]. Its main idea is the circumvention of a full game-tree analysis with the proof principle of induction, a concept which has turned out to increase proof efficiency to a level that matches the tight time restrictions of the General Game Playing Competition. Since this induction approach forms the basis of the induction proof method developed in this thesis, it uses some of the methodology which is already familiar to the reader. In the following, we point out the main differences to our extension.

Syntax and Semantics Properties are restricted to make knowledge-free statements about single states of a game. More formally, formulas φ can be formulated using propositional logic according to the grammar

$$\varphi \quad ::= \quad p \quad | \quad \neg\varphi \quad | \quad \varphi \wedge \varphi,$$

where $p \in \mathcal{P}$ is a ground atom of G according to the Sequence Invariants Definition 3.1. For simplicity, we omit some additional syntactic constructs which can also be expressed using the given connectives. The truth of formulas can be evaluated using the semantics for sequence invariants from Definition 3.5, where sequences are required to be of length 0 only and hence collapse into single states. Similarly to our approach, formulas which are successfully verified by induction are proved to be valid and hence true in each reachable state of the game.

Encoding and Implementation As formulas are solely interpreted over states (in contrast to state *sequences* which are required in our extension), no temporal GDL extension is needed. Instead, the GDL specification together with an (informally given) encoding of the property can directly be used to construct an answer set program for both the base case and the induction step of the induction proof. An arbitrary answer set solver can check whether the constructed answer set programs are unsatisfiable, which then implies validity of the property. The practical implementation uses an earlier version of Clingo (cf. Section 5.3.1).

Comparison

Expressivity The difference in expressivity amounts to the missing unary connectives $\bigcirc$ and K_r. Hence, the knowledge of players, e.g. concerning their legal and illegal moves (cf. formula (4.4) at page 68), is not addressable. Furthermore, properties which involve an arbitrary finite amount of time steps, such as the monotonicity of the game (cf. Definition 2.21 and formula (5.5) at page 121) and the persistence of fluents (cf. formulas (5.6)), cannot be formulated. Still expressible are properties such as zero-sum, turn-taking, and playability.

Completeness As our method relies on the same induction principle that is used here, also our drawback of incompleteness in case of an inaccurate state(-set) generator is inherited from this approach. To circumvent this problem, a set of previously proved formulas can be added to restrict the overestimated set of reachable states. Furthermore, an algorithm is proposed which successively attempts proofs on all properties, performing repeated runs on all non-proved formulas whenever a new formula could be proved and hence added to restrict the state space. Our scheme from Section 3.6.3 can be seen as a generalisation of this algorithm which additionally takes care of property sets that can only be proved jointly and incorporates previously obtained information from counter examples.

Time Consumption Since the presented method and our extension are based on the same induction principle and both implemented on top of Fluxplayer, the time consumption on single properties which are expressible in both approaches is similar. This changes with our extension that proves a multitude of similar properties in one run: it often obtains runtimes similar to those for the proof of single properties and hence greatly increases overall performance.

6.2.3 Epistemic Properties via Epistemic Logic

The first approach which is concerned with the development of a method to prove *epistemic* properties of GDL specifications has been given in [RT11a] (and as workshop version in [RT11b]). Its underlying property language is standard epistemic logic [FHMV95].

Syntax and Semantics Formulas φ in standard epistemic logic are given by the grammar

$$\varphi \quad ::= \quad p \quad | \quad \neg\varphi \quad | \quad \varphi \wedge \varphi \quad | \quad K_r\varphi \quad | \quad C_B\varphi,$$

where $p \in \mathcal{P}$ is again a ground atom of G according to the Sequence Invariants Definition 3.1, r is a player, and B is a subset of the players. Corresponding to our setting, $K_r\varphi$ is used to express that player r knows φ. Additionally, $C_B\varphi$ can be used to formulate that φ is *common knowledge* among all players in B, which is true in case each player $r \in B$ knows φ, each player $r \in B$ knows that each player $r' \in B$ knows φ, each player $r \in B$ knows that each player $r' \in B$ knows that each player $r'' \in B$ knows φ, and so on ad infinitum. An example for the conceptional necessity of common knowledge is given with the famous *Muddy Children Puzzle*, which is described, e.g., in [FHMV95].

The work concentrates on two semantics for the interpretation of formulas: a standard semantics via Kripke structures [FHMV95], and a newly introduced semantics via an *epistemic game model*. The latter structure uses indistinguishable game developments to define accessibility relations for players, which are needed to interpret formulas containing knowledge operators K_r and C_B. The epistemic game model provides the basis of our semantics for epistemic sequence invariants in Definition 4.5. The semantics over Kripke structures and the semantics over epistemic game models are proved to be equivalent, which shows that the GDL for incomplete-information games is as expressive as the standard formalism of epistemic logic, even though all players are completely aware of the game rules and hence in particular the initial state.

Implementation The work puts its main focus on the before-mentioned equivalence result. However, an implementation for automated formula proofs is sketched shortly as well. To check if a formula is true with respect to all reachable states at a given depth k, as a first step, an answer set program is constructed such that each answer set corresponds to a development of length k that contains all information concerning the percepts of each player. Tools from Potassco (cf. Section 5.3) are used to compute all answer sets, which are further processed to construct a single Kripke structure. The model checker DEMO [vE07] then decides the truth of the formula with respect to the constructed structure and all game developments of length k.

Comparison

Expressivity Common knowledge involves an infinite mutual nesting of the players knowledge. and hence cannot be expressed as macro over the unary knowledge operator K_r. Consequently, the presented formalism provides additional expressivity via the additional unary operator C_B. In addition, the verification principle does not require the restriction to positive-knowledge formulas that is needed in our approach. On the other hand, operator $\bigcirc$ is not incorporated here, restricting the formulation of properties to single states. Epistemic state sequence invariants *without* $\bigcirc$, however, can in principle be checked by a full game-tree traversal.

Completeness and Time Consumption The presented approach is complete. However, checking validity of invariants such as whether a player r knows his legal and illegal moves in each reachable state, cf. formula (4.4) at page 68, again requires a complete traversal of the game tree and is hence practically applicable in exhaustively searchable games only.

6.3 Summary

We gave an overview of several approaches which deal with the verification of game properties in the setting of General Game Playing. In the first part, we focussed on methods which solve single player games. In addition to approaches which utilise Answer Set Programming in this endeavour, we gave a short account on the distinguished field of Automated Planning which is closely related to General Game Playing. In the second part, we focussed on two verification approaches which apply Model Checkers to perform a complete state-space analysis. We further discussed an induction approach for the verification of state invariants which laid the foundations for our approach.

Chapter 7

Conclusion

In this chapter, we first summarise the main contributions of the thesis in Section 7.1, and then provide directions for possible improvements and further developments which are left for future work in Section 7.2.

7.1 Main Contributions

The following list summarises our main contributions and points to own publications that resulted from this work. Furthermore, we point to the parts of the thesis which use and extend passages from a paper we have submitted to the Artificial Intelligence Journal.

Proof Method for State Sequence Invariants We developed a sound theory to prove rich temporal invariance properties for games formulated over the GDL. To this end, we introduced a simple yet expressive property description language to address game properties which may involve arbitrary finite sequences of game states, and provided a linear-time semantics based on *finite* state sequences. We defined an encoding of a formula to an answer set program and proved that the addition of a temporal GDL extension similar to the one given in [Thi09] and an encoding of a specific state sequence yields an answer set program whose unique answer set correctly indicates whether the formula is entailed by that state sequence. Based on this correspondence, we developed an induction proof theory in the spirit of an earlier method for (non-temporal) state invariants [ST09] which establishes the validity of a formula using Answer Set Programming. We formally proved the soundness of our method. Our implementation shows that properties can efficiently be proved even in complex games. The results are published in [TV10] and summarised in [HMST11].

Proving Multiple Properties At Once General game players typically aim at proving large sets of properties, and invoking a proof system for each property individually may be costly. This motivated an extension of our proof method which is able to prove multiple properties while invoking the system only once. We proved the soundness of our extension and showed that it succeeds in proving at least as many game properties as the original one. The implementation of this extension confirmed a significant performance gain. The results are published in [HT10].

Combination and Extension of the Material The material summarised in the previous two items has been consolidated and significantly extended to a paper in the Artificial Intelligence Journal [HST12]. The extensions include highly detailed versions of the before-mentioned proofs and a comprehensive section on our implementation together with reports on a variety of performed experiments. In the following, we list parts of this thesis which incorporate and extend text passages from this publication.

- Most sections of the Preliminaries Chapter 2 contain passages from the paper in significantly rearranged and extended form. We added further material on Answer Set Programming, Game Theory, game properties, and the General Game Playing execution model.

- The main content of Chapter 3 on our proof method for knowledge-free state sequence invariants consists of unaltered or slightly extended sections from the paper. New passages are the remark on the linear-time encoding of formulas in Section 3.4.2, Section 3.5.2 on the restricted completeness of the proof method, Section 3.5.3 on the sound and complete verification at fixed depth, Section 3.6.1 on solving single-player games, Section 3.6.3 on conjunctive formula proofs, and the discussion in Section 3.7.

- Concerning Chapter 5 on our implementation of the proof method, Section 5.1 on the calculation of domains summarises content of the paper which is now part of a doctoral thesis [Sch11]. Further material from the paper is incorporated nearly unaltered in several sections of the chapter. The newly added passages are Section 5.2.2 on formula encodings with variables, Section 5.3 on the answer set solving collection Potassco, Section 5.4.2 on experiments concerning epistemic state sequence invariants, Section 5.4.3 on experiments concerning weak winnability, and the summary and outlook in Section 5.5.

Proving Positive-Knowledge Formulas In Chapter 4, we developed a sound extension to our induction proof method which allows to prove state sequence invariants which involve positive player-specific knowledge. To this end, we incorporated a unary knowledge operator to our formula syntax, and provided a semantics which provably satisfies the S5 properties. We suggested an alternative semantics based on multiple state sequences, and showed equivalence of both semantics with respect to positive-knowledge formulas. We adapted our proof method to account for positive-knowledge formulas with respect to our alternative semantics, and were able to generalise our soundness result for knowledge-free formulas to this adaption. Moreover, we generalised the restricted completeness result and the result for the correct verification of properties with respect to a fixed depth of the game tree. We showed how to strengthen the base case proof, and sketched further generalisations to incorporate previously proved positive-knowledge formulas and simultaneous proofs for multiple properties. Experiments on games with incomplete information have shown the practicability of the generalised approach. The results have been published in [HT12] after submission of this thesis.

7.2 Future Work

Systematically Discovering Increasingly Complex Properties While we proposed several categories of interesting properties and showed how to efficiently verify which of them are valid, we did not concentrate on ways to find out which properties are actually worth being attempted for verification in the first place. In work such as [Clu07, Sch11, ST07], the discovery of interesting properties is extensively studied, and a key to success are random moves to reachable states with property violations. While random walks cannot formally prove properties, they can formally *reject* them, which is possible in our method only with respect to the initial state. This additional approach can hence help to significantly reduce the set of property candidates we consider for verification. The addition of a general falsification mechanism which operates over an intelligently selected set of reachable states hence forms a useful addition to our method.

The mentioned reduction of property candidates is a first step towards a comprehensive method for the discovery of *all* valid sequence invariants up to some temporal degree. Starting with very simple formulas such as $\mathit{true}(\mathit{cell}(1,1,x))$, the method could try to prove (or reject) increasingly complex formulas. We have given a general scheme for such a method in Section 3.6.3. Besides a generalisation to positive-knowledge formulas, it needs further instantiation with a sophisticated heuristic to select formulas according to their structural complexity or their dependency on other formulas. The heuristic will possibly have to incorporate the GDL clauses and information concerning previous proof attempts in order to select formulas which result in a potentially successful following attempt. In Section 6.1.1, we have already pointed to some approaches which aim for the systematic discovery of sequence invariants in the field of Automated Planning. Especially interesting is work which concerns the discovery of *all* state invariants in some specific planning domains via relaxation of the state space [Lin04], which could yield further ideas for the sketched approach.

Increasing Proof Efficiency In Section 5.5, we summarised tools which can help to decrease the processing time which is needed by an answer set solver. We further presented several methods to reduce the size of the generated answer set programs in Section 5.2. The mentioned optimisations do not modify the answers of our proof method. However, one can possibly further push efficiency by dropping this requirement, e.g. by introducing further incompleteness issues while retaining the soundness of the method. To this end, consider the following two examples.

- All generated induction step programs as well as all base case programs for formulas which contain the temporal operator $\bigcirc$ require the incorporation of all clauses which concern $\mathit{terminal}$, as both the action generator and $\bigcirc$ require this keyword. These clauses are quite complex in some games and can yield a processing speedup when replaced, say, by clauses which define a state to be (pseudo) terminal if one of the players has no legal move (which of course helps only when the replacement is less complex than the original). Since legal actions are often defined independent from $\mathit{terminal}$ in a game description, the provability of some properties is not affected. E.g., persistence formulas in Krieg-Tictactoe such as $\mathit{cell}(1,1,x) \supset \bigcirc \mathit{cell}(1,1,x)$ can still be proved with this abstraction.

- Further optimisation is possible by the omittance of useless induction hypothesis encodings. E.g., the induction step for formulas of the shape $\varphi = \mathit{terminal} \supset \psi$ amounts to find a sequence σ such that $\sigma \models \varphi \wedge \neg \bigcirc \varphi$. Here, the first state of σ has to be non-terminal anyway (due to the negation in front of $\bigcirc$) and hence does *always* satisfy the induction hypothesis φ. Consequently, the induction hypothesis does not provide any information and can be omitted from the proof, which additionally allows to lower the degree of the temporally extended GDL clauses by one. A similar effect arises with persistence formulas.

In general, an abstraction scheme is imaginable which first tries to quickly prove formulas with a considerably simplified GDL clause set, and then increases complexity and hence accuracy of the clauses for unprovable formulas on further iterative proof attempts.

Additional Functionality As mentioned in connection with a related approach to formula verification in Section 6.2.3, *common knowledge* is an interesting additional concept which cannot be handled with our approach but is certainly worthwhile to be investigated. Also interesting is the incorporation of *advisory functionality* to our proof method. E.g., if a newly designed game does not satisfy the properties which are desired by the game designer, the proof system could provide assistance such as "if you alter these clauses as follows, then the property will hold". Similarly, the game player might be interested in getting advice such as "when you perform this action, then your desired property will hold from the successor state on". While the former scenario appears to require a completely different technique, the latter scenario seems realisable by the incorporation of move information to our property language.

Bibliography

[ABW87] Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 2, pages 89–148. Morgan Kaufmann, 1987.

[AH99] Rajeev Alur and Thomas A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15:7–48, 1999.

[AHK02] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.

[AHM + 98] Rajeev Alur, Thomas A. Henzinger, Freddy Y. C. Mang, Shaz Qadeer, Sriram K. Rajamani, and Serdar Tasiran. Mocha: Modularity in model checking. In A. Hu and M. Vardi, editors, *Computer Aided Verification*, volume 1427 of *Lecture Notes in Computer Science*, pages 521–525. Springer Berlin / Heidelberg, 1998.

[Apt97] Krzysztof R. Apt. *From Logic Programming to Prolog*. International Series in Computer Science. Prentice Hall, 1997.

[BdRV01] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*. Cambridge University Press, 2001.

[BFM11] Meghyn Bienvenu, Christian Fritz, and Sheila A. McIlraith. Specifying and computing preferred plans. *Artificial Intelligence*, 175(7–8):1308–1345, 2011.

[BK00] Fahiem Bacchus and Froduald Kabanza. Using temporal logic to express search control knowledge for planning. *Artificial Intelligence*, 116(1–2):123–191, 2000.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[CHH02] Murray Campbell, A. Joseph Hoane, and Feng-Hsiung Hsu. Deep Blue. *Artificial Intelligence*, 134:57–83, 2002.

[Clu07] Jim Clune. Heuristic evaluation functions for general game playing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1134–1139, Vancouver, July 2007. AAAI Press.

[Cou07] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Proceedings of the 5th International Conference on Computers and Games*, pages 72–83, Berlin, Heidelberg, 2007. Springer.

[CSMG09] Evan Cox, Eric Schkufza, Ryan Madsen, and Michael R. Genesereth. Factoring general games using propositional automata. In *Proceedings of the IJCAI Workshop on General Intelligence in Game-Playing Agents (GIGA)*, pages 13–20, Pasadena, 2009.

[dlRM11] Tomas de la Rosa and Sheila A. McIlraith. Learning domain control knowledge for TLPlan and beyond. In *Proceedings of the ICAPS-11 Workshop on Planning and Learning (PAL)*, 2011.

[EGG$^+$09] Enrico Ellguth, Martin Gebser, Markus Gusowski, Roland Kaminski, Benjamin Kaufmann, Stefan Liske, Torsten Schaub, Lars Schneidenbach, and Bettina Schnor. A simple distributed conflict-driven answer set solver. In E. Erdem, F. Lin, and T. Schaub, editors, *Proceedings of the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, volume 5753 of *Lecture Notes in Artificial Intelligence*, pages 490–495. Springer, 2009.

[EL04] Selim T. Erdoğan and Vladimir Lifschitz. Definitions in answer set programming. In V. Lifschitz and I. Niemelä, editors, *Proeceedings of 7th International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 114–126, 2004.

[FB11] Hilmar Finnsson and Yngvi Björnsson. CadiaPlayer: Search-control techniques. *Künstliche Intelligenz*, 25(1):9–16, 2011.

[Fer05] Paolo Ferraris. Answer sets for propositional theories. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 119–131. Springer, 2005.

[FHMV95] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge.* MIT Press, 1995.

[FL05a] Paolo Ferraris and Vladimir Lifschitz. Mathematical foundations of answer set programming. In *We Will Show Them! Essays in Honour of Dov Gabbay*, pages 615–664. King's College Publications, 2005.

[FL05b] Paolo Ferraris and Vladimir Lifschitz. Weight constraints as nested expressions. In *Theory and Practice of Logic Programming*, volume 5, pages 45–74, 2005.

[Gel08] Michael Gelfond. Answer sets. In F. van Harmelen, V. Lifschitz, and B. Porter, editors, *Handbook of Knowledge Representation*, pages 285–316. Elsevier, 2008.

[GGKS11] Martin Gebser, Torsten Grote, Roland Kaminski, and Torsten Schaub. Reactive answer set programming. In J. Delgrande and W. Faber, editors, *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*, volume 6645 of *Lecture Notes in Artificial Intelligence*, pages 54–66. Springer, 2011.

[GKK$^+$08] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Sven Thiele. Engineering an incremental ASP solver. In M. G. de la Banda and E. Pontelli, editors, *Proceedings of the Twenty-fourth International Conference on Logic Programming (ICLP'08)*, volume 5366 of *Lecture Notes in Computer Science*, pages 190–205. Springer, 2008.

[GKK$^+$11a] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Marius Schneider. Potassco: The Potsdam answer set solving collection. *AI Communications*, 24(2):105–124, 2011.

[GKK$^+$11b] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Torsten Schaub, Marius Schneider, and Stefan Ziller. A portfolio solver for answer set programming: Preliminary report. In J. Delgrande and W. Faber, editors, *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*, volume 6645 of *Lecture Notes in Artificial Intelligence*, pages 352–357. Springer, 2011.

[GKKS09] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. On the implementation of weight constraint rules in conflict-driven ASP solvers. In P. Hill and D. Warren, editors, *Proceedings of the Twenty-fifth International Conference on Logic Programming (ICLP'09)*, volume 5649 of *Lecture Notes in Computer Science*, pages 250–264. Springer, 2009.

[GKKS11a] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Challenges in answer set solving. In M. Balduccini and T. Son, editors, *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays in Honor of Michael Gelfond*, volume 6565, pages 74–90. Springer, 2011.

[GKKS11b] Martin Gebser, Roland Kaminski, Arne König, and Torsten Schaub. Advances in *gringo* series 3. In J. Delgrande and W. Faber, editors, *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*, volume 6645 of *Lecture Notes in Artificial Intelligence*, pages 345–351. Springer, 2011.

[GKNS07] Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. Conflict-driven answer set solving. In M. Veloso, editor, *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 386–392. AAAI Press/The MIT Press, 2007.

[GL88] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Proceedings of the International Joint Conference and Symposium on Logic Programming (IJCSLP)*, pages 1070–1080, Seattle, 1988. MIT Press.

[GL91] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–386, 1991.

[GL05] Alfonso Gerevini and Derek Long. Plan constraints and preferences in PDDL3 — the language of the fifth international planning competition. Technical report, University of Brescia, 2005.

[GLL+04] Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner. Nonmonotonic causal theories. *Artificial Intelligence*, 153(1–2):49–104, 2004.

[GLP05] Michael Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the AAAI competition. *AI Magazine*, 26(2):62–72, 2005.

[GS00] Alfonso Gerevini and Lenhart Schubert. Discovering state constraints in discoplan: Some new results. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-2000)*, pages 761–767. MIT Press, 2000.

[HMST11] Sebastian Haufe, Daniel Michulke, Stephan Schiffel, and Michael Thielscher. Knowledge-based general game playing. *Künstliche Intelligenz*, 25(1):25–33, 2011.

[HST12] Sebastian Haufe, Stephan Schiffel, and Michael Thielscher. Automated verification of state sequence invariants in general game playing. *Artificial Intelligence Journal*, 187–188:1–30, 2012.

[HT10] Sebastian Haufe and Michael Thielscher. Pushing the envelope: General game players prove theorems. In J. Li, editor, *Proceedings of the Australasian Joint Conference on Artificial Intelligence*, volume 6464 of *LNCS*, pages 1–10, Adelaide, December 2010. Springer.

[HT12] Sebastian Haufe and Michael Thielscher. Automated verification of epistemic properties for general game playing. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 339–349, Rome, June 2012.

[IK94] Shigeki Iwata and Takumi Kasai. The Othello game on an n*n board is PSPACE-complete. *Theoretical Computer Science*, 123(2):329–340, 1994.

[Kai07] David M. Kaiser. Automatic feature extraction for autonomous general game playing agents. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, number 93 in AAMAS '07, pages 1–7. ACM, 2007.

[KDS06] Gregory Kuhlmann, Kurt Dresner, and Peter Stone. Automatic heuristic construction in a complete general game player. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1457–1462, Boston, July 2006. AAAI Press.

[KE11] Peter Kissmann and Stefan Edelkamp. Gamer, a general game playing agent. *Künstliche Intelligenz*, 25(1):49–52, 2011.

[Kis03] Zachary Kissel. Associative memory and the board game Quarto. *Crossroads. The ACM Magazine for Students*, 10(2), December 2003.

[KM08] Fred Kröger and Stephan Merz. *Temporal Logic and State Systems.* Springer, 2008.

[KP97] Daphne Koller and Avi Pfeffer. Representations and solutions for game-theoretic problems. *Artificial Intelligence*, 94(1):167–215, 1997.

[KS06] Levente Kocsis and Csaba Szepesvri. Bandit based monte-carlo planning. In *ECML-06. Number 4212 in LNCS*, pages 282–293. Springer, 2006.

[KSS11] Mesut Kirci, Nathan Sturtevant, and Jonathan Schaeffer. A ggp feature learning algorithm. *Künstliche Intelligenz*, 25(1):35–42, 2011.

[LHH$^+$06] Nathaniel Love, Timothy Hinrichs, David Haley, Eric Schkufza, and Michael Genesereth. General Game Playing: Game Description Language Specification. Technical Report LG–2006–01, Stanford Logic Group, Computer Science Department, Stanford University, 353 Serra Mall, Stanford, CA 94305, 2006.

[Lif96] Vladimir Lifschitz. Foundations of logic programming. In *Principles of Knowledge Representation*, pages 23–37. MIT Press, 1996.

[Lin04] Fangzhen Lin. Discovering state invariants. In *Principles of Knowledge Representation and Reasoning*, pages 536–544, 2004.

[Llo87] John W. Lloyd. *Foundations of Logic Programming.* Series Symbolic Computation. Springer, second, extended edition, 1987.

[LT86] John W. Lloyd and Rodney W. Topor. A basis for deductive database systems II. *Journal of Logic Programming*, 3(1):55–67, 1986.

[LT94] Vladimir Lifschitz and Hudson Turner. Splitting a logic program. In *Principles of Knowledge Representation*, pages 23–37. MIT Press, 1994.

[LTT99] Vladimir Lifschitz, Lappoon R. Tang, and Hudson Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25:369–389, 1999.

[MC11] Jean Méhat and Tristan Cazenave. A parallel general game player. *Künstliche Intelligenz*, 25(1):43–47, 2011.

[McC63] John McCarthy. *Situations and Actions and Causal Laws.* Stanford Artificial Intelligence Project, Memo 2, Stanford University, CA, 1963.

[McD98] Drew McDermott. PDDL — the Planning Domain Definition Language. Technical Report TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[McD00] Drew McDermott. The 1998 ai planning systems competition. *AI Magazine*, 21(2):35–55, 2000.

[MM11] Leora Morgenstern and Sheila A. McIlraith. John McCarthy's legacy. *Artificial Intelligence*, 175(1):1–24, January 2011.

[MS11] Daniel Michulke and Stephan Schiffel. Distance features for general game playing. In *Proceedings of the IJCAI Workshop on General Intelligence in Game-Playing Agents (GIGA)*, pages 7–14, Barcelona, 2011.

[MSWS11] Maximilian Möller, Marius Schneider, Martin Wegner, and Torsten Schaub. Centurio, a general game player: Parallel, java- and asp-based. *Künstliche Intelligenz*, 25(1):17–24, 2011.

[MT99] Victor W. Marek and Miroslaw Truszczynski. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer, 1999.

[MT09] Daniel Michulke and Michael Thielscher. Neural networks for state evaluation in general game playing. In W. Buntine, M. Grobelnik, D. Mladenic, and J. Shawe-Taylor, editors, *Proceedings of the European Conference on Machine Learning (EMCL)*, volume 5803 of *LNCS*, pages 95–110, Bled, Slovenia, September 2009. Springer.

[Nie99] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25:241–273, 1999.

[NSS99] Ilkka Niemelä, Patrik Simons, and Timo Soininen. Stable model semantics of weight constraint rules. In *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'99), volume 1730 of Lecture*, pages 317–331. Springer. LNAI, 1999.

[Osb04] Martin J. Osborne. *An Introduction to Game Theory.* Oxford University Press, 2004.

[Pel93] Barney Pell. *Strategy Generation and Evaluation for Meta-Game Playing.* PhD thesis, Trinity College, University of Cambridge, 1993.

[Pit68] Jacques Pitrat. Realization of a general game playing program. In A. Morrell, editor, *Proceedings of IFIP Congress*, pages 1570–1574, Edinburgh, August 1968.

[Pri94] David Pritchard. *The Encyclopedia of Chess Variants.* Godalming, 1994.

[Ras07] Eric Rasmusen. *Games and Information: an Introduction to Game Theory.* Blackwell, 4th edition, 2007.

[Rin08] Jussi Rintanen. Regression for classical and nondeterministic planning. In *Proceeding of the 2008 conference on ECAI 2008: 18th European Conference on Artificial Intelligence*, pages 568–572. IOS Press, 2008.

[RN03] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, second edition, 2003.

[Ros09] Jason Rosenhouse. *The Monty Hall Problem.* Oxford University Press, 2009.

[RT11a] Ji Ruan and Michael Thielscher. The epistemic logic behind the game description language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 840–845, San Francisco, August 2011. AAAI Press.

[RT11b] Ji Ruan and Michael Thielscher. On the comparative expressiveness of epistemic models and GDL-II. In *Proceedings of the IJCAI Workshop on General Intelligence in Game-Playing Agents (GIGA)*, pages 53–60, Barcelona, 2011.

[Rua09] Ji Ruan. *Reasoning about Time, Action and Knowledge in Multi-Agent Systems.* PhD thesis, University of Liverpool, 2009.

[RvW09] Ji Ruan, Wiebe van der Hoek, and Michael Wooldridge. Verification of games in the game description language. *Journal of Logic and Computation*, 19(6):1127–1156, 2009.

[RW05] Stuart Russell and Jason Wolfe. Efficient belief-state AND-OR search with application to kriegspiel. In L. Kaelbling and A. Saffiotti, editors, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 278–285, Edinburgh, UK, August 2005.

[SBB$^+$07] Jonathan Schaeffer, Neil Burch, Yngvi Björnsson, Akihiro Kishimoto, Martin Müller, Robert Lake, Paul Lu, and Steve Sutphen. Checkers is solved. *Science*, 317:1518–1522, 2007.

[Sch00] Jonathan Schaeffer. The games computers (and people) play. *Advances in Computers*, 52:190–268, 2000.

[Sch10] Stephan Schiffel. Symmetry detection in general game playing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 980–985, Atlanta, July 2010. AAAI Press.

[Sch11] Stephan Schiffel. *Knowledge-Based General Game Playing.* PhD thesis, Technische Universität Dresden, 2011.

[SLB09] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations.* Cambridge University Press, 2009.

[ST07] Stephan Schiffel and Michael Thielscher. Fluxplayer: A successful general game player. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1191–1196, Vancouver, July 2007. AAAI Press.

[ST09] Stephan Schiffel and Michael Thielscher. Automated theorem proving for general game playing. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 911–916, Pasadena, July 2009.

[ST10] Stephan Schiffel and Michael Thielscher. A multiagent semantics for the game description language. In J. Filipe, A. Fred, and B. Sharp, editors, *Agents and Artificial Intelligence*, volume 67 of *Communications in Computer and Information Science*, pages 44–55. Springer, 2010.

[ST11] Stephan Schiffel and Michael Thielscher. Reasoning about general games described in GDL-II. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 846–851, San Francisco, August 2011. AAAI Press.

[Thi09] Michael Thielscher. Answer set programming for single-player games in general game playing. In P. Hill and D. Warren, editors, *Proceedings of the International Conference on Logic Programming (ICLP)*, volume 5649 of *LNCS*, pages 327–341, Pasadena, July 2009. Springer.

[Thi10] Michael Thielscher. A general game description language for incomplete information games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 994–999, Atlanta, July 2010.

[Thi11a] Michael Thielscher. The general game playing description language is universal. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1107–1112, Barcelona, July 2011. AAAI Press.

[Thi11b] Michael Thielscher. General game playing in AI research and education. In J. Bach and S. Edelkamp, editors, *Proceedings of the German Annual Conference on Artificial Intelligence (KI)*, volume 7006 of *LNAI*, pages 26–37, Berlin, Germany, October 2011. Springer.

[Thi11c] Michael Thielscher. Translating general game descriptions into an action language. In M. Balduccini and T. Son, editors, *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays in Honor of Michael Gelfond*, volume 6565 of *LNAI*, pages 300–314. Springer, 2011.

[TV10] Michael Thielscher and Sebastian Voigt. A temporal proof system for general game playing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1000–1005, Atlanta, July 2010. AAAI Press.

[vdHRW07a] Wiebe van der Hoek, Mark Roberts, and Michael Wooldridge. Social laws in alternating time: Effectiveness, feasibility, and synthesis. In *Synthese*, volume 156, pages 1–19, 2007.

[vdHRW07b] Wiebe van der Hoek, Ji Ruan, and Michael Wooldridge. Strategy logics and the game description language. In *Proceedings of the Workshop on Logic, Rationality and Interaction*, Bejing, August 2007.

[vE07] Jan van Eijck. DEMO — a demo of epistemic modelling. In J. van Benthem, D. Gabbay, and B. Löwe, editors, *Interactive Logic — Proceedings of the 7th Augustus de Morgan Workshop*, Texts in Logic and Games 1, pages 305–363, 2007.

[ZST09] Dengji Zhao, Stephan Schiffel, and Michael Thielscher. Decomposition of multi-player games. In A. Nicholson and X. Li, editors, *Proceedings of the Australasian Joint Conference on Artificial Intelligence*, volume 5866 of *LNCS*, pages 475–484, Melbourne, December 2009. Springer.