INFINITE STATES VERIFICATION IN GAME-THEORETIC LOGICS

SLAWOMIR KMIEC

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

MASTER OF COMPUTER SCIENCE

GRADUATE PROGRAM IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO
MAY 2013

INFINITE STATES VERIFICATION IN GAME-THEORETIC LOGICS

by **Slawomir Kmiec**

a thesis submitted to the Faculty of Graduate Studies of York University in partial fulfilment of the requirements for the degree of

MASTER OF COMPUTER SCIENCE

INFINITE STATES VERIFICATION IN GAME-THEORETIC LOGICS

by **Slawomir Kmiec**

Abstract

Many practical problems where the environment is not in the system's control such as service orchestration and contingent and multi-agent planning can be modelled in game-theoretic logics. This thesis demonstrates that the verification techniques based on regression and fixpoint approximation introduced in De Giacomo, Lespérance and Pearce [DLP10] do work on several game-theoretic problems. De Giacomo, Lespérance and Pearce [DLP10] emphasize that their study is essentially theoretical and call for complementing their work with experimental studies to understand whether these techniques are effective in practical cases. Several example problems with varying properties have been developed and, although not exhaustive nor complete, our results nevertheless demonstrate that the techniques work on some problems. Our results show that the methods introduced in [DLP10] work for infinite domains where very few verification methods are available and allow reasoning about a wide range of game problems. Our examples also demonstrate the use of a rich language for specifying temporal properties proposed in [DLP10]. While classical model checking is well known and utilized, it is mostly restricted to finite-state models. A important aspect of the work is the demonstration of the use and effectiveness of characteristic graphs (Claßen and Lakemeyer [CL08]) in verifying properties of games in infinite domains. A special-purpose programming language GameGolog proposed in De Giacomo, Lespérance and Pearce [DLP10] allows such game-theoretic systems to be specified procedurally at a high-level of abstraction. We show its practicality to model game structures in a convenient way that combines declarative and procedural elements. We provided examples to show the verification of GameGolog specifications using characteristic graphs. This thesis also proposes a refinement to the formalism in [DLP10] to incorporate action constraints as a mechanism to incorporate user strategies and for the modeller to supply heuristic guidance in temporal property verification. It also presents an implementation of evaluation-based fixpoint verifier that handles Situation Calculus game structures, as well as GameGolog specifications, for temporal property verification in the initial or a given situation. The verifier supports player action constraints.

Acknowledgements

The author wishes to thank several people. It would not have been possible to write this thesis without the help and support of the kind people around me, to only some of whom it is possible to give particular mention here. I would like to thank my wife, Beata, for her love, kindness, support and great patience at all times she has shown during the past years it has taken me to finalize this thesis. This thesis would not have been possible without the help, support and patience of my supervisor, Prof. Yves Lespérance, not to mention his advice and unsurpassed knowledge of the field. Besides my supervisor, I would like to thank the rest of my thesis committee: Prof. Zbigniew Stachniak, Prof. Jonathan Ostroff, and Prof. Marin Litoiu, for their encouragement, insightful comments, and hard questions. The good advice and support of the members of the examination committee has been invaluable, for which I am extremely grateful.

Table of Contents

1 Introduction

1.1 Overview

Many types of problems, from contingent and multi-agent planning to process/service orchestration, can be viewed as games, where one or more agents try to ensure that certain objectives (or in general, properties) hold no matter how the environment and other agents behave. Often a strategy can be determined from verification that allows the agents to ensure that the property holds. Our framework supports reasoning about game structures – any type of multi-agent problem that requires strategic thinking. The purpose of the work put forward by this thesis is to contribute to the advancement of knowledge representation and reasoning in the area of reasoning about action and processes, and to develop techniques for process verification and synthesis. Many practical problems such as service orchestration and contingent and multi-agent planning can be closely modelled in game-theoretic logics. In such systems, some agents cannot be controlled and thus the systems can naturally be views as games. Here game-theoretic logics allow specifying properties to verify on the system. A special-purpose programming language GameGolog (De Giacomo, Lespérance and Pearce [DLP10]) has been proposed to allow such systems to be specified procedurally at a high-level of abstraction. GameGolog programs can be described by characteristic graphs (Claßen and Lakemeyer [CL08]), which are used to compactly represent all the possible configurations that a GameGolog program may visit during its execution. Such action theories, their GameGolog programs and their characteristic graphs can be examined for several high-level properties that can be defined based on alternating-time temporal logic (ATL) (Alur, Henzinger and Kupferman [AHK02]). While classical model checking is well known and utilized, it is mostly restricted to finite-state models and affected by how large the state space is. Currently there is very little work dealing with infinite-state domains (Claßen and Lakemeyer [CL08]). The approach proposed by this thesis allows and supports infinite-state domains. The objective of the this thesis is to show that several new techniques based on regression and fixpoint approximation (De Giacomo, Lespérance and Pearce [DLP10]) do work on non-trivial game theories with infinite state space. Also verification techniques based on evaluation and fixpoint approximation are implemented and evaluated with respect to their effectiveness in property verification for such game theoretic logics.

1.2 Thesis Focus

The work described by this thesis is based on and focuses on the framework of De Giacomo, Lespérance and Pearce [DLP10]. One of the main points of this thesis is to examine the viability symbolic computation techniques in verifying alternating-time μ-calculus temporal properties. Some attention is devoted to the special-purpose programming language GameGolog (De Giacomo, Lespérance and Pearce [DLP10]) that allows game-like environments to be specified procedurally at a high-level of abstraction. GameGolog semantics is also employed to axiomatise such environments in a formalism in which typical temporal properties can also be expressed. Such GameGolog constructs and their characteristic graphs are examined for several high-level properties that can be defined based on alternating-time temporal logic (ATL) (Alur, Henzinger and Kupferman [AHK02]), also for infinite-state domains. The objective is to show that new techniques based on regression and fixpoint approximation proposed by De Giacomo, Lespérance and Pearce [DLP10] do work on non-trivial game theories with infinite-state space. Also verification techniques based on evaluation and fixpoint approximation are implemented and evaluated with respect to their effectiveness in property verification for such game theoretic logics. Other implementation work is performed and tests are done to examine the viability of using GameGolog, and to fully or partially specify strategies and to see how temporal property verification results change if opponent strategies are known.

In particular this thesis puts the focus on:

- the process of modelling and verifying game structures as proposed in [DLP10],
- using an ATL-like language for specifying properties to be verified as proposed in [DLP10],
- the use of GameGolog to conveniently specify and verify game structures as proposed in [DLP10],
- the symbolic-computation verification techniques for alternating-time μ-Calculsus as proposed in [DLP10], and
- evaluation-based verification techniques implemented in Prolog.

1.3 Thesis Motivation

De Giacomo, Lespérance and Pearce [DLP10] emphasize that their study is essentially theoretical and they acknowledge that completeness guarantees will only be available for very specific cases (finite states or structures that allow quantifier elimination). They called for complementing their work with experimental studies to understand whether these techniques, especially those based on the labelling of characteristic graphs, are effective in practical cases. This thesis shows that concrete problems can be specified in the framework and that fixpoint convergence in a finite number of steps based on the techniques of [DLP10] does occur in many cases with or without a specification of the initial state. It also describes cases where fixpoint approximation does not converge in a finite number of steps. The thesis shows the viability and practicality of the techniques. For infinite-state verification one cannot expect completeness in the general case. The additional goal is to prove that GameGolog is of value and works for some real-world problems.

1.4 Thesis Contributions

The work presented in this thesis is intended to be a demonstration that the techniques introduced in De Giacomo, Lespérance and Pearce [DLP10] do work on several problems with infinite state space. Several example problems with varying properties have been developed and, although not exhaustive nor complete, our results nevertheless show that the techniques work on some problems. A important point of the work is the use of characteristic graphs (Claßen and Lakemeyer [CL08]) in verifying properties of games which can help in dealing with complex infinite domains. Additionally this thesis proposes a refinement to the formalism in [DLP10] to incorporate player action/strategy constraints and to subsequently to perform temporal property verification under such constraints. Also, an implementation of evaluation-based verifier has been conducted and employed to verify some temporal properties on some examples.

The key contributions can be listed as:

- demonstration that the verification techniques based on regression and fixpoint approximation introduced in [DLP10] do work on several interesting problems; we provide examples to show verification of Situation Calculus game structures (Chapter 3); the sample domains that we developed involve infinite domains, and our results show that the methods introduced in [DLP10] work for infinite domains, where very few verification methods are available, and allow reasoning about a wide range of game problems; our examples also demonstrate the use of a rich language for specifying temporal properties proposed in [DLP10]; we also found problems where the technique does not work and we extend the method to consider some additional knowledge about the initial situation to make the method work;
- demonstration of the use and effectiveness of characteristic graphs (Claßen and Lakemeyer [CL08]) in verifying properties of games in infinite domains; we also show the practicality of the GameGolog programming language to model game structures in a convenient way that combines declarative and procedural elements; we provide an example to show verification of GameGolog specifications using characteristic graphs (Chapter 4);
- an implementation of an evaluation-based fixpoint verifier that handles Situation Calculus game structures as well as GameGolog specifications for temporal property verification in the initial or a given situation (Chapter 5); the system is tested on some sample game domains.
- a refinement to the formalism in [DLP10] to incorporate action constraints as a mechanism to incorpo-

rate user strategies and for the modeller to supply heuristic guidance in temporal property verification; we provide a implementation of the framework by modifying our evaluation-based verifier to incorporate player action constraints (Chapter 5).

Note that a paper summarizing the results of chapter 3 has been accepted for presentation at Tenth International Workshop on Nonmonotonic Reasoning, Action and Change (NRAC 2013) in Beijing.

1.5 Outline

Chapter 2 provides the necessary background for understanding the research proposed by this thesis. It explains how problems that require reasoning can be formalized using the Situation Calculus language and Basic Action Theories. Next it talks about the most common tasks in reasoning about action such as planning, projection and legality testing. It discusses complex actions and higher-level languages that can represent such complex actions and model such environments in a procedural way. Following that, characteristic graphs are introduced. Next some formalisms are introduced that allow us to model game-type environments/problems, and also to express temporal properties that a game domain may have. Next, some methods used in this thesis for verifying such temporal properties over game structures are explained. This section also provides a brief review and discussion of existing work that is related to the research presented in this thesis. It explains the similarities and highlights the differences to the methods researched here. Finally existing verification methods are discussed that are alternative to the ones presented in this thesis.

Chapter 3, 4 and 5 present the actual research results of this thesis. The work is built around the formalisms and methods presented in De Giacomo, Lespérance and Pearce [DLP10]. It starts off by providing a description of some representative game-theoretic problems with various characteristics. These problems and some properties of interest are specified in the proposed formalisms. Then the verification techniques from [DLP10] are employed and their feasibility examined. Next, a characteristic graph-based technique is used on one of the examples and the results compared with the symbolic method. After that an evaluation-based technique is explained and its Prolog implementation is presented. Additionally some extensions to the mentioned techniques and formalisms are proposed to allow partial or full specification of agent strategies. These extensions also allow us to examine the influence of possible existing strategies on the temporal properties of game-theoretic problems and it can make the verification easier. This thesis also provides an implementation and experimental verification of the proposed extensions.

Chapter 6 concludes by summarizing the work put forward by this thesis. It evaluates the results against the initial objectives and compares our approach to others. I also presents some interesting ideas and problems that could be researched in the future.

The thesis uses a consistent line of examples to illustrate the various concepts and techniques. The appendix contains the code of the implementations developed to illustrate and to verify the researched techniques. It also shows detailed results of the tests.

2 Background

This section provides the necessary background to set the context of the main research proposed by this thesis. First it explains how problems that require reasoning can be formalized via the Situation Calculus language and Basic Action Theories. Next it briefly talks about most common tasks in reasoning about actions such as planning, projection and legality. This leads to introduction of complex actions and higher-level languages that can represent such complex actions and model such environments in procedural ways. Following that, characteristic graphs are introduced as a method to compactly represent the states that a high-level program may visit during its execution. Next some formalisms are introduced that allow to mode game-specific problems, and also some formalisms are introduced to express temporal properties that a game-specific environment may have. Finally some methods used in this thesis of verifying such temporal properties over game structures are explained.

Here is where the background for understanding the various chapters can be found. Chapter 3 requires sections 2.5.2, 2.6.3, and 2.7.2. Chapter 4 requires sections 2.4, 2.5.3, 2.6.4, and 2.7.3. Chapter 5 requires sections 2.5.3, 2.6.4, and 2.7.3.

The key sections of this chapter are as follows. Section 2.1 provides overall background on representing dynamically changing worlds; it is needed especially in section 2.5. Section 2.2 explains the task of reasoning about action and this is needed for understanding strategies.; it is needed especially in section 2.6. Section 2.3 explains the reasons and the semantics for the GameGolog programming language which is used in chapter 4 and 5. Section 2.4 introduces characteristic graphs which are used in chapter 4. Section 2.5 explains how axiomatization of game structures is done and provides a few examples. Section 2.6 explains how temporal properties can be expressed and provides a few examples; it is used in chapter 3. Section 2.7 explains how the verification of temporal properties is performed in this thesis; it is used in chapter 3 and 4. Section 2.8 briefly presents other main verification methods; it is needed for chapter 5 and the conclusions in chapter 6.

2.1 Representing Dynamic Worlds

In order to reason about actions and dynamically changing worlds, a language was needed that would allow to express such worlds and deal with actions performed in them. The situation calculus is a logical language specifically designed for representing and reasoning about dynamically changing worlds. The situation calculus id explained in the next subsection. Within the situation calculus, one can formulate action theories that describe how the world changes as the result of the available actions. A specialization of action theories called basic action theories has been proposed in Pirri and Reiter [PR99] and Reiter [Rei01]. Basic action theories is a formalism that builds on the situation calculus to allow defining the laws describing the conditions and effects of actions in the modelled worlds, and thus to reason about worlds that change as a result of actions. Basic action theories are defined at the end of this section.

2.1.1 The Situation Calculus

The Situation Calculus (SitCalc) is a formalism for representing dynamically changing worlds in which all changes are the result of named actions. Although it is a dialect of First-Order Logic, the situation calculus is a second-order logic formalism. As such it is incomplete - there is no axiomatization of second-order logic that will yield all the valid second-order sentences i.e. the valid sentences of second-order logic are not recursively enumerable. Despite this it is sufficient and convenient for modelling problems that deal with actions and planning.

There are two distinguished sorts of terms in the situation calculus: **actions** and **situations**.

All changes to the world are the result of actions. Actions are terms in the language Action terms are denoted by α with possible subscripts to differentiate different action terms. Action variables are denoted by lower case letters a with possible subscripts to differentiate different action variables. Action types, i.e. actions that require a parameter, are denoted by upper case letters A with possible subscripts to differentiate different action types. For example $pickup(R, X)$ could be an action of a robot R picking an object X, $walk(R, Y)$ could be an action of a robot R walking to an object Y.

Situations represent possible world histories. Situations are terms in the language. The distinguished constant S_0 denotes the initial situation where no action has been performed. The distinguished function symbol *do* is used to build sequences of actions such that $do(a, s)$ denotes the successor situation which resulted from performing action a in situation s.

Other elements in the language are: **fluents** and **formulas**.

Fluents are predicates or functions whose values may vary from situation to situation. Fluents represent properties of the world in the current situation. Fluents are denoted by symbols that take a situation term as their last argument. A distinguished predicate symbol *Poss(a,s)* is used to state that an action a may be performed in a situation s.

Predicate logic formulas can be constructed from the elements of the language. Situation-suppressed formulas are formulas in the language where all the situation arguments in fluents have been suppressed. This is useful when reasoning about actions. The situation argument can be easily restored by adding it to all fluents. If ϕ is a situation-suppressed formula then $\phi[s]$ denotes situation calculus formula obtained from ϕ by restoring situation argument s into all fluents in ϕ. Situation calculus formula uniform in s is a situation calculus formula where s is the only situation term appearing in such formula i.e. there is no $do(\ldots)$ construct in the subject formula.

The formalism includes a set of domain independent **foundational axioms** Σ of:

- unique name axioms for situations

  $do(a_1, s_1) = do(a_2, s_2) \supset a_1 = a_2 \wedge s_1 = s_2$

- minimum set axioms

  $(\forall P).P(S_0) \wedge (\forall a, s)[P(s) \supset P(do(a, s))] \supset (\forall s)P(s)$

  it describes the second-order induction and has the effect of limiting the sort situation to the smallest set containing S_0, and closed under the application of the function do an action and a situation

- initial situation constant S_0

  $S_0 \in \Sigma$

- successor situation axioms via distinguished operator *do*

  $\neg s \sqsubset S_0$

  $s \sqsubset do(a, s') \equiv s \sqsubset s' \vee s = s'$

  where $\sqsubset$ is a binary predicate symbol defining an ordering relation on situations [Rei01]

  i.e. $do(a, s)$ is the successor situation to situation s, which results from doing action a in situation s

NOTE: This thesis assumes that there is a finite number of action types in the considered domains.

SitCalc is a simple formalism and does not allow to represent many of the problems that are commonly found in representing real-world. Some of the problems of SitCalc are:

- no time: it cannot express about how long actions take, or when they occur
- only known actions: no hidden exogenous actions, no unnamed events
- no concurrency: cannot express about doing two actions at once
- only discrete situations: no continuous actions, like pushing an object from A to B
- only hypothetical: cannot express that an action has occurred or will occur

- only primitive actions: no actions made up of other parts, like conditionals or iterations

EXAMPLE:

Simple Situation Calculus Model

Here is a simple situation calculus model of an environment where a robot can pick blocks and move them between the floor and a table:

- primitive actions: $\mathit{pickup}(x)$, $\mathit{putonfloor}(x)$, $\mathit{putontable}(x)$
- fluents: $Holding(x, s)$, $OnTable(x, s)$, $OnFloor(x, s)$
- initial situation:

  $\forall x.\ \neg Holding(x, S_0)$

  $OnTable(x, S_0) \equiv x = A \vee x = B$
- some situations: $\mathit{do}(\mathit{putontable}(A), \mathit{do}(\mathit{pickup}(A), S_0))$

  situation where an object A was picked up and put on table $\square$

2.1.2 Basic Action Theories

Sets of formulas within the SitCalc language that describe how the world changes as a result of available actions are called action theories. Basic action theories are special cases of action theories and were proposed by Pirri and Reiter [PR99] and by Reiter [Rei01].

A basic action theory, denoted by $\mathcal{D}$, is a union of the following disjoint sets of axioms:

$$\mathcal{D} = \Sigma \cup \mathcal{D}_{poss} \cup \mathcal{D}_{ssa} \cup \mathcal{D}_{ca} \cup \mathcal{D}_{S_0}$$

where

Σ - foundational, domain independent, axioms of the situation calculus

$\mathcal{D}_{poss}$ - precondition axioms describing when actions can be legally performed

$\mathcal{D}_{ssa}$ - successor state axioms describing how fluents change between situations

$\mathcal{D}_{ca}$ - unique name axioms for actions and domain closure on action types

$\mathcal{D}_{S_0}$ - axioms describing the fluents in the initial situation of the world

These sets of axioms are described and defined below.

The foundational, domain independent, set of axioms Σ is the same as that of the situation calculus.

The precondition axioms $\mathcal{D}_{poss}$ describing when actions can be legally performed are defined by a special predicate *Poss*. The predicate $Poss(a, s)$ is used to define if action a is executable in situation s. $Poss(a, s)$ defines the requirements that must be satisfied whenever the action can be executed in the given situation and it uses fluents to do so. There is one precondition axiom per action i.e. the number of axioms is equal to the number of actions $|A|$. A precondition axiom has a form of "it is possible to do an action a iff a formula of fluents holds":

$$Poss(a(\vec{x}), s) \equiv \Pi_a(\vec{x}, s)$$

where $\Pi_a(\vec{x}, s)$ is a first-order formula containing fluents and no $do(\ldots)$ construct appears in Π_a

Successor state axioms $\mathcal{D}_{ssa}$ describe how fluents change between situations. They encode the casual laws of the modelled world. They replace the effect and frame axioms under unique action name assumption. Effect axioms describe what changes (i.e. what becomes true and what becomes false) for each fluent as a result of an action thus how actions affect the values of fluents. Effect axiom sentences are not very intuitive. Frame axioms describe what is unchanged for each fluent as a result of an action. Frame axioms specify fluents unaffected be performance of action. Just using the effect and frame axioms requires 2 axioms per fluent per action and this can be a significant number. Additionally, any introduction of new fluents requires several new effect and frame axioms and this may be prone to errors. On the other hand only one successor state axiom is defined per fluent. If the number of actions in the subject theory is $|A|$ and the number of fluents is $|F|$ then using $|F|$ successor state axioms effectively replaces $2 * |F| * |A|$ effect and frame axioms. Additionally if relatively few actions affect each fluent and if it can be assumed that effect axioms capture complete conditions then we do not get fewer axioms at the expense of prohibitively long ones - the length of a successor state axioms is roughly proportional to the number of actions which affect the truth value of the fluent. For deterministic actions and given that an action is possible the effect axioms are in the following normal form:

$R_+(\vec{x}, a, s) \supset F(\vec{x}, do(a, s))$ - positive relational effect i.e. F becomes true if R_+

$R_-(\vec{x}, a, s) \supset \neg F(\vec{x}, do(a, s))$ - negative relational effect i.e. F becomes false if R_-

$R(\vec{x}, y, a, s) \supset f(\vec{x}, do(a, s)) = y$ - functional effect i.e. f takes a value y

where R is the first-order formula specifying conditions under which its action will have effect

Reiter [Rei01] proposes a systematic solution to effect and frame axiom multiplicity: new actions just require effect axioms and the complete successor state axioms are generated and frame conditions are computed - no accidental omission of frame axioms. The conversion from effect axioms to successor state axioms can be done under the unique names assumption for actions and the completeness assumption i.e. that the normal forms of the effect axioms characterize all the conditions under which an action a changes the value of fluent F or function f. Given that an action is possible, the formula for obtaining successor state axioms from the normal forms of the effect axioms is:

$F(\vec{x}, do(a, s)) \equiv R_+(\vec{x}, a, s) \vee F(\vec{x}, s) \wedge \neg R_-(\vec{x}, a, s)$ for relational fluents

$f(\vec{x}, do(a, s)) = y \equiv R(\vec{x}, y, a, s) \vee f(\vec{x}, s) = y \wedge \neg(\exists y'.R(\vec{x}, y', a, s))$ for functional fluents

The unique name axioms $\mathcal{D}_{ca}$ for actions and domain closure on action types is the assumption of unique names for actions:

$a(\vec{x}) \neq b(\vec{y})$ where a and b are distinct actions

$a(\vec{x}) = a(\vec{y}) \supset \vec{x} = \vec{y}$

Finally the axioms $\mathcal{D}_{S_0}$ describing the fluents in the initial situation of the world describe the state or value of the fluents in the initial situation S_0.

EXAMPLE:

Simple Action Theory

Here is a simple action theory that models an environment where a robot can pick blocks and move them between the floor and a table:

- primitive actions: $\mathit{pickup}(x)$, $\mathit{putonfloor}(x)$, $\mathit{putontable}(x)$
- fluents: $Holding(x,s)$, $OnTable(x,s)$, $OnFloor(x,s)$
- action preconditions:

  $Poss(\mathit{pickup}(x), s) \equiv \forall z. \neg Holding(z,s)$

  $Poss(\mathit{putonfloor}(x), s) \equiv Holding(x,s)$

  $Poss(\mathit{putontable}(x), s) \equiv Holding(x,s)$

- successor state axioms:

  $Holding(x, do(a,s)) \equiv a = \mathit{pickup}(x) \vee Holding(x,s) \wedge a \neq \mathit{putontable}(x) \wedge a \neq \mathit{putonfloor}(x)$

  $OnTable(x, do(a,s)) \equiv a = \mathit{putontable}(x) \vee OnTable(x,s) \wedge a \neq \mathit{pickup}(x)$

  $OnFloor(x, do(a,s)) \equiv a = \mathit{putonfloor}(x) \vee OnFloor(x,s) \wedge a \neq \mathit{pickup}(x)$

- initial situation:

$\forall x.\ \neg Holding(x, S_0)$

$OnTable(x, S_0) \equiv x = A \lor x = B$ □

2.2 Reasoning about Action

One of the typical tasks of reasoning about actions is planning. In general planning means to figure out what to do to make an arbitrary condition true. A condition to be achieved is called a goal and the sequence of actions that will make the goal true is called a plan.

Somewhat more formally the **planning problem** can be defined as follows: given an axiomatized initial situation, and a goal statement, find an action sequence that will lead to a state in which the goal will be true. The formal definition of the planning problem is:

> given a formula $Goal(s)$, find a sequence of actions a such that $\mathcal{D} \models Goal(do(\vec{a}, S_0)) \land Legal(do(\vec{a}, S_0))$
>
> where $\vec{a} = \langle a_1, \ldots, a_n \rangle$
>
> and $do(\vec{a}, S_0)$ is an abbreviation for $do(a_n, do(a_{n-1}, \ldots, do(a_2, do(a_1, S_0)) \ldots))$
>
> and $Legal(\vec{a}, S_0)$ is an abbreviation for $Poss(a_1, S_0) \land Poss(a_2, do(a_1, S_0)) \land \ldots \land Poss(a_n, do(\ldots, S_0))$
>
> here a plan is some such sequence $\vec{a}$

Here $\models$ stands for second order entailment. But [Rei01] has shown that in many cases, planning and the other reasoning tasks mentioned below can be performed using regression and first order reasoning.

Another typical task of reasoning about actions is the projection task. The **projection task** can be defined as follows: given a sequence of actions, determine what would be true in the situation that results from performing that sequence. To find out if $R(s)$ would be true after performing $\vec{a} = \langle a_1, \ldots, a_n \rangle$ in the initial situation, we determine whether or not $\mathcal{D} \models R(do(a_n, do(a_{n-1}, \ldots, do(a_1, S_0) \ldots)))$

The projection task asks if a condition would hold after performing a sequence of actions, but not whether that sequence can in fact be properly executed. Therefore usually there is a requirement to perform the

legality task. We call a situation legal if it is the initial situation or the result of performing an action whose preconditions are satisfied starting in a legal situation. In other words the legality task is the task of determining whether a sequence of actions leads to a legal situation. To find out if the sequence $\vec{a} = \langle a_1, \ldots, a_n \rangle$ can be legally performed in the initial situation, we determine whether or not $\mathcal{D} \models Poss(a_i, do(a_{i-1}, \ldots, do(a_1, S_0) \ldots))$ for every i such that $1 \leq i \leq n$.

2.3 High-Level Languages

To represent and reason about complex actions or processes obtained by suitably executing atomic actions, various so-called high-level programming languages have been defined, most notably Golog and its concurrent extension ConGolog (De Giacomo, Yves Lespérance and Levesque [DLL00]). All these languages overcome the main shortfalls of the classical situation calculus - they allow for complex actions. The next subsection introduces complex actions and the reasons for them. Then the Golog programming language is described. Golog is one of the most popular languages for planning problems and also it is the basis for other specialized languages that are more task oriented. ConGolog is such a language and it introduces concurrency for multi-agent interaction. ConGolog is also described in a subsection. The GameGolog language based on ConGolog has been proposed (De Giacomo, Yves Lespérance and Pearce [DLP10]) to better model multi-agent interaction problems as games and to represent game structures in a more procedural way while also utilizing the situation calculus to specify the dynamic domain. GameGolog is explained at the end of this section.

2.3.1 Complex Actions

The situation calculus deals with elementary actions and how they represented in situations and the state of the fluents. As such the resulting framework can only deal with discrete progression within a predefined sequence of elementary actions. This may be sufficient for some simple planning problems i.e. to find a sequence of actions that will lead to a situation where desired objectives hold but not much more beyond this. Hence some other types actions are necessary that better model real-world problems. Such complex actions, as opposed to the simple elementary ones of the situation calculus, are needed to deal with:

- conditionals e.g. ff the car is in the driveway then drive else walk
- iterations e.g. while there is a block on the table, remove one
- non-deterministic choices e.g. pickup up some block and put it on the floor
- define such actions in terms of the primitive actions, and inherit their solution to the frame problem

The semantics of complex actions can be expressed via a distinguished predicate $Do(a, s, s')$, that says that a possibly-complex action a when started in situation s may legally terminate in situation s'. Now the various types of actions can be defined as:

- primitive actions: $Do(A, s, s') \equiv Poss(A, s) \wedge s' = do(A, s)$
- sequence: $Do([A; B], s, s') = \exists s".Do(A, s, s") \wedge Do(B, s", s')$
- conditionals: $Do([if \phi then A else B], s, s') \equiv \phi(s) \wedge Do(A, s, s')) \vee \neg\phi(s) \wedge Do(B, s, s')$
- non-deterministic branch: $Do([A|B], s, s') = Do(A, s, s') \vee Do(B, s, s')$
- non-deterministic choice: $Do([\pi x.A], s, s') = \exists x.Do(A, s, s')$

Several high-level languages have been defined to represent and reason about complex actions, and processes obtained by suitably executing atomic actions. Some of these languages are described in the following subsections.

2.3.2 Golog and ConGolog

Golog was proposed by Levesque, Reiter, Lespérance, Lin and Scherl [LRL+97]. Golog is short for "Algol in logic" and it is a programming language that generalizes conventional imperative programming and logic. It includes the usual imperative constructs but also allows to model processes that require non-determinism. It bottoms out not on operations on internal states (assignment statements, pointer updates) but on primitive actions in the world. What the primitive actions do is user-specified by precondition and successor state axioms and defined by logic similar to the basic action theories.

To "execute" a Golog program A means to find a sequence of primitive actions such that performing them starting in some initial situation s would lead to a situation s' where the formula $Do(A, s, s')$ holds (as explained in the preceding Complex Action subsection). In other words, given domain theory $\mathcal{D}$ and program d, it is to find a sequence of actions a such that: $\mathcal{D} \models Do(d, S_0, do(\vec{a}, S_0))$.

The Golog constructs are:

- primitive action: α
- test a condition: $\phi?$
- sequence: $(\rho_1; \rho_2)$
- conditional: $if\ \phi\ then\ \rho_1\ else\ \rho_2\ endIf$
- loop: $while\ \phi\ do\ \rho\ endWhile$
- procedure definition: $proc\ \beta(x)\ :\ \rho\ endProc$
- procedure call: $\beta(t)$
- non-deterministic branch: $(\rho_1 \mid \rho_2)$
- non-deterministic choice of arguments: $\pi\ x\ [\rho]$
- non-deterministic iteration: $\rho*$

It should be explained here how non-determinism is understood. Non-determinism is mean as some choice which turns valid - somehow the program knows (maybe by trying all options) which choices turn valid. One can think of non-deterministic execution as branching that goes in many copies at once and executions that turn invalid are discarded.

NOTE: If what is known about the actions and the initial state can be expressed as Horn clauses, then Golog evaluation can be done in Prolog.

EXAMPLE:

Simple Golog Program

Here is a simple Golog program that models an environment where a robot can pick blocks and move them between the floor and a table:

- primitive actions: $pickup(x)$, $putonfloor(x)$, $putontable(x)$
- fluents: $Holding(x, s)$, $OnTable(x, s)$, $OnFloor(x, s)$
- action preconditions:

  $Poss(pickup(x), s) \equiv \forall z. \neg Holding(z, s)$

  $Poss(putonfloor(x), s) \equiv Holding(x, s)$

  $Poss(putontable(x), s) \equiv Holding(x, s)$
- successor state axioms:

  $Holding(x, do(a, s)) \equiv a = pickup(x) \lor Holding(x, s) \land a \neq putontable(x) \land a \neq putonfloor(x)$

  $OnTable(x, do(a, s)) \equiv a = putontable(x) \lor OnTable(x, s) \land a \neq pickup(x)$

  $OnFloor(x, do(a, s)) \equiv a = putonfloor(x) \lor OnFloor(x, s) \land a \neq pickup(x)$
- initial situation:

  $\forall x. \neg Holding(x, S_0)$

  $OnTable(x, S_0) \equiv x = A \lor x = B$
- some complex actions:

$$proc\ ClearTable\ :\ while\ \exists b.\ OnTable(b)\ do\ \pi\ b\ [OnTable(b)?\ ;\ RemoveBlock(b)]\ endProc$$

$$proc\ RemoveBlock(x)\ :\ pickup(x)\ ;\ putonfloor(x)\ endProc$$

□

The key limitation of Golog were its lack of support for concurrent processes: the inability to program several agents within a single Golog program and the inability to specify an agent's behaviour using concurrent processes. This made Golog inconvenient when it comes to program reactive or event-driven behaviours. Concurrent Golog, in short ConGolog, was proposed by De Giacomo, Lespérance and Levesque [DLL00] to overcome the mentioned limitations. ConGolog introduced concurrent processes with possibly different priorities and concurrent processes as interleavings of the primitive actions.

Here are some of the features of ConGolog:

- $\rho_1\ \rangle\rangle\ \rho_2$

  ρ_1 has higher priority than ρ_2; ρ_2 executes only when ρ_1 is done or blocked

- $\rho^{||}$

  this construct is like non-deterministic iteration ρ^*,

  but the instances of ρ are executed concurrently rather than in sequence

- $\langle\phi \rightarrow \rho\rangle$

  high-level interrupts - an interrupt has trigger condition ϕ and body ρ;

  if interrupt gets control from higher priority processes and condition is true,

  it triggers and body is executed; once body completes execution it may trigger again

- arbitrary exogenous actions

  exogenous actions that can occur at random;

  it is achieved by defining the Exo predicate: $Exo(a)\ \equiv\ a = a_1\ \vee\ \cdots\ \vee\ a = a_n$

In De Giacomo, Yves Lespérance and Pearce [DLP10], which is the main focus of this thesis, the focus is put on a subset of ConGolog which includes most constructs of ConGolog except for recursive procedures. This subset is sufficient to model game structures and is used by this thesis. Let ϕ be a situation calculus situation-suppressed formula, then a ConGolog program is any of the following (recursive) constructs:

- α - atomic action
- $\phi?$ - test for a condition

    it provides for the ConGolog program to advance if formula ϕ holds;

    it is not considered an action and nothing changes in the state after the test is passed
- $\rho_1\ ;\ \rho_2$ - sequence

  it provides for execution of ConGolog program ρ_1 followed by execution of ConGolog program ρ_2
- $if\ \phi\ then\ \rho_1\ else\ \rho_2$ - conditional

    it provides for execution of ConGolog program ρ_1

    if formula ϕ holds otherwise it allows execution of ConGolog program ρ_2
- $while\ \phi\ do\ \rho$ - while loop

    it provides for executing ConGolog program ρ while formula ϕ holds
- $\rho_1 \mid \rho_2$ - non-deterministic branch

    it provides for non-deterministic choice between ConGolog programs ρ_1 and ρ_2
- $\pi\ x\ .\ \rho$ - non-deterministic choice of argument

    it provides for execution of ConGolog program ρ for some non-deterministic choice

    (in general unbounded) of legal binding for variable x;

    [DLP10] requires that x occurs in some non-variable action term in ConGolog program ρ;

    it is disallowed for x to occur only in tests or as an action itself;

    effectively the construct acts as non-deterministic choice of action parameters;

it is assumed that each occurrence of this construct uses a unique fresh variable x

that no two occurrences of this construct use the same variable

- ρ_* - non-deterministic iteration

  it provides for non-deterministic number of iterations of ConGolog program ρ, possibly none

- $\rho_1 \parallel \rho_2$ - concurrent execution

  it provides for concurrent execution of programs ρ_1 and ρ_2

  and is interpreted as interleaving of programs ρ_1 and ρ_2

The semantic of ConGolog constructs can be specified formally as single-step transitions via two predicates $Trans(\rho, s, \rho', s')$ and $Final(\rho, s)$ as defined in De Giacomo, Lespérance and Levesque [DLL00]. One can interpret the predicate $Trans(\rho, s, \rho', s')$ to hold if one step of program ρ in situation s can lead to situation s' and program ρ' remaining for execution. You can interpret the predicate $Final(\rho, s)$ to hold if program ρ in situation s can legally terminate (is considered done, has nothing else to execute). The formal definitions of $Trans$ and $Final$ as used in this thesis are from Sardina and De Giacomo [SD09]. They differ from the usual ones [DLL00] in the definition of the test construct i.e. $\phi?$ does not yield any transition but is final when satisfied. The definitions of $Trans$ and $Final$ are as follows:

$$Trans(\alpha, s, \delta', s') \equiv s' = do(\alpha, s) \wedge Poss(\alpha, s) \wedge \delta' = True?$$

$$Trans(\varphi?, s, \delta', s') \equiv False$$

$$Trans(\delta_1; \delta_2, s, \delta', s') \equiv Trans(\delta_1, s, \delta_1', s') \wedge \delta' = \delta_1'; \delta_2 \vee Final(\delta_1, s) \wedge Trans(\delta_2, s, \delta', s')$$

$$Trans(\textbf{if } \varphi \textbf{ then } \delta_1 \textbf{ else } \delta_2, s, \delta', s') \equiv \varphi[s] \wedge Trans(\delta_1, s, \delta', s') \vee \neg\varphi[s] \wedge Trans(\delta_2, s, \delta', s')$$

$$Trans(\textbf{while } \varphi \textbf{ do } \delta, s, \delta', s') \equiv \varphi[s] \wedge Trans(\delta, s, \delta'', s') \wedge \delta' = \delta''; (\textbf{while } \varphi \textbf{ do } \delta)$$

$$Trans(\delta_1 | \delta_2, s, \delta', s') \equiv Trans(\delta_1, s, \delta', s') \vee Trans(\delta_2, s, \delta', s')$$

$$Trans(\pi x.\delta, s, \delta', s') \equiv \exists x. Trans(\delta, s, \delta', s')$$

$Trans(\delta^*, s, \delta', s') \equiv Trans(\delta, s, \delta'', s') \wedge \delta' = \delta''; \delta^*$

$Trans(\delta_1 \| \delta_2, s, \delta', s') \equiv Trans(\delta_1, s, \delta_1', s') \wedge \delta' = \delta_1' \| \delta_2 \vee Trans(\delta_2, s, \delta_2', s') \wedge \delta' = \delta_1 \| \delta_2'$

$Final(\alpha, s) \equiv False$

$Final(\varphi?, s) \equiv \varphi[s]$

$Final(\delta_1; \delta_2, s) \equiv Final(\delta_1, s) \wedge Final(\delta_2, s)$

$Final(\textbf{if } \varphi \textbf{ then } \delta_1 \textbf{ else } \delta_2, s) \equiv \varphi[s] \wedge Final(\delta_1, s) \vee \neg\varphi[s] \wedge Final(\delta_2)$

$Final(\textbf{while } \varphi \textbf{ do } \delta, s) \equiv \varphi[s] \wedge Final(\delta, s') \vee \neg\varphi[s]$

$Final(\delta_1 | \delta_2, s) \equiv Final(\delta_1, s) \vee Final(\delta_2, s)$

$Final(\pi x.\delta, s) \equiv \exists x.Final(\delta, s)$

$Final(\delta^*, s) \equiv True$

$Final(\delta_1 \| \delta_2, s) \equiv Final(\delta_1, s) \wedge Final(\delta_2, s)$

It should be noted that another programming language IndiGolog has been developed to yet better model multi-agent interactions. IndiGolog extends ConGolog to support interleaved search and execution, to perform on-line sensing, and to allow detecting of exogenous actions.

2.3.3 GameGolog

GameGolog, a short form of Game Structure ConGolog, was proposed by Giacomo, Lespérance and Pearce [DLP10]. It is intended as an alternative, more procedural, way of specifying game structures. In the language all non-deterministic choices are made by some agent that has control in the situation and these choices are recorded in the situation. Therefore the history of non-deterministic choices is recorded in the situation. Also the agent that gets to act next is always specified. GameGolog programs are denoted by ρ

possibly with subscripts or superscripts. GameGolog is a variant of ConGolog programming language where the non-deterministic constructs and the concurrency construct have been replaced by new constructs that explicitly specify which agent is responsible for the non-deterministic choice. GameGolog new constructs are:

- $[agt\ \rho_1\ |\rho_2]$

  non-deterministic branch that replaces ConGolog non-deterministic branch;

  agent agt is responsible for making a non-deterministic choice to execute ρ_1 or ρ_2

  i.e. the agent agt makes a choice whether to continue with ρ_1 or with ρ_2

- $[agt\ \pi x.\ \rho]$

  non-deterministic choice of argument that replaces corresponding ConGolog construct;

  agent agt is responsible for making a non-deterministic choice of argument x then ρ is executed

  i.e. the agent agt makes a choice for the binding for variable x to continue with ρ

- $[agt\ \rho^*]$

  non-deterministic iteration that replaces ConGolog non-deterministic iteration;

  agent agt is responsible for making a non-deterministic choice of executing program ρ

  or to terminate the iteration i.e. the agent agt makes a choice when to stop the iteration of ρ

- $[agt\ \rho_1 \parallel \rho_2]$

  concurrency that replaces ConGolog concurrency construct;

  the agent agt chooses how to interleave the execution of ρ_1 and ρ_2

In fact De Giacomo, Lespérance and Pearce [DLP10] show how any GameGolog program without the new concurrency construct can be translated into ConGolog. For convenience the GameGolog language allows to write $[agt\ \rho]$ where ρ is a program that may mix GameGolog and ConGolog where agent agt controls all the non-deterministic choices in ρ that are not already controlled by other agents. [DLP10] explains how to get the standard GameGolog program from $[agt\ \rho]$ notation.

The semantics of GameGolog constructs, similarly to ConGolog, can be specified formally as single-step transitions via two predicates $Trans(\rho, s, \rho', s')$ and $Final(\rho, s)$ as defined in De Giacomo, Lespérance and Levesque [DLL00] and already explained in the previous section. In fact these definitions are the same as in ConGolog except for the new constructs:

$Trans([agt\ \rho_1 |\ \rho_2], s, \rho', s') \equiv s' = do(left(agt), s) \wedge \rho' = \rho_1 \vee s' = do(right(agt), s) \wedge \rho' = \rho_2$

the non-deterministic choice is recorded in the situation;

the agent may choose to go "left" by performing $left(agt)$ choice action and then execute ρ_1

or the agent may choose to go "right" by performing $right(agt)$ choice action and then execute ρ_2

$Trans([agt\ \pi x.\ \rho], s, \rho', s') \equiv \exists x. s' = pick(agt, x) \wedge \rho' = \rho$

the non-deterministic choice is recorded in the situation;

the agent makes a choice for binding of x by performing $pick(agt, x)$ choice action and then execute ρ for this binding of x

$Trans([agt\ \rho^*], s, \rho', s') \equiv s' = do(continue(agt), s) \wedge \rho' = \rho; [agt\ \rho^*] \vee s' = do(stop(agt), s) \wedge \rho' = True?$

the non-deterministic choice is recorded in the situation;

agent may choose to continue iteration by performing $continue(agt)$ choice action and execute ρ

or agent may choose to stop iteration by performing $stop(agt)$ choice action and terminating

$Trans([agt\ \rho_1 \| \rho_2], s, \rho', s') \equiv$

$s' = do(left(agt), s) \wedge \rho' = [agt\ \rho_1 \langle||\ \rho_2] \vee s' = do(right(agt), s) \wedge \rho' = [agt\ \rho_1 ||\rangle\ \rho_2]$

$Final([agt\ \rho_1 |\ \rho_2], s) \equiv False$

$Final([agt\ \pi x. \rho], s) \equiv False$

$Final([agt\ \rho^*], s) \equiv False$

$Final([agt\ \rho_1 \| \rho_2], s) \equiv False$

EXAMPLE:

GameGolog Program of the TTT (Tic-Tac-Toe) Game

```
$\rho_{TTT} =$
    **while** $\neg Finished()$ **do** (
        $[X\ \pi r, c.\ move(X, r, c)]$;
        **if** $\neg Finished()$ **then** $[O\ \pi r, c.\ move(O, r, c)]$ **else** $True?$
    )
```

The program requires the Situation Calculus axiomatization of the game (provided in this thesis). ρ_{TTT} simply alternates the moves of X and O, starting from X, until a player has won or the board no longer has blanks and hence the fluent $Finished()$ holds.

□

2.4 Characteristic Graphs

Characteristic graphs and verification methods based on characteristic graphs have been proposed by Claßen and Lakemeyer [CL08]. De Giacomo, Lespérance and Pearce [DLP10] use a variant of characteristic graph - characteristic graph for a Golog program. They employ characteristic graphs to compactly represent all possible configurations that a GameGolog program may visit during its execution. The characteristic graph of a Golog program ρ_0 is a graph $\mathcal{G}$ where:

- the nodes are tuples of the form $\langle \rho, \chi \rangle$

  meaning that ρ is a possible remaining program during ρ_0 execution

  and χ characterizes the conditions under which ρ can terminate

- the initial node is $\langle \rho_0, \chi_0 \rangle$

- the edges are tuples of the form $\langle \pi\vec{x}.\alpha, \omega\rangle$

    they represent single transitions between program configurations;

    α is an action term with specific action type, it is not an action variable;

    $\vec{x}$ is a tuple of variables that may appear free in α and ω;

    ω is a condition that must hold for the action $\pi\vec{x}.\alpha$ to be possible;

    if ω is $True$ then it can be omitted and the edge is labelled with just $\pi\vec{x}.\alpha$;

    $\pi\vec{x}.\alpha$ can be reduced to just term α if there are no variable bindings to be made;

    in essence an edge represents a transition from configuration to configuration when

    the binding for variable $\pi\vec{x}$ is chosen and action α is performed in a situation where ω holds

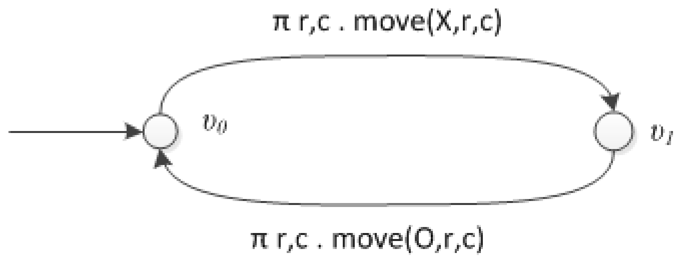
EXAMPLE:

Characteristic Graph of the TTT (Tic-Tac-Toe) Game

The GameGolog program for the TTT game is given as:

```
ρ_TTT =
    while ¬Finished() do (
        [X πr, c. move(X, r, c)];
        if ¬Finished() then [O πr, c. move(O, r, c)] else True?
    )
```

The corresponding characteristic graph $\mathcal{G}_{TTT}$ is:



Where:

- $\upsilon_0 = \langle [X\ \pi r, c.\ move(X, r, c)]; [O\ \pi r, c.\ move(O, r, c)], Finished() \rangle$
- $\upsilon_1 = \langle [O\ \pi r, c.\ move(O, r, c)]; [X\ \pi r, c.\ move(X, r, c)]; [O\ \pi r, c.\ move(O, r, c)], Finished() \rangle$

□

2.5 Game Structures

Several specifications have been proposed that model games in a natural way, allow us to specify games with procedural constructs, and allow us to represent various properties in logic. These specifications also allow us to express multi-agent interaction problems. The Game Description Language (GDL) by Genesereth, Love and Pell [GLP05] provides a declarative specification language to represent discrete complete information on games. But GDL does not provide for temporally extended game property representation and verification. One formalism for representing games via logic is to employ the situation calculus and action theories as introduced by Reiter [Rei01]. This thesis focuses on three such languages that have been utilized or developed in De Giacomo, Lespérance and Pearce [DLP10]. The first one is the Basic Action Theory formalism that is a generic situation calculus-based logical language that is possibly the simplest way to represent an environment that requires reasoning about actions and situations. The second is the Situation Calculus Game Structure formalism that is a specialization of the Situation Calculus and more conveniently models agent interaction. The third is the GameGolog Theory formalism that utilizes Situation Calculus Game Structure axiomatization of the world to model, but agent interaction is defined procedurally as a GameGolog program. There are some other languages that can be used to represent game structures although they have not been used in this thesis as they were not appropriate or required for the main research presented here. These languages are described in the Other Related Work section of this thesis.

2.5.1 Basic Action Theory Models

The simplest formalisms to describe game problems are basic action theories. Basic action theories have been defined in a previous section of this thesis but, as a reiteration, a basic action theory of a given game, denoted by $\mathcal{D}$, is a union of the following disjoint sets:

$$\mathcal{D} = \Sigma \cup \mathcal{D}_{poss} \cup \mathcal{D}_{ssa} \cup \mathcal{D}_{ca} \cup \mathcal{D}_{S_0}$$

where

Σ - foundational, domain independent, axioms of the situation calculus

these have been defined in a previous section of this thesis

$\mathcal{D}_{poss}$ - precondition axioms describing when actions can be legally performed

these have been described in a previous section of this thesis and a defined according to the problem being modelled

$\mathcal{D}_{ssa}$ - successor state axioms describing how fluents change between situations

these have been described in a previous section of this thesis and a defined according to the problem being modelled

$\mathcal{D}_{ca}$ - unique name axioms for actions and domain closure on action types

these have been defined in a previous section of this thesis

$\mathcal{D}_{S_0}$ - axioms describing the fluents in the initial situation of the world

these have been described in a previous section of this thesis and a defined according to the problem being modelled

All the above axioms are defined and explained in a previous section of this thesis and the axioms that are problem-specific are summarized below.

The precondition axioms $\mathcal{D}_{poss}$ describing when actions can be legally performed are defined by a special predicate $Poss$. The predicate $Poss(a, s)$ is used to define if action a is executable in situation s. $Poss(a, s)$ defines the requirements that must be satisfied whenever the action can be executed in the given situation and it uses fluents to do so. Minor conditions, including unknown ones, are ignored.

Successor state axioms $\mathcal{D}_{ssa}$ describe how fluents change between situations. They encode the casual laws of the modelled world. They replace the effect and frame axioms under unique action name assumption. Effect axioms describe what changes (i.e. what becomes true and what becomes false) for each fluent as a result of an action thus how actions affect the values of fluents. Frame axioms describe what is unchanged for each fluent as a result of an action i.e. frame axioms specify fluents unaffected be performance of action. Reiter [Rei01] proposes a complete systematic solution to computing successor state axioms from effect axioms while respecting implicit frame conditions. The conversion from effect axioms to successor state axioms can be done under the unique names assumption for actions and the completeness assumption i.e. that the normal forms of the effect axioms characterize all the conditions under which an action a changes the value of fluent F or function f.

Finally the axioms $\mathcal{D}_{S_0}$ describing the fluents in the initial situation of the world describe the state or value of the fluents in the initial situation S_0.

EXAMPLE:

Basic Action Theory Axiomatization of the TTT (Tic-Tac-Toe) Game

Fluents

- $turn(s)$ - functional fluent indicating which agent is to take next action; the domain is the set of agents
- $Cell(m, r, c, s)$ - relational fluent indicating if the content of a cell row r and column c is m
- $Wins(p, s) \equiv$

  $\exists c. Cell(p, 1, c, s) \wedge Cell(p, 2, c, s) \wedge Cell(p, 3, c, s) \vee$

$\exists r.Cell(p, r, 1, s) \wedge Cell(p, r, 2, s) \wedge Cell(p, r, 3, s) \vee$

$Cell(p, 1, 1, s) \wedge Cell(p, 2, 2, s) \wedge Cell(p, 3, 3, s) \vee$

$Cell(p, 1, 3, s) \wedge Cell(p, 2, 2, s) \wedge Cell(p, 3, 1, s)$

constructed fluent indicating if agent p is considered a winner

- $Finished(s) \equiv \exists p.Wins(p, s) \vee \forall r, c.InRange(r, c) \wedge \neg Cell(B, r, c, s)$

  constructed fluent indicating if there should be no more actions

- $InRange(r, c) \equiv (r = 1 \vee r = 2 \vee r = 3) \wedge (c = 1 \vee c = 2 \vee c = 3)$

  situation invariant predicate for the domain of cells

- $Agent(p) \equiv p = X \vee p = O$

  situation invariant predicate for the domain of agents

Actions

- $move(p, r, c)$ - agent p marks the cell row r and column c with mark p

Axioms Σ

Standard foundational domain-independent axioms of Situation Calculus.

Axioms $\mathcal{D}_{poss}$

Precondition axioms, one per action, indicate when actions can be legally performed:

$Poss(move(p, r, c), s) \equiv Agent(p) \wedge turn(s) = p \wedge InRange(r, c) \wedge Cell(B, r, c, s)$

players can only move on blank cells

Axioms $\mathcal{D}_{ssa}$

Successor state axioms describing how the fluents change as a result of actions. They are derived from the effect axioms.

Effect axioms in the normal form:

$$a = move(m, r, c, s) \supset Cell(m, r, c, do(a, s))$$

$$\exists p.a = move(p, r, c, s) \supset \neg Cell(m, r, c, do(a, s))$$

$$\neg turn(s) = p \supset turn(do(a, s)) = p$$

Derived successor state axioms (after simplification and domain closure assumption for actions):

$$Cell(m, r, c, do(a, s)) \equiv a = move(m, r, c, s) \lor Cell(m, r, c, s) \land \forall p.a \neq move(p, r, c)$$

$$turn(do(a, s)) = p \equiv \neg turn(s) = p$$

Axioms $\mathcal{D}_{ca}$

Standard unique name axioms for actions and domain closure on action types.

Axioms $\mathcal{D}_{S_0}$

Description of the initial situation:

$$turn(S_0) = X$$

$$\forall r, c.InRange(r, c) \land Cell(r, c, B, S_0)$$

□

2.5.2 Situation Calculus Game Structures

Situation Calculus Game Structures, or SitCalc Game Structures for short, are a specialization of the situation calculus that allows to better model games. The language is similar in nature to Basic Action Theories where some functions and predicates are distinguished and some format standardization is imposed. The descriptions presented in this section follow those of De Giacomo, Lesperance and Pearce [DLP10] which are in general similar to the typical descriptions in the literature. The main difference is that situation calculus game structure in De Giacomo, Lesperance and Pearce [DLP10] refers to a situation calculus game theory and not just single model.

In SitCalc Game Structures every action has an agent parameter and the distinguished function $agent(a)$ takes a parameter a which is an action and returns the agent of the action. The axioms for the $agent$ function are defined for every action type and by convention the agent parameter is the first argument of any action type. It is assumed that there is a finite set $Agents$ of agents who denoted by unique names. A distinguished predicate $Poss(a, s)$ specifies if an action a is physically possible (i.e. executable) in situation s. Actions are divided into two groups: choice actions and standard actions. Choice actions model the decisions of agents and they are assumed to have no effect on any fluent other than $Poss$, $Legal$, and $Control$. Choice actions are always physically possible. Standard actions are the other non-choice actions. A distinguished predicate $Legals(s)$ is a stronger version of possibility / legality and models the game structure of interest. It encapsulates the ability of an agent to execute actions and perform decisions according to the rules of the game and it is axiomatized according to the game being modelled. It is required that Legal entails 3 properties:

1. $Legal$ implies physically possible

   $Legal(s) \supset s = S_0 \vee \exists a, s'.s = do(a, s') \wedge Poss(a, s')$

2. legal situations are result of an action performed in legal situations

   $Legal(s) \supset s = S_0 \vee \exists a, s'.s = do(a, s') \wedge Legal(s')$

3. only one agent can act in a legal situation

   $Legal(do(a, s)) \wedge Legal(do(a', s)) \supset agent(a) = agent(a')$

The distinguished predicate $Control(agt, s)$ is a convenience predicate that holds if an agent can act in a given legal situation:

$$Control(agt, s) \doteq \exists a.Legal(do(a, s)) \wedge agent(a) = agt$$

As a result of the constraints on the predicate *Legal* it follows that the predicate *Control* holds for only one agent in a given legal situation. To model games where several agents can act simultaneously a round-robin choice actions among the agents involved can be used. To model games where several agents want to act non-deterministically but one player can succeed in performing his actions an extra player can be introduced who is in charge of making the "non-deterministic" decisions i.e. that player will decide which agent will actually act among the agents that may act and this decision will automatically be recorded in the situation. It is worth noting that the state of the game in situation s is captured by the fluents.

A situation calculus game structures, denoted by $\mathcal{D}_{GS}$, is a union of the following disjoint sets:

$$\mathcal{D}_{GS} = \Sigma \cup \mathcal{D}_{poss} \cup \mathcal{D}_{ssa} \cup \mathcal{D}_{ca} \cup \mathcal{D}_{S_0} \cup \mathcal{D}_{legal}$$

where

Σ - foundational, domain independent, axioms of the situation calculus (as in Basic Action Theories)

$\mathcal{D}_{poss}$ - precondition axioms describing when actions can be physically performed

$\mathcal{D}_{ssa}$ - successor state axioms describing how fluents change (as in Basic Action Theories)

$\mathcal{D}_{ca}$ - unique name axioms for actions and domain closure on action types (as in Basic Action Theories)

$\mathcal{D}_{S_0}$ - axioms describing the fluents in the initial situation of the world (as in Basic Action Theories)

$\mathcal{D}_{legal}$ - axioms for predicates *Legal* and *Control*, and for function $agent()$

EXAMPLE:

SitCalc Game Structure Axiomatization of the TTT (Tic-Tac-Toe) Game

Fluents

- $Cell(m, r, c, s)$ - relational fluent indicating if the content of a cell row r and column c is m
- $turn(s)$ - functional fluent indicating the agent that is to take next action
- $InRange(r, c) \equiv (r = 1 \vee r = 2 \vee r = 3) \wedge (c = 1 \vee c = 2 \vee c = 3)$

  situation invariant predicate for the domain of cells
- $Agent(p) \equiv p = X \vee p = O$

  situation invariant predicate for the domain of agents
- $Wins(p, s) \ \equiv$

  $\exists c.Cell(p, 1, c, s) \wedge Cell(p, 2, c, s) \wedge Cell(p, 3, c, s) \ \vee$

  $\exists r.Cell(p, r, 1, s) \wedge Cell(p, r, 2, s) \wedge Cell(p, r, 3, s) \ \vee$

  $Cell(p, 1, 1, s) \wedge Cell(p, 2, 2, s) \wedge Cell(p, 3, 3, s) \ \vee$

  $Cell(p, 1, 3, s) \wedge Cell(p, 2, 2, s) \wedge Cell(p, 3, 1, s)$

  constructed fluent indicating if agent p is considered a winner
- $Completed(s) \doteq \forall r, c.InRange(r, c) \wedge Cell(m, r, c, s) \supset m \neq B$

  (convenience) formula abbreviation
- $Finished(s) \doteq Completed(s) \vee Wins(X, s) \vee Wins(O, s)$

  (convenience) formula abbreviation

Actions

- $move(p, r, c)$ - agent p marks the cell row r and column c with mark p

Axioms Σ

Standard foundational domain-independent axioms of Situation Calculus.

Axioms $\mathcal{D}_{poss}$

Precondition axioms, one per action, indicate when actions can be legally performed:

$Poss(move(p, r, c), s) \equiv Agent(p) \wedge InRange(r, c) \wedge Cell(B, r, c, s)$

players can only move on blank cells

Axioms $\mathcal{D}_{ssa}$

Successor state axioms describing how the fluents change as a result of actions. They are derived from the effect axioms.

Effect axioms in the normal form:

$a = move(m, r, c, s) \supset Cell(m, r, c, do(a, s))$

$\exists p.a = move(p, r, c, s) \supset \neg Cell(m, r, c, do(a, s))$

$\neg turn(s) = p \supset turn(do(a, s)) = p$

Derived successor state axioms (after simplification and domain closure assumption for actions):

$Cell(m, r, c, do(a, s)) \equiv a = move(m, r, c, s) \vee Cell(m, r, c, s) \wedge \forall p.a \neq move(p, r, c)$

$turn(do(a, s)) = p \equiv \neg turn(s) = p$

Axioms $\mathcal{D}_{ca}$

Standard unique name axioms for actions and domain closure on action types.

Axioms $\mathcal{D}_{S_0}$

Description of the initial situation:

$\forall r, c.InRange(r, c) \wedge Cell(r, c, B, S_0)$

$turn(S_0) = X$

$Legal(S_0)$

Axioms $\mathcal{D}_{legal}$

Axioms for predicates *Legal* and *Control*, and for function *agent*(). They define the rules of the game:

$$Legal(do(a,s)) \equiv Legal(s) \wedge turn(s) = p \wedge \exists a.agent(a) = p \wedge Poss(a,s)$$

$$Control(agt,s) \doteq \exists a.Legal(do(a,s)) \wedge agent(a) = agt$$

$$agent(move(m,r,c)) = m$$

□

2.5.3 GameGolog Theories

The semantics of GameGolog constructs (those shared from ConGolog and those changed ones in GameGolog as described in one of the previous sections of this thesis) can be defined using axioms for predicates *Trans* and *Final*. A GameGolog theory is denoted $\mathcal{D}_{GG}$. The definitions of the *Trans* and *Final* are as given in section 2.3 and the axioms for the replaced constructs are as follows:

$$Trans([agt\ \rho_1|\rho_2],s,\rho',s') \equiv s' = do(left(agt),s) \wedge \rho' = \rho_1 \vee s' = do(right(agt),s) \wedge \rho' = \rho_2$$

the non-deterministic choice is recorded in the situation;

the agent may chose to go "left" by performing $left(agt)$ choice action and then executes ρ_1

or the agent may chose to go "right" by performing $right(agt)$ choice action and then executes ρ_2

$$Trans([agt\ \pi x.\rho],s,\rho',s') \equiv \exists x.s' = pick(agt,x) \wedge \rho' = \rho$$

the non-deterministic choice is recorded in the situation;

the agent makes a choice for binding of x by performing $pick(agt,x)$ choice action

and then executes ρ for this binding of x

$$Trans([agt\ \rho^*],s,\rho',s') \equiv s' = do(continue(agt),s) \wedge \rho' = \rho; [agt\ \rho^*] \vee s' = do(stop(agt),s) \wedge \rho' = True?$$

the non-deterministic choice is recorded in the situation;

the agent makes a choice of continuing the iteration by performing *continue*(agt) choice action and then executes ρ or the agent makes a choice to stop the iteration by performing *stop*(agt) choice action and terminating

$$Trans([agt\ \rho_1 \| \rho_2], s, \rho', s') \equiv$$

$$s' = do(left(agt), s) \wedge \rho' = [agt\ \rho_1 \langle || \ \rho_2] \vee s' = do(right(agt), s) \wedge \rho' = [agt\ \rho_1 || \rangle\ \rho_2]$$

$$Trans([agt\ \rho_1\ \langle ||\ \rho_2], s, \rho', s') \equiv Trans(\rho_1, s, \rho_1', s) \wedge \rho' = [agt\ \rho_1' || \rho_2]$$

$$Trans([agt\ \rho_1\ || \rangle\ \rho_2], s, \rho', s') \equiv Trans(\rho_2, s, \rho_2', s) \wedge \rho' = [agt\ \rho_1 || \rho_2']$$

$$Final([agt\ \rho_1 | \rho_2], s) \equiv False$$

$$Final([agt\ \pi x.\rho], s) \equiv False$$

$$Final([agt\ \rho^*], s) \equiv False$$

$$Final([agt\ \rho_1 \| \rho_2], s) \equiv False$$

$$Final([agt\ \rho_1\ \langle ||\ \rho_2], s) \equiv Final(\rho_1, s) \wedge Final(\rho_2, s)$$

$$Final([agt\ \rho_1\ || \rangle\ \rho_2], s) \equiv Final(\rho_1, s) \wedge Final(\rho_2, s)$$

The semantics of GameGolog and the GameGolog program represent the meaning of Legal in a game structure. Legal situations are those that can be reached from the initial situation by performing transitions on the program ρ_0 that models the subject game structure. *Legal*, and likewise *Final*, can be defined as:

$$Legal(s) \equiv \exists \rho'.Trans^*(\rho_0, S_0, \rho', s)$$

$$Final(s) \equiv \exists \rho'.Trans^*(\rho_0, S_0, \rho', s) \wedge Final(\rho', s)$$

Theories of this form where *Legal*, and optionally *Final*, is defined by GameGolog programs are called GameGolog theories and are denoted $\mathcal{D}_{GGT}$.

EXAMPLE:

GameGolog Theory Axiomatization of the TTT (Tic-Tac-Toe) Game

The GameGolog Theory axiomatization $\mathcal{D}_{GGT}$ of the TTT (Tic-Tac-Toe) Game uses all the same axioms as the Basic Action Theory for the TTT Game (already provided in this thesis) with the addition GameGolog semantics axioms D_{GG} and the following axioms:

```
$\rho_{TTT} =$
    **while** $\neg Finished()$ **do** (
        $[X\ \pi r, c.\ move(X, r, c)]$;
        **if** $\neg Finished()$ **then** $[O\ \pi r, c.\ move(O, r, c)]$ **else** $True?$
    )
```

$Legal(s) \equiv \exists \rho'.Trans^*(\rho_{TTT}, S_0, \rho', s)$

$Final(s) \equiv \exists \rho'.Trans^*(\rho_{TTT}, S_0, \rho', s) \wedge Final(\rho', s)$

2.6 Expressing Properties of Systems in Game-Theoretic Logics

Several specifications have been proposed that model games in natural ways. Some of these formalisms allow us to specify games with procedural constructs, as well as, allow us to represent various temporal properties in logic. Several properties and characteristics of games have been identified and proven useful. These can be very high level properties of good equity (i.e. all players have a chance to win the game), liveness (i.e. all requests will eventually be handled), or the existence of winning strategy (i.e. a player can ensure to win eventually). Additionally a critical characteristic of a representation is the ability to support incomplete specifications of the application domain i.e. when action theories do not have to have a single model. These properties can be expressed in formalisms such as alternating-time temporal logic (ATL) (Alur, Henzinger and Kupferman [AHK02]), or the Ł-language of De Giacomo, Lespérance and Pearce [DLP10] which uses the alternating time μ-calculus and provides a rich grammar for representing various important properties. Such temporal properties can subsequently be verified via model checking or other methods like those that are proposed by De Giacomo, Lespérance and Pearce [DLP10] and that are researched by this thesis. In De Giacomo, Lespérance and Pearce [DLP10] complex temporal dynamic properties can be expressed using least and greatest fixpoint constructions and it is shown how one can formally verify (via regression and fixpoint approximation) that a formula of their logic is satisfied in a situation calculus-based game structure. This technique can be automated easily but does not always terminate - it is sound but not complete.

2.6.1 Alternating-time Temporal Logic (ATL)

Alternating-time Temporal Logic (ATL) has been proposed by Alur, Henzinger, and Kupferman [AHK02]. is a temporal logic i.e. it is logic used to describe rules and symbolism for representing and reasoning about, propositions quantified in terms of time. In temporal logic questions can have truth value that can vary in time and hence time quantification can often be used to eliminate this dependency. Temporal logic has found an important application in formal verification, where it is used to express the important operating requirements of hardware or software systems.

ATL Syntax

The temporal logic ATL (Alternating-time Temporal Logic) is defined with respect to a set Π of propositions and a finite set $\Sigma = \{1, \ldots, k\}$ of players. An ATL formula is one of the following:

(S1) p, for proposition $p \in \Pi$

(S2) $\neg\varphi$ and $\varphi_1 \vee \varphi_2$, where φ, φ_1, and φ_2 are ATL formulas

(S3) $\langle\langle A\rangle\rangle \bigcirc \varphi$, $\langle\langle A\rangle\rangle\Box\varphi$, or $\langle\langle A\rangle\rangle\varphi_1\mathcal{U}\varphi_2$, where $A \in \Sigma$ is a set of players, and φ, φ_1, and φ_2 are ATL formulas

The operator $\langle\langle\ \rangle\rangle$ is a path quantifier parameterized by sets of players. $\bigcirc$ ("next"), $\Box$ ("always"), and $\mathcal{U}$ ("until") are temporal operators. Additional Boolean connectives are defined, for example $\langle\langle A\rangle\rangle\Diamond\varphi$ is for $\langle\langle A\rangle\rangle true\mathcal{U}\varphi$.

Definition of Strategy

Before the semantics of ATL can be properly explained it is helpful to define the notion of strategies. This will be done using a very generic definition of a game structure. This generic definition of a game structure is not used to define game problems researched in this thesis and just gives enough formalism to define some of the concepts used later in this subsection. Consider a game structure $S = \langle k, Q, \Pi, \pi, d, \sigma\rangle$ where:

- k a natural number of players, let the players be identified by numbers $1, \cdots, k$
- finite set Q of states
- finite set Π of propositions
- for each state $q \in Q$, a set $\pi(q) \subset \Pi$ of propositions true in q
- for each player $a \in \{1, \cdots, k\}$ and each state $q \in Q$, a natural number $d_a(q) \geq 1$ of moves available at state q to player a; given a state $q \in Q$, $D(q)$ is the set $\{1, \ldots, d_1(q)\} \times \cdots \times \{1, \ldots, d_k(q)\}$
- σ is the transition function where for each state $q \in Q$ and each vector $\langle j_1, \cdots,\ j_k\rangle$ of moves available to players $a = 1, \cdots, k$ then $\sigma(q, j_1, \cdots,\ j_k) \in Q$ is a state that results from q by each player making a move j_a

A strategy for player $a \in \Sigma$ is a function f_a that maps every nonempty finite state sequence $\lambda \in Q^+$ to a natural number such that if the last state of λ is q, then $f_a(\lambda) \leq d_a(q)$. Thus, the strategy f_a for player a determines for every finite prefix λ of a computation a move $f_a(\lambda)$ for player a. The outcome of $F_A = \{f_a | a \in A\}$ from state q is the set $out(q, F_A)$ computations $\lambda = q_0, q_1, \ldots$ if $q_0 = q$ and for all positions $i \geq 0$, there is a move vector $\langle j_1, \ldots, j_k \rangle \in D(q_i)$ such that $j_a = f_a(\lambda[0, i])$ for all players $ainA$, and $\sigma(q_i, j_1, \ldots, j_k) = q_{i+1}$.

ATL Semantics

The semantics of ATL is defined recursively as follows, where $q \models \varphi$ means that state $q \in \Pi$ satisfies ATL formula φ:

- $q \models p$ iff $p \in \pi(q)$, for propositions $q \in \Pi$
- $q \models \neg\varphi$ iff $q \not\models \varphi$
- $q \models \varphi_1 \vee \varphi_2$ iff $q \models \varphi_1$ or $q \models \varphi_2$
- $q \models \langle\langle A \rangle\rangle \bigcirc \varphi$ iff there exists a set F_A of strategies, one for each player in A, such that for all computations $\lambda \in out(q, F_A)$ and all positions $i \geq 0$, we have $\lambda[i] \models \varphi$
- $q \models \langle\langle A \rangle\rangle \Box \varphi$ iff there exists a set F_A of strategies, one for each player in A, such that for all computations $\lambda \in out(q, F_A)$, we have $\lambda[1] \models \varphi$
- $q \models \langle\langle A \rangle\rangle \varphi_1 \mathcal{U} \varphi_2$ iff there exists a set F_A of strategies, one for each player in A, such that for all computations $\lambda \in out(q, F_A)$, there exists a position $i \geq 0$ such that $\lambda[i] \models \varphi_2$ and for all positions $0 \geq j \geq i$ and $\lambda[j] \models \varphi_1$

It is often useful to express an ATL formula in a dual form. It uses $[\![A]\!]$ quantifier for a set A of players. While $\langle\langle A \rangle\rangle \varphi$ means that the players in A can cooperate to make φ true (they can "enforce" φ), the dual formula $[\![A]\!]\varphi$ means that the players in A cannot cooperate to make φ false (they cannot "avoid" φ). In turn-based synchronous and turn-based asynchronous game structures the players in A can enforce a set Λ of computations iff the players in $\Sigma - A$ cannot avoid Λ.

2.6.2 Alternating-time μ-Calculus (AMC)

Alternating-time μ-Calculus (AMC) has been proposed by Alur, Henzinger, and Kupferman [AHK02] as an extension of μ-Calculus as proposed by E. Allen Emerson [Eme96]. AMC is a temporal logic i.e. it is logic used to describe rules and symbolism for representing and reasoning about, propositions quantified in terms of time. In temporal logic questions can have truth value that can vary in time and hence time quantification can often be used to eliminate this dependency.

AMC Syntax

The temporal logic AMC (Alternating-time μ-Calculus) is defined with respect to a set Π of propositions, a set V of propositional variables (for modal/second order quantification), and a set $\Sigma = \{1, \ldots, k\}$ of players. An AMC formula is one of the following:

- p, for proposition $p \in \Pi$
- X, for variable $X \in V$
- $\neg\varphi$ or $\varphi_1 \vee \varphi_2$, where φ, φ_1, and φ_2 are AMC formulas
- $\langle\langle A \rangle\rangle \bigcirc \varphi$, where $A \subseteq \Sigma$ is a set of players, and φ is an AMC formula
- $\mu X.\varphi$ is the least fixpoint operator, where φ is an AMC formula in which all free occurrences of X fall under even number of negations

The operator $\langle\langle\ \rangle\rangle$ is a path quantifier parameterized by sets of players which is the main change as compared to E. Allen Emerson [Eme96] where the next-time operator $\bigcirc$ is quantified by existential or universal path quantifier.

For example $\mu X.p \vee \langle\langle \Sigma \rangle\rangle X$ says that p eventually holds along some path from the current state, or more formally that the current state is in the least set of states that are fixed points of $\varphi(X) = p \vee \langle\langle \Sigma \rangle\rangle X$

Some abbreviations are proposed by Alur, Henzinger, and Kupferman [AHK02] which are constructed from the elements of the language:

- the dual: $[\![A]\!] \bigcirc \varphi = \neg\langle\langle A\rangle\rangle \bigcirc \neg\varphi$
- $\exists = \langle\langle\Sigma\rangle\rangle$
- $\forall = [\![\Sigma]\!]$
- greatest fixpoint: $\nu X.\varphi = \neg\mu X.\neg\varphi$

AMC Semantics

Before the semantics of AMC can be properly explained it is helpful to provide some definitions. This will be done using a very generic definition of a game structure as used in the background section on ATL. This generic definition of a game structure is not used to define game problems researched in this thesis and just gives enough formalism to define some of the concepts used later in this subsection. Consider a game structure $S = \langle k, Q, \Pi, \pi, d, \sigma\rangle$ where:

- k a natural number of players, let the players be identified by numbers $1, \cdots, k$
- finite set Q of states
- finite set Π of propositions
- for each state $q \in Q$, a set $\pi(q) \subset \Pi$ of propositions true in q
- for each player $a \in \{1, \cdots, k\}$ and each state $q \in Q$, a natural number $d_a(q) \geq 1$ of moves available at state q to player a; given a state $q \in Q$, $D(q)$ is the set $\{1, \ldots, d_1(q)\} \times \cdots \times \{1, \ldots, d_k(q)\}$
- σ is the transition function where for each state $q \in Q$ and each vector $\langle j_1, \cdots,\ j_k\rangle$ of moves available to players $a = 1, \cdots, k$ then $\sigma(q, j_1, \cdots,\ j_k) \in Q$ is a state that results from q by each player making a move j_a

- let's consider $\Sigma = \{1, \cdots, k\}$ to be the set of players

A valuation $\mathcal{V}$ is a function from the propositional variables V to subsets of Q. For valuation $\mathcal{V}$, a propositional variable X, and a set $\rho \subseteq Q$ of states, let $\mathcal{V}[X := \rho]$ be a valuation that maps X to ρ and agrees with $\mathcal{V}$ on all other variables. An AMC formula φ is interpreted as a mapping φ^S from valuations to state sets. Let $\varphi^S(\mathcal{V})$ denote the set of states that satisfies the AMC formula φ under the valuation $\mathcal{V}$ in the structure S. Then the semantics of AMC is the mappings φ^S defined recursively as follows:

- for a proposition $p \in \Pi$, we have $p^S(\mathcal{V}) = \{q \in Q \mid p \in \pi(q)\}$
- for a propositional variable $X \in V$, we have $X^S(\mathcal{V}) = \mathcal{V}(X)$
- $(\neg\varphi)^S(\mathcal{V}) = Q - \varphi^S(\mathcal{V})$
- $(\varphi_1 \vee \varphi_2)^S(\mathcal{V}) = \varphi_1^S(\mathcal{V}) \cup \varphi_2^S(\mathcal{V})$
- $(\langle\langle A\rangle\rangle \bigcirc \varphi) = \{q \in Q \mid \mathit{for\ every\ player\ } a \in A, \mathit{\ there\ exists\ a\ move\ } j_a \in \{1, \dots, d_a(q)\}$

  $\quad \mathit{such\ that\ for\ all\ players\ } b \in \Sigma - A \mathit{\ and\ moves\ } j_b \in \{1, \dots, d_b(q)\},$

  $\quad \mathit{we\ have\ } \sigma(q, j_1, \dots, j_k) \in \varphi^S(\mathcal{V})\}$
- $(\mu X.\varphi)^S(\mathcal{V}) = \bigcap\{\rho \subseteq Q \mid \varphi^S(\mathcal{V}[X := \rho]) \subseteq \rho\}$

Looking at an AMC formula $\mu X.\varphi$, and given a valuation $\mathcal{V}$, the subformula φ can be viewed as a function $h^S_{\varphi,\mathcal{V}}$ that maps each state set $\rho \subseteq Q$ to the state set $\varphi^S(\mathcal{V}[X := \rho])$. By the standard fixed-point theory $(\mu X.\varphi)^S(\mathcal{V})$ has a least fixpoint, namely, $\bigcap\{\rho \subseteq Q \mid \varphi^S(\mathcal{V}[X := \rho]) \subseteq \rho\}$ which can be computed by interactive approximation:

$$(\mu X.\varphi)^S(\mathcal{V}) = \textstyle\bigcup_{i \geq 0}(h + S_{\varphi,\mathcal{V}})^i([false])$$

A sentence of AMC is a formula that that contains no free occurrences of propositional variables.

2.6.3 Ł-Logic

De Giacomo, Lespérance and Pearce [DLP10] propose a specific logic, called Ł-logic, to express properties of game structures. It is convenient to logically express temporal properies of Basic Action Theory-axiomatized game structures. It is inspired by ATL (Alur, Henzinger and Kupferman [AHK02]) and it is based on μ-calculus (Park [Par76]). It focuses on μ-calculus over game structures as initiated in Bradfield and Stirling [BS07].

The key element of the Ł-logic is the $\langle\langle G\rangle\rangle \bigcirc \varphi$ operator defined as follows:

$$\langle\langle G\rangle\rangle \bigcirc \varphi \doteq$$

$$(\exists agt \in G.\ Control(agt, now) \wedge \exists a.\ agent(a) = agt \wedge Legal(do(a, now)) \wedge \varphi[do(a, now)]) \vee$$

$$(\forall agt \notin G.\ Control(agt, now) \wedge \forall a.\ agent(a) = agt \wedge Legal(do(a, now)) \supset \varphi[do(a, now)])$$

This operator in essence defines whether a formula ϕ can hold after one more action as follows. If the coalition G of agents is in control in current situation then all we need is some action of some agent in coalition G that will make the formula ϕ hold after such action. If the coalition G of agents is not in control in current situation then what we need is that regardless of action (for all) and regardless of agent (for all) not in coalition G that the formula ϕ holds after such action. *Control* was defined in section 2.5.2.

The whole logic Ł can be defined as follows. Let ϕ be a situation-suppressed situation calculus uniform formula and let Z be a predicate variable of a given arity then a formula Ψ in language Ł(also called Ł-formula) is any of the following (recursive) constructs:

ϕ – a situation-suppressed situation calculus uniform formula is a formula of Ł

$Z(\vec{x})$ – predicate variable of a given arity is a formula in Ł

$\Psi_1 \wedge \Psi_2$ – conjunction of language Ł-formulas Ψ_1 and Ψ_2 is a formula in Ł

$\Psi_1 \vee \Psi_2$ – disjunction of language Ł-formulas Ψ_1 and Ψ_2 is a formula in Ł

$\exists x.\Psi$ – existential quantification of of language Ł-formula Ψ is a formula in Ł

$\forall x.\Psi$ – universal quantification of of language Ł-formula Ψ is a formula in Ł

$\langle\langle G\rangle\rangle \bigcirc \Psi$ – next of language Ł-formula Ψ, as defined above, is a formula in Ł

$[[G]] \bigcirc \Psi$ – the dual of $\langle\langle G\rangle\rangle \bigcirc \Psi$ (i.e. $[[G]] \bigcirc \Psi \equiv \neg\langle\langle G\rangle\rangle \bigcirc \neg\Psi$) is a formula in Ł

$\mu Z(\vec{x}).\Psi(Z(\vec{x}))$ – the least fixpoint operator from the μ-calculus is a formula in Ł

$\Psi(Z(\vec{x}))$ is the notation to emphasize that $Z(\vec{x})$ may occur free, i.e. not quantified by μ or ν in Ψ

$\nu Z(\vec{x}).\Psi(Z(\vec{x}))$ – the greatest fixpoint operator from the μ-calculus is a formula in Ł

$\Psi(Z(\vec{x}))$ is the notation to emphasize that $Z(\vec{x})$ may occur free, i.e. not quantified by μ or ν in Ψ

Language Ł allows to express arbitrary temporal/dynamic properties.

For example a strategy to achieve $\varphi(\vec{x})$ by group G, where $\varphi(\vec{x})$ is a situation suppressed formula with free variables $\vec{x}$, can be defined by the following least fixpoint construction:

$$\langle\langle G\rangle\rangle\Diamond\varphi(\vec{x}) \doteq \mu Z(\vec{x}).\ \varphi(\vec{x}) \vee \langle\langle G\rangle\rangle \bigcirc Z(\vec{x})$$

$$\langle\langle G\rangle\rangle\Diamond\varphi(\vec{x}) \doteq \mu Z(\vec{x}).\ \varphi(\vec{x}) \vee \langle\langle G\rangle\rangle \bigcirc Z(\vec{x})$$

Similarily an ability to maintain a propert $\varphi(\vec{x})$ by group G, where $\varphi(\vec{x})$ is a situation suppressed formula with free variables $\vec{x}$, can be defined by the following greatest fixpoint construction:

$$\langle\langle G\rangle\rangle\Box\varphi(\vec{x}) \doteq \nu Z(\vec{x}).\varphi(\vec{x}) \wedge \langle\langle G\rangle\rangle \bigcirc Z(\vec{x})$$

The property that there is a path where $\varphi(\vec{x})$ holds next, where $\varphi(\vec{x})$ is a situation suppressed formula with free variables $\vec{x}$, can be defined as the set of all agents can ensure that $\varphi(\vec{x})$ holds next:

$$\exists \bigcirc \varphi(\vec{x}) \doteq \langle\langle Agents\rangle\rangle \bigcirc \varphi(\vec{x})$$

The property that there is a path where $\varphi(\vec{x})$ holds in the future, where $\varphi(\vec{x})$ is a situation suppressed formula with free variables $\vec{x}$, can be defined as the set of all agents has a strategy to achieve $\varphi(\vec{x})$:

$$\exists\Diamond\varphi(\vec{x}) \doteq \langle\langle Agents\rangle\rangle\Diamond\varphi(\vec{x})$$

2.6.4 Ł$_p$-Logic

De Giacomo, Lespérance and Pearce [DLP10] propose program-constraint logic for game structure properties denoted Ł$_p$. The language allows to define temporal properties of game structures by incorporating the $\mathcal{D}_{GG}$ axioms for the GameGolog-based Legal predicate of the game being modeled and utilizing the axioms for GameGolog semantics.

Similarly to Ł-language $\langle\langle G\rangle\rangle \bigcirc \varphi$ operator, the program-constrained $\ll G \gg \bigcirc \varphi$ operator is defined in terms of GameGolog program semantics as follows:

$$\ll G \gg \bigcirc \varphi \doteq$$

$$(\exists agt \in G, a.\exists \rho'.\ agent(a) = agt \wedge Trans(\rho_{now}, now, \rho', do(a, now)) \wedge \varphi[\rho', do(a, now)]) \vee$$

$$(\forall agt \notin G, a.\exists \rho'.\ (agent(a) = agt \wedge Trans(\rho_{now}, now, \rho', do(a, now)) \supset \varphi[\rho', do(a, now)])$$

A formula $\widehat{\Psi}$ in Ł$_p$ logic is related to an Ł-formula from Ł logic (as described earlier) by suppressing the current situation argument *now* and the assumption of a suppressed argument of the current program ρ_{now} still to execute in situation *now*. It should be noted that if the initial situation S_0 and the GameGolog program ρ_0 for the problem of interest are given then ρ_S for situation S is functionally determined. Following the theorem from De Giacomo, Lespérance and Pearce [DLP10] we have that for every GameGolog theory $\mathcal{D}_{GGT}$ and associated program ρ_0, and Ł-formula Ψ, the corresponding Ł$_p$ formula $\widehat{\Psi}$ is such that:

$$\mathcal{D}_{GGT} \models \forall \rho, s.Trans^*(\rho_0, S_0, \rho, s) \supset (\Psi[s] \equiv \widehat{\Psi}[\rho, s])$$

EXAMPLE:

Example Temporal Property in Ł$_P$ Logic of the TTT (Tic-Tac-Toe) Game

The GameGolog Theory axiomatization $\mathcal{D}_{GGT}$ of the TTT (Tic-Tac-Toe) Game uses all the same axioms as the Basic Action Theory for the TTT Game (already provided in this thesis) with the addition GameGolog semantics axioms D_{GG} and the following axioms:

```
$\rho_{TTT} =$
    **while** $\neg Finished()$ **do** (
        $[X\ \pi r, c.\ move(X, r, c)]$;
        **if** $\neg Finished()$ **then** $[O\ \pi r, c.\ move(O, r, c)]$ **else** $True?$
    )
```

$Legal(s) \equiv \exists \rho'.Trans^*(\rho_{TTT}, S_0, \rho', s)$

$Final(s) \equiv \exists \rho'.Trans^*(\rho_{TTT}, S_0, \rho', s) \wedge Final(\rho', s)$

$\mathcal{D}_{GGT} \models \forall \rho, s.Trans^*(\rho_0, S_0, \rho, s) \supset (\Psi[s] \equiv \widehat{\Psi}[\rho, s])$

The property of having a strategy to achieve $\varphi(\vec{x})$ by group G, where $\varphi(\vec{x})$ is a situation suppressed formula with free variables $\vec{x}$, can be defined by the following least fixpoint construction:

$$\ll G \gg \Diamond \varphi(\vec{x}) \doteq \mu Z(\vec{x}).\ \varphi(\vec{x}) \vee \ll G \gg \bigcirc Z(\vec{x})$$

□

2.7 Verification of Properties of Systems in Game-Theoretic Logics

2.7.1 Symbolic Manipulation

De Giacomo, Lespérance and Pearce [DLP10] propose the method to verify if formulas of logic Ł are satisfied in a situation calculus-based game structure. The method is based on 3 elements: regression in situation calculus (Pirri abd Reiter [PR99]), the fixpoint approximation, and classical Knaster and Tarski results [Tar55]. The method makes the assumption of finite set of action types and agents which is the case for typical games structures. Regression is needed to keep the computed formulas to be situation-uniform i.e. to talk about the same situation by eliminating the *do* function. In principle regression is to utilize successor state axioms to replace the *do* situation calculus constructs.

The method is based on the observation that least fixpoint approximation of a formula can be sometimes computed by the general technique of fixpoint approximation (Knaster and Tarski [Tar55])

$Z_0 \doteq \Psi(False)$

$Z_1 \doteq \Psi(Z_0)$

$Z_2 \doteq \Psi(Z_1)$

$\ldots$

where all of Z_i are situation suppressed formulas that talk about the same situation and the computation may require situation calculus regression to achieve that.

To verify an Ł-formula Ψ in situation S is to check if the formula holds in situation S. Now if $\mathcal{D}_G S$ is a situation calculus game structure and S is a situation, then

if for some i $\mathcal{D}_{GS} \models Z_{i+1}[S] \equiv Z_i[S]$ then $\mathcal{D}_{GS} \models \mu Z.\Psi(Z)[S]$

and therefore verification of Ł-formula is equivalent to translating the Ł-formula to a situation calculus situation-uniform formula and checking if this formula holds in a given situation.

2.7.2 $\tau(\Psi)$ Procedure for Ł-Logic Formulas

De Giacomo, Lespérance and Pearce [DLP10] propose a procedure to verify if formulas of logic Ł are satisfied in a situation calculus-based game structure. This recursive procedure $\tau(\Psi)$ tries to compute a first-order formula uniform in current situation *now* and that is equivalent to Ψ:

- $\tau(\phi) = \phi$

  where ϕ is an arbitrary situation-suppressed situation calculus uniform formula

- $\tau(Z) = Z$

  where Z is a predicate variable

- $\tau(\Psi_1 \wedge \Psi_2) = \tau(\Psi_1) \wedge \tau(\Psi_2)$

- $\tau(\Psi_1 \vee \Psi_2) = \tau(\Psi_1) \vee \tau(\Psi_2)$

- $\tau(\exists x.\Psi) = \exists x.\tau(\Psi)$

- $\tau(\forall x.\Psi) = \forall x.\tau(\Psi)$

- $\tau(\langle\langle G\rangle\rangle \bigcirc \Psi) = \mathcal{R}(\langle\langle G\rangle\rangle \bigcirc \tau(\Psi))$

  where $\mathcal{R}$ represents regression operation and $\langle\langle G\rangle\rangle \bigcirc \Psi$ is regressible if Ψ is regressible

- $\tau([[G]] \bigcirc \Psi) = \neg\mathcal{R}(\langle\langle G\rangle\rangle \bigcirc \tau(NNF(\neg\Psi)))$

  where NNF stands for negation normal form of $\neg\Psi$ with the provison that for variables $NNF(Z) \doteq Z$

- $\tau(\mu Z.\Psi) = lfpZ.\tau(\Psi)$

  where $lfpZ.\Psi$ is the formula R resulting from the least fixpoint procedure:

```
$R := False$;
$R_{new} := \Psi(False)$;
**while** ($\mathcal{D}_{ca} \not\models R \equiv R_{new}$) {
    $R := R_{new}$;
    $R_{new} := \Psi(R)$;
}
```

  The procedure tests if $R \equiv R_{new}$ under unique name and domain closure assumptions for actions in $\mathcal{D}_{GS}$. In general there is no guarantee that the procedure will ever stop i.e. that $\mathcal{D}_{ca} \models R_i \equiv R_{i+1}$, but if it does then $\mathcal{D}_{GS} \models R_i[S] \equiv \mu Z.\Psi(Z)[S]$ and R_i is first-order and uniform in S.

  Also $\mathcal{D}_{GS} \models R_i[S_0]$ iff $\mathcal{D}_{S_0} \cup \mathcal{D}_{ca} \models R_i[S_0]$ that is if $\mathcal{D}_{S_0} \cup \mathcal{D}_{ca} \models R_i[S_0]$ then $\mathcal{D}_{GS} \models \mu Z.\Psi(Z)[S_0]$. This means that the task of verifying a fixpoint formula in the situation calculus is reduced to verifying a first-order formula.

- $\tau(\nu Z.\Psi) = gfpZ.\tau(\Psi)$

  where $gfpZ.\Psi$ is the formula R resulting from the greatest fixpoint procedure:

```
$R := True$;
$R_{new} := \Psi(True)$;
```

```
while ($\mathcal{D}_{ca} \not\models R \equiv R_{new}$) {
    $R := R_{new}$;
    $R_{new} := \Psi(R)$;
}
```

the procedure tests if $R \equiv R_{new}$ under unique name and domain closure assumptions for actions in $\mathcal{D}_{GS}$. In general there is no guarantee that the procedure will ever stop i.e. that $\mathcal{D}_{ca} \models R_i \equiv R_{i+1}$, but if it does then $\mathcal{D}_{GS} \models R_i[S] \equiv \mu Z.\Psi(Z)[S]$ and R_i is first-order and uniform in S.

Also $\mathcal{D}_{GS} \models R_i[S_0]$ iff $\mathcal{D}_{S_0} \cup \mathcal{D}_{ca} \models R_i[S_0]$ that is if $\mathcal{D}_{S_0} \cup \mathcal{D}_{ca} \models R_i[S_0]$ then $\mathcal{D}_{GS} \models \nu Z.\Psi(Z)[S_0]$. This means that the task of verifying a fixpoint formula in the situation calculus is reduced to verifying a first-order formula.

Now if $\mathcal{D}_{GS}$ is situation calculus game structure and Ψ is an Ł-formula then if the algorithm τ terminates then

$\mathcal{D}_{GS} \models \Psi[S_0]$ iff $\mathcal{D}_{S_0} \cup \mathcal{D}_{ca} \models \tau(\Psi)[S_0]$

It should benoted that for the least fixpoint formula $\langle\langle G\rangle\rangle\Diamond\varphi$ the fixpoint approximations are:

$Z_0 \doteq \phi \vee \langle\langle G\rangle\rangle \bigcirc False$ i.e. $Z_0 \doteq \phi$

$Z_1 \doteq \phi \vee \langle\langle G\rangle\rangle \bigcirc Z_0$

$Z_2 \doteq \phi \vee \langle\langle G\rangle\rangle \bigcirc Z_1$

...

In procedure τ the regression is applied at each step of the computation of the approximate so the formulas $Z_i[S]$ are equivalent to the corresponding formulas $R_i[S]$:

$R_0 \doteq \phi$

$R_1 \doteq \phi \vee \mathcal{R}[\langle\langle G\rangle\rangle \bigcirc R_0]$

$R_2 \doteq \phi \vee \mathcal{R}[\langle\langle G\rangle\rangle \bigcirc R_1]$

...

The formulas $R_i[S]$ are equivalent to the corresponding formulas $Z_i[S]$ the difference between formula $R_i[S]$

and formula $Z_i[S]$ is that in $R_i[S]$ the only situation term that appears is S whereas $Z_i[S]$ may have S and other situation terms that may be up to i steps in the future.

2.7.3 $[\![\Psi]\!]$ Procedure for $Ł_p$-Logic Formulas

De Giacomo, Lespérance and Pearce [DLP10] propose a procedure to verify if formulas of logic $Ł_p$ are satisfied in a GameGolog game structure. The procedure is based on regression (Pirri abd Reiter [PR99]), fixpoint approximation, and characteristic graphs verification methods proposed by Claßen and Lakemeyer [CL08].

When looking at characteristic graphs, which have been described earlier in this thesis, and the nature of game problems it can be observed that for GameGolog characteristic graph $\mathcal{G}$ all outgoing edges of every node v will be labelled by actions of same agent. That agent controls the node and the agent of the node can be denoted as $agent(v)$. Now the technique is based on recursive procedure denoted $[\![\Psi]\!]$ that labels nodes of characteristic graph $\mathcal{G}$ for any $Ł_p$-formula Ψ. An assumption is made that free variables occurring in formulas to be checked are distinct from those occurring in GameGolog program ρ_0 that are quantified by the π construct. Also if the procedure $[\![\Psi]\!]$ terminates then it produces labelling set $\mathcal{Z} = \{\langle v, \phi\rangle | v \in \mathcal{G}\}$ of nodes in the graph $\mathcal{G}$ where ϕ are first-order formulas and this labelling can be used to check whether the property of interest Ψ holds. The procedure uses a few definitions that need to be introduced first:

- $[\![\varphi]\!] \doteq \{\langle v, \varphi\rangle | v \in \mathcal{G}\}$ where φ is any first-order formula
- $\mathcal{Z}_1$ AND $\mathcal{Z}_2 \doteq \{\langle v, \phi_1 \wedge \phi_2\rangle | \langle v, \phi_1\rangle \in \mathcal{Z}_1, \langle v, \phi_2\rangle \in \mathcal{Z}_2\}$
- $\mathcal{Z}_1$ OR $\mathcal{Z}_2 \doteq \{\langle v, \phi_1 \vee \phi_2\rangle | \langle v, \phi_1\rangle \in \mathcal{Z}_1, \langle v, \phi_2\rangle \in \mathcal{Z}_2\}$
- EXISTS $x.\mathcal{Z} \doteq \{\langle v, \exists x.\phi\rangle | \langle v, \phi\rangle \in \mathcal{Z}\}$
- ALL $x.\mathcal{Z} \doteq \{\langle v, \forall x.\phi\rangle | \langle v, \phi\rangle \in \mathcal{Z}\}$

- $Pre(G, \mathcal{Z}) \doteq \{$

  $\langle v, \phi \rangle | v \in \mathcal{G}$, where

  if $agent(v) \in G$ then

  $\phi = \bigvee_{v \overset{\pi \vec{x}:\alpha,\omega}{\longrightarrow} v' \in \mathcal{G}, (v',\phi') \in \mathcal{Z}} \exists \vec{x}.\omega(\vec{x}) \wedge \mathcal{R}(\phi'(do(\alpha, now)))$

  and if $agent(v) \notin G$ then

  $\phi = \bigvee_{v \overset{\pi \vec{x}:\alpha,\omega}{\longrightarrow} v' \in \mathcal{G}, (v',\phi') \in \mathcal{Z}} \exists \vec{x}.\omega(\vec{x}) \wedge$

  $\bigwedge_{v \overset{\pi \vec{x}:\alpha,\omega}{\longrightarrow} v' \in \mathcal{G}, (v',\phi') \in \mathcal{Z}} \forall \vec{x}.\omega(\vec{x}) \supset \mathcal{R}(\phi'(do(\alpha, now)))$

  $\}$

- $\overline{Pre}(G, \mathcal{Z}) \doteq \{\langle v, \text{NNF}(\neg\phi) \rangle | \langle v, \phi \rangle \in Pre(G, \mathcal{Z})\}$

- LFP$\mathcal{Z}.\Psi(\mathcal{Z})$, where $\Psi(\mathcal{Z})$ denotes an parametrized expression in which $\mathcal{Z}$ occurs as a parameter (possibly together with other parameters), stands for the result of the following procedure (in which $\mathcal{Z} \neq \mathcal{Z}_{new}$ is an abbreviation for $\mathcal{D}_{ca} \not\models \bigwedge_{\langle v,\varphi \rangle \in \mathcal{Z}, \langle v,\varphi_{new} \rangle \in \mathcal{Z}_{old}} \varphi \equiv \varphi_{new}$):

```
$\mathcal{Z} := [\![False]\!]$;
$\mathcal{Z}_{new} := [\![\Psi(\mathcal{Z})]\!]$;
while ($\mathcal{Z} \neq \mathcal{Z}_{new}$){
    $\mathcal{Z} := \mathcal{Z}_{new}$;
    $\mathcal{Z}_{new} := [\![\Psi(\mathcal{Z})]\!]$
}
```

- GFP$\mathcal{Z}.\Psi(\mathcal{Z})$, where $\Psi(\mathcal{Z})$ denotes a parametrized expression in which $\mathcal{Z}$ occurs as a parameter, stands for the result of the following procedure:

```
$\mathcal{Z} := [\![True]\!]$;
$\mathcal{Z}_{new} := [\![\Psi(\mathcal{Z})]\!]$;
while ($\mathcal{Z} \neq \mathcal{Z}_{new}$){
    $\mathcal{Z} := \mathcal{Z}_{new}$;
    $\mathcal{Z}_{new} := [\![\Psi(\mathcal{Z})]\!]$
```

}

The general labelling procedure $[\![\Psi]\!]$ for any $Ł_p$ formula Ψ on a characteristic graph $\mathcal{G}$ is as follows:

- $[\![\varphi]\!]$ as defined earlier where φ is first-order formula
- $[\![\mathcal{Z}]\!] \doteq \mathcal{Z}$ where $\mathcal{Z}$ is any labeling
- $[\![\Psi_1 \wedge \Psi_2]\!] \doteq [\![\Psi_1]\!]$ AND $[\![\Psi_2]\!]$
- $[\![\Psi_1 \vee \Psi_2]\!] \doteq [\![\Psi_1]\!]$ OR $[\![\Psi_2]\!]$
- $[\![\exists x.\Psi]\!] \doteq$EXISTS $x.[\![\Psi]\!]$
- $[\![\forall x.\Psi]\!] \doteq$ALL $x.[\![\Psi]\!]$
- $[\![\ll G \gg \bigcirc \Psi]\!] \doteq Pre(G, [\![\Psi]\!])$
- $[\![[[G]] \bigcirc \Psi]\!] \doteq \overline{Pre}(G, [\![\Psi]\!])$
- $[\![\mu Z.\Psi(Z)]\!] \doteq$ LFP$\mathcal{Z}.[\![\Psi(Z)]\!]$
- $[\![\nu Z.\Psi(Z)]\!] \doteq$ GFP$\mathcal{Z}.[\![\Psi(Z)]\!]$

Based on a theorem from De Giacomo, Lespérance and Pearce [DLP10], when $[\![\Psi]\!]$ terminates, the uniform formulas that label the nodes in the resulting labeling can be used to check whether Ψ holds:

> For every $Ł_p$-formula Ψ, if $[\![\Psi]\!]$ terminates and $\langle v, \varphi\rangle$, with $v = \langle \rho, \chi\rangle$, is in the returned set,
>
> then for all situation terms s, $\mathcal{D}_{GGT} \models \Psi[\rho, s] \equiv \varphi[s]$.
>
> For every $Ł_p$-formula Ψ, if $[\![\Psi]\!]$ terminates and $\langle v_0, \varphi\rangle$, with $v = \langle \rho_0, \chi_0\rangle$, is in the returned set,
>
> then $\mathcal{D}_{GGT} \models \Psi[\rho_0, S_0]$ iff $\mathcal{D}_{S_0} \cup \mathcal{D}_{ca} \models \varphi[S_0]$.

2.8 Other Verification Methods

There is an increasing interest in producing proofs of correctness of systems by automated means. Formal methods [Mon01] and [BBS09] are a set of mathematically-based techniques for the specification, development and verification of software and hardware systems. Automated verification techniques fall into two general categories. The first category is the automated theorem proving, in which a system attempts to produce a formal proof from scratch, given a description of the system, a set of logical axioms, and a set of inference rules. The second category is model checking, in which a system verifies certain properties by means of an exhaustive search of all possible states that a system could enter during its execution. Model checkers can quickly get bogged down in checking millions of uninteresting states if not given a sufficiently abstract model. Quite often they make a pronouncement of truth, yet give no explanation of that truth and also there is the problem of "verifying the verifier" (if the program which aids in the verification is itself unproven, there may be reason to doubt the soundness of the produced results). In general, existing verification methods require significant manual preparations and time to produce, use low-level specifications of the systems, and most are are limited to finite state structures. The following sections briefly explain the main verification methods that can be used in verifying temporal properties of game problems.

2.8.1 Theorem Proving

Theorem proving approaches to verification are in competition to our approach but they are not mainstream today. There is not much work on game-theoretic logic theorem proving in general. Some of the theorem proving techniques used these days are based on first-order resolution with unification and follow the work of [Hoa69], while some others such as [BKM95] are based on mathematical induction. The work involved in producing a good proof requires a high level of mathematical sophistication and expertise and standard theorem proving approaches are difficult to automate completely [KMM00] and [Mil06].

Currently many applications use Satisfiability Modulo Theory (SMT) [Ume10] which has its roots in automated theorem proving. SMT greatly relies on the SAT technology and can be seen as an extension of SAT solving. In essence in SMT an instance of the Boolean satisfiability problem (SAT) has some of the binary variables replaced by predicates over a suitable set of non-binary variables. SMT is the problem of determining whether an SMT formula is satisfiable.

2.8.2 Model Checking

While theorem proving uses axioms and can work with possibly incomplete specification for property verification, model checking uses models and requires complete specification. Non-symbolic model checking techniques [CE81] explore program states and state transitions by performing an enumeration. In practice they are often hampered by severe "state space explosion" – a situation where the state space grows dramatically with the evaluation of state transitions. To tame this problem, symbolic algorithm research [McM92] gained focus and an example of such research for temporal logics is the work of Lomuscio and Penczek [LP12]. In symbolic model checking, sets of states (a state represented as a vector of state variables) are manipulated rather than individual states. These sets are encoded as formulas that are satisfied if and only if the subject states belong to the represented set (for example a formula $x_1 \vee x_2 \vee x_3$ would represent 7 states in an 8 state space with 3 variables). Symbolic representation of sets of states can be much more succinct then the corresponding enumeration. State exploration is performed via the symbolic transformation of the mentioned set representations. The power of symbolic techniques comes from significant advances in the performance of constraint solvers. These solvers underline symbolic checking technologies [CABN97] where the system model and the property to verify are turned into a formula which satisfiability needs to be checked in the end. Temporal properties and fixpoint computations become iterative formula manipulations. For finite state systems the verification and falsification results are guaranteed. Here the formulas can be represented and evaluated as binary decision diagrams (BDDs) [Ake78] providing efficient implementations. Still, to construct BDDs manual intervention may be required, as some Boolean operations and existential quantification can be quadratic in the size of the BDD, and the size of the BDD is sensitive to the variable ordering and in the wost case can be exponential in the number of variables. This is the symbolic analogue to the state space explosion problem. Some of the popular symbolic model checkers are: SPIN [Hol03], NUSMV [CCGR00], and Java Pathfinder [Hav99]. A leading example of a symbolic model checker for multi-agent systems is MCMAS (Lomuscio, Qu and Raimondi [LQR09]). Two general references on model checking are Clarke et al. [CGP01] and Baier and Katoen [BK08].

2.8.3 Bounded Model Checking

Bounded Model Checking (Biere [Bie09]), acronymed as BMC, uses symbolic representation for sets of states and the transition relation in the form of Boolean logic formulas. Just like symbolic model checkers, BMC operates on symbolic (as opposed to concrete) representations. For temporal logic properties, instead of computing fixpoints, it unrolls the transition relation up to certain fixed bound and searches for violations (counter examples) of the property within such unrolled formula. The search is performed as a propositional satisfiability problem and is using modern SAT and Satisfiability Modulo Theory (SMT) solvers [Ume10] that have become quite efficient. It can be shown that if a temporal property holds for a finite state transition system then then it also holds for that system using bounded semantics for some bound k, so if we keep increasing the bound we will find a path that satisfies the formula if such path exists. Now if there is no solution then the procedure would not terminate but quite often we can define the diameter of the transition system so that the property holds if and only if it is not violated on a path bounded by the diameter (for example for the Tic-Tac-Toe game the maximum number of consecutive moves is 9 and therefore it could be assumed as the diameter of its transition system). In bounded model checking the falsification of a property can be guaranteed and verification of a property can be guaranteed if we know the binding diameter of the system. It needs to be noted that variable domains need to be bounded otherwise would not be able to convert the problems to Boolean SAT. It is also worth noting that bounded model checking usually finds counterexamples fast (due to the depth-first approach of SAT solvers) and it finds counterexamples of minimal length which makes such counterexamples more easily understandable. BMC also does not require the manual intervention that was needed in constructing BDDs for symbolic model checking.

2.8.4 Program Synthesis

Another direction in verification is design synthesis from a temporal logic specification (Piterman, Pnueli and Sa'ar [PPS06]). The ability to synthesize a program from a given specification that includes the desired temporal properties can be considered a proof that such program exhibits the desired temporal properties. Although the general complexity of such synthesis can be doubly exponential, in Piterman, Pnueli and Sa'ar [PPS06] a more efficient technique is proposed for a special type of temporal formulas – the General Reactivity (1) formulas. Although it is a restricted class of temporal logic formulas, this type of formulas addresses a large number of practical problems. In Piterman, Pnueli and Sa'ar [PPS06] the realizablility of the synthesis of a temporal formula is reduced to the decision of a winner in games. Two player games are considered that are played between a system and an environment, and the goal of the system is to satisfy the specification regardless of the actions of the environment. The game is solved by attempting to decide whether the system has a strategy that guarantees winning. If the environment can win then the system is unrealizable. If the system is guaranteed to win then the winning strategy is synthesized into an implementation. It should be noted that in the presence of incomplete information about the initial situation, getting synthesis techniques out of verification techniques is much more complex. Strategies not only need to exists (as guaranteed by verification), but also need to be epistemically feasible, i.e. the agent performing a strategy needs to have sufficient information to actually make all the choices needed for its execution.

2.8.5 Infinite-States Domains

To deal with verification in infinite-states domains, methods based on fixpoint approximation and characteristic graphs have been proposed by Claßen and Lakemeyer [CL08]. There, a concept of characteristic graphs is proposed as a way to represent all possible configurations that a program written in Golog to represent a multi-agent interaction may visit. A characteristic graph is constructed for a given program where nodes represent program configurations and edges are labelled with actions and conditions under which these actions can be taken. There is no need to ground the program configurations and transitions to obtain its characteristic graph. Conditions involving temporal operators are then tested by computing fixpoint with respect to these graphs, using methods adapted from symbolic model checking. They also provide an automated method to verify typical program properties. However their specification language is not a game structure logic. Interesting methods for specifying game structures in a logical and procedural manner have been also proposed and potential verification techniques have been outlined by De Giacomo, Lespérance and Pearce [DLP10]. In there, a logical framework for specifying and solving game theoretic problems based on the situation calculus and the ConGolog agent programming language is proposed. Also a new programming language GameGolog is proposed to allow better modelling of game-like or multi-agent interaction problems using a procedural formalism. Additionally a language based on the μ-calculus, game-theoretic path quantifiers, and first-order logic for specifying program properties is proposed. They also devise techniques for the verification of properties over game structures by using fixpoint approximation, as well as, verification of properties over GameGolog programs by using characteristic graphs. These techniques apply to incomplete specifications of game structures and infinite-states domains.

3 Symbolic Manipulation-Based Verification of Properties of Game Structures

This section presents the first part of the research conducted for this thesis. Several game domains have been developed that have various levels of complexity and are representative of problems for which game logics are used. These domains range from simple setups such as arrays of controlable lights to simple infinite state variants of typical games such as a version of tic-tac-toe played on an infinite vector of cells. Many involve incomplete specification of the initial state. The purpose of such diversity is twofold: to have a range of problems that the techniques will be tested on, and to determine the effectiveness of the techniques for problems of varying characteristics. These domains have then been modelled in game theories in the situation calculus, as well as, in GameGolog. The symbolic fixpoint approximation method (De Giacomo, Lespérance and Pearce [DLP10]) has been applied on the created problems to examine if it works in verifying ATL-type properties of finite and infinite-states game structures. This has been done on complete and incomplete game theories (classes of games) and the fixpoint evaluation convergence has been examined to extract the required conditions and strategies for verifying desired properties.

The developed games have an infinite state space. For the majority of them the verification procedure applied to the selected temporal properties, i.e., the possibility of winning and the existence of a winning strategy for one player, converges to a fixpoint in a small finite number of steps and we can show the successive approximations become equivalent without using the initial state axioms, i.e., without any knowledge of the initial state. In one case we did need to use our knowledge of the initial situation to get the fixpoint approximation to terminate in a finite number of steps; without such knowledge, the fixpoint approximation formulas keep growing covering more situations indefinitely. The formulas obtained by the fixpoint procedures show what needs to be true in the initial situation if the given properties are to hold.

3.1 Light World (LW)

In this section we verify some temporal properties using the fixpoint approximation method of De Giacomo, Lespérance and Pearce [DLP10] with one small modification: when checking whether the two successive approximates are equivalent, we assume that we have a suitable axiomatization of the integers D_Z in addition to the unique names and domain closure axioms for actions (as our game domain involves one light that may be on or not for every integer). [1]

The Light World (LW) game that we have designed involves an infinite row of lights – one for each integer. A light can be on or off. A light has a switch that can be flipped and will turn the light off if it was on or it will turn the light on if it was off. There are 2 players X and O in the game. Players take turns and initially it is player X's turn. The goal of player X is to have lights 1 and 2 on in which case player X wins. It is possible for the game to go on forever and the goal cannot be reached: player O keeps switching off light 1 or light 2 (depending which one happens to be on) and in the initial situation both lights 1 and 2 are off.

3.1.1 Situation Calculus Game Structure Axiomatization of LW Domain

The Situation Calculus Game Structure Axiomatization is defined as:

$$\mathcal{D}_{GS}^{LW} = \Sigma \cup \mathcal{D}_{poss}^{LW} \cup \mathcal{D}_{ssa}^{LW} \cup \mathcal{D}_{ca}^{LW} \cup \mathcal{D}_{S_0}^{LW} \cup \mathcal{D}_{Legal}^{LW} \cup \mathcal{D}_Z$$

where $\mathcal{D}_Z$ is a suitable axiomatization of the integers as discussed earlier.

[1] Our axioms and the properties we attempt to verify only use a very simple part of integer arithmetic [End72] and [Ham82]. It should be possible to generate the proofs using the decidable theory of Presburger arithmetic after encoding integers as pairs of natural numbers in the standard way. Most theorem proving systems include sophisticated solvers for dealing with formulas involving integer constraints and it should be possible to use these to perform the reasoning about integers that we require.

Fluents

- $On(t, s)$ - predicate fluent indicating whether the light t is on in situation s
- $turn(s)$ - functional fluent indicating which agent is to take the next action, with its domain being the agent set $\{X, O\}$

Derived Fluents

- $Wins(p, s) \doteq Legal(s) \land On(1, s) \land On(2, s)$
- $Finished(s) \doteq \exists p.\ Wins(p, s)$

  This fluent is not used in this chapter but later in Chapter 4.

Actions

- $flip(p, t)$ - agent p flips the switch of light t

Precondition Axioms $\mathcal{D}^{LW}_{poss}$

These are precondition axioms, one per action, specifying when an action can physically be performed:

- $Poss(flip(p, t), s) \equiv Agent(p)$

Successor State Axioms $\mathcal{D}^{LW}_{ssa}$

These are successor state axioms specifying how the fluents change as a result of actions:

- $On(t, do(a, s)) \equiv \neg On(t, s) \land \exists p.a = flip(p, t) \lor On(t, s) \land \forall p.a \neq flip(p, t)$
- $turn(do(a, s)) = p \equiv p = O \land turn(s) = X \lor p = X \land turn(s) = O$

Initial State Axioms $\mathcal{D}_{S_0}^{LW}$

These describe the initial situation:

- $turn(S_0) = X$
- $On(5, S_0)$
- $Legal(S_0)$

Legal Moves Axioms $\mathcal{D}_{legal}^{LW}$

These encode the rules of the game:

- $agent(flip(p,t)) = p$
- $Control(p,s) \doteq \exists a. Legal(do(a,s)) \wedge agent(a) = p$
- $Legal(do(a,s)) \equiv Legal(s) \wedge \exists p,t.\ Agent(p) \wedge turn(s) = p \wedge a = flip(p,t)$

Unique Name and Domain Closure Axioms for Actions $\mathcal{D}_{ca}^{LW}$

These describe the uniqueness of action names and the fact that the domain of actions s closed:

- $\forall a.\ \{\ \exists p,t.\ a = flip(p,t)\}$
- $\forall p,p',t,t'.\ \{\ flip(p,t) = flip(p',t') \supset p = p' \wedge t = t'\ \}$
- $\forall p.\ \{\ Agent(p) \equiv (p = X \vee p = O)\}$
- $X \neq O$

The complete theory also includes the foundational axioms Σ of the Situation Calculus. It can be noticed that this is clearly an infinite state domain as the set of light that can be turned on or off is infinite.

Propositions and Lemmas

We derived the following propositions, some of which are used in later proofs:

P 3.1.1. *Regression for concrete cases for the turn functional fluent.*

$\mathcal{D}_{ca}^{LW} \models \mathcal{R}(turn(do(flip(X,t),s)) = X) \equiv False$

$\mathcal{D}_{ca}^{LW} \models \mathcal{R}(turn(do(flip(O,t),s)) = X) \equiv turn(s) = O$

$\mathcal{D}_{ca}^{LW} \models \mathcal{R}(turn(do(flip(X,t),s)) = O) \equiv turn(s) = X$

$\mathcal{D}_{ca}^{LW} \models \mathcal{R}(turn(do(flip(O,t),s)) = O) \equiv False$

P 3.1.2. *Regression of a concrete case for the On fluent.*

$\mathcal{D}_{ca}^{LW} \cup \mathcal{D}_Z \models \mathcal{R}(On(v, do(flip(p,t),s))) \equiv \neg On(v,s) \wedge t = v \vee On(v,s) \wedge t \neq v$

P 3.1.3. *Regression of a concrete case for the Legal fluent.*

$\mathcal{D}_{ca}^{LW} \models \mathcal{R}(Legal(do(flip(p,t),s))) \equiv Legal(s) \wedge turn(s) = p$

In this domain, for groups $G = \{X\}$ and $G = \{X, O\}$ ensuring that φ holds next can be established by considering only the $flip(t)$ actions as the following lemmas show:

L 3.1.1. *Regression of the concrete case of the $\exists \bigcirc \varphi$ definition based on domain closure.*

$\mathcal{D}_{ca}^{LW} \models \mathcal{R}(\exists \bigcirc \varphi[s]) \equiv$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\exists t.\varphi[do(flip(X,t),s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\exists t.\varphi[do(flip(O,t),s)])$

Proof Sketch: this is the case of the $\langle\langle G \rangle\rangle \bigcirc \varphi$ operator where all agents (X and O) are in the coalition G, leaving no agent outside the coalition. Then by the definition of $\langle\langle G \rangle\rangle \bigcirc \varphi$ from [DLP10]

$\mathcal{R}(\langle\langle G \rangle\rangle \bigcirc \varphi) \doteq$

$(\bigvee_{agt \in G} \mathcal{R}(Control(agt,s)) \wedge \bigvee_{a \in A} \exists \vec{x}.\ agent(a(\vec{x})) = agt \wedge$

$\mathcal{R}(Legal(do(a(\vec{x}),s))) \wedge \mathcal{R}(\varphi(do(a(\vec{x}),s)))) \vee$

$(\bigvee_{agt \notin G} \mathcal{R}(Control(agt,s)) \wedge \bigwedge_{a \in A} \forall \vec{x}.\ agent(a(\vec{x})) = agt \wedge$

$\mathcal{R}(Legal(do(a(\vec{x}), s))) \supset \mathcal{R}(\varphi(do(a(\vec{x}), s))))$

$\equiv$

$(\bigvee_{agt \in G} \mathcal{R}(Control(agt, s)) \wedge \bigvee_{a \in A} \exists \vec{x}.\ agent(a(\vec{x})) = agt \wedge$

$\mathcal{R}(Legal(do(a(\vec{x}), s))) \wedge \mathcal{R}(\varphi(do(a(\vec{x}), s)))) \vee$

$\equiv$ *by enumerating the agents and* $\mathcal{R}$ *for* $Control(agt, s)$ *and the definition of the agent function*

$Control(X, s) \wedge \bigvee_{a \in A} \exists t.\ a = flip(X, t) \wedge \mathcal{R}(Legal(do(a, s))) \wedge \mathcal{R}(\varphi(do(a, s))) \vee$

$Control(O, s) \wedge \bigvee_{a \in A} \exists t.\ a = flip(O, t) \wedge \mathcal{R}(Legal(do(a, s))) \wedge \mathcal{R}(\varphi(do(a, s))))$

$\equiv$ *by FOL and combining the quantifiers*

$Control(X, s) \wedge \exists t.\ \mathcal{R}(Legal(do(flip(X, t), s))) \wedge \mathcal{R}(\varphi(do(flip(X, t), s))) \vee$

$Control(O, s) \wedge \exists t.\ \mathcal{R}(Legal(do(flip(O, t), s))) \wedge \mathcal{R}(\varphi(do(flip(O, t), s))))$

$\equiv$ *by definition of* $Control$ *and P3.1.3*

$Legal(s) \wedge turn(s) = X \wedge \exists t.\ Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\varphi(do(flip(X, t), s))) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists t.\ Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\varphi(do(flip(O, t), s)))$

$\equiv$ *by FOL*

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\exists t. \varphi(do(flip(X, t), s))) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\exists t. \varphi(do(flip(O, t), s)))$ □

L 3.1.2. *Regression of the concrete case of the* $\langle\langle\{X\}\rangle\rangle \bigcirc \varphi$ *definition based on domain closure.*

$\mathcal{D}_{ca}^{LW} \models \mathcal{R}(\langle\langle\{X\}\rangle\rangle \bigcirc \varphi[s]) \equiv$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\exists t. \varphi[do(flip(X, t), s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\forall t. \varphi[do(flip(O, t), s)])$

Proof Sketch: this is the case of the $\langle\langle G\rangle\rangle \bigcirc \varphi$ *operator where agent X is in the coalition G, leaving agent O outside the coalition. Then by the definition of* $\langle\langle G\rangle\rangle \bigcirc \varphi$ *from [DLP10]*

$\mathcal{R}(\langle\langle G\rangle\rangle \bigcirc \varphi) \doteq$

$(\bigvee_{agt \in G} \mathcal{R}(Control(agt, s)) \wedge \bigvee_{a \in A} \exists \vec{x}.\ agent(a(\vec{x})) = agt \wedge$

$\mathcal{R}(Legal(do(a(\vec{x}), s))) \wedge \mathcal{R}(\varphi(do(a(\vec{x}), s)))) \vee$

$(\bigvee_{agt \notin G} \mathcal{R}(Control(agt, s)) \wedge \bigwedge_{a \in A} \forall \vec{x}.\ agent(a(\vec{x})) = agt \wedge$

$\quad \mathcal{R}(Legal(do(a(\vec{x}), s))) \supset \mathcal{R}(\varphi(do(a(\vec{x}), s))))$

$\equiv$ *by enumerating the agents and $\mathcal{R}$ for $Control(agt, s)$*

$Control(X, s) \wedge \bigvee_{a \in A} \exists \vec{x}.\ agent(a(\vec{x})) = X \wedge \mathcal{R}(Legal(do(a(\vec{x}), s))) \wedge \mathcal{R}(\varphi(do(a(\vec{x}), s))) \vee$

$Control(O, s) \wedge \bigwedge_{a \in A} \forall \vec{x}.\ agent(a(\vec{x})) = O \wedge \mathcal{R}(Legal(do(a(\vec{x}), s))) \supset \mathcal{R}(\varphi(do(a(\vec{x}), s)))$

$\equiv$ *by enumerating the actions and the definition of the agent function*

$Control(X, s) \wedge \exists t.\mathcal{R}(Legal(do(flip(X, t), s))) \wedge \mathcal{R}(\varphi(do(flip(X, t), s))) \vee$

$Control(O, s) \wedge \forall t.\mathcal{R}(Legal(do(flip(O, t), s))) \supset \mathcal{R}(\varphi(do(flip(O, t), s)))$

$\equiv$ *by definition of Control*

$Legal(s) \wedge turn(s) = X \wedge \exists t.\mathcal{R}(Legal(do(flip(X, t), s))) \wedge \mathcal{R}(\varphi(do(flip(X, t), s))) \vee$

$Legal(s) \wedge turn(s) = O \wedge \forall t.\mathcal{R}(Legal(do(flip(O, t), s))) \supset \mathcal{R}(\varphi(do(flip(O, t), s)))$

$\equiv$ *by P3.1.3*

$Legal(s) \wedge turn(s) = X \wedge \exists t.Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\varphi(do(flip(X, t), s))) \vee$

$Legal(s) \wedge turn(s) = O \wedge \forall t.(Legal(s) turn(s) = O \wedge \ \supset \mathcal{R}(\varphi(do(flip(O, t), s))))$

$\equiv$ *by FOL*

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\exists t.\varphi(do(flip(X, t), s))) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\forall t.\varphi(do(flip(O, t), s))))$ $\square$

3.1.2 Possibility of Winning

The property that it is possible for X to eventually win can be represented by the following formula:

$$\exists \Diamond Wins(X) \doteq \mu Z.Wins(X) \ \vee \ \exists \bigcirc Z$$

We begin by applying the De Giacomo et al. [DLP10] method and try to show that successive approximates are equivalent using only the unique name and domain closure axioms for actions $\mathcal{D}_{ca}^{LW}$ and the axiomatization of the integers $\mathcal{D}_Z$:

$\mathcal{D}_{ca}^{LW} \cup \mathcal{D}_Z \models \exists \Diamond Wins(X)$

This formula can be verified by employing the [DLP10] method using regression and fixpoint approximation, until we converge. The technique does not always converge and this needs to be checked as we proceed. The regressed approximations are as follows:

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{LW}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_0(s)} \doteq \mathbf{Wins(X,s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{False}) \equiv$ by lemma L3.1.1

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\exists t.False) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\exists t.False)$

$\equiv$

$Legal(s) \wedge On(1,s) \wedge On(2,s)$

This approximation evaluates to true if s is legal and such that X is winning in s already (in no steps). These are situations where light 1 and light 2 are on. $\square$

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{LW}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_1(s)} \doteq \mathbf{Wins(X,s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{R_0}) \equiv$ by lemma L3.1.1

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\exists t.R_0[do(flip(X,t),s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\exists t.R_0[do(flip(O,t),s)])$

$\equiv$ by R_0 substitution of $\equiv$ from previous step

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\exists t.($

$Legal(do(flip(X,t),s)) \wedge On(1,do(flip(X,t),s)) \wedge On(2,do(flip(X,t),s)))) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\exists t.($

$Legal(do(flip(O,t),s)) \wedge On(1,do(flip(O,t),s)) \wedge On(2,do(flip(O,t),s))))$

$\equiv$ by P3.1.2 and P3.1.3

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists t.(\mathbf{Legal(s)} \wedge \mathbf{turn(s)} = \mathbf{X} \wedge$

$(\neg On(1,s) \wedge t = 1 \vee On(1,s) \wedge t \neq 1) \wedge (\neg On(2,s) \wedge t = 2 \vee On(2,s) \wedge t \neq 2)) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists t.(\mathbf{Legal(s)} \wedge \mathbf{turn(s)} = \mathbf{O} \wedge$

$(\neg On(1,s) \wedge t = 1 \vee On(1,s) \wedge t \neq 1) \wedge (\neg On(2,s) \wedge t = 2 \vee On(2,s) \wedge t \neq 2))$

$\equiv$ by FOL and combining line 2/3 and 3/4

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge (turn(s) = X \vee turn(s) = O) \wedge$

$\exists t.((\neg On(1,s) \wedge t = 1 \vee On(1,s) \wedge t \neq 1) \wedge (\neg On(2,s) \wedge t = 2 \vee On(2,s) \wedge t \neq 2))$

$\equiv$ by distribution of $\wedge$

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge (turn(s) = X \vee turn(s) = O) \wedge \exists t.(\neg On(1,s) \wedge \mathbf{t} = \mathbf{1} \wedge \neg On(2,s) \wedge \mathbf{t} = \mathbf{2}) \vee$

$Legal(s) \wedge (turn(s) = X \vee turn(s) = O) \wedge \exists t.(On(1,s) \wedge \mathbf{t} \neq \mathbf{1} \wedge \neg On(2,s) \wedge \mathbf{t} = \mathbf{2}) \vee$

$Legal(s) \wedge (turn(s) = X \vee turn(s) = O) \wedge \exists t.(\neg On(1,s) \wedge \mathbf{t} = \mathbf{1} \wedge On(2,s) \wedge \mathbf{t} \neq \mathbf{2}) \vee$

$Legal(s) \wedge (turn(s) = X \vee turn(s) = O) \wedge \exists t.(On(1,s) \wedge \mathbf{t} \neq 1 \wedge On(2,s) \wedge \mathbf{t} \neq \mathbf{2})$

$\equiv$ by: inconsistent line 2 as $1 \neq 2$, by FOL and elimination of quantifiers

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge (turn(s) = X \vee turn(s) = O) \wedge On(1,s) \wedge \neg On(2,s) \vee$

$Legal(s) \wedge (turn(s) = X \vee turn(s) = O) \wedge \neg On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge (turn(s) = X \vee turn(s) = O) \wedge On(1,s) \wedge On(2,s)$

$\equiv$ by FOL (line 4 subsumed by line 1, line 2 and 4 combined, line 3 and 4 combined)

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge (turn(s) = X \vee turn(s) = O) \wedge On(1,s) \vee$

$Legal(s) \wedge (turn(s) = X \vee turn(s) = O) \wedge On(2,s)$

$\equiv$ by R_0 substitution of $\equiv$ from previous step

$R_0(s) \vee$

$Legal(s) \wedge (turn(s) = X \vee turn(s) = O) \wedge On(1, s) \vee$

$Legal(s) \wedge (turn(s) = X \vee turn(s) = O) \wedge On(2, s)$

This approximation evaluates to true if s is legal and such that X can win in at most 1 step. These are legal situations where player X wins already or one of lights 1 or 2 is on (X or O can turn the other light at the next step). □

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{LW}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_2(s)} \doteq \mathbf{Wins(X, s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{R_1}) \equiv$ by lemma L3.1.1

$Legal(s) \wedge On(1, s) \wedge On(2, s) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\exists t.R_1[do(flip(X, t), s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\exists t.R_1[do(flip(O, t), s)])$

$\equiv$ by R_1 substitution of $\equiv$ from previous step

$Legal(s) \wedge On(1, s) \wedge On(2, s) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\exists t.($

$R_0(do(flip(X, t), s)) \vee$

$Legal(do(flip(X, t), s)) \wedge (turn(do(flip(X, t), s)) = X \vee turn(do(flip(X, t), s)) = O) \wedge$

$On(1, do(flip(X, t), s)) \vee$

$Legal(do(flip(X, t), s)) \wedge (turn(do(flip(X, t), s)) = X \vee turn(do(flip(X, t), s)) = O) \wedge$

$On(2, do(flip(X, t), s))$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\exists t.($

$R_0(do(flip(O, t), s)) \vee$

$Legal(do(flip(O, t), s)) \wedge (turn(do(flip(O, t), s)) = X \vee turn(do(flip(O, t), s)) = O) \wedge$

$On(1, do(flip(O, t), s)) \vee$

$Legal(do(flip(O, t), s)) \wedge (turn(do(flip(O, t), s)) = X \vee turn(do(flip(O, t), s)) = O) \wedge$

$On(2, do(flip(O, t), s)))$

$\equiv$ by splitting the quantifiers and reorganization

$Legal(s) \wedge On(1, s) \wedge On(2, s) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\exists t.R_0(do(flip(X, t), s)) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\exists t.R_0(do(flip(O, t), s)) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\exists t.Legal(do(flip(X, t), s)) \wedge$

$(turn(do(flip(X, t), s)) = X \vee turn(do(flip(X, t), s)) = O) \wedge On(1, do(flip(X, t), s)) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\exists t.Legal(do(flip(X, t), s)) \wedge$

$(turn(do(flip(X, t), s)) = X \vee turn(do(flip(X, t), s)) = O) \wedge On(2, do(flip(X, t), s)) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\exists t.Legal(do(flip(O, t), s)) \wedge$

$(turn(do(flip(O, t), s)) = X \vee turn(do(flip(O, t), s)) = O) \wedge On(1, do(flip(O, t), s)) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\exists t.Legal(do(flip(O, t), s)) \wedge$

$(turn(do(flip(O, t), s)) = X \vee turn(do(flip(O, t), s)) = O) \wedge On(2, do(flip(O, t), s)))$

$\equiv$

— by R_1 substitution of $\equiv$ from previous step (second step of equivalence proof for $R_1(s)$ on p.68)

— by P3.1.1, P3.1.2 and P3.1.3

$R_1(s) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists t.Legal(s) \wedge turn(s) = X \wedge$

$(False \vee turn(s) = X) \wedge (\neg On(1, s) \wedge t = 1 \vee On(1, s) \wedge t \neq 1) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists t.Legal(s) \wedge turn(s) = X \wedge$

$(False \vee turn(s) = X) \wedge (\neg On(2, s) \wedge t = 2 \vee On(2, s) \wedge t \neq 2) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists t.Legal(s) \wedge turn(s) = O \wedge$

$(turn(s) = O \vee False) \wedge (\neg On(1, s) \wedge t = 1 \vee On(1, s) \wedge t \neq 1) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists t.Legal(s) \wedge turn(s) = O \wedge$

$(turn(s) = O \vee False) \wedge (\neg On(2, s) \wedge t = 2 \vee On(2, s) \wedge t \neq 2)$

$\equiv$ by FOL

$R_1(s) \vee$

$Legal(s) \land turn(s) = X \land \exists t.(\neg On(1, s) \land t = 1 \lor On(1, s) \land t \neq 1) \lor$

$Legal(s) \land turn(s) = X \land \exists t.(\neg On(2, s) \land t = 2 \lor On(2, s) \land t \neq 2) \lor$

$Legal(s) \land turn(s) = O \land \exists t.(\neg On(1, s) \land t = 1 \lor On(1, s) \land t \neq 1) \lor$

$Legal(s) \land turn(s) = O \land \exists t.(\neg On(2, s) \land t = 2 \lor On(2, s) \land t \neq 2)$

$\equiv$ by by splitting the quantifier and quantifier elimination

$R_1(s) \lor$

$Legal(s) \land turn(s) = X \land \neg On(1, s) \lor$

$Legal(s) \land turn(s) = X \land On(1, s) \lor$

$Legal(s) \land turn(s) = X \land \neg On(2, s) \lor$

$Legal(s) \land turn(s) = X \land On(2, s) \lor$

$Legal(s) \land turn(s) = O \land \neg On(1, s) \lor$

$Legal(s) \land turn(s) = O \land On(1, s) \lor$

$Legal(s) \land turn(s) = O \land \neg On(2, s) \lor$

$Legal(s) \land turn(s) = O \land On(2, s)$

$\equiv$ by combining lines 2/3/4/5 and 6/7/8/9

$R_1(s) \lor$

$Legal(s) \land turn(s) = X \lor$

$Legal(s) \land turn(s) = O$

$\equiv$ by R_1 substitution of $\equiv$ from previous step

$Legal(s) \land On(1, s) \land On(2, s) \lor$

$Legal(s) \land (turn(s) = X \lor turn(s) = O) \land On(1, s) \lor$

$Legal(s) \land (turn(s) = X \lor turn(s) = O) \land On(2, s) \lor$

$Legal(s) \land turn(s) = X \lor$

$Legal(s) \land turn(s) = O$

$\equiv$ by FOL (line 2 subsumed by 4/5, line 3 subsumed by 4/5)

$Legal(s) \land On(1, s) \land On(2, s) \lor$

$Legal(s) \wedge (turn(s) = X \vee turn(s) = O)$

This approximation evaluates to true if s is such that X can win in at most 2 steps. Here it is true if player X is winning already or for all legal situations where it is one of the player's turn (as one step can turn light 1 on and the second step can turn light 2 on). □

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{LW}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_3(s)} \equiv \mathbf{Wins(X,s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{R_2}) \equiv$ by lemma L3.1.1

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\exists t.R_2[do(flip(X,t),s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\exists t.R_2[do(flip(O,t),s)])$

$\equiv$ by R_2 substitution of $\equiv$ from previous step

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\exists t.$

$\quad Legal(do(flip(X,t),s)) \wedge On(1,do(flip(X,t),s)) \wedge On(2,do(flip(X,t),s)) \vee$

$\quad Legal(do(flip(X,t),s)) \wedge turn(do(flip(X,t),s)) = X \vee$

$\quad Legal(do(flip(X,t),s)) \wedge turn(do(flip(X,t),s)) = O$

$) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\exists t.$

$\quad Legal(do(flip(O,t),s)) \wedge On(1,do(flip(O,t),s)) \wedge On(2,do(flip(O,t),s)) \vee$

$\quad Legal(do(flip(O,t),s)) \wedge turn(do(flip(O,t),s)) = X \vee$

$\quad Legal(do(flip(O,t),s)) \wedge turn(do(flip(O,t),s)) = O\ )$

$\equiv$ by distributing $\wedge$ over $\vee$ and reorganization

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\exists t.Legal(do(flip(X,t),s)) \wedge On(1,do(flip(X,t),s)) \wedge On(2,do(flip(X,t),s))) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\exists t.Legal(do(flip(O,t),s)) \wedge On(1,do(flip(O,t),s)) \wedge On(2,do(flip(O,t),s))) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\exists t.Legal(do(flip(X,t),s)) \wedge turn(do(flip(X,t),s)) = X) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\exists t. Legal(do(flip(X,t),s)) \wedge turn(do(flip(X,t),s)) = O) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\exists t. Legal(do(flip(O,t),s)) \wedge turn(do(flip(O,t),s)) = X) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\exists t. Legal(do(flip(O,t),s)) \wedge turn(do(flip(O,t),s)) = O)$

$\equiv$ by R_1 substitution of $\equiv$ from previous steps for lines 1/2/3, and P3.1.2, P3.1.3 for lines 4/5/6/7

$R_1(s) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists t. Legal(s) \wedge turn(s) = X \wedge False \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists t. Legal(s) \wedge turn(s) = X \wedge turn(s) = X \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists t. Legal(s) \wedge turn(s) = O \wedge turn(s) = O \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists t. Legal(s) \wedge turn(s) = O \wedge False$

$\equiv$ by FOL and quantifier elimination

$R_1(s) \vee$

$Legal(s) \wedge turn(s) = X \vee$

$Legal(s) \wedge turn(s) = O$

$\equiv$ by R_1 substitution of $\equiv$ from previous step

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge (turn(s) = X \vee turn(s) = O) \wedge On(1,s) \vee$

$Legal(s) \wedge (turn(s) = X \vee turn(s) = O) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \vee$

$Legal(s) \wedge turn(s) = O$

$\equiv$ by FOL (line 2 subsumed by 4/5, line 3 subsumed by 4/5)

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \vee$

$Legal(s) \wedge turn(s) = O$

$\square$

Thus the fixpoint expansion procedure converges in the 4^{th} step as we have:

$$\mathcal{D}_{ca}^{LW} \cup \mathcal{D}_Z \models R_2(s) \;\equiv\; R_3(s)$$

And therefore by Theorem 1 of [DLP10]:

$$\mathcal{D}_{GS}^{LW} \cup \mathcal{D}_Z \models \exists\Diamond Wins(X)[s] \;\equiv\; Legal(s) \wedge \{On(1,s) \wedge On(2,s) \vee turn(s) = X \vee turn(s) = O\}$$

It follows by the initial state axioms that $\mathcal{D}_{GS}^{LW} \cup \mathcal{D}_Z \models \exists\Diamond Wins(X)[S_0]$, i.e., the goal may eventually be reached from the initial situation as it is legal and it is player X's turn.

3.1.3 Existence of a Winning Strategy

The existence of a strategy to ensure $Wins(X)$ by group $G = \{X\}$ can be represented by the following formula:

$$\langle\langle\{X\}\rangle\rangle\Diamond Wins(X) \doteq \mu Z.\, Wins(X) \vee \langle\langle\{X\}\rangle\rangle \bigcirc Z$$

We begin by applying the De Giacomo et al. [DLP10] method and try to show that successive approximates are equivalent using only the unique name and domain closure axioms for actions $\mathcal{D}_{ca}^{LW}$:

$$\mathcal{D}_{ca}^{LW} \cup \mathcal{D}_Z \models \langle\langle\{X\}\rangle\rangle\Diamond Wins(X)$$

This formula can be verified by employing the [DLP10] method using regression and fixpoint approximation, until we converge. As discussed previously, the technique does not always converge in a finite number of steps and this needs to be checked as we proceed. The regressed approximations are as follows:

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{LW}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_0(s)} \doteq \mathbf{Wins(X,s)} \vee \mathcal{R}(\langle\langle\{\mathbf{X}\}\rangle\rangle \bigcirc \mathbf{False}) \equiv$ by lemma L3.1.2

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\exists t.False) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\forall t.False)$

$\equiv$

$Legal(s) \wedge On(1,s) \wedge On(2,s)$

This approximation evaluates to true if s is such that X is winning in s already (in no steps) i.e. these are situations where lights 1 and 2 are already on. $\square$

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{LW}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_1(s)} \doteq \mathbf{Wins(X,s)} \vee \mathcal{R}(\langle\langle\{\mathbf{X}\}\rangle\rangle \bigcirc \mathbf{R_0}) \equiv$ by lemma L3.1.2

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\exists t.R_0[do(flip(X,t),s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\forall t.R_0[do(flip(O,t),s)])$

$\equiv$ by R_0 substitution of $\equiv$ from previous step

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\exists t.($

$Legal(do(flip(X,t),s)) \wedge On(1, do(flip(X,t),s)) \wedge On(2, do(flip(X,t),s)))) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\forall t.($

$Legal(do(flip(O,t),s)) \wedge On(1, do(flip(O,t),s)) \wedge On(2, do(flip(O,t),s))))$

$\equiv$ by P3.1.1 and P3.1.2 and P3.1.3

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists t.(\mathbf{Legal(s)} \wedge \mathbf{turn(s)} = \mathbf{X} \wedge$

$(\neg On(1,s) \wedge t = 1 \vee On(1,s) \wedge t \neq 1) \wedge (\neg On(2,s) \wedge t = 2 \vee On(2,s) \wedge t \neq 2)) \vee$

$Legal(s) \wedge turn(s) = O \wedge \forall t.(\mathbf{Legal(s)} \wedge \mathbf{turn(s)} = \mathbf{O} \wedge$

$(\neg On(1,s) \wedge t = 1 \vee On(1,s) \wedge t \neq 1) \wedge (\neg On(2,s) \wedge t = 2 \vee On(2,s) \wedge t \neq 2))$

$\equiv$ by FOL

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists t.($

$\quad (\neg On(1,s) \wedge t = 1 \vee On(1,s) \wedge t \neq 1) \wedge (\neg On(2,s) \wedge t = 2 \vee On(2,s) \wedge t \neq 2)) \vee$

$Legal(s) \wedge turn(s) = O \wedge \forall t.($

$\quad (\neg On(1,s) \wedge t = 1 \vee On(1,s) \wedge t \neq 1) \wedge (\neg On(2,s) \wedge t = 2 \vee On(2,s) \wedge t \neq 2))$

$\equiv$ by distribution of $\wedge$

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists t.(\neg On(1,s) \wedge t = 1 \wedge \neg On(2,s) \wedge t = 2) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists t.(\neg On(1,s) \wedge t = 1 \wedge On(2,s) \wedge t \neq 2) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists t.(On(1,s) \wedge t \neq 1 \wedge \neg On(2,s) \wedge t = 2) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists t.(On(1,s) \wedge t \neq 1 \wedge On(2,s) \wedge t \neq 2) \vee$

$Legal(s) \wedge turn(s) = O \wedge \forall t.($

$\quad (\neg On(1,s) \wedge t = 1 \vee On(1,s) \wedge t \neq 1) \wedge (\neg On(2,s) \wedge t = 2 \vee On(2,s) \wedge t \neq 2))$

$\equiv$ by quantifier elimination and FOL, by $1 \neq 2$ of D_Z

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge \neg On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge On(1,s) \wedge \neg On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = O \wedge \forall t.($

$\quad (\neg On(1,s) \wedge t = 1 \vee On(1,s) \wedge t \neq 1) \wedge (\neg On(2,s) \wedge t = 2 \vee On(2,s) \wedge t \neq 2))$

$\equiv$ by FOL (line 2 and 4 combined, line 3 and 4 combined)

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge On(1,s) \vee$

$Legal(s) \wedge turn(s) = O \wedge \forall t.($

$(\neg On(1,s) \wedge t = 1 \vee On(1,s) \wedge t \neq 1) \wedge (\neg On(2,s) \wedge t = 2 \vee On(2,s) \wedge t \neq 2))$

$\equiv$ by splitting $\forall$ into cases and D_Z

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge On(1,s) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists t.(\mathbf{t} = \mathbf{1} \wedge$

$(\neg On(1,s) \wedge t = 1 \vee On(1,s) \wedge t \neq 1) \wedge (\neg On(2,s) \wedge t = 2 \vee On(2,s) \wedge t \neq 2)) \wedge$

$Legal(s) \wedge turn(s) = O \wedge \exists t.(\mathbf{t} = \mathbf{2} \wedge$

$(\neg On(1,s) \wedge t = 1 \vee On(1,s) \wedge t \neq 1) \wedge (\neg On(2,s) \wedge t = 2 \vee On(2,s) \wedge t \neq 2)) \wedge$

$Legal(s) \wedge turn(s) = O \wedge \forall t.((\mathbf{t} < \mathbf{1} \vee \mathbf{t} > \mathbf{2}) \wedge$

$(\neg On(1,s) \wedge t = 1 \vee On(1,s) \wedge t \neq 1) \wedge (\neg On(2,s) \wedge t = 2 \vee On(2,s) \wedge t \neq 2))$

$\equiv$ by FOL and quantifier elimination

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge On(1,s) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists t.(\neg On(1,s) \wedge On(2,s)) \wedge$

$Legal(s) \wedge turn(s) = O \wedge \exists t.(On(1,s) \wedge \neg On(2,s)) \wedge$

$Legal(s) \wedge turn(s) = O \wedge \forall t.(On(1,s) \wedge On(2,s))$

$\equiv$ by quantifier elimination

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge On(1,s) \vee$

$Legal(s) \wedge turn(s) = O \wedge \neg On(1,s) \wedge On(2,s) \wedge$

$Legal(s) \wedge turn(s) = O \wedge On(1,s) \wedge \neg On(2,s) \wedge$

$Legal(s) \wedge turn(s) = O \wedge On(1, s) \wedge On(2, s)$

$\equiv$ by FOL (lines 4, 5, and 6 clearly contradictory) and reorganization

$Legal(s) \wedge On(1, s) \wedge On(2, s) \vee$

$Legal(s) \wedge turn(s) = X \wedge On(1, s) \vee$

$Legal(s) \wedge turn(s) = X \wedge On(2, s)$

This approximation evaluates to true if s is such that X can win in at most 1 step i.e. these are situations where lights 1 and 2 are already on or one of lights 1 or 2 is on and it is player X's turn (X can then turn the other light at the next step). □

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{LW}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_2(s)} \equiv \mathbf{Wins(X, s)} \vee \mathcal{R}(\langle\langle\{\mathbf{X}\}\rangle\rangle \bigcirc \mathbf{R_1}) \equiv$ by lemma L3.1.2

$Legal(s) \wedge On(1, s) \wedge On(2, s) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\exists t. R_1[do(flip(X, t), s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\forall t. R_1[do(flip(O, t), s)])$

$\equiv$ by R_1 substitution of $\equiv$ from previous step

$Legal(s) \wedge On(1, s) \wedge On(2, s) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\exists t.($

$Legal(do(flip(X, t), s)) \wedge On(1, do(flip(X, t), s)) \wedge On(2, do(flip(X, t), s)) \vee$

$Legal(do(flip(X, t), s)) \wedge turn(do(flip(X, t), s)) = X \wedge On(1, do(flip(X, t), s)) \vee$

$Legal(do(flip(X, t), s)) \wedge turn(do(flip(X, t), s)) = X \wedge On(2, do(flip(X, t), s)))) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\forall t.($

$Legal(do(flip(O, t), s)) \wedge On(1, do(flip(O, t), s)) \wedge On(2, do(flip(O, t), s)) \vee$

$Legal(do(flip(O, t), s)) \wedge turn(do(flip(O, t), s)) = X \wedge On(1, do(flip(O, t), s)) \vee$

$Legal(do(flip(O, t), s)) \wedge turn(do(flip(O, t), s)) = X \wedge On(2, do(flip(O, t), s))))$

$\equiv$ by P3.1.1 and P3.1.2 and P3.1.3

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists t.($

$\mathbf{Legal(s)} \wedge \mathbf{turn(s)} = \mathbf{X} \wedge On(1, do(flip(X,t),s)) \wedge On(2, do(flip(X,t),s)) \vee$

$\mathbf{Legal(s)} \wedge \mathbf{turn(s)} = \mathbf{X} \wedge False \wedge On(1, do(flip(X,t),s)) \vee$

$\mathbf{Legal(s)} \wedge \mathbf{turn(s)} = \mathbf{X} \wedge False \wedge On(2, do(flip(X,t),s))) \vee$

$Legal(s) \wedge turn(s) = O \wedge \forall t.($

$\mathbf{Legal(s)} \wedge \mathbf{turn(s)} = \mathbf{O} \wedge On(1, do(flip(O,t),s)) \wedge On(2, do(flip(O,t),s)) \vee$

$\mathbf{Legal(s)} \wedge \mathbf{turn(s)} = \mathbf{O} \wedge \mathbf{turn(s)} = \mathbf{O} \wedge On(1, do(flip(O,t),s)) \vee$

$\mathbf{Legal(s)} \wedge \mathbf{turn(s)} = \mathbf{O} \wedge \mathbf{turn(s)} = \mathbf{O} \wedge On(2, do(flip(O,t),s)))$

$\equiv$ by FOL

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists t.(On(1, do(flip(X,t),s)) \wedge On(2, do(flip(X,t),s))) \vee$

$Legal(s) \wedge turn(s) = O \wedge \forall t.($

$On(1, do(flip(O,t),s)) \wedge On(2, do(flip(O,t),s)) \vee$

$On(1, do(flip(O,t),s)) \vee$

$On(2, do(flip(O,t),s)))$

$\equiv$ by FOL (line 4 subsumed by line 5 and 6)

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists t.(On(1, do(flip(X,t),s)) \wedge On(2, do(flip(X,t),s))) \vee$

$Legal(s) \wedge turn(s) = O \wedge \forall t.($

$On(1, do(flip(O,t),s)) \vee$

$On(2, do(flip(O,t),s)))$

$\equiv$ by P3.1.2

$Legal(s) \land On(1, s) \land On(2, s) \lor$

$Legal(s) \land turn(s) = X \land \exists t.($

$\quad (\neg On(1, s) \land t = 1 \lor On(1, s) \land t \neq 1) \land (\neg On(2, s) \land t = 2 \lor On(2, s) \land t \neq 2))\lor$

$Legal(s) \land turn(s) = O \land \forall t.($

$\quad (\neg On(1, s) \land t = 1 \lor On(1, s) \land t \neq 1) \lor (\neg On(2, s) \land t = 2 \lor On(2, s) \land t \neq 2))$

$\equiv$ by distribution of $\land$ and splitting of the $\exists$ quantifier

$Legal(s) \land On(1, s) \land On(2, s) \lor$

$Legal(s) \land turn(s) = X \land \exists t.(\neg On(1, s)\land \mathbf{t} = \mathbf{1} \land \neg On(2, s)\land \mathbf{t} = \mathbf{2})\lor$

$Legal(s) \land turn(s) = X \land \exists t.(\neg On(1, s) \land t = 1 \land (On(2, s) \land t \neq 2)\lor$

$Legal(s) \land turn(s) = X \land \exists t.(On(1, s) \land t \neq 1 \land \neg On(2, s) \land t = 2)\lor$

$Legal(s) \land turn(s) = X \land \exists t.(On(1, s) \land t \neq 1 \land (On(2, s) \land t \neq 2)\lor$

$Legal(s) \land turn(s) = O \land \forall t.($

$\quad (\neg On(1, s) \land t = 1 \lor On(1, s) \land t \neq 1) \lor (\neg On(2, s) \land t = 2 \lor On(2, s) \land t \neq 2))$

$\equiv$ by FOL and elimination of the $\exists$ quantifier, by $1 \neq 2$ of D_Z

$Legal(s) \land On(1, s) \land On(2, s) \lor$

$Legal(s) \land turn(s) = X \land \neg On(1, s) \land (On(2, s)\lor$

$Legal(s) \land turn(s) = X \land On(1, s) \land \neg On(2, s)\lor$

$Legal(s) \land turn(s) = X \land On(1, s) \land (On(2, s)\lor$

$Legal(s) \land turn(s) = O \land \forall t.($

$\quad (\neg On(1, s) \land t = 1 \lor On(1, s) \land t \neq 1) \lor (\neg On(2, s) \land t = 2 \lor On(2, s) \land t \neq 2))$

$\equiv$ by FOL (line 2 and 4 combined, line 3 and 4 combined)

$Legal(s) \land On(1, s) \land On(2, s) \lor$

$Legal(s) \land turn(s) = X \land On(2, s)\lor$

$Legal(s) \land turn(s) = X \land On(1, s)\lor$

$Legal(s) \wedge turn(s) = O \wedge \forall t.($

$\quad (\neg On(1,s) \wedge t = 1 \vee On(1,s) \wedge t \neq 1) \vee (\neg On(2,s) \wedge t = 2 \vee On(2,s) \wedge t \neq 2))$

$\equiv$ by splitting $\forall$ into cases and D_Z

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge On(1,s) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists t.(\mathbf{t} = \mathbf{1} \wedge$

$\quad (\neg On(1,s) \wedge t = 1 \vee On(1,s) \wedge t \neq 1) \vee (\neg On(2,s) \wedge t = 2 \vee On(2,s) \wedge t \neq 2)) \wedge$

$Legal(s) \wedge turn(s) = O \wedge \exists t.(\mathbf{t} = \mathbf{2} \wedge$

$\quad (\neg On(1,s) \wedge t = 1 \vee On(1,s) \wedge t \neq 1) \vee (\neg On(2,s) \wedge t = 2 \vee On(2,s) \wedge t \neq 2)) \wedge$

$Legal(s) \wedge turn(s) = O \wedge \forall t.((\mathbf{t} < \mathbf{1} \vee \mathbf{t} > \mathbf{2}) \wedge$

$\quad (\neg On(1,s) \wedge t = 1 \vee On(1,s) \wedge t \neq 1) \vee (\neg On(2,s) \wedge t = 2 \vee On(2,s) \wedge t \neq 2))$

$\equiv$ by FOL and quantifier elimination

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge On(1,s) \vee$

$Legal(s) \wedge turn(s) = O \wedge (\neg On(1,s) \vee On(2,s)) \wedge$

$Legal(s) \wedge turn(s) = O \wedge (On(1,s) \vee \neg On(2,s)) \wedge$

$Legal(s) \wedge turn(s) = O \wedge (On(1,s) \vee On(2,s))$

$\equiv$ by splitting line 6 disjunction

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge On(1,s) \vee$

$Legal(s) \wedge turn(s) = O \wedge (\neg On(1,s) \vee On(2,s)) \wedge$

$Legal(s) \wedge turn(s) = O \wedge (On(1,s) \vee \neg On(2,s)) \wedge$

$Legal(s) \wedge turn(s) = O \wedge On(1,s) \vee$

$Legal(s) \wedge turn(s) = O \wedge (\neg On(1,s) \vee On(2,s)) \wedge$

$Legal(s) \wedge turn(s) = O \wedge (On(1,s) \vee \neg On(2,s)) \wedge$

$Legal(s) \wedge turn(s) = O \wedge On(2,s)$

$\equiv$ by FOL

$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge On(1,s) \vee$

$Legal(s) \wedge turn(s) = X \wedge On(2,s)$

$\square$

Thus the fixpoint expansion procedure converges in the 3rd step as we have:

$$\mathcal{D}_{ca}^{LW} \cup \mathcal{D}_Z \models R_1(s) \; \equiv \; R_2(s)$$

And therefore by Theorem 1 of [DLP10]:

$$\mathcal{D}_{GS}^{LW} \cup \mathcal{D}_Z \models \langle\langle\{X\}\rangle\rangle \Diamond Wins(X)[s] \; \equiv \; R_1(s)$$

It follows by the initial state axioms that

$$\mathcal{D}_{GS}^{LW} \cup \mathcal{D}_Z \models \neg\langle\langle\{X\}\rangle\rangle \Diamond Wins(X)[S_0]$$

i.e., there is no winning strategy for X in the initial situation S_0.

However we have that

$$\mathcal{D}_{GS}^{LW} \cup \mathcal{D}_Z \models \langle\langle\{X\}\rangle\rangle \Diamond Wins(X)[S_1]$$

where $S_1 = do(flip(O,3), do(flip(X,1), S_0))$

i.e., there is a winning strategy for X in the situation S_1 where X has first turned light 1 on and then O has turned light 3 on, as X can turn on light 2 next.

3.2 Oil Lamp World (OLW)

The De Giacomo et al. [DLP10] fixpoint iteration method tries to detect convergence by checking if the i-th approximate is equivalent to the $i+1$-th approximate using only the unique name and domain closure axioms for actions (to which we have added the axiomatization of the integers). In this section we give an example where this method does not converge in a finite number of steps. However, we also show that if we use some additional facts that are entailed by the complete theory (including the initial state axioms) when checking if successive approximates are equivalent, we do get convergence in a finite number of steps.

The Oil Lamp World (OLW) is a variant of the Light World (LW) Domain discussed in the previous section. It also involves an infinite row of lamps one for each integer. A lamp can be on or off. A lamp has an igniter that can be flipped and will turn the lamp on if the lamp immediately to the right is on, i.e., flipping an igniter in lamp t will turn it on if lamp $t+1$ is on. There is only one acting agent. The goal of player X is to have lamp 1 on in which case player X wins. It is possible for the game go on forever without the goal being reached: for example if player X keeps switching some lamp other than lamp 1 on and off repeatedly.

3.2.1 Situation Calculus Game Structure Axiomatization of OLW Domain

The Situation Calculus Game Structure Axiomatization is defined as:

$$\mathcal{D}_{GS}^{OLW} = \Sigma \cup \mathcal{D}_{poss}^{OLW} \cup \mathcal{D}_{ssa}^{OLW} \cup \mathcal{D}_{ca}^{OLW} \cup \mathcal{D}_{S_0}^{OLW} \cup \mathcal{D}_Z$$

where $\mathcal{D}_Z$ is a suitable axiomatization of integers.

Fluents

- $On(t, s)$ - predicate fluent indicating whether the lamp t is on in situation s

Derived Fluents

- $Wins(p, s) \doteq Legal(s) \wedge On(1, s)$

  This fluent indicates if the agent X has won.

- $Finished(s) \doteq \exists p.\ Wins(p.s)$

  This fluent is not used in this chapter but later in Chapter 4.

Actions

- $flip(p, t)$ - agent p flips the igniter on lamp t

Precondition Axioms $\mathcal{D}^{OLW}_{poss}$

These are precondition axioms, one per action, specifying when an action can physically be performed:

- $Poss(flip(p, t), s) \equiv Agent(p)$

Successor State Axioms $\mathcal{D}^{OLW}_{ssa}$

These are successor state axioms specifying how the fluents change as a result of actions:

- $On(t, do(a, s)) \equiv \exists p.\{a = flip(p, t) \wedge On(t + 1, s)\} \vee On(t, s)$

  We assume that once a lamp is turned on it remains on.

Initial State Axioms $\mathcal{D}^{OLW}_{S_0}$

These describe the initial situation S_0:

- $Legal(S_0)$

- $On(3, S_0)$

Unique Name and Domain Closure Axioms for Actions $\mathcal{D}_{ca}^{OLW}$

These describe the uniqueness of action names and the fact that the domain of actions s closed:

- $\forall a.\ \{\ \exists p, t.\ a = flip(p, t)\ \}$

- $\forall p, p', t, t'.\ \{\ flip(p, t) = flip(p', t') \supset p = p' \wedge t = t'\ \}$

- $\forall p.\ \{\ Agent(p) \equiv p = X\}$

Legal Moves Axioms $\mathcal{D}_{legal}^{OLW}$

These encode the rules of the game:

- $agent(flip(p, t)) = p$

- $Control(p, s) \doteq \exists a.Legal(do(a, s)) \wedge agent(a) = p$

- $Legal(do(a, s)) \equiv Legal(s) \wedge \exists t.\ a = flip(X, t)$

The complete theory also includes the foundational axioms Σ of the Situation Calculus. It can be noticed that this is clearly an infinite state domain as the set of lamps that can be turned on is infinite.

Propositions and Lemmas

We derived the following propositions, some of which are used in later proofs:

P 3.2.1. *Regression for concrete case for the On fluent.*

$$\mathcal{D}_{ca}^{OLW} \models \mathcal{R}(On(v, do(do(flip(X, t), s))) \equiv On(t + 1, s) \wedge t = v \vee On(v, s)$$

P 3.2.2. *Regression for concrete case for the Legal fluent.*

$$\mathcal{D}_{ca}^{OLW} \models \mathcal{R}(Legal(do(flip(X,t),s))) \equiv Legal(s)$$

In this domain, the possibility that φ holds next can be established by considering only the $flip(X,t)$ actions as the following lemma shows:

L 3.2.1. *Regression for the concrete case of the $\exists \bigcirc \varphi$ definition based on domain closure.*

$$\mathcal{D}_{ca}^{OLW} \models \mathcal{R}(\exists \bigcirc \varphi[s]) \equiv Legal(s) \wedge \mathcal{R}(\exists t.\ \varphi[do(flip(X,t),s)])$$

We can show by induction on situations that in this domain, if a lamp is on initially, it remains on in all situations:

L 3.2.2. *The persistence of the On fluent.*

$$\mathcal{D}_{GS}^{OLW} \cup \mathcal{D}_Z \models \forall t.\ \{On(t,S_0) \supset \forall s.\ On(t,s)\}$$

Proof: By induction on situations. Take an arbitrary t and assume the antecedent. Then the base case $On(t,S_0)$ trivially follows. Suppose that $On(t,s)$ holds. Then it follows by the SSA that $On(t,do(a,s))$. Thus by the principle of induction on situations [Rei01], $\forall s.On(t,s)$. □

Also, if no lamps are on initially, than no lamp will ever be on:

L 3.2.3. *Absence of any light persist*

$$\mathcal{D}_{GS}^{OLW} \cup \mathcal{D}_Z \models \{\forall t.\ \neg On(t,S_0)\} \supset \forall s \forall t.\ \neg On(t,s)$$

Proof: By induction on situations. Assume the antecedent for all t. Then the base case $\forall t.\neg On(t,S_0)$ trivially follows. Suppose that $\forall t.\neg On(t,s)$ holds. Then it follows by the SSA that $\forall t.\neg On(t,do(a,s))$. Thus by the principle of induction on situations [Rei01], $\forall s \forall t.\neg On(t,s)$. □

L 3.2.4. *Legality of every situation.*

$$\mathcal{D}_{GS}^{OLW} \models \forall s.\ Legal(s)$$

Proof: By induction on situations. The base case $Legal(S_0)$ trivially follows from D_{S_0}. Suppose that $Legal(s)$ holds. Then it follows by the SSA and the domain closure for actions that $Legal(do(a, s))$. Thus by the principle of induction on situations [Rei01], $\forall s.Legal(s)$. □

3.2.2 Possibility of Winning

The property that it is possible for X to eventually win can be represented by the following formula:

$$\exists\Diamond Wins(X) \doteq \mu Z.\{\ Wins(X)\ \lor\ \exists\bigcirc Z\ \}$$

We begin by applying the De Giacomo et al. [DLP10] method and try to show that successive approximates are equivalent using only the unique name and domain closure axioms for actions $\mathcal{D}_{ca}^{OLW}$ and the axiomatization of the integers $\mathcal{D}_Z$:

$$\mathcal{D}_{ca}^{OLW} \cup \mathcal{D}_Z \models \exists\Diamond Wins(X)$$

This formula can be verified by employing the [DLP10] method using regression and fixpoint approximation, until we converge. The technique does not always converge and this needs to be checked as we proceed. The approximations are as follows:

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{OLW}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_0(s)} \doteq \mathbf{Wins(X,s)} \lor \mathcal{R}(\exists\bigcirc \mathbf{False}) \equiv$ by L 3.2.1

$Legal(s) \land On(1, s) \lor$

$Legal(s) \land \mathcal{R}(\exists t.\ False)$

$\equiv$

$Legal(s) \land On(1, s)$

This approximation evaluates to true if s is such that X is winning in s already (in no steps) i.e. these are situations where lamp 1 is on. □

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{OLW}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_1(s)} \doteq \mathbf{Wins(X,s)} \lor \mathcal{R}(\exists\bigcirc \mathbf{R_0}) \equiv$ by L 3.2.1

$Legal(s) \wedge On(1, s) \vee$

$Legal(s) \wedge \mathcal{R}(\exists t.\ R_0[do(flip(X, t), s)])$

$\equiv$ by R_0 substitution of $\equiv$ from previous step

$Legal(s) \wedge On(1, s) \vee$

$Legal(s) \wedge \mathcal{R}(\exists t.\ Legal(do(flip(X, t), s)) \wedge On(1, do(flip(X, t), s)))$

$\equiv$ by P3.2.1, P3.2.2

$Legal(s) \wedge On(1, s) \vee$

$Legal(s) \wedge \exists t.\ \mathbf{Legal(s)} \wedge (On(t + 1, s) \wedge \mathbf{t = 1} \vee On(1, s))$

$\equiv$ by FOL

$Legal(s) \wedge On(1, s) \vee$

$Legal(s) \wedge (On(2, s) \vee On(1, s))$

$\equiv$ by distribution of $\wedge$

$Legal(s) \wedge On(1, s) \vee$

$Legal(s) \wedge On(2, s) \vee$

$Legal(s) \wedge On(1, s)$

$\equiv$ by FOL (line 1 subsumes line 3)

$Legal(s) \wedge On(1, s) \vee$

$Legal(s) \wedge On(2, s)$

This approximation evaluates to true if s is such that X can win in at most 1 step i.e. these are situations where lamp 1 is on or lamp 2 is on (and X can turn lamp 1 on at the next step). $\square$

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{OLW}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_2(s)} \doteq \mathbf{Wins(X, s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{R_1}) \equiv$ by L 3.2.1

$Legal(s) \wedge On(1, s) \vee$

$Legal(s) \wedge \mathcal{R}(\exists t.\ R_1[do(flip(X, t), s)])$

$\equiv$ by R_1 substitution of $\equiv$ from previous step

$Legal(s) \wedge On(1, s) \vee$

$Legal(s) \wedge \mathcal{R}(\exists t.\ ($

$Legal(do(flip(X,t),s)) \wedge On(1, do(flip(X,t),s)) \vee$

$Legal(do(flip(X,t),s)) \wedge On(2, do(flip(X,t),s))))$

$\equiv$ by P3.2.1, P3.2.2

$Legal(s) \wedge On(1,s) \vee$

$Legal(s) \wedge \exists t.\ ($

$\mathbf{Legal(s)} \wedge (On(t+1,s) \wedge \mathbf{t = 1} \vee On(1,s)) \vee$

$\mathbf{Legal(s)} \wedge (On(t+1,s) \wedge \mathbf{t = 2} \vee On(2,s)))$

$\equiv$ by FOL

$Legal(s) \wedge On(1,s) \vee$

$Legal(s) \wedge (On(2,s) \vee On(1,s)) \vee (On(3,s) \vee On(2,s))$

$\equiv$ by FOL and distribution of $\wedge$

$Legal(s) \wedge On(1,s) \vee$

$Legal(s) \wedge On(2,s) \vee$

$Legal(s) \wedge On(1,s) \vee$

$Legal(s) \wedge On(3,s)$

$\equiv$ by FOL (line 1 subsumes line 3)

$Legal(s) \wedge On(1,s) \vee$

$Legal(s) \wedge On(2,s) \vee$

$Legal(s) \wedge On(3,s)$

This approximation evaluates to true if s is such that X can win in at most 2 steps i.e. these are situations where lamp 1 is on or lamp 2 is on (and X can turn lamp 1 on at the next step) or lamp 3 is on (and X can turn lamp 2 on at the next step and lamp 1 at the following step). □

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{OLW}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_3(s)} \doteq \mathbf{Wins(X,s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{R_2}) \equiv$ by L 3.2.1

$Legal(s) \wedge On(1,s) \vee$

$Legal(s) \wedge \mathcal{R}(\exists t.\ R_2[do(flip(X,t),s)])$

$\equiv$ by R_2 substitution of $\equiv$ from previous step

$Legal(s) \wedge On(1,s) \vee$

$Legal(s) \wedge \mathcal{R}(\exists t.\ ($

$\quad Legal(do(flip(X,t),s)) \wedge On(1, do(flip(X,t),s)) \vee$

$\quad Legal(do(flip(X,t),s)) \wedge On(2, do(flip(X,t),s)) \vee$

$\quad Legal(do(flip(X,t),s)) \wedge On(3, do(flip(X,t),s))))$

$\equiv$ by P3.2.1, P3.2.2

$Legal(s) \wedge On(1,s) \vee$

$Legal(s) \wedge \exists t.\ ($

$\quad \mathbf{Legal(s)} \wedge (On(t+1,s) \wedge \mathbf{t = 1} \vee On(1,s)) \vee$

$\quad \mathbf{Legal(s)} \wedge (On(t+1,s) \wedge \mathbf{t = 2} \vee On(2,s)) \vee$

$\quad \mathbf{Legal(s)} \wedge (On(t+1,s) \wedge \mathbf{t = 3} \vee On(3,s)))$

$\equiv$ by FOL

$Legal(s) \wedge On(1,s) \vee$

$Legal(s) \wedge (On(2,s) \vee On(1,s)) \vee On(3,s) \vee \mathbf{On(2,s)} \vee On(4,s) \vee \mathbf{On(3,s)})$

$\equiv$ by FOL and distribution of $\wedge$

$Legal(s) \wedge On(1,s) \vee$

$Legal(s) \wedge On(2,s) \vee$

$Legal(s) \wedge On(1,s) \vee$

$Legal(s) \wedge On(3,s) \vee$

$Legal(s) \wedge On(4,s)$

$\equiv$ by FOL (line 1 subsumes line 3)

$Legal(s) \wedge On(1,s) \vee$

$Legal(s) \wedge On(2,s) \vee$

$Legal(s) \wedge On(3,s) \vee$

$Legal(s) \wedge On(4, s)$

This approximation evaluates to true if s is such that X can win in at most 3 steps i.e. these are situations where lamp 1 is on or lamp 2 is on (X can turn lamp 1 on at the next step) or lamp 3 is on (X can turn lamp 2 on at the next step and lamp 1 at the following step) or lamp 4 is on (X can turn lamp 3 on at the next step and lamp 2 and 1 at the following steps). $\square$

The subsequent results can be generalized to the following formula:

$\mathbf{D_{ca}^{OLW} \cup D_Z \models R_i \equiv Legal(s) \wedge \bigvee_{1 \leq j \leq i+1} On(j, s)}$

Proof: The proof is by induction. The base case is R_0 and it holds by the calculation for R_0 above. Let's assume that for some i it holds that $D_{ca}^{OLW} \cup D_Z \models R_i \equiv Legal(s) \wedge \bigvee_{1 \leq j \leq i+1} On(j, s)$, and let's prove $D_{ca}^{OLW} \cup D_Z \models R_{i+1} \equiv Legal(s) \wedge \bigvee_{1 \leq j \leq i+2} On(j, s)$ as follows:

$\mathcal{D}_{ca}^{OLW} \cup \mathcal{D}_Z \models R_{i+1}(s) \doteq Wins(X, s) \vee \mathcal{R}(\exists \bigcirc R_i) \equiv$ by L 3.2.1

$Legal(s) \wedge On(1, s) \vee$

$Legal(s) \wedge \mathcal{R}(\exists t.\ R_i[do(flip(X, t), s)])$

$\equiv$ by R_0 substitution of $\equiv$ from I.H.

$Legal(s) \wedge On(1, s) \vee$

$Legal(s) \wedge \mathcal{R}(\exists t.\ Legal(do(flip(X, t), s)) \wedge \bigvee_{1 \leq j \leq i+1} On(j, do(flip(X, t), s)))$

$\equiv$ by P3.2.1, P3.2.2

$Legal(s) \wedge On(1, s) \vee$

$Legal(s) \wedge \exists t.\ Legal(s) \wedge \bigvee_{1 \leq j \leq i+1} \{On(t + 1, s) \wedge t = j \vee On(j, s)\}$

$\equiv$ by FOL

$Legal(s) \wedge On(1, s) \vee$

$Legal(s) \wedge \exists t.\ \bigvee_{1 \leq j \leq i+1} \{On(t + 1, s) \wedge t = j\} \vee \bigvee_{1 \leq j \leq i+1} On(j, s)$

$\equiv$ by FOL (line 1 subsumed by 2) and reorganization

$Legal(s) \wedge (\bigvee_{1 \leq j \leq i+1} On(j,s) \vee \exists t.\ \bigvee_{1 \leq j \leq i+1} \{On(t+1,s) \wedge t = j\})$

$\equiv$ by FOL

$Legal(s) \wedge (\bigvee_{1 \leq j \leq i+1} On(j,s) \vee \bigvee_{1 \leq j \leq i+1} \{\exists t.\ On(t+1,s) \wedge t = j\})$

$\equiv$ by FOL

$Legal(s) \wedge (\bigvee_{1 \leq j \leq i+1} On(j,s) \vee \bigvee_{1 \leq j \leq i+1} \{On(j+1,s)\})$

$\equiv$ by FOL

$Legal(s) \wedge (\bigvee_{1 \leq j \leq i+1} On(j,s) \vee \bigvee_{2 \leq j \leq i+2} On(j,s))$

$\equiv$ by FOL

$Legal(s) \wedge \bigvee_{1 \leq j \leq i+2} On(j,s)$

$\square$

Is there a convergence in a finite number of steps? No.

We can observe that for all natural numbers i, $D_{ca}^{OLW} \cup D_Z \not\models R_i \equiv R_{i+1}$, since one can always construct a model of $D_{ca}^{OLW} \cup D_Z$ where every light is off except light $i+2$.

Will there be a convergence if we use additional facts entailed by the whole theory ? Yes.

The proof is by 2 cases.

Case 1: If there is a lamp k that is on in the initial situation S_0, i.e. $On(k, S_0)$ then by L3.2.2 that lamp will be on in any situation s. Then it follows that for any natural numbers i, j, $i \leq j$:

$$\mathcal{D}_{ca}^{OLW} \cup \mathcal{D}_Z \cup \{On(i+1, S_0), L3.2.2\} \models R_j \equiv Legal(s)$$

In essence, X can eventually win in any legal situation where some lamp n is known to be on.

It follows that:

$$\mathcal{D}_{ca}^{OLW} \cup \mathcal{D}_Z \cup \{On(n, S_0), L3.2.2\} \models R_{n-1} \equiv R_n$$

Thus the fixpoint approximation method converges in a finite number of steps if we use the facts that some lamp n is known to be on initially and that a lamp that is on initially remains on forever.

By Theorem 1 of [DLP10] and L3.2.4 we have that if there is a lamp n that is known to be on then:

$$\mathcal{D}_{ca}^{OLW} \cup \mathcal{D}_Z \cup \{On(n, S_0), L3.2.2, L3.2.4\} \models \forall s.\exists\Diamond Wins(X)[s] \equiv Legal(s) \qquad \Box$$

Case 2: If there are no lamps that on in the initial situation S_0, i.e. $\forall k.\ \neg On(k, S_0)$ then by L3.2.3 no lamp will be on in any situation s. In this case

$\mathcal{D}_{ca}^{OLW} \cup \mathcal{D}_Z \cup \{\forall k.\ \neg On(k, S_0), L3.2.3\} \models R_0 \equiv False$

$\mathcal{D}_{ca}^{OLW} \cup \mathcal{D}_Z \cup \{\forall k.\ \neg On(k, S_0), L3.2.3\} \models R_1 \equiv False$

therefore

$\mathcal{D}_{ca}^{OLW} \cup \mathcal{D}_Z \cup \{\forall k.\ \neg On(k, S_0), L3.2.3\} \models R_0 \equiv R_1$ and the procedure converges in the second step.

By Theorem 1 of [DLP10] we have:

$$\mathcal{D}_{ca}^{OLW} \cup \mathcal{D}_Z \cup \{\forall k.\ \neg On(k, S_0), L3.2.3\} \models \neg\exists\Diamond Wins(X)[s] \qquad \Box$$

In our axiomatization it follows by the initial state axioms that $\mathcal{D}_{GS}^{OLW} \models \exists\Diamond Wins(X)[S_0]$, i.e., player can eventually win from the initial situation as it is legal and there is a lamp that is on.

3.3 In-Line Tic-Tac-Toe (TTT1D)

In this section we look at an example that is more like a traditional game. We apply the De Giacomo et al. [DLP10] fixpoint iteration method to verify some temporal properties, and the method does converge in a finite number of steps for these examples.

The In-Line Tic-Tac-Toe (TTT1D) game that we have designed involves an infinite vector of cells one for each integer. The cells are initially blank and may be marked with X or O. There are 2 pointers maintained in the game: the left marking position and the right marking position. Initially the left marking position points at cell number 0 and the right marking position points at cell number 1. There are 2 players X and O and they can mark the cells with their mark at the right marking position, or at the left marking position. If the mark is done at the left marking position then that left marking position is decreased by 1. If the mark is done at the right marking position then that right marking position is increased by 1. Initially it is player X's turn. Players take turns. The goal of player X is to have 3 consecutive cells marked with X to the right of the left marking position or to the left of the left marking position in which case X wins, and similarly for player O. A sample sequence of moves could be as follows: X at 1, O at 0, X at 2, O at -1, X at 3 and the game is over, the goal is reached to the left of the right marking position $[O_{-1}O_0X_1X_2X_3]$. It is possible for the game to go on forever and player O can prevent the player X from winning: player O needs to mark at the right marking position if player X did so in the previous move and to mark at the left marking position if X did so in the previous move. An example of such scenario would be: $[O_{-1}X_0X_1O_2X_3O_4]$ where X marks 1, O marks 2, X marks 3, O marks 4, X marks 0, O marks -1 and on and on. Please note that this version of the game permits a player X to win after player O reached a winning situation in the past.

3.3.1 Situation Calculus Game Structure Axiomatization of TTT1D Domain

The Situation Calculus Game Structure Axiomatization is defined as:

$$\mathcal{D}_{GS}^{TTT1D} = \Sigma \cup \mathcal{D}_{poss}^{TTT1D} \cup \mathcal{D}_{ssa}^{TTT1D} \cup \mathcal{D}_{ca}^{TTT1D} \cup \mathcal{D}_{S_0}^{TTT1D} \cup \mathcal{D}_Z$$

where $\mathcal{D}_Z$ is a suitable axiomatization of integers.

Fluents

- $curn(s)$ - functional fluent indicating the latest left marking position, domain of integer numbers
- $curp(s)$ - functional fluent indicating the latest right marking position, domain of integer numbers
- $turn(s)$ - functional fluent indicating which agent is to take the next action, with its domain being the agent set $\{X, O\}$
- $cell(k, s)$ - functional fluent indicating the content of cell k, domain of $\{B, X, O\}$

Derived Fluents

- $Wins(p, s) \doteq$

  $\exists k(Legal(s) \wedge curn(s) = k - 2 \wedge cell(k-1, s) = p \wedge cell(k, s) = p \wedge cell(k+1, s) = p \;\vee$

  $Legal(s) \wedge curp(s) = k + 2 \wedge cell(k+1, s) = p \wedge cell(k, s) = p \wedge cell(k-1, s) = p)$

- $Finished(s) \doteq Wins(X, s)$

  This fluent is not used in this chapter but later in Chapter 4.

Actions

- *markn(p)* - agent p marks the current left marking position with p and reduces the current left marking position by 1
- *markp(p)* - agent p marks the current right marking position with p and increases the current right marking position by 1

Precondition Axioms $\mathcal{D}_{poss}^{TTT1D}$

These are precondition axioms, one per action, specifying when an action can physically be performed:

- $Poss(markn(p), s) \equiv True$
- $Poss(markp(p), s) \equiv True$

Successor State Axioms $\mathcal{D}_{ssa}^{TTT1D}$

These are successor state axioms specifying how the fluents change as a result of actions:

- $curn(do(a, s)) = k \equiv curn(s) = k + 1 \wedge \exists p.\{a = markn(p)\} \;\vee curn(s) = k \wedge \forall p.\{a \neq markn(p)\}$

- $curp(do(a,s)) = k \equiv curp(s) = k - 1 \wedge \exists p.\{a = markp(p)\} \ \vee curp(s) = k \wedge \forall p.\{a \neq markn(p)\}$

- $cell(k, do(a,s)) = p \equiv$

  $a = markp(p) \wedge curp(s) = k \ \vee \ a = markn(p) \wedge curn(s) = k \ \vee$

  $cell(k,s) = p \wedge \neg \exists p'.\{a = markp(p') \wedge curp(s) = k\} \wedge \neg \exists p'.\{a = markn(p') \wedge curn(s) = k\}$

- $turn(do(a,s)) = p \equiv agent(a) = X \wedge p = O \wedge turn(s) = X \vee agent(a) = O \wedge p = X \wedge turn(s) = O$

Initial State Axioms $\mathcal{D}_{S_0}^{TTT1D}$

These describe the initial situation:

- $curn(S_0) = 0$
- $curp(S_0) = 1$
- $turn(S_0) = X$
- $Legal(S_0)$

Legal Moves Axioms $\mathcal{D}_{legal}^{TTT1D}$

These encode the rules of the game:

- $agent(markn(p)) = p$
- $agent(markp(p)) = p$
- $Control(p,s) \doteq \exists a.Legal(do(a,s)) \wedge agent(a) = p$
- $Legal(do(a,s)) \equiv Legal(s) \wedge \exists p.\{\ turn(s) = p \wedge (a = markn(p) \ \vee a = markp(p))\ \}$

Unique Name and Domain Closure Axioms for Actions $\mathcal{D}_{ca}^{TTT1D}$

These describe the uniqueness of action names and the fact that the domain of actions s closed:

- $\forall a.\ \{\ \exists p.\ a = markn(p) \vee \exists p.\ a = markp(p)\ \}$
- $\forall p, p'.\ \{\ markn(p) \neq markp(p')\ \}$
- $\forall p, p'.\ \{\ markn(p) = markn(p') \supset p = p'\ \}$
- $\forall p, p'.\ \{\ markp(p) = markp(p') \supset p = p'\ \}$
- $\forall p.\ \{\ Agent(p) \equiv (p = X \vee p = O)\}$
- $X \neq O$

The complete theory also includes the foundational axioms Σ of the Situation Calculus. It can be noticed that this is clearly an infinite state domain as the set of positions that can be marked is infinite.

Propositions and Lemmas

We derived the following propositions, some of which are used in later proofs:

P 3.3.1. *Transformation of the $Wins$ fluent.*

$$\mathcal{D}_{ca}^{TTT1D} \cup \mathcal{D}_Z \models Wins(p,s) \equiv \exists k($$

$$Legal(s) \wedge cell(k-1,s) = p \wedge cell(k,s) = p \wedge cell(k+1,s) = p \wedge curn(s) = k-2\ \vee$$

$$Legal(s) \wedge cell(k-1,s) = p \wedge cell(k,s) = p \wedge cell(k+1,s) = p \wedge curp(s) = k+2\ )$$

P 3.3.2. *Regression for concrete cases for the turn fluent.*

$$\mathcal{D}_{ca}^{TTT1D} \models \mathcal{R}(turn(do(a,s)) = X) \equiv agent(a) = O \wedge turn(s) = O$$

$$\mathcal{D}_{ca}^{TTT1D} \models \mathcal{R}(turn(do(a,s)) = O) \equiv agent(a) = X \wedge turn(s) = X$$

P 3.3.3. *Regression for concrete cases for the curn fluent.*

$$\mathcal{D}_{ca}^{TTT1D} \cup \mathcal{D}_Z \models \mathcal{R}(curn(do(markn(p),s) = k) \equiv curn(s) = k+1$$

$$\mathcal{D}_{ca}^{TTT1D} \cup \mathcal{D}_Z \models \mathcal{R}(curn(do(markp(p),s) = k) \equiv curn(s) = k$$

P 3.3.4. *Regression for concrete cases for the curp fluent.*

$\mathcal{D}_{ca}^{TTT1D} \cup \mathcal{D}_Z \models \mathcal{R}(curp(do(markn(p), s) = k) \equiv curp(s) = k$

$\mathcal{D}_{ca}^{TTT1D} \cup \mathcal{D}_Z \models \mathcal{R}(curp(do(markp(p), s) = k) \equiv curp(s) = k - 1$

P 3.3.5. *Regression for concrete cases for the Control fluent.*

$\mathcal{D}_{ca}^{TTT1D} \models \mathcal{R}(Control(X, s)) \equiv Legal(s) \wedge turn(s) = X$

$\mathcal{D}_{ca}^{TTT1D} \models \mathcal{R}(Control(O, s)) \equiv Legal(s) \wedge turn(s) = O$

P 3.3.6. *Regression for concrete cases for the Legal fluent.*

$\mathcal{D}_{ca}^{TTT1D} \models \mathcal{R}(Legal(do(markn(p), s))) \equiv Legal(s) \wedge turn(s) = p$

$\mathcal{D}_{ca}^{TTT1D} \models \mathcal{R}(Legal(do(markp(p), s))) \equiv Legal(s) \wedge turn(s) = p$

P 3.3.7. *Regression for concrete cases for the cell fluent.*

$\mathcal{D}_{ca}^{TTT1D} \models \mathcal{R}(cell(k, do(markn(X), s)) = X) \equiv curn(s) = k \vee cell(k, s) = X$

$\mathcal{D}_{ca}^{TTT1D} \models \mathcal{R}(cell(k, do(markn(O), s)) = O) \equiv curn(s) = k \vee cell(k, s) = O$

$\mathcal{D}_{ca}^{TTT1D} \models \mathcal{R}(cell(k, do(markp(X), s)) = X) \equiv curp(s) = k \vee cell(k, s) = X$

$\mathcal{D}_{ca}^{TTT1D} \models \mathcal{R}(cell(k, do(markp(O), s)) = O) \equiv curp(s) = k \vee cell(k, s) = O$

$\mathcal{D}_{ca}^{TTT1D} \models \mathcal{R}(cell(k, do(markn(O), s)) = X) \equiv curn(s) \neq k \wedge cell(k, s) = X$

$\mathcal{D}_{ca}^{TTT1D} \models \mathcal{R}(cell(k, do(markp(O), s)) = X) \equiv curp(s) \neq k \wedge cell(k, s) = X$

P 3.3.8. *Relation between functional fluents curn and curp based on $\mathcal{D}_{GS}^{TTT1D}$.*

$\mathcal{D}_{GS}^{TTT1D} \models \forall s.\ curn(s) < curp(s)$

Proof: the proof is by induction on the situation. For the base case S_0 we have $curn(S_0) < curn(S_0)$. Let's assume $curn(s) < curn(s)$ holds for some situation s. The successor situation to s is $do(markn(p), s)$ or $do(markp(p), s)$ where p is O or X. In the first case $curn(do(makn(p), s)) = curn(s) - 1$ and $curp(do(makn(p), s)) = curp(s)$, and thus by these and by the inductive assumption $curn(do(makn(p), s)) < curn(s) < curp(s) = curp(do(makn(p), s))$. In the second case $curn(do(makp(p), s)) = curn(s)$ and $curp(do(makp(p), s)) = curp(s) + 1$, thus with inductive assumption $curn(do(makn(p), s)) = curn(s) < curp(s) < curp(do(makn(p), s))$. This completes the proof that in any situation s led from S_0 we have that $curn(s) < curp(s)$.

P 3.3.9. *Player O influence on Wins fluent (follows from propositions above).*

$$\mathcal{D}_{ca}^{TTT1D} \cup \mathcal{D}_{S_0}^{TTT1D} \models Wins(X, do(markn(O), s)) \supset Wins(X, s)$$

$$\mathcal{D}_{ca}^{TTT1D} \cup \mathcal{D}_{S_0}^{TTT1D} \models Wins(X, do(markp(O), s)) \supset Wins(X, s)$$

In this domain, for groups $G = \{X\}$ and $G = \{X, O\}$ ensuring that φ holds next can be established by considering only the *markn* and *markp* actions as the following lemmas show:

L 3.3.1. *Regression of the concrete case of the* $\exists \bigcirc \varphi$ *definition based on domain closure.*

$$\mathcal{D}_{ca}^{TTT1D} \models \mathcal{R}(\exists \bigcirc \varphi[s]) \equiv$$

$$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\varphi[do(markn(X), s)]) \vee$$

$$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\varphi[do(markp(X), s)]) \vee$$

$$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\varphi[do(markn(O), s)]) \vee$$

$$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\varphi[do(markp(O), s)])$$

L 3.3.2. *Regression of the concrete case of the* $\langle\langle\{X\}\rangle\rangle \bigcirc \varphi$ *definition based on domain closure.*

$$\mathcal{D}_{ca}^{TTT1D} \models \mathcal{R}(\langle\langle\{X\}\rangle\rangle \bigcirc \varphi[s]) \equiv$$

$$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\varphi[do(markn(X), s)]) \vee$$

$$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\varphi[do(markp(X), s)]) \vee$$

$$Legal(s) \wedge turn(s) = O \ \wedge \mathcal{R}(\varphi[do(markn(O), s)]) \wedge \mathcal{R}(\varphi[do(markp(O), s)])$$

3.3.2 Possibility of Winning

The property that it is possible for X to eventually win can be represented by the following formula:

$$\exists \Diamond Wins(X) \doteq \mu Z. Wins(X) \ \vee \ \exists \bigcirc Z$$

We begin by applying the De Giacomo et al. [DLP10] method and try to show that successive approximates are equivalent using only the unique name and domain closure axioms for actions $\mathcal{D}_{ca}^{TTT1D}$ and the axiomatization of the integers $\mathcal{D}_Z$:

$$\mathcal{D}_{ca}^{TTT1D} \cup \mathcal{D}_Z \models \exists \Diamond Wins(X)$$

This formula can be verified by employing the [DLP10] method using regression and fixpoint approximation, until we converge. The technique does not always converge and this needs to be checked as we proceed. The proof is very long and tedious and there are numerous cases to deal with. The reason for this is that we cannot use the fact that *curn* is always less than *curp* and that the cells that are between them are non-blank and that the other cells are blank, which are consequences of the initial state axioms. More details are available in the on-Line appendix `http://www.cse.yorku.ca/~skmiec`, but here the first few steps and last few steps are collected. The approximations are as follows:

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{TTT1D}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_0(s)} \doteq \mathbf{Wins(X,s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{False}) \equiv$ by lemma L 3.3.1

$\exists k(Legal(s) \wedge cell(k-1,s) = X \wedge cell(k,s) = X \wedge cell(k+1,s) = X \wedge curn(s) = k-2) \vee$

$\exists k(Legal(s) \wedge cell(k-1,s) = X \wedge cell(k,s) = X \wedge cell(k+1,s) = X \wedge curp(s) = k+2) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(False) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(False) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(False) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(False)$

$\equiv$

$Legal(s) \wedge \exists k(curn(s) = k-2 \wedge cell(k-1,s) = X \wedge cell(k,s) = X \wedge cell(k+1,s) = X) \vee$

$Legal(s) \wedge \exists k(cell(k-1,s) = X \wedge cell(k,s) = X \wedge cell(k+1,s) = X \wedge curp(s) = k+2)$

This approximation evaluates to true if s is such that X is winning in s already (in no steps), i.e., these are situations where there are 3 X marks in a row on either side. □

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{TTT1D}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_1(s)} \doteq \mathbf{Wins(X,s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{R_0}) \equiv$ by lemma L 3.3.1

$Legal(s) \wedge \exists k(curn(s) = k-2 \wedge cell(k-1,s) = X \wedge cell(k,s) = X \wedge cell(k+1,s) = X) \vee$

$Legal(s) \wedge \exists k(cell(k-1,s) = X \wedge cell(k,s) = X \wedge cell(k+1,s) = X \wedge curp(s) = k+2) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(R_0[do(markn(X),s)]) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(R_0[do(markp(X),s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(R_0[do(markn(O),s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(R_0[do(markp(O),s)])$

$\equiv$ by R_0 from the previous step

$Legal(s) \wedge \exists k(curn(s) = k-2 \wedge cell(k-1,s) = X \wedge cell(k,s) = X \wedge cell(k+1,s) = X) \vee$

$Legal(s) \wedge \exists k(cell(k-1,s) = X \wedge cell(k,s) = X \wedge cell(k+1,s) = X \wedge curp(s) = k+2) \vee$

$Legal(s) \wedge turn(s) = X \wedge Legal(do(markn(X),s)) \wedge \mathcal{R}(\exists k(curn(do(markn(X),s)) = k-2 \wedge$

$cell(k-1, do(markn(X),s)) = X \wedge cell(k, do(markn(X),s)) = X \wedge cell(k+1, do(markn(X),s)) = X)) \vee$

$Legal(s) \wedge turn(s) = X \wedge Legal(do(markn(X),s)) \wedge \mathcal{R}(\exists k(cell(k-1, do(markn(X),s)) = X \wedge$

$cell(k, do(markn(X),s)) = X \wedge cell(k+1, do(markn(X),s)) = X \wedge curp(do(markn(X),s)) = k+2)) \vee$

$Legal(s) \wedge turn(s) = X \wedge Legal(do(markp(X),s)) \wedge \mathcal{R}(\exists k(curn(do(markp(X),s)) = k-2 \wedge$

$cell(k-1, do(markp(X),s)) = X \wedge cell(k, do(markp(X),s)) = X \wedge cell(k+1, do(markp(X),s)) = X)) \vee$

$Legal(s) \wedge turn(s) = X \wedge Legal(do(markp(X),s)) \wedge \mathcal{R}(\exists k(cell(k-1, do(markp(X),s)) = X \wedge$

$cell(k, do(markp(X),s)) = X \wedge cell(k+1, do(markp(X),s)) = X \wedge curp(do(markp(X),s)) = k+2)) \vee$

$Legal(s) \wedge turn(s) = O \wedge Legal(do(markn(O),s)) \wedge \mathcal{R}(\exists k(curn(do(markn(O),s)) = k-2 \wedge$

$cell(k-1, do(markn(O),s)) = X \wedge cell(k, do(markn(O),s)) = X \wedge cell(k+1, do(markn(O),s)) = X)) \vee$

$Legal(s) \wedge turn(s) = O \wedge Legal(do(markn(O),s)) \wedge \mathcal{R}(\exists k(cell(k-1, do(markn(O),s)) = X \wedge$

$cell(k, do(markn(O),s)) = X \wedge cell(k+1, do(markn(O),s)) = X \wedge curp(do(markn(O),s)) = k+2)) \vee$

$Legal(s) \wedge turn(s) = O \wedge Legal(do(markp(O),s)) \wedge \mathcal{R}(\exists k(curn(do(markp(O),s)) = k-2 \wedge$

$cell(k-1, do(markp(O),s)) = X \wedge cell(k, do(markp(O),s)) = X \wedge cell(k+1, do(markp(O),s)) = X)) \vee$

$Legal(s) \wedge turn(s) = O \wedge Legal(do(markp(O),s)) \wedge \mathcal{R}(\exists k(cell(k-1, do(markp(O),s)) = X \wedge$

$cell(k, do(markp(O),s)) = X \wedge cell(k+1, do(markp(O),s)) = X \wedge curp(do(markp(O),s)) = k+2))$

$\equiv$ by P 3.3.6, P 3.3.3, P 3.3.4, P 3.3.7

$Legal(s) \wedge \exists k(curn(s) = k-2 \wedge cell(k-1, s) = X \wedge cell(k, s) = X \wedge cell(k+1, s) = X) \vee$

$Legal(s) \wedge \exists k(cell(k-1, s) = X \wedge cell(k, s) = X \wedge cell(k+1, s) = X \wedge curp(s) = k+2) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k(curn(s) = k-1 \wedge (curn(s) = k-1 \vee cell(k-1, s) = X) \wedge$

$\quad (curn(s) = k \vee cell(k, s) = X) \wedge (curn(s) = k+1 \vee cell(k+1, s) = X)) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k((curn(s) = k-1 \vee cell(k-1, s) = X) \wedge (curn(s) = k \vee cell(k, s) = X) \wedge$

$\quad (curn(s) = k+1 \vee cell(k+1, s) = X) \wedge curp(s) = k+2) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k(curn(s) = k-2 \wedge (curp(s) = k-1 \vee cell(k-1, s) = X) \wedge (curp(s) = k \vee$

$\quad cell(k, s) = X) \wedge (curp(s) = k+1 \vee cell(k+1, s) = X)) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k((curp(s) = k-1 \vee cell(k-1, s) = X) \wedge (curp(s) = k \vee cell(k, s) = X) \wedge$

$\quad (curp(s) = k+1 \vee cell(k+1, s) = X) \wedge curp(s) = k+1) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k(curn(s) = k-1 \wedge curn(s) \neq k-1 \wedge cell(k-1, s) = X \wedge curn(s) \neq k \wedge$

$\quad cell(k, s) = X \wedge curn(s) \neq k+1 \wedge cell(k+1, s) = X) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k(curn(s) \neq k-1 \wedge cell(k-1, s) = X \wedge curn(s) \neq k \wedge cell(k, s) = X \wedge$

$\quad curn(s) \neq k+1 \wedge cell(k+1, s) = X \wedge curp(s) = k+2) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k(curn(s) = k-2 \wedge curp(s) \neq k-1 \wedge cell(k-1, s) = X \wedge curp(s) \neq k \wedge$

$\quad cell(k, s) = X \wedge curp(s) \neq k+1 \wedge cell(k+1, s) = X) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k(curp(s) \neq k-1 \wedge cell(k-1, s) = X \wedge curp(s) \neq k \wedge cell(k, s) = X \wedge$

$\quad curp(s) \neq k+1 \wedge cell(k+1, s) = X \wedge curp(s) = k+1)$

$\equiv$ by FOL (eliminate contradiction and simplification)

$Legal(s) \wedge \exists k(curn(s) = k-2 \wedge cell(k-1, s) = X \wedge cell(k, s) = X \wedge cell(k+1, s) = X) \vee$

$Legal(s) \wedge \exists k(cell(k-1, s) = X \wedge cell(k, s) = X \wedge cell(k+1, s) = X \wedge curp(s) = k+2) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k(curn(s) = k-1 \wedge cell(k, s) = X \wedge cell(k+1, s) = X) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k((curn(s) = k-1 \vee cell(k-1, s) = X) \wedge (curn(s) = k \vee cell(k, s) = X) \wedge$

$\quad (curn(s) = k+1 \vee cell(k+1, s) = X) \wedge curp(s) = k+2) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k(curn(s) = k-2 \wedge (curp(s) = k-1 \vee cell(k-1, s) = X) \wedge$

$(curp(s) = k \lor cell(k, s) = X) \land (curp(s) = k + 1 \lor cell(k + 1, s) = X)) \lor$

$Legal(s) \land turn(s) = X \land \exists k(cell(k - 1, s) = X \land cell(k, s) = X \land curp(s) = k + 1) \lor$

$Legal(s) \land turn(s) = O \land \exists k(curn(s) \neq k - 1 \land cell(k - 1, s) = X \land curn(s) \neq k \land$

$cell(k, s) = X \land curn(s) \neq k + 1 \land cell(k + 1, s) = X \land curp(s) = k + 2) \lor$

$Legal(s) \land turn(s) = O \land \exists k(curn(s) = k - 2 \land curp(s) \neq k - 1 \land cell(k - 1, s) = X \land$

$curp(s) \neq k \land cell(k, s) = X \land curp(s) \neq k + 1 \land cell(k + 1, s) = X)$

$\equiv$ by subsumption (line 7 by 2, line 8 by 1) and reordering, and distribution or $\land$

$Legal(s) \land \exists k(curn(s) = k - 2 \land cell(k - 1, s) = X \land cell(k, s) = X \land cell(k + 1, s) = X) \lor$

$Legal(s) \land \exists k(cell(k - 1, s) = X \land cell(k, s) = X \land cell(k + 1, s) = X \land curp(s) = k + 2) \lor$

$Legal(s) \land turn(s) = X \land \exists k(curn(s) = k - 1 \land cell(k, s) = X \land cell(k + 1, s) = X) \lor$

$Legal(s) \land turn(s) = X \land \exists k(cell(k - 1, s) = X \land cell(k, s) = X \land curp(s) = k + 1) \lor$

$Legal(s) \land turn(s) = X \land \exists k(curn(s) = k - 1 \land curn(s) = k \land curn(s) = k + 1 \land curp(s) = k + 2) \lor$

$Legal(s) \land turn(s) = X \land \exists k(curn(s) = k - 1 \land curn(s) = k \land cell(k + 1, s) = X \land curp(s) = k + 2) \lor$

$Legal(s) \land turn(s) = X \land \exists k(curn(s) = k - 1 \land cell(k, s) = X \land curn(s) = k + 1 \land curp(s) = k + 2) \lor$

$Legal(s) \land turn(s) = X \land \exists k(curn(s) = k - 1 \land cell(k, s) = X \land cell(k + 1, s) = X \land curp(s) = k + 2) \lor$

$Legal(s) \land turn(s) = X \land \exists k(cell(k - 1, s) = X \land curn(s) = k \land curn(s) = k + 1 \land curp(s) = k + 2) \lor$

$Legal(s) \land turn(s) = X \land \exists k(cell(k - 1, s) = X \land curn(s) = k \land cell(k + 1, s) = X \land curp(s) = k + 2) \lor$

$Legal(s) \land turn(s) = X \land \exists k(cell(k - 1, s) = X \land cell(k, s) = X \land curn(s) = k + 1 \land curp(s) = k + 2) \lor$

$Legal(s) \land turn(s) = X \land \exists k(cell(k - 1, s) = X \land cell(k, s) = X \land cell(k + 1, s) = X \land curp(s) = k + 2) \lor$

$Legal(s) \land turn(s) = X \land \exists k(curn(s) = k - 2 \land curp(s) = k - 1 \land curp(s) = k \land curp(s) = k + 1) \lor$

$Legal(s) \land turn(s) = X \land \exists k(curn(s) = k - 2 \land curp(s) = k - 1 \land curp(s) = k \land cell(k + 1, s) = X) \lor$

$Legal(s) \land turn(s) = X \land \exists k(curn(s) = k - 2 \land curp(s) = k - 1 \land cell(k, s) = X \land curp(s) = k + 1) \lor$

$Legal(s) \land turn(s) = X \land \exists k(curn(s) = k - 2 \land curp(s) = k - 1 \land cell(k, s) = X \land cell(k + 1, s) = X) \lor$

$Legal(s) \land turn(s) = X \land \exists k(curn(s) = k - 2 \land cell(k - 1, s) = X \land curp(s) = k \land curp(s) = k + 1) \lor$

$Legal(s) \land turn(s) = X \land \exists k(curn(s) = k - 2 \land cell(k - 1, s) = X \land curp(s) = k \land cell(k + 1, s) = X) \lor$

$Legal(s) \wedge turn(s) = X \wedge \exists k(curn(s) = k-2 \wedge cell(k-1, s) = X \wedge cell(k, s) = X \wedge curp(s) = k+1) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k(curn(s) = k-2 \wedge cell(k-1, s) = X \wedge cell(k, s) = X \wedge cell(k+1, s) = X) \vee$

$\equiv$ by subsumption (line 8/19 by 3, line 12 by 2, line 20 and 1) and removal of contradictions

$Legal(s) \wedge \exists k(curn(s) = k-2 \wedge cell(k-1, s) = X \wedge cell(k, s) = X \wedge cell(k+1, s) = X) \vee$

$Legal(s) \wedge \exists k(cell(k-1, s) = X \wedge cell(k, s) = X \wedge cell(k+1, s) = X \wedge curp(s) = k+2) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k(curn(s) = k-1 \wedge cell(k, s) = X \wedge cell(k+1, s) = X) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k(cell(k-1, s) = X \wedge cell(k, s) = X \wedge curp(s) = k+1) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k(cell(k-1, s) = X \wedge curn(s) = k \wedge cell(k+1, s) = X \wedge curp(s) = k+2) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k(cell(k-1, s) = X \wedge cell(k, s) = X \wedge curn(s) = k+1 \wedge curp(s) = k+2) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k(curn(s) = k-2 \wedge curp(s) = k-1 \wedge cell(k, s) = X \wedge cell(k+1, s) = X) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k(curn(s) = k-2 \wedge cell(k-1, s) = X \wedge curp(s) = k \wedge cell(k+1, s) = X)$

$\equiv$ by R_0 from the previous step

$R_0 \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k.(curn(s) = k-1 \wedge cell(k, s) = X \wedge cell(k+1, s) = X) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k.(cell(k-1, s) = X \wedge cell(k, s) = X \wedge curp(s) = k+1) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k.(cell(k-1, s) = X \wedge cell(k, s) = X \wedge curn(s) = k+1 \wedge curp(s) = k+2) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k.(curn(s) = k-2 \wedge curp(s) = k-1 \wedge cell(k, s) = X \wedge cell(k+1, s) = X) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k.(cell(k-1, s) = X \wedge curn(s) = k \wedge cell(k+1, s) = X \wedge curp(s) = k+2) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k.(curn(s) = k-2 \wedge cell(k-1, s) = X \wedge curp(s) = k \wedge cell(k+1, s) = X)$

This approximation evaluates to true if s is such that X can win in at most 1 step, i.e., these are situations where there are 3 X marks in a row on either side, or there are 2 X marks in a row on either side and it is player X's turn (X can put the third X mark at the next step), or there are 2 X marks separated by a cell in any of the marking positions and it is player X's turn (X can put the third X mark at the next step). □

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{TTT1D}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_2(s)} \doteq \mathbf{Wins(X, s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{R_1}) \equiv$ by lemma L 3.3.1

(the proof is similar to R_1 and the details are available at `http://www.cse.yorku.ca/~skmiec`)

$\equiv$

$Legal(s) \wedge \exists k(curn(s) = k-2 \wedge cell(k-1,s) = X \wedge cell(k,s) = X \wedge cell(k+1,s) = X) \vee$

$Legal(s) \wedge \exists k(cell(k-1,s) = X \wedge cell(k,s) = X \wedge cell(k+1,s) = X \wedge curp(s) = k+2) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k.(curn(s) = k-1 \wedge cell(k,s) = X \wedge cell(k+1,s) = X) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k.(cell(k-1,s) = X \wedge cell(k,s) = X \wedge curp(s) = k+1) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k.(cell(k-1,s) = X \wedge cell(k,s) = X \wedge curn(s) = k+1 \wedge curp(s) = k+2) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k.(curn(s) = k-2 \wedge curp(s) = k-1 \wedge cell(k,s) = X \wedge cell(k+1,s) = X) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k.(cell(k-1,s) = X \wedge curn(s) = k \wedge cell(k+1,s) = X \wedge curp(s) = k+2) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k.(curn(s) = k-2 \wedge cell(k-1,s) = X \wedge curp(s) = k \wedge cell(k+1,s) = X)$

$Legal(s) \wedge turn(s) = O \wedge \exists k.(curp(s) < k-1 \wedge curn(s) = k-1 \wedge cell(k,s) = X \wedge cell(k+1,s) = X) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k.(cell(k-1,s) = X \wedge cell(k,s) = X \wedge curp(s) = k+1 \wedge k+1 < curn(s)) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k.(curn(s) < k-1 \wedge cell(k-1,s) = X \wedge cell(k,s) = X \wedge curp(s) = k+1) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k.(curn(s) = k-1 \wedge cell(k,s) = X \wedge cell(k+1,s) = X \wedge k+1 < curp(s)) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k.(cell(k-1,s) = X \wedge cell(k,s) = X \wedge k+1 = curn(s) \wedge curp(s) = k+1) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k.(cell(k-1,s) = X \wedge cell(k,s) = X \wedge curn(s) = k+2 \wedge curp(s) = k+2) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k.(curn(s) = k-1 \wedge curp(s) = k-1 \wedge cell(k,s) = X \wedge cell(k+1,s) = X) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k.(curn(s) = k-2 \wedge curp(s) = k-2 \wedge cell(k,s) = X \wedge cell(k+1,s) = X)$

This approximation evaluates to true if s is such that X can win in at most 2 steps. $\square$

. . .

(the proofs for steps R_3 to R_8 are available at `http://www.cse.yorku.ca/~skmiec`)

. . .

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{TTT1D}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_9(s)} \doteq \mathbf{Wins(X,s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{R_8}) \equiv$ by lemma L 3.3.1

(the proof is similar to R_1 and the details are available at `http://www.cse.yorku.ca/~skmiec`)

$\equiv$

$Legal(s) \wedge \exists k(curn(s) = k-2 \wedge cell(k-1,s) = X \wedge cell(k,s) = X \wedge cell(k+1,s) = X) \vee$

$Legal(s) \wedge \exists k(cell(k-1,s) = X \wedge cell(k,s) = X \wedge cell(k+1,s) = X \wedge curp(s) = k+2) \vee$

$Legal(s) \wedge turn(s) = X \wedge curp(s) < curn(s) - 2 \vee$

$Legal(s) \wedge turn(s) = X \wedge curp(s) = curn(s) - 2 \vee$

$Legal(s) \wedge turn(s) = X \wedge curp(s) = curn(s) - 1 \vee$

$Legal(s) \wedge turn(s) = X \wedge curp(s) = curn(s) \vee$

$Legal(s) \wedge turn(s) = X \wedge curn(s) < curp(s) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k.(curn(s) = k-1 \wedge cell(k,s) = X \wedge cell(k+1,s) = X) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k.(cell(k-1,s) = X \wedge cell(k,s) = X \wedge curp(s) = k+1) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k.(curp(s) = k-1 \wedge curn(s) = k \wedge cell(k+1,s) = X) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k.(cell(k-1,s) = X \wedge curp(s) = k \wedge curn(s) = k+1) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k.(cell(k-1,s) = X \wedge cell(k,s) = X \wedge curp(s) = k+2 \wedge curn(s) = k+3) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k.(curp(s) = k-3 \wedge curn(s) = k-2 \wedge cell(k,s) = X \wedge cell(k+1,s) = X) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k.(curp(s) < k-1 \wedge curn(s) = k \wedge cell(k+1,s) = X) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k.(cell(k-1,s) = X \wedge curp(s) = k \wedge k+1 < curn(s)) \vee$

$Legal(s) \wedge turn(s) = O \wedge curp(s) < curn(s) - 3$

$Legal(s) \wedge turn(s) = O \wedge curp(s) = curn(s) - 2$

$Legal(s) \wedge turn(s) = O \wedge curp(s) = curn(s) - 1 \vee$

$Legal(s) \wedge turn(s) = O \wedge curn(s) = curp(s) \vee$

$Legal(s) \wedge turn(s) = O \wedge curn(s) < curp(s) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k.(curp(s) < k-1 \wedge curn(s) = k-1 \wedge cell(k,s) = X \wedge cell(k+1,s) = X) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k.(cell(k-1,s) = X \wedge cell(k,s) = X \wedge curp(s) = k+1 \wedge k+1 < curn(s)) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k.(curn(s) < k-1 \wedge cell(k-1,s) = X \wedge cell(k,s) = X \wedge curp(s) = k+1) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k.(curn(s) = k - 1 \wedge cell(k, s) = X \wedge cell(k + 1, s) = X \wedge k + 1 < curp(s)) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k.(cell(k - 1, s) = X \wedge cell(k, s) = X \wedge k + 1 = curn(s) \wedge curp(s) = k + 1) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k.(cell(k - 1, s) = X \wedge cell(k, s) = X \wedge curn(s) = k + 2 \wedge curp(s) = k + 2) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k.(curn(s) = k - 1 \wedge curp(s) = k - 1 \wedge cell(k, s) = X \wedge cell(k + 1, s) = X) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k.(curn(s) = k - 2 \wedge curp(s) = k - 2 \wedge cell(k, s) = X \wedge cell(k + 1, s) = X)$

$Legal(s) \wedge turn(s) = O \wedge \exists k.(cell(k - 1, s) = X \wedge curp(s) = k \wedge curn(s) = k + 2) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k.(curn(s) < k - 1 \wedge cell(k - 1, s) = X \wedge curp(s) = k) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k.(cell(k - 1, s) = X \wedge cell(k, s) = X \wedge curp(s) = k + 2 \wedge curn(s) = k + 4) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k.(curp(s) = k - 3 \wedge curn(s) = k - 1 \wedge cell(k, s) = X \wedge cell(k + 1, s) = X) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k.(cell(k - 1, s) = X \wedge curp(s) = k \wedge k + 2 < curn(s)) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k.(curp(s) = k - 2 \wedge curn(s) = k \wedge cell(k + 1, s) = X) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k.(curn(s) = k \wedge cell(k + 1, s) = X \wedge k + 1 < curp(s)) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k.(cell(k - 1, s) = X \wedge cell(k, s) = X \wedge curp(s) = k + 1 \wedge curn(s) = k + 3) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k.(curp(s) = k - 4 \wedge curn(s) = k - 2 \wedge cell(k, s) = X \wedge cell(k + 1, s) = X) \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists k.(curp(s) < k - 2 \wedge curn(s) = k \wedge cell(k + 1, s) = X)$

This approximation evaluates to true if s is such that X can win in at most 9 steps. □

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{TTT1D}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_{10}(s)} \doteq \mathbf{Wins(X, s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{R_9}) \equiv$ by lemma L 3.3.1

(the proof is similar to R_1 and the details are available at `http://www.cse.yorku.ca/~skmiec`)

$\equiv$

$Legal(s)$

This approximation evaluates to true if s is such that X can win in at most 10 steps. □

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{TTT1D}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_{11}(s)} \doteq \mathbf{Wins(X, s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{R_{10}}) \equiv$ by lemma L 3.3.1

(the proof is similar to R_1 and the details are available at `http://www.cse.yorku.ca/~skmiec`)

$\equiv$

$Legal(s)$

This approximation evaluates to true if s is such that X can win in at most 11 steps. Here is an interesting fact – indeed it may take up to 11 steps to win the game even if the agents cooperate. If we have no restrictions (and here we reason about all games) that $curn(S) \leq curp(S)$ then we can imagine an initial situation S_x like "pBBn" where n represents the cell pointed to by $curn(S_x)$, p represents the cell pointed to by $curp(S_x)$, and B represents blank cell. This is a situation S_x where the cell pointed to by the "right" marking position is followed by two blank cells and that is followed by a cell pointed to by the "left" marking position. If in a such configuration if it is player O's turn then indeed it will require 11 steps to win the game as player actions interfere and overwrite the other player's markings.

$\square$

Thus the fixpoint expansion procedure converges in the 11^{th} step as we have:

$$\mathcal{D}_{ca}^{TTT1D} \cup \mathcal{D}_Z \models R_{10}(s) \ \equiv \ R_{11}(s)$$

And therefore by Theorem 1 of [DLP10]:

$$\mathcal{D}_{GS}^{TTT1D} \ \models \exists\Diamond Wins(X)[s] \ \equiv \ R_{10}(s) \ \equiv Legal(s)$$

It follows by the initial state axioms that $\mathcal{D}_{GS}^{TTT1D} \models \exists\Diamond Wins(X)[S_0]$, i.e., X may eventually win in the initial situation as it is legal.

Note: the method converges more quickly and the proof becomes much easier if you add P 3.3.8 to the set of axioms used by the method.

3.3.3 Existence of a Winning Strategy

The existence of a strategy to ensure $Wins(X)$ by group $G = \{X\}$ can be represented by the following formula:

$$\langle\langle\{X\}\rangle\rangle\Diamond Wins(X) \doteq \mu Z.\, Wins(X) \vee \langle\langle\{X\}\rangle\rangle \bigcirc Z$$

We begin by applying the De Giacomo et al. [DLP10] method and try to show that successive approximates are equivalent using axioms $\mathcal{D}_{GS}^{TTT1D}$ without foundational axioms Σ:

$$\mathcal{D}_{GS}^{TTT1D} \setminus \Sigma \models \langle\langle\{X\}\rangle\rangle\Diamond Wins(X)$$

This formula can be verified by employing the [DLP10] method using regression and fixpoint approximation, until we converge. The technique does not always converge and this needs to be checked as we proceed. The symbolic manipulations are quite numerous and more details are available in the on-line appendix `http://www.cse.yorku.ca/~skmiec`, but here the results are collected. The approximations are as follows:

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{TTT1D}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_0(s)} \doteq \mathbf{Wins(X, s)} \vee \mathcal{R}(\langle\langle\{\mathbf{X}\}\rangle\rangle \bigcirc \mathbf{False})$

$\equiv$ by lemma L 3.3.2

$\exists k(Legal(s) \wedge cell(k-1, s) = X \wedge cell(k, s) = X \wedge cell(k+1, s) = X \wedge curn(s) = k-2) \vee$

$\exists k(Legal(s) \wedge cell(k-1, s) = X \wedge cell(k, s) = X \wedge cell(k+1, s) = X \wedge curp(s) = k+2) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(False) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(False) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(False) \wedge \mathcal{R}(False)$

$\equiv$

$Legal(s) \wedge \exists k(curn(s) = k-2 \wedge cell(k-1, s) = X \wedge cell(k, s) = X \wedge cell(k+1, s) = X \vee$

$Legal(s) \wedge \exists k(cell(k-2, s) = X \wedge cell(k-1, s) = X \wedge cell(k, s) = X \wedge curp(s) = k+1)$

This approximation evaluates to true if s is such that X is winning in s already (in no steps), i.e., these are situations where there are 3 X marks in a row on either side. $\square$

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{TTT1D}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_1(s)} \doteq \mathbf{Wins(X,s)} \vee \mathcal{R}(\langle\langle\{\mathbf{X}\}\rangle\rangle \bigcirc \mathbf{R_0})$

(the proof is by FOL, L3.3.2, P3.3.2, P3.3.6, P3.3.3, P3.3.4, P3.3.7)

$\equiv$

$Legal(s) \wedge \exists k(cell(k-1,s) = X \wedge cell(k,s) = X \wedge cell(k+1,s) = X \wedge curn(s) = k-2) \vee$

$Legal(s) \wedge \exists k(cell(k-1,s) = X \wedge cell(k,s) = X \wedge cell(k+1,s) = X \wedge curp(s) = k+2) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k(curn(s) = k-1 \wedge cell(k,s) = X \wedge cell(k+1,s) = X) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k cell(k-2,s) = X \wedge cell(k-1,s) = X \wedge curp(s) = k) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k(cell(k-2,s) = X \wedge cell(k-1,s) = X \wedge curn(s) = k \wedge curp(s) = k+1) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k(curn(s) = k-2 \wedge curp(s) = k-1 \wedge cell(k,s) = X \wedge cell(k+1,s) = X) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k(cell(k-2,s) = X \wedge curn(s) = k-1 \wedge cell(k,s) = X \wedge curp(s) = k+1) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k(curn(s) = k-2 \wedge cell(k-1,s) = X) \wedge curp(s) = k \wedge cell(k+1,s) = X)$

This approximation evaluates to true if s is such that X can win in at most 1 step, i.e.,

As in R_0 and if its X's turn then if XX on right of *curn*, or XX on left of *curp*, or configurations: XnXp, or XXnp, or npXX, or nXpX, where n is *curn* and p is *curp*. $\square$

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{TTT1D}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_2(s)} \doteq \mathbf{Wins(X,s)} \vee \mathcal{R}(\langle\langle\{\mathbf{X}\}\rangle\rangle \bigcirc \mathbf{R_1})$

(the proof is by FOL, L3.3.2, P3.3.2, P3.3.6, P3.3.3, P3.3.4, P3.3.7)

$\equiv$

$Legal(s) \wedge \exists k(cell(k-1,s) = X \wedge cell(k,s) = X \wedge cell(k+1,s) = X \wedge curn(s) = k-2) \vee$

$Legal(s) \wedge \exists k(cell(k-1,s) = X \wedge cell(k,s) = X \wedge cell(k+1,s) = X \wedge curp(s) = k+2) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k(curn(s) = k-1 \wedge cell(k,s) = X \wedge cell(k+1,s) = X) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k cell(k-2,s) = X \wedge cell(k-1,s) = X \wedge curp(s) = k) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k(cell(k-2,s) = X \wedge cell(k-1,s) = X \wedge curn(s) = k \wedge curp(s) = k+1) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k(curn(s) = k-2 \wedge curp(s) = k-1 \wedge cell(k,s) = X \wedge cell(k+1,s) = X) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k(cell(k-2, s) = X \wedge curn(s) = k-1 \wedge cell(k, s) = X \wedge curp(s) = k+1) \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists k(curn(s) = k-2 \wedge cell(k-1, s) = X) \wedge curp(s) = k \wedge cell(k+1, s) = X) \vee$

$Legal(s) \wedge turn(s) = O \wedge$

$\quad \exists m.(curn(s) < m-2 \wedge cell(m-2, s) = X \wedge cell(m-1, s) = X \wedge curp(s) = m) \wedge$

$\quad \exists n.(curn(s) = n \wedge cell(n+1, s) = X \wedge cell(n+2, s) = X \wedge n+2 < curp(s))$

This approximation evaluates to true if s is such that X can win in at most 2 step. □

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{TTT1D}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_3(s)} \doteq \mathbf{Wins(X, s)} \vee \mathcal{R}(\langle\langle\{\mathbf{X}\}\rangle\rangle \bigcirc \mathbf{R_2})$

(the proof is by FOL, L3.3.2, P3.3.2, P3.3.6, P3.3.3, P3.3.4, P3.3.7)

$\equiv R_2$

This approximation evaluates to true if s is such that X can win in at most 3 step.

——————————————————————— □

Thus the fixpoint expansion procedure converges in the 3rd step as we have:

$$\mathcal{D}_{ca}^{TTT1D} \cup \mathcal{D}_Z \models R_2(s) \;\equiv\; R_3(s)$$

And therefore by Theorem 1 of [DLP10]:

$$\mathcal{D}_{GS}^{TTT1D} \models \langle\langle\{X\}\rangle\rangle\Diamond Wins(X)[s] \;\equiv\; R_2(s)$$

It follows by the initial state axioms that $\mathcal{D}_{GS}^{TTT1D} \models \neg\langle\langle\{X\}\rangle\rangle\Diamond Wins(X)[S_0]$ i.e., there is no winning strategy for X in the initial situation. But $\mathcal{D}_{GS}^{TTT1D} \models \langle\langle\{X\}\rangle\rangle\Diamond Wins(X)[S_1]$
where $S_1 = do(markn(O), do(markp(X), do(markn(O), do(markp(X), S_0))))$ i.e., there is a winning strategy for X in a situation where X marked twice on the left and O marked twice on the right.

3.4 Mark Down (MD)

In this section we specify another simple game and apply the De Giacomo et al. [DLP10] fixpoint iteration method to verify some temporal properties. If we apply the method as originally specified and only use the unique name and domain closure axioms for actions (to which we have added the axiomatization of the integers), we don't get convergence in a finite number of steps. But if use add some additional facts entailed by the whole theory, then we do get convergence in a finite number of steps.

The Mark Down (MD) game that we have designed involves an infinite vector of cells, one for each integer. There is a pointer maintained in the game, which we call the current position. Initially the current position is set to $+\infty$. There are 2 players in the game: X and O. Players take turns and initially it is player X's turn. The actions are $move1$ to decrease the current position by 1 if the current position is not $+\infty$, $move2$ to decrease the current position by 2 if the current position is not $+\infty$, and action $init(p, n)$ that is only allowed for X and when the current position is $+\infty$, i.e. in the initial position. The action $init(p, n)$ sets the current position to an integer of choice between some given constants $MinInit$ and $MaxInit$ defined by the game. For these constatnts it is true that $3 \leq MinInit < MaxInit$ and $MaxInit \neq +\infty$. Player O can never make a move when the current position is $+\infty$. The goal of player X is to reach a state where the current position is 1 (after player X's turn) in which case player X wins, and similarly for player O. A sample sequence of moves for a game where $MinInit = 3$ and $MaxInit = 5$ could be as follows: X performs $init(4)$ and the current position is set to 4, O performs $move2$ and the current position is set to 2, X performs $move1$ and the current position is set to 1 and X wins. It is possible for the game go on forever without the goal being reached. An example of such scenario for a game where $MinInit = 3$ and $MaxInit = 5$ would be as follows: X performs $init(4)$ and the current position is set to 4, O performs $move2$ and the current position is set to 2, X performs $move2$ and the current position is set to 0, the actions beyond this do not matter as the current position can only decrease and it cannot reach cell 1.

3.4.1 Situation Calculus Game Structure Axiomatization of MD Domain

The Situation Calculus Game Structure Axiomatization is defined as:

$\mathcal{D}_{GS}^{MD} = \Sigma \cup \mathcal{D}_{poss}^{MD} \cup \mathcal{D}_{ssa}^{MD} \cup \mathcal{D}_{Legal}^{MD} \cup \mathcal{D}_{ca}^{MD} \cup \mathcal{D}_{S_0}^{MD} \cup \mathcal{D}_{Z}$

where $\mathcal{D}_Z$ is a suitable axiomatization of integers

Fluents

- $cur(s)$ - functional fluent indicating the latest marking position, whose domain is the integers and $+\infty$
- $turn(s)$ - functional fluent indicating which agent is to take the next action, with its domain being the agent set $\{X, O\}$

Derived Fluents

- $Wins(p, s) \doteq Legal(s) \land Agent(p) \land cur(s) = 1 \land \exists p'.\ Agent(p') \land turn(s) = p' \land p \neq p'$

Actions

- $init(p, n)$ - agent p sets the current position to be n
- $move1(p)$ - agent p reduces the current position by 1
- $move2(p)$ - agent p reduces the current position by 2

Precondition Axioms $\mathcal{D}_{poss}^{MD}$

These are precondition axioms, one per action, specifying when an action can physically be performed:

- $Poss(init(p, n), s) \equiv cur(s) = +\infty \land MinInit \leq n \leq MaxInit \land Agent(p)$

- $Poss(move1(p), s) \equiv cur(s) \neq +\infty \wedge Agent(p)$
- $Poss(move2(p), s) \equiv cur(s) \neq +\infty \wedge Agent(p)$

Successor State Axioms $\mathcal{D}^{MD}_{ssa}$

These are successor state axioms specifying how the fluents change as a result of actions:

- $cur(do(a, s)) = k \equiv cur(s) = +\infty \wedge \exists p.\ a = init(p, k)$

  $\vee\ cur(s) = k + 1 \wedge \exists p.\ a = move1(p) \vee cur(s) = k + 2 \wedge \exists p.\ a = move2(p)$
- $turn(do(a, s)) = p \equiv p = O \wedge turn(s) = X \vee p = X \wedge turn(s) = O$

Initial State Axioms $\mathcal{D}^{MD}_{S_0}$

These describe the initial situation:

- $cur(S_0) = +\infty$
- $turn(S_0) = X$
- $Legal(S_0)$
- $MinInit = 3$
- $MaxInit = 1000$

Legal Moves Axioms $\mathcal{D}^{MD}_{legal}$

These encode the rules of the game:

- $agent(init(p, n)) = p$
- $agent(move1(p)) = p$

- $agent(move2(p)) = p$
- $Control(p, s) \doteq \exists a.Legal(do(a, s)) \wedge agent(a) = p$
- $Legal(do(a, s)) \equiv Legal(s) \wedge \{$

$$a = init(X, n) \wedge MinInit \leq n \leq MaxInit \wedge cur(s) = +\infty \vee$$

$$\exists p.\ a = move1(p) \wedge turn(s) = p \wedge cur(s) \neq +\infty \vee$$

$$\exists p.\ a = move2(p) \wedge turn(s) = p \wedge cur(s) \neq +\infty\}$$

Unique Name and Domain Closure Axioms for Actions $\mathcal{D}_{ca}^{MD}$

These describe the uniqueness of action names and the fact that the domain of actions s closed:

- $\forall a.\ \{\ \exists p, n.\ a = init(p, n) \vee \exists p.\ a = move1(p) \vee \exists p.\ a = move2(p)\ \}$
- $\forall n, p, p'.\ \{\ init(p, n) \neq move2(p')\ \}$
- $\forall n, p, p'.\ \{\ init(p, n) \neq move2(p')\ \}$
- $\forall p, p'.\ \{\ move1(p) \neq move2(p')\ \}$
- $\forall p, p'.\ \{\ move1(p) = move1(p') \supset p = p'\ \}$
- $\forall p, p'.\ \{\ move2(p) = move2(p') \supset p = p'\ \}$
- $\forall p, p', n, n'.\ \{\ init(p, n) = init(p', n') \supset p = p' \wedge n = n'\ \}$
- $\forall p.\ \{\ Agent(p) \equiv (p = X \vee p = O)\}$
- $X \neq O$
- $3 \leq MinInit \leq MaxInit \wedge MaxInit \neq +\infty$
- $\forall n.\ n \neq +\infty$, where n ranges over the set of integers

The complete theory also includes the foundational axioms Σ of the Situation Calculus. It can be noticed that this is clearly an infinite state domain as the set of positions that can be visited is infinite.

Propositions and Lemmas

We derived the following propositions, some of which are used in later proofs:

P 3.4.1. *The concrete case of the Wins fluent.*

$$\mathcal{D}_{ca}^{MD} \models Wins(X,s) \equiv Legal(s) \wedge turn(s) = O \wedge cur(s) = 1$$

P 3.4.2. *Regression of concrete cases of the turn fluent.*

$$\mathcal{D}_{ca}^{MD} \models \mathcal{R}(turn(do(a,s)) = X) \equiv turn(s) = O$$

$$\mathcal{D}_{ca}^{MD} \models \mathcal{R}(turn(do(a,s)) = O) \equiv turn(s) = X$$

P 3.4.3. *Regression of concrete cases of the cur fluent.*

$$\mathcal{D}_{ca}^{MD} \cup \mathcal{D}_Z \models \mathcal{R}(cur(do(init(p,n),s) = k) \equiv n = k$$

$$\mathcal{D}_{ca}^{MD} \cup \mathcal{D}_Z \models \mathcal{R}(cur(do(move1(p),s) = k) \equiv cur(s) = k+1$$

$$\mathcal{D}_{ca}^{MD} \cup \mathcal{D}_Z \models \mathcal{R}(cur(do(move2(p),s) = k) \equiv cur(s) = k+2$$

P 3.4.4. *Regression of concrete cases for the Legal fluent.*

$$\mathcal{D}_{ca}^{MD} \models \mathcal{R}(Legal(do(init(O,n),s)) \equiv False$$

$$\mathcal{D}_{ca}^{MD} \models \mathcal{R}(Legal(do(init(X,n),s)) \equiv Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty$$
$$\wedge MinInit \leq nleqMaxInit$$

$$\mathcal{D}_{ca}^{MD} \models \mathcal{R}(Legal(do(move1(p),s))) \equiv Legal(s) \wedge turn(s) = p \wedge cur(s) \neq +\infty$$

$$\mathcal{D}_{ca}^{MD} \models \mathcal{R}(Legal(do(move2(p),s))) \equiv Legal(s) \wedge turn(s) = p \wedge cur(s) \neq +\infty$$

$$\mathcal{D}_{ca}^{MD} \models \mathcal{R}(Legal(do(move1(p),s))) \supset cur(s) \neq +\infty$$

$$\mathcal{D}_{ca}^{MD} \models \mathcal{R}(Legal(do(move2(p),s))) \supset cur(s) \neq +\infty$$

P 3.4.5. *Concrete cases for the Control fluent from $\mathcal{D}_{legal}^{MD}$.*

$$\mathcal{D}_{ca}^{MD} \models Control(X,s) \equiv Legal(s) \wedge turn(s) = X \vee cur(s) = +\infty$$

$$\mathcal{D}_{ca}^{MD} \models Control(O,s) \equiv Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty$$

In this domain, for groups $G = \{X\}$ or $G = \{X, O\}$ ensuring that φ holds next can be established by considering only the *init*, *move*1 and *move*2 actions as the following lemmas show:

L 3.4.1. *Regression of the concrete case of the* $\exists \bigcirc \varphi$ *definition based on domain closure.*

$$\begin{aligned}
&\mathcal{D}_{ca}^{MD} \models \mathcal{R}(\exists \bigcirc \varphi[s]) \equiv \\
&\quad Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge \\
&\qquad \exists n.\ MinInit \leq n \leq MaxInit \wedge \mathcal{R}(\varphi[do(init(X,n),s)]) \vee \\
&\quad Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge \mathcal{R}(\varphi[do(move1(X),s)]) \vee \\
&\quad Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge \mathcal{R}(\varphi[do(move2(X),s)]) \vee \\
&\quad Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge \mathcal{R}(\varphi[do(move1(O),s)]) \vee \\
&\quad Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge \mathcal{R}(\varphi[do(move1(O),s)])
\end{aligned}$$

L 3.4.2. *Regression of the concrete case of the* $\langle\langle\{X\}\rangle\rangle \bigcirc \varphi$ *definition based on domain closure.*

$$\begin{aligned}
&\mathcal{D}_{ca}^{MD} \models \mathcal{R}(\langle\langle\{X\}\rangle\rangle \bigcirc \varphi[s]) \equiv \\
&\quad Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge \\
&\qquad \exists n.\ MinInit \leq n \leq MaxInit \wedge \mathcal{R}(\varphi[do(init(X,n),s)]) \vee \\
&\quad Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge \mathcal{R}(\varphi[do(move1(X),s)]) \vee \\
&\quad Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge \mathcal{R}(\varphi[do(move2(X),s)]) \vee \\
&\quad Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge \mathcal{R}(\varphi[do(move1(O),s)]) \wedge \mathcal{R}(\varphi[do(move2(O),s)])
\end{aligned}$$

3.4.2 Possibility of Winning

The property that it is possible for X to eventually win can be represented by the following formula:

$$\exists \Diamond Wins(X) \doteq \mu Z.\{\ Wins(X)\ \vee\ \exists \bigcirc Z\ \}$$

We begin by applying the De Giacomo et al. [DLP10] method and try to show that successive approximates are equivalent using only the unique name and domain closure axioms for actions $\mathcal{D}_{ca}^{MD}$ and the axiomatization of the integers $\mathcal{D}_Z$:

$\mathcal{D}_{ca}^{MD} \cup \mathcal{D}_Z \models \exists\Diamond Wins(X)$

This formula can be verified by employing the [DLP10] method using regression and fixpoint approximation, until we converge. The technique does not always converge and this needs to be checked as we proceed. The approximations are as follows:

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MD}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_0(s)} \doteq \mathbf{Wins(X, s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{False}) \equiv$ by P 3.4.1, L 3.4.1 and FOL

$Legal(s) \wedge turn(s) = O \wedge cur(s) = 1$

This approximation evaluates to true if s is such that X is winning in s already (in no steps) i.e. these are situations where the current position is 1 and now it is player O's turn. □

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MD}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_1(s)} \doteq \mathbf{Wins(X, s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{R_0}) \equiv$ by P 3.4.1, L 3.4.1 and FOL

$Legal(s) \wedge turn(s) = O \wedge cur(s) = 1 \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge$

$\exists n.\ MinInit \leq n \leq MaxInit \wedge \mathcal{R}(R_0[do(init(X, n), s)]) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_0[do(move1(X), s)]) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_0[do(move2(X), s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_0[do(move1(O), s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_0[do(move1(O), s)])$

$\equiv$ by R_0 from previous step and P 3.4.2, P 3.4.3, P 3.4.4

$Legal(s) \wedge turn(s) = O \wedge cur(s) = 1 \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge$

$\exists n.\ MinInit \leq n \leq MaxInit \wedge$

$\mathcal{R}(Legal(do(init(X, n), s))) \wedge turn(s) = X \wedge n = 1 \vee$ contradiction for n

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge cur(s) = 2 \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge cur(s) = 3 \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge$

$\mathcal{R}(Legal(do(move1(O), s))) \wedge turn(s) = X \wedge \mathcal{R}(cur(do(move1(O), s)) = 1) \vee$ contradiction

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge$

$\mathcal{R}(Legal(do(move1(O), s))) \wedge turn(s) = X \wedge \mathcal{R}(cur(do(move1(O), s)) = 1)$ contradiction

$\equiv$ by FOL

$Legal(s) \wedge turn(s) = O \wedge cur(s) = 1 \vee$

$Legal(s) \wedge turn(s) = X \wedge (cur(s) = 2 \vee cur(s) = 3)$

This approximation evaluates to true if s is a legal situation such that X can win in at most 1 step. These are legal situations where the current position is 1 and now it is player O's turn or the current position is 2 or 3 and it is player X's turn (X can perform *move*1 or *move*2 to win). □

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MD}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_2(s)} \doteq \mathbf{Wins(X, s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{R_1}) \equiv$ by P 3.4.1, L 3.4.1 and FOL

$Legal(s) \wedge turn(s) = O \wedge cur(s) = 1 \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge$

$\exists n.\ MinInit \leq n \leq MaxInit \wedge \mathcal{R}(R_1[do(init(X, n), s)]) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_1[do(move1(X), s)]) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_1[do(move2(X), s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_1[do(move1(O), s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_1[do(move1(O), s)])$

$\equiv$ by R_1 from previous step and P 3.4.2, P 3.4.3, P 3.4.4

$Legal(s) \wedge turn(s) = O \wedge cur(s) = 1 \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge \exists n.\ MinInit \leq n \leq MaxInit \wedge \mathcal{R}($

$Legal(do(init(X, n), s)) \wedge turn(s) = X \wedge n = 1 \vee$ contradiction on n

$Legal(do(init(X, n), s)) \wedge turn(s) = O \wedge cur(do(init(X, n), s)) = 2 \vee$ contradiction

$Legal(do(init(X, n), s)) \wedge turn(s) = O \wedge cur(do(init(X, n), s)) = 3) \vee$ contradiction

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge \mathcal{R}($

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge cur(s) = 2 \vee$

$Legal(do(move1(X), s)) \wedge turn(s) = O \wedge cur(do(move1(X), s)) = 2 \vee$ contradiction

$Legal(do(move1(X), s)) \wedge turn(s) = O \wedge cur(do(move1(X), s)) = 3) \vee$ contradiction

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge \mathcal{R}($

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge cur(s) = 3 \vee$

$Legal(do(move2(X), s)) \wedge turn(s) = O \wedge cur(do(move2(X), s)) = 2 \vee$ contradiction

$Legal(do(move2(X), s)) \wedge turn(s) = O \wedge cur(do(move2(X), s)) = 3) \vee$ contradiction

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge \mathcal{R}($

$Legal(do(move1(O), s)) \wedge turn(s) = X \wedge cur(do(move1(O), s)) = 1 \vee$ contradiction

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(s) = 3 \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(s) = 4) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge \mathcal{R}($

$Legal(do(move2(O), s)) \wedge turn(s) = X \wedge cur(do(move1(O), s)) = 1 \vee$ contradiction

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(s) = 4 \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(s) = 5)$

$\equiv$ by FOL

$Legal(s) \wedge turn(s) = X \wedge (cur(s) = 2 \vee cur(s) = 3) \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) = 1 \vee cur(s) = 3 \vee cur(s) = 4 \vee cur(s) = 5)$

This approximation evaluates to true if s is legal and such that X can win in at most 2 steps. These are situations where the current position is 1 and now it is player O's turn or the current position is 2 or 3 and it is player X's turn (X can perform *move*1 or *move*2 to win) or the current position is 3 or 4 or 5 and now it is player O's turn (O can take an action to allow X to finish in the next turn). □

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MD}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_3(s)} \doteq \mathbf{Wins(X, s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{R_2}) \equiv$ by P 3.4.1, L 3.4.1 and FOL

$Legal(s) \wedge turn(s) = O \wedge cur(s) = 1 \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge$

$\quad \exists n.\ MinInit \leq n \leq MaxInit \wedge \mathcal{R}(R_2[do(init(X, n), s)]) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_2[do(move1(X), s)]) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_2[do(move2(X), s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_2[do(move1(O), s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_2[do(move1(O), s)])$

$\equiv$ by R_2 from previous step and P 3.4.2, P 3.4.3, P 3.4.4

$Legal(s) \wedge turn(s) = O \wedge cur(s) = 1 \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge \exists n.\ MinInit \leq n \leq MaxInit \wedge ($

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge turn(s) = O \wedge (n = 2 \vee n = 3) \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge turn(s) = X \wedge (n = 1 \vee n = 3 \vee n = 4 \vee n = 5)$

$) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge ($

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge$

$\quad\quad (cur(s) = 3 \vee cur(s) = 4) \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge$

$\quad\quad (cur(s) = 2 \vee cur(s) = 4 \vee cur(s) = 5 \vee cur(s) = 6)$

$) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge ($

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge$

$\quad\quad (cur(s) = 4 \vee cur(s) = 5) \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge$

$\quad\quad (cur(s) = 3 \vee cur(s) = 5 \vee cur(s) = 6 \vee cur(s) = 7)$

$) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge ($

$\quad Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge$

$(cur(s) = 3 \vee cur(s) = 4) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge$

$(cur(s) = 2 \vee cur(s) = 4 \vee cur(s) = 5 \vee cur(s) = 6)$

$) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge ($

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge$

$(cur(s) = 4 \vee cur(s) = 5) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge$

$(cur(s) = 3 \vee cur(s) = 5 \vee cur(s) = 6 \vee cur(s) = 7)$

$)$

$\equiv$ by FOL

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge 3 \leq MinInit \leq MaxInit \wedge MinInit \leq 5 \vee$

$Legal(s) \wedge turn(s) = X \wedge (2 \leq cur(s) \leq 7) \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) = 1 \vee 3 \leq cur(s) \leq 5)$

This approximation evaluates to true if s is legal such that X can win in at most 3 steps. The intuitive interpretation follows from how the 2 players can cooperate or if the game constants are such that $MinInit$ is less or equal 5 (then there exists an initial move that the game can be won in up to 3 steps). ☐

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MD}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_4(s)} \doteq \mathbf{Wins(X,s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{R_3}) \equiv$ by P 3.4.1, L 3.4.1 and FOL

$Legal(s) \wedge turn(s) = O \wedge cur(s) = 1 \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge$

$\exists n.\ MinInit \leq n \leq MaxInit \wedge \mathcal{R}(R_3[do(init(X,n),s)]) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_3[do(move1(X),s)]) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_3[do(move2(X),s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_3[do(move1(O),s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_3[do(move1(O),s)])$

$\equiv$ by R_3 from previous step and P 3.4.2, P 3.4.3, P 3.4.4

$Legal(s) \wedge turn(s) = O \wedge cur(s) = 1 \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge \exists n.\ MinInit \leq n \leq MaxInit \wedge ($

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge turn(s) = O \wedge n = +\infty \wedge$

$3 \leq MinInit \leq MaxInit \wedge MinInit \leq 5 \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge turn(s) = O \wedge (2 \leq n \leq 7) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge turn(s) = X \wedge (n = 1 \vee 3 \leq n \leq 5)$

$) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge \mathcal{R}($

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(s) - 1 = +\infty \wedge$

$3 \leq MinInit \leq MaxInit \wedge MinInit \leq 5 \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge (2 \leq cur(s) - 1 \leq 7) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge (cur(s) - 1 = 1 \vee 3 \leq cur(s) - 1 \leq 5)$

$) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge \mathcal{R}($

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(s) - 2 = +\infty \wedge$

$3 \leq MinInit \leq MaxInit \wedge MinInit \leq 5 \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge (2 \leq cur(s) - 2 \leq 7) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge (cur(s) - 2 = 1 \vee 3 \leq cur(s) - 2 \leq 5)$

$) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge \mathcal{R}($

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(s) - 1 = +\infty \wedge$

$3 \leq MinInit \leq MaxInit \wedge MinInit \leq 5 \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge (2 \leq cur(s) - 1 \leq 7) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge (cur(s) - 1 = 1 \vee 3 \leq cur(s) - 1 \leq 5)$

$) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge \mathcal{R}($

$\quad Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(s) - 2 = +\infty \wedge$

$\quad\quad 3 \leq MinInit \leq MaxInit \wedge MinInit \leq 5 \vee$

$\quad Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge (2 \leq cur(s) - 2 \leq 7) \vee$

$\quad Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge (cur(s) - 2 = 1 \vee 3 \leq cur(s) - 2 \leq 5)$

$)$

$\equiv$ by FOL

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge 3 \leq MinInit \leq MaxInit \wedge MinInit \leq 5 \vee$

$Legal(s) \wedge turn(s) = X \wedge 2 \leq cur(s) \leq 7$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) = 1 \vee 3 \leq cur(s) \leq 9)$

This approximation evaluates to true if s legal and is such that X can win in at most 4 steps. The intuitive interpretation follows from how the 2 players can cooperate or if the game constants are such that $MinInit$ is less or equal 5 (then there exists an initial move that the game can be won in up to 4 steps). □

The subsequent results can be generalized (by induction) to the following formulas:

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MD}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_{2i-1}(s)} \equiv$

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge 3 \leq MinInit \leq MaxInit \wedge MinInit \leq 4i - 3 \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge (2 \leq cur(s) \leq 4i - 1) \vee$

$\quad Legal(s) \wedge turn(s) = O \wedge (cur(s) = 1 \vee 3 \leq cur(s) \leq 4i - 3)$

This approximation evaluates to true if s is legal and such that X can win in at most $2i + 1$ (odd) steps.

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MD}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_{2i}(s)} \equiv$

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge 3 \leq MinInit \leq MaxInit \wedge MinInit \leq 4i - 3 \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge 2 \leq cur(s) \leq 4i - 1 \vee$

$\quad Legal(s) \wedge turn(s) = O \wedge (cur(s) = 1 \vee 3 \leq cur(s) \leq 4i + 1)$

This approximation evaluates to true if s is legal and such that X can win in at most $2i$ (even) steps.

Is there a convergence in a finite number of steps? No.

The De Giacomo et al. [DLP10] iterative method for logical formula manipulation does not work using only the unique name and domain closure axioms for actions $\mathcal{D}_{ca}^{MD}$ and the axiomatization of the integers $\mathcal{D}_Z$. It will not converge in a finite number of steps as it can be observed that for all natural numbers i, $D_{ca}^{MD} \cup D_Z \not\models R_{i-1} \equiv R_i$, since one can always construct a model of $D_{ca}^{MD} \cup D_Z$ that will satisfy R_i and not R_{i-1}.

Will there be a convergence if we use additional facts entailed by the whole theory ? Yes.

The procedure will converge if we consider the following lemma:

L 3.4.3. *Existence of upper bound for fluent cur.*

$$\mathcal{D}_{GS}^{MD} \models \forall s.\ cur(s) = +\infty \lor cur(s) \leq MaxInit$$

Proof: We will prove by induction on situation. Indeed our base case is S_0 and we have $cur(S_0) = +\infty$. Now assume that for some situation s_i we have that $cur(s_i) = +\infty \lor cur(s_i) \leq MaxInit$. Based on the domain closure for actions we have that the successor situations to s_i can only be $do(init(p, n), s_i)$, or $do(move1(p), s_i)$, or $do(move2(p), s_i)$ where p is X or O. Based on the inductive assumption and regression we have that $cur(do(init(n), s_i)) = n \leq MaxInit$, and $cur(do(move1(p), s_i)) = cur(s_i) - 1 \leq MaxInit - 1 \leq MaxInit$ and $cur(do(move2(p), s_i)) = cur(s_i) - 2 \leq MaxInit - 2 \leq MaxInit$. Therefore $\forall a.\ cur(do(a, s)) = +\infty \lor cur(do(a, s)) \leq MaxInit$. $\square$

Since $MaxInit$ is a constant defined in the game, let us consider an integer number $x = \lceil (MaxInit + 3)/4 \rceil$. First, we can observe that $MinInit \leq MaxInit \leq 4x - 3$ and thus based on L3.4.3 we also have that either $cur(s) = +\infty$ or $cur(s) \leq 4x - 3$ holds. Likewise it holds that $MinInit \leq MaxInit \leq 4x + 1$ and also $cur(s) \leq 4x + 1$. The two consecutive steps R_{2x-1} and R_{2x} will simplify to:

$\mathcal{D}^{\mathbf{MD}}_{\mathbf{ca}} \cup \mathcal{D}_{\mathbf{Z}} \cup \{\mathbf{L3.4.3}\} \models \mathbf{R_{2x-1}(s)} \equiv$

$$\begin{aligned}
&Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \vee\\
&Legal(s) \wedge turn(s) = X \wedge 2 \leq cur(s) \vee\\
&Legal(s) \wedge turn(s) = O \wedge (cur(s) = 1 \vee 3 \leq cur(s))
\end{aligned}$$

$\mathcal{D}^{\mathbf{MD}}_{\mathbf{ca}} \cup \mathcal{D}_{\mathbf{Z}} \cup \{\mathbf{L3.4.3}\} \models \mathbf{R_{2x}(s)} \equiv$

$$\begin{aligned}
&Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \vee\\
&Legal(s) \wedge turn(s) = X \wedge 2 \leq cur(s) \vee\\
&Legal(s) \wedge turn(s) = O \wedge (cur(s) = 1 \vee 3 \leq cur(s))
\end{aligned}$$

Thus the fixpoint expansion procedure converges no later than in the step $2x$ as we have:

$$\mathcal{D}^{MD}_{ca} \cup \mathcal{D}_{Z} \ \cup \ \{L3.4.3\} \models R_{2x-1}(s) \equiv R_{2x}(s)$$

And therefore by Theorem 1 of [DLP10]:

$$\begin{aligned}
&\mathcal{D}^{MD}_{GS} \ \models \exists\Diamond Wins(X)[s] \ \equiv\\
&Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \vee\\
&Legal(s) \wedge turn(s) = X \wedge 2 \leq cur(s) \vee\\
&Legal(s) \wedge turn(s) = O \wedge (cur(s) = 1 \vee 3 \leq cur(s))
\end{aligned}$$

It follows by the initial state axioms that $\mathcal{D}^{MD}_{GS} \models \exists\Diamond Wins(X)[S_0]$, i.e., the goal may eventually be reached from the initial situation as it is legal and it is player X's turn and $cur(s) = +\infty$.

3.4.3 Existence of a Winning Strategy

The existence of a strategy to ensure $Wins(X)$ by group $G = \{X\}$ can be represented by the following formula:

$$\langle\langle\{X\}\rangle\rangle\Diamond Wins(X) \doteq \mu Z.\, Wins(X) \vee \langle\langle\{X\}\rangle\rangle \bigcirc Z$$

We begin by applying the De Giacomo et al. [DLP10] method:

$$\mathcal{D}_{ca}^{MD} \cup \mathcal{D}_Z \models \langle\langle \{X\} \rangle\rangle \Diamond Wins(X)$$

The regressed approximations are as follows:

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MD}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_0(s)} \doteq \mathbf{Wins(X, s)} \vee \mathcal{R}(\langle\langle \{\mathbf{X}\} \rangle\rangle \bigcirc \mathbf{False}) \equiv$ by P 3.4.1, L 3.4.2 and FOL

$Legal(s) \wedge turn(s) = O \wedge cur(s) = 1$

This approximation evaluates to true if s is such that X is winning in s already (in no steps). These are situations where the current position is 1 and now it is player O's turn. □

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MD}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_1(s)} \doteq \mathbf{Wins(X, s)} \vee \mathcal{R}(\langle\langle \{\mathbf{X}\} \rangle\rangle \bigcirc \mathbf{R_0}) \equiv$ by P 3.4.1, L 3.4.2 and FOL

$Legal(s) \wedge turn(s) = O \wedge cur(s) = 1 \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge$

$\quad \exists n.\ MinInit \leq n \leq MaxInit \wedge \mathcal{R}(R_0[do(init(X, n), s)]) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_0[do(move1(X), s)]) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_0[do(move2(X), s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_0[do(move1(O), s)]) \wedge \mathcal{R}(R_0[do(move2(O), s)])$

$\equiv$ by R_0 from previous step and P 3.4.2, P 3.4.3, P 3.4.4

$Legal(s) \wedge turn(s) = O \wedge cur(s) = 1 \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge \exists n.\ MinInit \leq n \leq MaxInit \wedge ($

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge turn(s) = X \wedge n = 1) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge ($

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge cur(s) = 2) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge ($

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge cur(s) = 3) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty$

$\wedge(Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge cur(s) = 2)$

$\wedge(Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge cur(s) = 3)$

$\equiv$ by FOL

$Legal(s) \wedge turn(s) = O \wedge cur(s) = 1 \vee$

$Legal(s) \wedge turn(s) = X \wedge (cur(s) = 2 \vee cur(s) = 3)$

This approximation evaluates to true if s is legal and such that X can win in at most 1 step. These are situations where the current position is 1 and now it is player O's turn or the current position is 2 or 3 and it is player X's turn (X can perform $move1$ or $move2$ to win). □

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MD}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_2(s)} \doteq \mathbf{Wins(X, s)} \vee \mathcal{R}(\langle\langle\{\mathbf{X}\}\rangle\rangle \bigcirc \mathbf{R_1}) \equiv$ by P 3.4.1, L 3.4.2 and FOL

$Legal(s) \wedge turn(s) = O \wedge cur(s) = 1 \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge$

$\exists n.\ MinInit \leq n \leq MaxInit \wedge \mathcal{R}(R_1[do(init(X, n), s)]) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_1[do(move1(X), s)]) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_1[do(move2(X), s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_1[do(move1(O), s)]) \wedge \mathcal{R}(R_1[do(move2(O), s)])$

$\equiv$ by R_1 from previous step and P 3.4.2, P 3.4.3, P 3.4.4

$Legal(s) \wedge turn(s) = O \wedge cur(s) = 1 \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge \exists n.\ MinInit \leq n \leq MaxInit \wedge ($

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge turn(s) = X \wedge n = 1 \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge turn(s) = O \wedge n = 2 \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge turn(s) = O \wedge n = 3) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge ($

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge cur(s) = 2 \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(s) = 3 \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(s) = 4) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge ($

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge cur(s) = 3 \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(s) = 4 \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(s) = 5) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge ($

$\quad Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge cur(s) = 2 \vee$

$\quad Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(s) = 3 \vee$

$\quad Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(s) = 4)$

$\wedge($

$\quad Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge cur(s) = 3 \vee$

$\quad Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(s) = 4 \vee$

$\quad Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(s) = 5)$

$\equiv$ by FOL

$Legal(s) \wedge turn(s) = O \wedge (cur(s) = 1 \vee cur(s) = 4) \vee$

$Legal(s) \wedge turn(s) = X \wedge (cur(s) = 2 \vee cur(s) = 3)$

This approximation evaluates to true if s is legal and such that X can win in at most 2 steps. These are situations where the current position is 1 and now it is player O's turn or the current position is 2 or 3 and it is player X's turn (X can perform $move1$ or $move2$ to win) or the current position is 4 and now it is player O's turn (no matter what action player O takes player X can finish in the next turn). $\square$

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MD}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_3(s)} \doteq \mathbf{Wins(X, s)} \vee \mathcal{R}(\langle\langle\{\mathbf{X}\}\rangle\rangle \bigcirc \mathbf{R_2}) \equiv$ by P 3.4.1, L 3.4.2 and FOL

$Legal(s) \wedge turn(s) = O \wedge cur(s) = 1 \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge \exists n.\ MinInit \leq n \leq MaxInit \wedge \mathcal{R}(R_2[do(init(X, n), s)]) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_2[do(move1(X), s)]) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_2[do(move2(X), s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_2[do(move1(O), s)]) \wedge \mathcal{R}(R_2[do(move2(O), s)])$

$\equiv$ by R_2 from previous step and P 3.4.2, P 3.4.3, P 3.4.4

$Legal(s) \wedge turn(s) = O \wedge cur(s) = 1 \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge \exists n.\ MinInit \leq n \leq MaxInit \wedge ($

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge turn(s) = X \wedge n = 1 \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge turn(s) = X \wedge n = 4 \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge turn(s) = O \wedge n = 2 \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge turn(s) = O \wedge n = 3) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge ($

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge cur(s) = 2 \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge cur(s) = 5 \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(s) = 3 \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(s) = 4) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge ($

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge cur(s) = 3 \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge cur(s) = 6 \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(s) = 4 \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(s) = 5) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge ($

$\quad Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge cur(s) = 2 \vee$

$\quad Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge cur(s) = 5 \vee$

$\quad Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(s) = 3 \vee$

$\quad Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(s) = 4)$

$\wedge($

$\quad Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge cur(s) = 3 \vee$

$\quad Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge cur(s) = 6 \vee$

$\quad Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(s) = 4 \vee$

$\quad Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(s) = 5)$

$\equiv$ by FOL

$Legal(s) \wedge turn(s) = O \wedge (cur(s) = 1 \vee cur(s) = 4) \vee$

$Legal(s) \wedge turn(s) = X \wedge (cur(s) = 2 \vee cur(s) = 3 \vee cur(s) = 5 \vee cur(s) = 6) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge MinInit \leq 4 \leq MaxInit$

This approximation evaluates to true if s is legal and such that X can win in at most 3 steps. These are situations where the current position is 1 and now it is player O's turn or the current position is 2 or 3 and it is player X's turn (X can perform *move*1 or *move*2 to win), or the current position is 4 and now it is player O's turn (no matter what action player O takes player X can finish in the next turn), or the $MinInit$ and $MaxInit$ constants are such that X can mark cell 4 in the initial move and then it is player O's turn (no matter what action player O takes player X can finish in the next turn). □

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MD}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_4(s)} \doteq \mathbf{Wins(X, s)} \vee \mathcal{R}(\langle\langle\{\mathbf{X}\}\rangle\rangle \bigcirc \mathbf{R_3}) \equiv$ by P 3.4.1, L 3.4.2 and FOL

$Legal(s) \wedge turn(s) = O \wedge cur(s) = 1 \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge \exists n.\ MinInit \leq n \leq MaxInit \wedge \mathcal{R}(R_3[do(init(X, n), s)]) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_3[do(move1(X), s)]) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_3[do(move2(X), s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge \mathcal{R}(R_3[do(move1(O), s)]) \wedge \mathcal{R}(R_3[do(move2(O), s)])$

$\equiv$ by R_3 from previous step and P 3.4.2, P 3.4.3, P 3.4.4

$Legal(s) \wedge turn(s) = O \wedge cur(s) = 1 \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge \exists n.\ MinInit \leq n \leq MaxInit \wedge ($

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge turn(s) = X \wedge (n = 1 \vee n = 4) \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge turn(s) = O \wedge (n = 2 \vee n = 3 \vee n = 5 \vee n = 6) \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge turn(s) = O \wedge n = +\infty \wedge MinInit \leq 4 \leq MaxInit) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge ($

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge (cur(s) = 2 \vee cur(s) = 5) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge (cur(s) = 3 \vee cur(s) = 4 \vee cur(s) = 6 \vee cur(s) = 7) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(s) - 1 = +\infty \wedge MinInit \leq 4 \leq MaxInit) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge ($

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge (cur(s) = 3 \vee cur(s) = 6) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge (cur(s) = 4 \vee cur(s) = 5 \vee cur(s) = 7 \vee cur(s) = 8) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(do(move2(X), s)) = +\infty \wedge MinInit \leq 4 \leq MaxInit) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge ($

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge (cur(s) = 2 \vee cur(s) = 5) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge (cur(s) = 3 \vee cur(s) = 4 \vee cur(s) = 6 \vee cur(s) = 7) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(s) - 1 = +\infty \wedge MinInit \leq 4 \leq MaxInit)$

$\wedge($

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = X \wedge (cur(s) = 3 \vee cur(s) = 6) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge (cur(s) = 4 \vee cur(s) = 5 \vee cur(s) = 7 \vee cur(s) = 8) \vee$

$Legal(s) \wedge turn(s) = O \wedge cur(s) \neq +\infty \wedge turn(s) = O \wedge cur(do(move2(O), s)) = +\infty \wedge MinInit \leq 4 \leq MaxInit)$

$\equiv$ by FOL

$Legal(s) \wedge turn(s) = O \wedge (cur(s) = 1 \vee cur(s) = 4 \vee cur(s) = 7) \vee$

$Legal(s) \wedge turn(s) = X \wedge (cur(s) = 2 \vee cur(s) = 3 \vee cur(s) = 5 \vee cur(s) = 6) \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge MinInit \leq 4 \leq MaxInit$

This approximation evaluates to true if s is legal and such that X can win in at most 4 steps. The intuitive interpretation becomes more complicated but it follows from how the 2 players can interact. □

The subsequent results can be generalized (by induction) to the following formulas:

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MD}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_{2i-1}(s)} \equiv$

$Legal(s) \wedge turn(s) = O \wedge \exists j.\ 0 \leq j < i \wedge cur(s) = 1 + 3j \vee$

$Legal(s) \wedge turn(s) = X \wedge 2 \leq cur(s) \leq 3i \wedge \forall j.\ 0 < j \leq i \wedge cur(s) \neq 1 + 3j \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge \exists j. 0 < j < i \wedge MinInit \leq 1 + 3j \leq MaxInit$

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MD}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_{2i}(s)} \equiv$

$Legal(s) \wedge turn(s) = O \wedge \exists j.\ 0 \leq j \leq i \wedge cur(s) = 1 + 3j \vee$

$Legal(s) \wedge turn(s) = X \wedge 2 \leq cur(s) \leq 3i \wedge \forall j.\ 0 < j \leq i \wedge cur(s) \neq 1 + 3j \vee$

$Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge \exists j. 0 < j < i \wedge MinInit \leq 1 + 3j \leq MaxInit$

This approximations evaluate to true if s is legal and such that X can win in at most $2i - 1$ (odd) or $2i$ (even) number of steps. These are situations where the current position is multiple of 3 plus 1 and now it is player O's turn, or the $MinInit$ and $MaxInit$ constants are such that X can mark cell that is multiple of 3 plus 1 in the initial move.

Is there a convergence in a finite number of steps? No.

The De Giacomo et al. [DLP10] iterative method for logical formula manipulation does not work using only the unique name and domain closure axioms for actions $\mathcal{D}_{ca}^{MD}$ and the axiomatization of the integers $\mathcal{D}_Z$. It will not converge in a finite number of steps as it can be observed that for all natural numbers i, $D_{ca}^{MD} \cup D_Z \not\models R_{i-1} \equiv R_i$, since one can always construct a model of $D_{ca}^{MD} \cup D_Z$ that will satisfy R_i and not R_{i-1}.

Will there be a convergence if we use additional facts entailed by the whole theory ? Yes.

The procedure will converge if we add lemma L3.4.3 from the previous section to the axioms used by the method. Since $MaxInit$ is a constant defined in the game, let us consider an integer number $x = \lceil (MaxInit - 1)/3 \rceil$. First, we can observe that $MinInit \leq MaxInit \leq 3x + 1$ and thus based on L3.4.3 we also have that either $cur(s) = +\infty$ or $cur(s) \leq 3x + 1$ holds. That is, it is false that we could have a legal situation that $cur(s) = 3x + 1$.

Based on this observation, the two consecutive steps R_{2x-1} and R_{2x} will simplify to:

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MD}} \cup \mathcal{D}_{\mathbf{Z}} \cup \{\mathbf{L3.4.3}\} \models \mathbf{R_{2x-1}(s)} \equiv$

$\quad Legal(s) \wedge turn(s) = O \wedge \exists j.\ 0 \leq j < x \wedge cur(s) = 1 + 3j \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge 2 \leq cur(s) \leq 3x \wedge \forall j.\ 0 < j \leq x \wedge cur(s) \neq 1 + 3j \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge \exists j. 0 < j < x \wedge MinInit \leq 1 + 3j \leq MaxInit$

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MD}} \cup \mathcal{D}_{\mathbf{Z}} \cup \{\mathbf{L3.4.3}\} \models \mathbf{R_{2x}(s)} \equiv$

$\quad Legal(s) \wedge turn(s) = O \wedge \exists j.\ 0 \leq j < x \wedge cur(s) = 1 + 3j \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge 2 \leq cur(s) \leq 3x \wedge \forall j.\ 0 < j \leq x \wedge cur(s) \neq 1 + 3j \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge \exists j. 0 < j < x \wedge MinInit \leq 1 + 3j \leq MaxInit$

Thus the fixpoint expansion procedure converges no later than in the step $2x$ as we have:

$$\mathcal{D}_{ca}^{MD} \cup \mathcal{D}_{Z}\ \cup\ \{L3.4.3\} \models R_{2x-1}(s) \equiv R_{2x}(s)$$

And therefore by Theorem 1 of [DLP10]:

$\mathcal{D}_{GS}^{MD}\ \models \langle\langle\{X\}\rangle\rangle\Diamond Wins(X)[s]\ \equiv$

$\quad Legal(s) \wedge turn(s) = O \wedge \exists j.\ 0 \leq j < x \wedge cur(s) = 1 + 3j \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge 2 \leq cur(s) \leq 3x \wedge \forall j.\ 0 < j \leq x \wedge cur(s) \neq 1 + 3j \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge cur(s) = +\infty \wedge \exists j.\ 0 < j < x \wedge MinInit \leq 1 + 3j \leq MaxInit$

$\quad$ where $x = \lceil (MaxInit - 1)/3 \rceil$

It follows by the initial state axioms that $\mathcal{D}_{GS}^{MD} \models \langle\langle\{X\}\rangle\rangle\Diamond Wins(X)[S_0]$, i.e., the goal can be ensured from the initial situation as it is a legal situation and it is player X's turn and $MinInit \leq 4 \leq MaxInit$. On the other hand, for an initial situation S_1 where $MinInit = 5$ and $MaxInit = 6$ we cannot ensure that player X can win since $\mathcal{D}_{GS}^{MD} \not\models \langle\langle\{X\}\rangle\rangle\Diamond Wins(X)[S_1]$.

3.5 Mark Up (MU)

In this section we specify another simple game that is similar to the MD game from the previous section but where the current position can increase infinitely. The idea was to make MD more like a real game where the players are actively kept involved trying to win the game. We then apply the De Giacomo et al. [DLP10] fixpoint iteration method to verify some temporal properties. If we apply the method as originally specified and only use the unique name and domain closure axioms for actions (to which we have added the axiomatization of the integers), we don't get convergence in a finite number of steps. But if use add some additional facts entailed by the whole theory, then we do get convergence in a finite number of steps.

The Mark Up (MU) game that we have designed involves an infinite vector of cells one for each natural number. There is a pointer maintained in the game and it is called the current position. Initially the current position is set to 0. There are 2 players X and O in the game. Players take turns and initially it is player X's turn. The actions are *move*1 to increase the current position by 1 and *move*2 to increase the current position by 2. The goal of player X is to reach a state where the current position is greater than 100 and divided modulo 10 it is 0 (after player X's turn) in which case player X wins, and similarly for player O. A sample sequence of moves for a game could be as follows: X performs *move*1, then O and X perform *move*2 53 times, finally O performs *move*1 and X finishes the game by performing *move*2. It is possible for the game go on forever and the goal cannot be reached. An example of such scenario for a game would be as follows: X performs *move*1, then O and X perform only *move*2 actions.

3.5.1 Situation Calculus Game Structure Axiomatization of MU Domain

The Situation Calculus Game Structure Axiomatization is defined as:

$$\mathcal{D}_{GS}^{MU} = \Sigma \cup \mathcal{D}_{poss}^{MU} \cup \mathcal{D}_{ssa}^{MU} \cup \mathcal{D}_{Legal}^{MU} \cup \mathcal{D}_{ca}^{MU} \cup \mathcal{D}_{S_0}^{MU} \cup \mathcal{D}_{Z}$$

where $\mathcal{D}_Z$ is a suitable axiomatization of integers.

Fluents

- $cur(s)$ - functional fluent indicating the latest marking position, whose domain is the integers
- $turn(s)$ - functional fluent indicating which agent is to take the next action, with its domain being the agent set $\{X, O\}$

Derived Fluents

- $Wins(p,s) \doteq Legal(s) \wedge Agent(p) \wedge 100 < cur(s) \wedge cur(s)\ mod\ 10 = 0 \wedge \exists p'.\ Agent(p') \wedge turn(s) = p' \wedge p \neq p'$

Actions

- $move1(p)$ - agent p increases the current position by 1
- $move2(p)$ - agent p increases the current position by 2

Precondition Axioms $\mathcal{D}_{poss}^{MU}$

These are precondition axioms, one per action, specifying when an action can physically be performed:

- $Poss(move1(p), s) \equiv True$
- $Poss(move2(p), s) \equiv True$

Successor State Axioms $\mathcal{D}_{ssa}^{MU}$

These are successor state axioms specifying how the fluents change as a result of actions:

- $cur(do(a,s)) = k \equiv cur(s) = cur(s) = k - 1 \wedge \exists p.a = move1(p) \vee cur(s) = k - 2 \wedge \exists p.a = move2(p)$

- $turn(do(a,s)) = p \equiv p = O \wedge turn(s) = X \vee p = X \wedge turn(s) = O$

Initial State Axioms $\mathcal{D}_{S_0}^{MU}$

These describe the initial situation:

- $cur(S_0) = 0$
- $turn(S_0) = X$
- $Legal(S_0)$

Legal Moves Axioms $\mathcal{D}_{legal}^{MU}$

These encode the rules of the game:

- $agent(move1(p)) = p$
- $agent(move2(p)) = p$
- $Control(p,s) \doteq \exists a. Legal(do(a,s)) \wedge agent(a) = p$
- $Legal(do(a,s)) \equiv Legal(s) \wedge \{\exists p.\ a = move1(p) \wedge turn(s) = p \vee \exists p.\ a = move2(p) \wedge turn(s) = p\}$

Unique Name and Domain Closure Axioms for Actions $\mathcal{D}_{ca}^{MU}$

These describe the uniqueness of action names and the fact that the domain of actions s closed:

- $\forall a.\ \{\ \exists p.\ a = move1(p) \vee \exists p.\ a = move2(p)\ \}$
- $\forall p, p'.\ \{\ move1(p) \neq move2(p')\ \}$
- $\forall p, p'.\ \{\ move1(p) = move1(p') \supset p = p'\ \}$

- $\forall p, p'. \; \{ \; move2(p) = move2(p') \supset p = p' \; \}$
- $\forall p. \; \{ \; Agent(p) \equiv (p = X \lor p = O)\}$
- $X \neq O$

The complete theory also includes the foundational axioms Σ of the Situation Calculus. It can be noticed that this is clearly an infinite state domain as the set of positions that can be visited is infinite.

Propositions and Lemmas

We derived the following propositions, some of which are used in later proofs:

P 3.5.1. *Regression of the concrete case of the Wins fluent.*

$$\mathcal{D}_{ca}^{MU} \cup \mathcal{D}_Z \models \mathcal{R}(Wins(X, s)) \equiv Legal(s) \land turn(s) = O \land 100 < cur(s) \land cur(s) \; mod \; 10 = 0$$

P 3.5.2. *Regression of concrete cases of the turn fluent.*

$$\mathcal{D}_{ca}^{MU} \models \mathcal{R}(turn(do(a, s)) = X) \equiv turn(s) = O$$

$$\mathcal{D}_{ca}^{MU} \models \mathcal{R}(turn(do(a, s)) = O) \equiv turn(s) = X$$

P 3.5.3. *Regression of concrete cases of the cur fluent.*

$$\mathcal{D}_{ca}^{MU} \cup \mathcal{D}_Z \models \mathcal{R}(cur(do(move1(p), s) = k) \equiv cur(s) = k - 1$$

$$\mathcal{D}_{ca}^{MU} \cup \mathcal{D}_Z \models \mathcal{R}(cur(do(move2(p), s) = k) \equiv cur(s) = k - 2$$

P 3.5.4. *Regression of concrete cases for the Legal fluent.*

$$\mathcal{D}_{ca}^{MU} \cup \mathcal{D}_Z \models \mathcal{R}(Legal(do(move1(p), s))) \equiv Legal(s) \land turn(s) = p$$

$$\mathcal{D}_{ca}^{MU} \cup \mathcal{D}_Z \models \mathcal{R}(Legal(do(move2(p), s))) \equiv Legal(s) \land turn(s) = p$$

P 3.5.5. *Concrete cases for the Control fluent.*

$$\mathcal{D}_{ca}^{MU} \cup \mathcal{D}_Z \models Control(X, s) \equiv Legal(s) \land turn(s) = X$$

$$\mathcal{D}_{ca}^{MU} \cup \mathcal{D}_Z \models Control(O, s) \equiv Legal(s) \land turn(s) = O$$

In this domain, for groups $G = \{X\}$ and $G = \{X, O\}$ ensuring that φ holds next can be established by considering only the *move*1 and *move*2 actions as the following lemmas show:

L 3.5.1. *Regression of the concrete case of the* $\exists \bigcirc \varphi$ *definition based on domain closure.*

$$\begin{aligned}
\mathcal{D}_{ca}^{MU} \models \mathcal{R}(\exists \bigcirc \varphi[s]) \equiv \\
& Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\varphi[do(move1(X), s)]) \vee \\
& Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\varphi[do(move2(X), s)]) \vee \\
& Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\varphi[do(move1(O), s)]) \vee \\
& Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\varphi[do(move2(O), s)])
\end{aligned}$$

L 3.5.2. *Regression of the concrete case of the* $\langle\langle\{X\}\rangle\rangle \bigcirc \varphi$ *definition based on domain closure.*

$$\begin{aligned}
\mathcal{D}_{ca}^{MU} \models \mathcal{R}(\langle\langle\{X\}\rangle\rangle \bigcirc \varphi[s]) \equiv \\
& Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\varphi[do(move1(X), s)]) \vee \\
& Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(\varphi[do(move2(X), s)]) \vee \\
& Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(\varphi[do(move1(O), s)]) \wedge \mathcal{R}(\varphi[do(move2(O), s)])
\end{aligned}$$

3.5.2 Possibility of Winning

The property that it is possible for X to eventually win can be represented by the following formula:

$$\exists\Diamond Wins(X) \doteq \mu Z.\{\ Wins(X)\ \vee\ \exists \bigcirc Z\ \}$$

We begin by applying the De Giacomo et al. [DLP10] method and try to show that successive approximates are equivalent using only the unique name and domain closure axioms for actions $\mathcal{D}_{ca}^{MU}$ and the axiomatization of the integers $\mathcal{D}_Z$:

$$\mathcal{D}_{ca}^{MU} \cup \mathcal{D}_Z \models \exists\Diamond Wins(X)$$

This formula can be verified by employing the [DLP10] method using regression and fixpoint approximation, until we converge. The technique does not always converge and this needs to be checked as we proceed. The approximations are as follows:

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MU}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_0(s)} \doteq \mathbf{Wins(X, s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{False}) \equiv$ by P 3.5.1, L 3.5.1 and FOL

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) \wedge cur(s) \mod 10 = 0$

This approximation evaluates to true if s is legal and such that X is winning in s already (in no steps). These are situations where the current position is 0 away from a modulo 10 position above 100 and now it is player O's turn. □

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MU}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_1(s)} \doteq \mathbf{Wins(X, s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{R_0}) \equiv$ by P 3.5.1, L 3.5.1 and FOL

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) \wedge cur(s) \mod 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(R_0[do(move1(X), s)]) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(R_0[do(move2(X), s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(R_0[do(move1(O), s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(R_0[do(move2(O), s)])$

$\equiv$ by R_0 from previous step and P 3.5.2, P 3.5.3, P 3.5.4

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) \wedge cur(s) \mod 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge (Legal(s) \wedge turn(s) = X \wedge 100 < cur(s)+1 \wedge (cur(s)+1)\ mod\ 10 = 0 \wedge turn(s) = X) \vee$

$Legal(s) \wedge turn(s) = X \wedge (Legal(s) \wedge turn(s) = X \wedge 100 < cur(s)+2 \wedge (cur(s)+2)\ mod\ 10 = 0 \wedge turn(s) = X) \vee$

$Legal(s) \wedge turn(s) = O \wedge (Legal(s) \wedge turn(s) = O \wedge 100 < cur(s)+1 \wedge (cur(s)+1)\ mod\ 10 = 0 \wedge turn(s) = X) \vee$

$Legal(s) \wedge turn(s) = O \wedge (Legal(s) \wedge turn(s) = O \wedge 100 < cur(s)+2 \wedge (cur(s)+2)\ mod\ 10 = 0 \wedge turn(s) = X)$

$\equiv$ by FOL

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) \wedge cur(s)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge 100 < cur(s) + 1 \wedge (cur(s) + 1)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge 100 < cur(s) + 2 \wedge (cur(s) + 2)\ mod\ 10 = 0$

This approximation evaluates to true if s is legal and such that X can win in at most 1 step. These are situations where the current position is 0 away from a modulo 10 position above 100 and now it is player

O's turn or the current position is 1 or 2 away from modulo 10 position above 100 and it is player X's turn (X can perform *move*1 or *move*2 to win). $\square$

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MU}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_2(s)} \doteq \mathbf{Wins(X, s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{R_1}) \equiv$ by P 3.5.1, L 3.5.1 and FOL

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) \wedge cur(s)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(R_1[do(move1(X), s)]) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(R_1[do(move2(X), s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(R_1[do(move1(O), s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(R_1[do(move2(O), s)])$

$\equiv$ by R_1 from previous step and P 3.5.2, P 3.5.3, P 3.5.4

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) \wedge cur(s)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge ($

$\quad Legal(s) \wedge turn(s) = X \wedge turn(s) = X \wedge 100 < cur(s) + 1 \wedge (cur(s) + 1)\ mod\ 10 = 0 \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge turn(s) = O \wedge 100 < cur(s) + 2 \wedge (cur(s) + 2)\ mod\ 10 = 0 \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge turn(s) = O \wedge 100 < cur(s) + 3 \wedge (cur(s) + 3)\ mod\ 10 = 0) \vee$

$Legal(s) \wedge turn(s) = X \wedge ($

$\quad Legal(s) \wedge turn(s) = X \wedge turn(s) = X \wedge 100 < cur(s) + 2 \wedge (cur(s) + 2)\ mod\ 10 = 0 \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge turn(s) = O \wedge 100 < cur(s) + 3 \wedge (cur(s) + 3)\ mod\ 10 = 0 \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge turn(s) = O \wedge 100 < cur(s) + 4 \wedge (cur(s) + 4)\ mod\ 10 = 0) \vee$

$Legal(s) \wedge turn(s) = O \wedge ($

$\quad Legal(s) \wedge turn(s) = O \wedge turn(s) = X \wedge 100 < cur(s) + 1 \wedge (cur(s) + 1)\ mod\ 10 = 0 \vee$

$\quad Legal(s) \wedge turn(s) = O \wedge turn(s) = O \wedge 100 < cur(s) + 2 \wedge (cur(s) + 2)\ mod\ 10 = 0 \vee$

$\quad Legal(s) \wedge turn(s) = O \wedge turn(s) = O \wedge 100 < cur(s) + 3 \wedge (cur(s) + 3)\ mod\ 10 = 0) \vee$

$Legal(s) \wedge turn(s) = O \wedge ($

$\quad Legal(s) \wedge turn(s) = O \wedge turn(s) = X \wedge 100 < cur(s) + 2 \wedge (cur(s) + 2)\ mod\ 10 = 0 \vee$

$\quad Legal(s) \wedge turn(s) = O \wedge turn(s) = O \wedge 100 < cur(s) + 3 \wedge (cur(s) + 3)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge turn(s) = O \wedge 100 < cur(s) + 4 \wedge (cur(s) + 4)\ mod\ 10 = 0)$

$\equiv$ by FOL

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) \wedge cur(s)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) + 2 \wedge (cur(s) + 2)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) + 3 \wedge (cur(s) + 3)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) + 4 \wedge (cur(s) + 4)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge 100 < cur(s) + 1 \wedge (cur(s) + 1)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge 100 < cur(s) + 2 \wedge (cur(s) + 2)\ mod\ 10 = 0$

This approximation evaluates to true if s legal and is such that X can win in at most 2 steps. These are situations where the current position is 0 away from a modulo 10 position above 100 and now it is player O's turn or the current position is between 1 and 2 away from modulo 10 position above 100 and it is player X's turn (X can perform $move1$ or $move2$ to win) or the current position is between 2 and 4 away from modulo 10 position above 100 and it is player O's turn (players O and X can cooperate accordingly). $\square$

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MU}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_3(s)} \doteq \mathbf{Wins(X, s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{R_2}) \equiv$ by P 3.5.1, L 3.5.1 and FOL

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) \wedge cur(s)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(R_2[do(move1(X), s)]) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(R_2[do(move2(X), s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(R_2[do(move1(O), s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(R_2[do(move2(O), s)])$

$\equiv$ by R_2 from previous step and P 3.5.2, P 3.5.3, P 3.5.4

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) \wedge cur(s)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge ($

$Legal(s) \wedge turn(s) = X \wedge turn(s) = X \wedge 100 < cur(s) + 1 \wedge (cur(s) + 1)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge turn(s) = X \wedge 100 < cur(s) + 3 \wedge (cur(s) + 3)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge turn(s) = X \wedge 100 < cur(s) + 4 \wedge (cur(s) + 4)\ mod\ 10 = 0 \vee$

$Legal(s) \land turn(s) = X \land turn(s) = X \land 100 < cur(s) + 5 \land (cur(s) + 5)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = X \land turn(s) = O \land 100 < cur(s) + 2 \land (cur(s) + 2)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = X \land turn(s) = O \land 100 < cur(s) + 3 \land (cur(s) + 3)\ mod\ 10 = 0$

$) \lor$

$Legal(s) \land turn(s) = X \land ($

$Legal(s) \land turn(s) = X \land turn(s) = X \land 100 < cur(s) + 2 \land (cur(s) + 2)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = X \land turn(s) = X \land 100 < cur(s) + 4 \land (cur(s) + 4)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = X \land turn(s) = X \land 100 < cur(s) + 5 \land (cur(s) + 5)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = X \land turn(s) = X \land 100 < cur(s) + 6 \land (cur(s) + 6)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = X \land turn(s) = O \land 100 < cur(s) + 3 \land (cur(s) + 3)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = X \land turn(s) = O \land 100 < cur(s) + 4 \land (cur(s) + 4)\ mod\ 10 = 0$

$) \lor$

$Legal(s) \land turn(s) = O \land ($

$Legal(s) \land turn(s) = O \land turn(s) = X \land 100 < cur(s) + 1 \land (cur(s) + 1)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = O \land turn(s) = X \land 100 < cur(s) + 3 \land (cur(s) + 3)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = O \land turn(s) = X \land 100 < cur(s) + 4 \land (cur(s) + 4)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = O \land turn(s) = X \land 100 < cur(s) + 5 \land (cur(s) + 5)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = O \land turn(s) = O \land 100 < cur(s) + 2 \land (cur(s) + 2)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = O \land turn(s) = O \land 100 < cur(s) + 3 \land (cur(s) + 3)\ mod\ 10 = 0$

$) \lor$

$Legal(s) \land turn(s) = O \land ($

$Legal(s) \land turn(s) = O \land turn(s) = X \land 100 < cur(s) + 2 \land (cur(s) + 2)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = O \land turn(s) = X \land 100 < cur(s) + 4 \land (cur(s) + 4)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = O \land turn(s) = X \land 100 < cur(s) + 5 \land (cur(s) + 5)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = O \land turn(s) = X \land 100 < cur(s) + 6 \land (cur(s) + 6)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = O \land turn(s) = O \land 100 < cur(s) + 3 \land (cur(s) + 3)\ mod\ 10 = 0 \lor$

$Legal(s) \wedge turn(s) = O \wedge turn(s) = O \wedge 100 < cur(s) + 4 \wedge (cur(s) + 4)\ mod\ 10 = 0$

)

$\equiv$ by FOL

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) \wedge cur(s)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) + 2 \wedge (cur(s) + 2)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) + 3 \wedge (cur(s) + 3)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) + 4 \wedge (cur(s) + 4)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge 100 < cur(s) + 1 \wedge (cur(s) + 1)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge 100 < cur(s) + 2 \wedge (cur(s) + 2)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge 100 < cur(s) + 3 \wedge (cur(s) + 3)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge 100 < cur(s) + 4 \wedge (cur(s) + 4)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge 100 < cur(s) + 5 \wedge (cur(s) + 5)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge 100 < cur(s) + 6 \wedge (cur(s) + 6)\ mod\ 10 = 0$

This approximation evaluates to true if s is legal and such that X can win in at most 3 steps. These are situations where the current position is 0 away from a modulo 10 position above 100 and now it is player O's turn, or the current position is between 1 and 6 away from modulo 10 position above 100 and it is player X's turn, or the current position is between 2 and 4 away from modulo 10 position above 100 and it is player O's turn. $\square$

The subsequent results can be generalized (by induction) to the following formulas:

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MU}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_{2i-1}(s)} \equiv$

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) \wedge cur(s)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists j.\ \mathbf{2 \leq j \leq 4i - 4} \wedge 100 - j < cur(s) \wedge (cur(s) + j)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists j.\ 1 \leq j \leq 4i - 2 \wedge 100 - j < cur(s) \wedge (cur(s) + j)\ mod\ 10 = 0$

This approximation evaluates to true if s is legal and such that X can win in at most $2i - 1$ (odd) steps.

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MU}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_{2i}(s)} \equiv$

$$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) \wedge cur(s)\ mod\ 10 = 0 \vee$$

$$Legal(s) \wedge turn(s) = O \wedge \exists j.\ \mathbf{2 \leq j \leq 4i} \wedge 100 - j < cur(s) \wedge (cur(s) + j)\ mod\ 10 = 0 \vee$$

$$Legal(s) \wedge turn(s) = X \wedge \exists j.\ 1 \leq j \leq 4i - 2 \wedge 100 - j < cur(s) \wedge (cur(s) + j)\ mod\ 10 = 0$$

This approximation evaluates to true if s is legal and such that X can win in at most $2i$ (even) steps.

Is there a convergence in a finite number of steps? No.

The De Giacomo et al. [DLP10] iterative method for logical formula manipulation does not work using only the unique name and domain closure axioms for actions $\mathcal{D}_{ca}^{MU}$ and the axiomatization of the integers $\mathcal{D}_Z$. It will not converge in a finite number of steps as it can be observed that for all natural numbers i, $D_{ca}^{MU} \cup D_Z \not\models R_{i-1} \equiv R_i$, since one can always construct a model of $D_{ca}^{MU} \cup D_Z$ that will satisfy R_{2i} and not R_{2i-1}, e.g. where $cur(s) = 4i$ for steps $2i-1$ and $2i$.

Will there be a convergence if we use additional facts entailed by the whole theory ? Yes.

The procedure will converge if we consider the following lemma:

L 3.5.3. *Existence of lower bound for fluent cur.*

$\mathcal{D}_{GS}^{MU} \models \forall s.\ 0 \leq cur(s)$

Proof: We will prove by induction on situation. Indeed our base case is S_0 and we have $0 \leq cur(S_0) = 0$. Now assume that for some situation s_i we have that $0 \leq cur(s_i)$. Based on the domain closure for actions we have that the successor situations to s_i can only be $do(move1(p), s_i)$, or $do(move2(p), s_i)$ where p is X or O. Based on the inductive assumption and regression we have that $0 \leq cur(do(move1(p), s_i)) = cur(s_i) + 1$ and $0 \leq cur(do(move2(p), s_i)) = cur(s_i) + 2$. Therefore $\forall s, a.\ 0 \leq cur(do(a, s))$. □

Let us consider the formulas R_{57} and R_{58} (57 and 58 were chosen so that i will become 29 and $4i$ is a number that is more than 10 above 100 which greatly helps simplifying the generalized formulas):

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MU}} \cup \mathcal{D}_{\mathbf{Z}} \cup \{\mathbf{L3.5.3}\} \models \mathbf{R_{57}(s)} \equiv$

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) \wedge cur(s)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists j.\ \mathbf{2 \leq j \leq 112} \wedge 100 - j < cur(s) \wedge (cur(s) + j)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists j.\ \mathbf{1 \leq j \leq 114} \wedge 100 - j < cur(s) \wedge (cur(s) + j)\ mod\ 10 = 0$

$\equiv$

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) \wedge cur(s)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists j.\ \mathbf{2 \leq j \leq 102} \wedge 100 - j < cur(s) \wedge (cur(s) + j)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge\ -3 < cur(s) \wedge (cur(s) + 103)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge\ -4 < cur(s) \wedge (cur(s) + 104)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge\ -5 < cur(s) \wedge (cur(s) + 105)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge\ -6 < cur(s) \wedge (cur(s) + 106)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge\ -7 < cur(s) \wedge (cur(s) + 107)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge\ -8 < cur(s) \wedge (cur(s) + 108)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge\ -9 < cur(s) \wedge (cur(s) + 109)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge\ -10 < cur(s) \wedge (cur(s) + 110)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge\ -11 < cur(s) \wedge (cur(s) + 111)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge\ -12 < cur(s) \wedge (cur(s) + 112)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists j.\ \mathbf{1 \leq j \leq 100} \wedge 100 - j < cur(s) \wedge (cur(s) + j)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -1 < cur(s) \wedge (cur(s) + 101)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -2 < cur(s) \wedge (cur(s) + 102)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -3 < cur(s) \wedge (cur(s) + 103)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -4 < cur(s) \wedge (cur(s) + 104)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -5 < cur(s) \wedge (cur(s) + 105)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -6 < cur(s) \wedge (cur(s) + 106)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -7 < cur(s) \wedge (cur(s) + 107)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -8 < cur(s) \wedge (cur(s) + 108)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge -9 < cur(s) \wedge (cur(s) + 109)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge -10 < cur(s) \wedge (cur(s) + 110)\ mod\ 10 = 0$

$\equiv$ by L3.5.3

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) \wedge cur(s)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists j.\ \mathbf{2 \leq j \leq 102} \wedge 100 - j < cur(s) \wedge (cur(s) + j)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 103)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 104)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 105)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 106)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 107)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 108)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 109)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 110)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 111)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 112)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists j.\ \mathbf{1 \leq j \leq 100} \wedge 100 - j < cur(s) \wedge (cur(s) + j)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = x \wedge (cur(s) + 101)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = x \wedge (cur(s) + 102)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = x \wedge (cur(s) + 103)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = x \wedge (cur(s) + 104)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = x \wedge (cur(s) + 105)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = x \wedge (cur(s) + 106)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = x \wedge (cur(s) + 107)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = x \wedge (cur(s) + 108)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = x \wedge (cur(s) + 109)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = x \wedge (cur(s) + 110)\ mod\ 10 = 0$

$\equiv$ by the fact that over consecutive 10 numbers one is 0 modulo 10

$Legal(s) \land turn(s) = O \land 100 < cur(s) \land cur(s)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = O \land \exists j.\ 2 \leq j \leq 102 \land 100 - j < cur(s) \land (cur(s) + j)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = O \land True \lor$

$Legal(s) \land turn(s) = X \land \exists j.\ 1 \leq j \leq 100 \land 100 - j < cur(s) \land (cur(s) + j)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = x \land True$

$\equiv$ by FOL (line 3 subsumes line 1 and 2, line 5 subsumes line 4)

$\mathbf{Legal}(\mathbf{s}) \land \mathbf{turn}(\mathbf{s}) = \mathbf{X} \lor \mathbf{Legal}(\mathbf{s}) \land \mathbf{turn}(\mathbf{s}) = \mathbf{O}$ $\square$

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MU}} \cup \mathcal{D}_{\mathbf{Z}} \cup \{\mathbf{L3.5.3}\} \models \mathbf{R_{58}}(\mathbf{s}) \equiv$

$Legal(s) \land turn(s) = O \land 100 < cur(s) \land cur(s)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = O \land \exists j.\ \mathbf{2 \leq j \leq 116} \land 100 - j < cur(s) \land (cur(s) + j)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = X \land \exists j.\ \mathbf{1 \leq j \leq 114} \land 100 - j < cur(s) \land (cur(s) + j)\ mod\ 10 = 0$

$\equiv$

$Legal(s) \land turn(s) = O \land 100 < cur(s) \land cur(s)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = O \land \exists j.\ \mathbf{2 \leq j \leq 106} \land 100 - j < cur(s) \land (cur(s) + j)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = O \land -7 < cur(s) \land (cur(s) + 107)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = O \land -8 < cur(s) \land (cur(s) + 108)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = O \land -9 < cur(s) \land (cur(s) + 109)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = O \land -10 < cur(s) \land (cur(s) + 110)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = O \land -11 < cur(s) \land (cur(s) + 111)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = O \land -12 < cur(s) \land (cur(s) + 112)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = O \land -13 < cur(s) \land (cur(s) + 113)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = O \land -14 < cur(s) \land (cur(s) + 114)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = O \land -15 < cur(s) \land (cur(s) + 115)\ mod\ 10 = 0 \lor$

$Legal(s) \land turn(s) = O \land -16 < cur(s) \land (cur(s) + 116)\ mod\ 10 = 0 \lor$

$Legal(s) \wedge turn(s) = X \wedge \exists j.\ \mathbf{1 \leq j \leq 100} \wedge 100 - j < cur(s) \wedge (cur(s) + j)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -1 < cur(s) \wedge (cur(s) + 101)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -2 < cur(s) \wedge (cur(s) + 102)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -3 < cur(s) \wedge (cur(s) + 103)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -4 < cur(s) \wedge (cur(s) + 104)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -5 < cur(s) \wedge (cur(s) + 105)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -6 < cur(s) \wedge (cur(s) + 106)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -7 < cur(s) \wedge (cur(s) + 107)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -8 < cur(s) \wedge (cur(s) + 108)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -9 < cur(s) \wedge (cur(s) + 109)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -10 < cur(s) \wedge (cur(s) + 110)\ mod\ 10 = 0$

$\equiv$ by L 3.5.3

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) \wedge cur(s)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists j.\ \mathbf{2 \leq j \leq 106} \wedge 100 - j < cur(s) \wedge (cur(s) + j)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 107v\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 108)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 109)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 110)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 111)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 112)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 113)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 114)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 115)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 116)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists j.\ \mathbf{1 \leq j \leq 100} \wedge 100 - j < cur(s) \wedge (cur(s) + j)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = x \wedge (cur(s) + 101)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = x \wedge (cur(s) + 102)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = x \wedge (cur(s) + 103)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = x \wedge (cur(s) + 104)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = x \wedge (cur(s) + 105)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = x \wedge (cur(s) + 106)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = x \wedge (cur(s) + 107)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = x \wedge (cur(s) + 108)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = x \wedge (cur(s) + 109)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = x \wedge (cur(s) + 110)\ mod\ 10 = 0$

$\equiv$ by the fact that over consecutive 10 numbers one is 0 modulo 10

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) \wedge cur(s)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge \exists j.\ 2 \leq j \leq 106 \wedge 100 - j < cur(s) \wedge (cur(s) + j)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge True \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists j.\ 1 \leq j \leq 100 \wedge 100 - j < cur(s) \wedge (cur(s) + j)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = x \wedge True$

$\equiv$ by FOL (line 3 subsumes line 1 and 2, line 5 subsumes line 4)

$\mathbf{Legal(s) \wedge turn(s) = X \vee Legal(s) \wedge turn(s) = O}$ $\square$

Thus the fixpoint expansion procedure converges no later than in the step 58 as we have:

$$\mathcal{D}_{ca}^{MU} \cup \mathcal{D}_Z \ \cup \ \{L3.5.3\} \models R_{57}(s) \equiv R_{58}(s)$$

And therefore by Theorem 1 of [DLP10]:

$$\mathcal{D}_{GS}^{MU} \ \models \exists\Diamond Wins(X)[s] \ \equiv Legal(s) \wedge turn(s) = X \vee Legal(s) \wedge turn(s) = O$$

It follows by the initial state axioms that $\mathcal{D}_{GS}^{MU} \models \exists\Diamond Wins(X)[S_0]$, i.e. the goal may eventually be reached from the initial situation as it is legal and it is player X's turn.

3.5.3 Existence of a Winning Strategy

The existence of a strategy to ensure $Wins(X)$ by group $G = \{X\}$ can be represented by the following formula:

$$\langle\langle\{X\}\rangle\rangle\Diamond Wins(X) \doteq \mu Z.\, Wins(X) \vee \langle\langle\{X\}\rangle\rangle \bigcirc Z$$

We begin by applying the De Giacomo et al. [DLP10] method:

$$\mathcal{D}_{ca}^{MU} \cup \mathcal{D}_Z \models \langle\langle\{X\}\rangle\rangle\Diamond Wins(X)$$

The regressed approximations are as follows:

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MU}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_0(s)} \doteq \mathbf{Wins(X,s)} \vee \mathcal{R}(\langle\langle\{\mathbf{X}\}\rangle\rangle \bigcirc \mathbf{False}) \equiv$ by P 3.5.1, L 3.5.2 and FOL

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) \wedge cur(s)\ mod\ 10 = 0$

This approximation evaluates to true if s legal and is such that X is winning in s already (in no steps). These are situations where the current position is 0 away from a modulo 10 position above 100 and now it is player O's turn. □

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MU}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_1(s)} \doteq \mathbf{Wins(X,s)} \vee \mathcal{R}(\langle\langle\{\mathbf{X}\}\rangle\rangle \bigcirc \mathbf{R_0}) \equiv$ by P 3.5.1, L 3.5.2 and FOL

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) \wedge cur(s)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(R_0[do(move1(X), s)]) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(R_0[do(move2(X), s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(R_0[do(move1(O), s)]) \wedge \mathcal{R}(R_0[do(move2(O), s)])$

$\equiv$ by R_0 from previous step and P 3.5.2, P 3.5.3, P 3.5.4

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) \wedge cur(s)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge Legal(s) \wedge turn(s) = X \wedge turn(s) = X \wedge 100 < cur(s)+1 \wedge (cur(s)+1)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge Legal(s) \wedge turn(s) = X \wedge turn(s) = X \wedge 100 < cur(s)+2 \wedge (cur(s)+2)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge$

$\quad (Legal(s) \wedge turn(s) = O \wedge turn(s) = X \wedge 100 < cur(s) + 1 \wedge (cur(s) + 1)\ mod\ 10 = 0) \wedge$

$\quad (Legal(s) \wedge turn(s) = O \wedge turn(s) = X \wedge 100 < cur(s) + 2 \wedge (cur(s) + 2)\ mod\ 10 = 0)$

$\equiv$ by FOL

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) \wedge cur(s)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge 100 < cur(s) + 1 \wedge (cur(s) + 1)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge 100 < cur(s) + 2 \wedge (cur(s) + 2)\ mod\ 10 = 0$

This approximation evaluates to true if s is legal and such that X can win in at most 1 step. These are situations where the current position is 0 away from a modulo 10 position above 100 and now it is player O's turn or the current position is 1 or 2 away from modulo 10 position above 100 and it is player X's turn (X can perform *move*1 or *move*2 to win). □

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MU}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_2(s)} \doteq \mathbf{Wins(X, s)} \vee \mathcal{R}(\langle\langle\{\mathbf{X}\}\rangle\rangle \bigcirc \mathbf{R_1}) \equiv$ by P 3.5.1, L 3.5.2 and FOL

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) \wedge cur(s)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(R_1[do(move1(X), s)]) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(R_1[do(move2(X), s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(R_1[do(move1(O), s)]) \wedge \mathcal{R}(R_1[do(move2(O), s)])$

$\equiv$ by R_1 from previous step and P 3.5.2, P 3.5.3, P 3.5.4

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) \wedge cur(s)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge ($

$\quad Legal(s) \wedge turn(s) = X \wedge turn(s) = X \wedge 100 < cur(s) + 1 \wedge (cur(s) + 1)\ mod\ 10 = 0 \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge turn(s) = O \wedge 100 < cur(s) + 2 \wedge (cur(s) + 2)\ mod\ 10 = 0 \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge turn(s) = O \wedge 100 < cur(s) + 3 \wedge (cur(s) + 3)\ mod\ 10 = 0) \vee$

$Legal(s) \wedge turn(s) = X \wedge ($

$\quad Legal(s) \wedge turn(s) = X \wedge turn(s) = X \wedge 100 < cur(s) + 2 \wedge (cur(s) + 2)\ mod\ 10 = 0 \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge turn(s) = O \wedge 100 < cur(s) + 3 \wedge (cur(s) + 3)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge turn(s) = O \wedge 100 < cur(s) + 4 \wedge (cur(s) + 4)\ mod\ 10 = 0) \vee$

$Legal(s) \wedge turn(s) = O$

$\wedge($

$Legal(s) \wedge turn(s) = O \wedge turn(s) = X \wedge 100 < cur(s) + 1 \wedge (cur(s) + 1)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge turn(s) = O \wedge 100 < cur(s) + 2 \wedge (cur(s) + 2)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge turn(s) = O \wedge 100 < cur(s) + 3 \wedge (cur(s) + 3)\ mod\ 10 = 0)$

$\wedge($

$Legal(s) \wedge turn(s) = O \wedge turn(s) = X \wedge 100 < cur(s) + 2 \wedge (cur(s) + 2)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge turn(s) = O \wedge 100 < cur(s) + 3 \wedge (cur(s) + 3)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge turn(s) = O \wedge 100 < cur(s) + 4 \wedge (cur(s) + 4)\ mod\ 10 = 0)$

$\equiv$ by FOL

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) \wedge cur(s)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) + 3 \wedge (cur(s) + 3)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge 100 < cur(s) + 1 \wedge (cur(s) + 1)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge 100 < cur(s) + 2 \wedge (cur(s) + 2)\ mod\ 10 = 0$

This approximation evaluates to true if s is legal and such that X can win in at most 2 steps. These are situations where the current position is 0 away from a modulo 10 position above 100 and now it is player O's turn, or the current position is between 1 and 2 away from modulo 10 position above 100 and it is player X's turn (X can perform *move*1 or *move*2 to win), or the current position is 3 away from modulo 10 position above 100 and it is player O's turn (after players O's move player X can finish the game). □

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MU}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_3(s)} \doteq \mathbf{Wins(X, s)} \vee \mathcal{R}(\langle\langle\{\mathbf{X}\}\rangle\rangle \bigcirc \mathbf{R_2}) \equiv$ by P 3.5.1, L 3.5.2 and FOL

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) \wedge cur(s)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(R_2[do(move1(X), s)]) \vee$

$Legal(s) \wedge turn(s) = X \wedge \mathcal{R}(R_2[do(move2(X), s)]) \vee$

$Legal(s) \wedge turn(s) = O \wedge \mathcal{R}(R_2[do(move1(O), s)]) \wedge \mathcal{R}(R_2[do(move2(O), s)])$

$\equiv$ by R_2 from previous step and P 3.5.2, P 3.5.3, P 3.5.4

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) \wedge cur(s)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge ($

$\quad Legal(s) \wedge turn(s) = X \wedge turn(s) = X \wedge 100 < cur(s) + 1 \wedge (cur(s) + 1)\ mod\ 10 = 0 \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge turn(s) = X \wedge 100 < cur(s) + 4 \wedge (cur(s) + 4)\ mod\ 10 = 0 \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge turn(s) = O \wedge 100 < cur(s) + 2 \wedge (cur(s) + 2)\ mod\ 10 = 0 \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge turn(s) = O \wedge 100 < cur(s) + 3 \wedge (cur(s) + 3)\ mod\ 10 = 0) \vee$

$Legal(s) \wedge turn(s) = X \wedge ($

$\quad Legal(s) \wedge turn(s) = X \wedge turn(s) = X \wedge 100 < cur(s) + 2 \wedge (cur(s) + 2)\ mod\ 10 = 0 \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge turn(s) = X \wedge 100 < cur(s) + 5 \wedge (cur(s) + 5)\ mod\ 10 = 0 \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge turn(s) = O \wedge 100 < cur(s) + 3 \wedge (cur(s) + 3)\ mod\ 10 = 0 \vee$

$\quad Legal(s) \wedge turn(s) = X \wedge turn(s) = O \wedge 100 < cur(s) + 4 \wedge (cur(s) + 4)\ mod\ 10 = 0) \vee$

$Legal(s) \wedge turn(s) = O \wedge ($

$\quad Legal(s) \wedge turn(s) = O \wedge turn(s) = X \wedge 100 < cur(s) + 1 \wedge (cur(s) + 1)\ mod\ 10 = 0 \vee$

$\quad Legal(s) \wedge turn(s) = O \wedge turn(s) = X \wedge 100 < cur(s) + 4 \wedge (cur(s) + 4)\ mod\ 10 = 0 \vee$

$\quad Legal(s) \wedge turn(s) = O \wedge turn(s) = O \wedge 100 < cur(s) + 2 \wedge (cur(s) + 2)\ mod\ 10 = 0 \vee$

$\quad Legal(s) \wedge turn(s) = O \wedge turn(s) = O \wedge 100 < cur(s) + 3 \wedge (cur(s) + 3)\ mod\ 10 = 0)$

$\wedge($

$\quad Legal(s) \wedge turn(s) = O \wedge turn(s) = X \wedge 100 < cur(s) + 2 \wedge (cur(s) + 2)\ mod\ 10 = 0 \vee$

$\quad Legal(s) \wedge turn(s) = O \wedge turn(s) = X \wedge 100 < cur(s) + 5 \wedge (cur(s) + 5)\ mod\ 10 = 0 \vee$

$\quad Legal(s) \wedge turn(s) = O \wedge turn(s) = O \wedge 100 < cur(s) + 3 \wedge (cur(s) + 3)\ mod\ 10 = 0 \vee$

$\quad Legal(s) \wedge turn(s) = O \wedge turn(s) = O \wedge 100 < cur(s) + 4 \wedge (cur(s) + 4)\ mod\ 10 = 0)$

$\equiv$ by FOL

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) \wedge cur(s)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge 100 < cur(s) + 3 \wedge (cur(s) + 3)\ mod\ 10 = 0) \vee$

$Legal(s) \wedge turn(s) = X \wedge 100 < cur(s) + 1 \wedge (cur(s) + 1)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge 100 < cur(s) + 2 \wedge (cur(s) + 2)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge 100 < cur(s) + 4 \wedge (cur(s) + 4)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge 100 < cur(s) + 5 \wedge (cur(s) + 5)\ mod\ 10 = 0$

This approximation evaluates to true if s is legal and such that X can win in at most 3 steps. These are situations where the current position is 0 away from a modulo 10 position above 100 and now it is player O's turn, or the current position is between 1 and 6 but not 3 or 6 away from modulo 10 position above 100 and it is player X's turn, or the current position is 3 from modulo 10 position above 100 and it is player O's turn. $\square$

The subsequent results can be generalized (by induction) to the following formulas:

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MU}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_{2i-1}(s)} \equiv$

$Legal(s) \wedge turn(s) = O \wedge \exists j.\ \mathbf{0 \leq j < i} \wedge 100 - 3j < cur(s) \wedge (cur(s) + 3j)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists j.\ 0 \leq j < i \wedge 100 - 3j - 1 < cur(s) \wedge (cur(s) + 3j + 1)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists j.\ 0 \leq j < i \wedge 100 - 3j - 2 < cur(s) \wedge (cur(s) + 3j + 2)\ mod\ 10 = 0$

This approximation evaluates to true if s is legal and such that X can win in at most $2i - 1$ (odd) steps.

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MU}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_{2i}(s)} \equiv$

$Legal(s) \wedge turn(s) = O \wedge \exists j.\ \mathbf{0 \leq j \leq i} \wedge 100 - 3j < cur(s) \wedge (cur(s) + 3j)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists j.\ 0 \leq j < i \wedge 100 - 3j - 1 < cur(s) \wedge (cur(s) + 3j + 1)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists j.\ 0 \leq j < i \wedge 100 - 3j - 2 < cur(s) \wedge (cur(s) + 3j + 2)\ mod\ 10 = 0$

This approximation evaluates to true if s is legal and such that X can win in at most $2i$ (even) steps.

Is there a convergence in a finite number of steps? No.

The De Giacomo et al. [DLP10] iterative method for logical formula manipulation does not work using only the unique name and domain closure axioms for actions $\mathcal{D}_{ca}^{MU}$ and the axiomatization of the integers

$\mathcal{D}_Z$. It will not converge in a finite number of steps as it can be observed that for all natural numbers i, $D_{ca}^{MU} \cup D_Z \not\models R_{i-1} \equiv R_i$, since one can always construct a model of $D_{ca}^{MU} \cup D_Z$ that will satisfy R_{2i} and not R_{2i-1}, e.g. where $cur(s) = 3i$ for steps $2i - 1$ and $2i$.

Will there be a convergence if we use additional facts entailed by the whole theory ? Yes.

The procedure will converge if we add lemma L3.5.3 from the previous section to the axioms used by the method. Let us consider the formulas R_{87} and R_{88} (87 and 88 were chosen so that i will become 44 and $3i$ is a number that is at least 30 above 100 which greatly helps simplifying the generalized formulas):

$\mathbf{\mathcal{D}_{ca}^{MU} \cup \mathcal{D}_Z \cup \{L3.5.3\} \models R_{87}(s)} \equiv$

$$Legal(s) \wedge turn(s) = O \wedge \exists j.\ \mathbf{0 \leq j < 44} \wedge 100 - 3j < cur(s) \wedge (cur(s) + 3j)\ mod\ 10 = 0 \vee$$

$$Legal(s) \wedge turn(s) = X \wedge \exists j.\ \mathbf{0 \leq j < 44} \wedge 100 - 3j - 1 < cur(s) \wedge (cur(s) + 3j + 1)\ mod\ 10 = 0 \vee$$

$$Legal(s) \wedge turn(s) = X \wedge \exists j.\ 0 \leq j < 44 \wedge 100 - 3j - 2 < cur(s) \wedge (cur(s) + 3j + 2)\ mod\ 10 = 0$$

$\equiv$

$$Legal(s) \wedge turn(s) = O \wedge \exists j.\ \mathbf{0 \leq j < 34} \wedge 100 - 3j < cur(s) \wedge (cur(s) + 3j)\ mod\ 10 = 0 \vee$$

$$Legal(s) \wedge turn(s) = O \wedge -2 < cur(s) \wedge (cur(s) + 102)\ mod\ 10 = 0 \vee$$

$$Legal(s) \wedge turn(s) = O \wedge -5 < cur(s) \wedge (cur(s) + 105)\ mod\ 10 = 0 \vee$$

$$Legal(s) \wedge turn(s) = O \wedge -8 < cur(s) \wedge (cur(s) + 108)\ mod\ 10 = 0 \vee$$

$$Legal(s) \wedge turn(s) = O \wedge -11 < cur(s) \wedge (cur(s) + 111)\ mod\ 10 = 0 \vee$$

$$Legal(s) \wedge turn(s) = O \wedge -14 < cur(s) \wedge (cur(s) + 114)\ mod\ 10 = 0 \vee$$

$$Legal(s) \wedge turn(s) = O \wedge -17 < cur(s) \wedge (cur(s) + 117)\ mod\ 10 = 0 \vee$$

$$Legal(s) \wedge turn(s) = O \wedge -20 < cur(s) \wedge (cur(s) + 120)\ mod\ 10 = 0 \vee$$

$$Legal(s) \wedge turn(s) = O \wedge -23 < cur(s) \wedge (cur(s) + 123)\ mod\ 10 = 0 \vee$$

$$Legal(s) \wedge turn(s) = O \wedge -26 < cur(s) \wedge (cur(s) + 126)\ mod\ 10 = 0 \vee$$

$$Legal(s) \wedge turn(s) = O \wedge -29 < cur(s) \wedge (cur(s) + 129)\ mod\ 10 = 0 \vee$$

$$Legal(s) \wedge turn(s) = X \wedge \exists j.\ \mathbf{0 \leq j < 34} \wedge 100 - 3j - 1 < cur(s) \wedge (cur(s) + 3j + 1)\ mod\ 10 = 0 \vee$$

$Legal(s) \wedge turn(s) = X \wedge -3 < cur(s) \wedge (cur(s) + 103)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge -6 < cur(s) \wedge (cur(s) + 106)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge -9 < cur(s) \wedge (cur(s) + 109)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge -12 < cur(s) \wedge (cur(s) + 112)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge -15 < cur(s) \wedge (cur(s) + 115)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge -18 < cur(s) \wedge (cur(s) + 118)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge -21 < cur(s) \wedge (cur(s) + 121)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge -24 < cur(s) \wedge (cur(s) + 124)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge -27 < cur(s) \wedge (cur(s) + 127)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge -30 < cur(s) \wedge (cur(s) + 130)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists j.\ 0 \leq j < 44 \wedge 100 - 3j - 2 < cur(s) \wedge (cur(s) + 3j + 2)\ mod\ 10 = 0$

$\equiv$ by L3.5.3

$Legal(s) \wedge turn(s) = O \wedge \exists j.\ \mathbf{0} \leq \mathbf{j} < \mathbf{34} \wedge 100 - 3j < cur(s) \wedge (cur(s) + 3j)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 102)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 105)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 108)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 111)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 114)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 117)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 120)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 123)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 126)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 129)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists j.\ \mathbf{0} \leq \mathbf{j} < \mathbf{34} \wedge 100 - 3j - 1 < cur(s) \wedge (cur(s) + 3j + 1)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge (cur(s) + 103)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge (cur(s) + 106)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge (cur(s) + 109)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge (cur(s) + 112)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge (cur(s) + 115)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge (cur(s) + 118)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge (cur(s) + 121)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge (cur(s) + 124)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge (cur(s) + 127)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge (cur(s) + 130)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists j.\ 0 \leq j < 44 \wedge 100 - 3j - 2 < cur(s) \wedge (cur(s) + 3j + 2)\ mod\ 10 = 0$

$\equiv$ by the fact that the numbers 102, 105, 108, 111, 114, 117, 120, 123, 126, and 129 one is 0 modulo 10

$\equiv$ by the fact that the numbers 103, 106, 109, 112, 115, 118, 121, 124, 127, and 130 one is 0 modulo 10

$Legal(s) \wedge turn(s) = O \wedge \exists j.\ 0 \leq j < 34 \wedge 100 - 3j < cur(s) \wedge (cur(s) + 3j)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge True \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists j.\ 0 \leq j < 34 \wedge 100 - 3j - 1 < cur(s) \wedge (cur(s) + 3j + 1)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge True \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists j.\ 0 \leq j < 44 \wedge 100 - 3j - 2 < cur(s) \wedge (cur(s) + 3j + 2)\ mod\ 10 = 0$

$\equiv$ by FOL (line 2 subsumes line 1, line 4 subsumes line 3 and 5)

$\mathbf{Legal(s) \wedge turn(s) = X \vee Legal(s) \wedge turn(s) = O}$ $\square$

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{MU}} \cup \mathcal{D}_{\mathbf{Z}} \cup \{\mathbf{L3.5.3}\} \models \mathbf{R_{88}(s)} \equiv$

$Legal(s) \wedge turn(s) = O \wedge \exists j.\ \mathbf{0 \leq j \leq 44} \wedge 100 - 3j < cur(s) \wedge (cur(s) + 3j)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists j.\ \mathbf{0 \leq j < 44} \wedge 100 - 3j - 1 < cur(s) \wedge (cur(s) + 3j + 1)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists j.\ 0 \leq j < 44 \wedge 100 - 3j - 2 < cur(s) \wedge (cur(s) + 3j + 2)\ mod\ 10 = 0$

$\equiv$

$Legal(s) \wedge turn(s) = O \wedge \exists j.\ \mathbf{0 \leq j \leq 34} \wedge 100 - 3j < cur(s) \wedge (cur(s) + 3j)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge\ -5 < cur(s) \wedge (cur(s) + 105)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge\ -8 < cur(s) \wedge (cur(s) + 108)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge\ -11 < cur(s) \wedge (cur(s) + 111)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge\ -14 < cur(s) \wedge (cur(s) + 114)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge\ -17 < cur(s) \wedge (cur(s) + 117)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge\ -20 < cur(s) \wedge (cur(s) + 120)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge\ -23 < cur(s) \wedge (cur(s) + 123)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge\ -26 < cur(s) \wedge (cur(s) + 126)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge\ -29 < cur(s) \wedge (cur(s) + 129)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge\ -32 < cur(s) \wedge (cur(s) + 132)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists j.\ \mathbf{0} \leq \mathbf{j} < \mathbf{34} \wedge 100 - 3j - 1 < cur(s) \wedge (cur(s) + 3j + 1)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -3 < cur(s) \wedge (cur(s) + 103)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -6 < cur(s) \wedge (cur(s) + 106)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -9 < cur(s) \wedge (cur(s) + 109)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -12 < cur(s) \wedge (cur(s) + 112)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -15 < cur(s) \wedge (cur(s) + 115)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -18 < cur(s) \wedge (cur(s) + 118)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -21 < cur(s) \wedge (cur(s) + 121)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -24 < cur(s) \wedge (cur(s) + 124)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -27 < cur(s) \wedge (cur(s) + 127)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge\ -30 < cur(s) \wedge (cur(s) + 130)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists j.\ 0 \leq j < 44 \wedge 100 - 3j - 2 < cur(s) \wedge (cur(s) + 3j + 2)\ mod\ 10 = 0$

$\equiv$ by L3.5.3

$Legal(s) \wedge turn(s) = O \wedge \exists j.\ \mathbf{0} \leq \mathbf{j} \leq \mathbf{34} \wedge 100 - 3j < cur(s) \wedge (cur(s) + 3j)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 105)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 108)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 111)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 114)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 117)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 120)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 123)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 126)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 129)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge (cur(s) + 132)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists j.\ \mathbf{0 \leq j < 34} \wedge 100 - 3j - 1 < cur(s) \wedge (cur(s) + 3j + 1)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge (cur(s) + 103)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge (cur(s) + 106)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge (cur(s) + 109)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge (cur(s) + 112)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge (cur(s) + 115)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge (cur(s) + 118)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge (cur(s) + 121)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge (cur(s) + 124)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge (cur(s) + 127)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge (cur(s) + 130)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists j.\ 0 \leq j < 44 \wedge 100 - 3j - 2 < cur(s) \wedge (cur(s) + 3j + 2)\ mod\ 10 = 0$

$\equiv$ by the fact that the numbers 105, 108, 111, 114, 117, 120, 123, 126, 129, and 132 one is 0 modulo 10

$\equiv$ by the fact that the numbers 103, 106, 109, 112, 115, 118, 121, 124, 127, and 130 one is 0 modulo 10

$Legal(s) \wedge turn(s) = O \wedge \exists j.\ 0 \leq j \leq 34 \wedge 100 - 3j < cur(s) \wedge (cur(s) + 3j)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = O \wedge True \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists j.\ 0 \leq j < 34 \wedge 100 - 3j - 1 < cur(s) \wedge (cur(s) + 3j + 1)\ mod\ 10 = 0 \vee$

$Legal(s) \wedge turn(s) = X \wedge True \vee$

$Legal(s) \wedge turn(s) = X \wedge \exists j.\ 0 \leq j < 44 \wedge 100 - 3j - 2 < cur(s) \wedge (cur(s) + 3j + 2)\ mod\ 10 = 0$

$\equiv$ by FOL (line 2 subsumes line 1, line 4 subsumes line 3 and 5)

$\mathbf{Legal(s)} \wedge \mathbf{turn(s)} = \mathbf{X} \vee \mathbf{Legal(s)} \wedge \mathbf{turn(s)} = \mathbf{O}$ $\square$

Thus the fixpoint expansion procedure converges no later than in the step 88 as we have:

$$\mathcal{D}_{ca}^{MU} \cup \mathcal{D}_Z \ \cup \ \{L3.5.3\} \models R_{87}(s) \equiv R_{88}(s)$$

And therefore by Theorem 1 of [DLP10]:

$$\mathcal{D}_{GS}^{MU} \ \models \langle\langle\{X\}\rangle\rangle \Diamond Wins(X)[s] \ \equiv Legal(s) \wedge turn(s) = X \vee Legal(s) \wedge turn(s) = O$$

It follows by the initial state axioms that $\mathcal{D}_{GS}^{MU} \models \langle\langle\{X\}\rangle\rangle \Diamond Wins(X)[S_0]$, i.e. there is a strategy for player X from the initial situation as it is legal and it is player X's turn.

3.6 Discussion

Several game domains have been developed that are quite representative of the type of problem that the technique is trying to address. The presented example problems have varying properties, which although not exhaustive and complete, still allow to evaluate if the techniques work on some problems in infinite domains. The work is based on the assumption that all agents have the complete knowledge of the theory, that actions are observable by all agents, and that there are no sensing actions that allow agents to gain additional private knowledge. The technique supports incomplete specifications of the application domain – the basic action theories do not need to have a single model. The results in this chapter demonstrate that the verification technique based on symbolic manipulation for properties in our game-theoretic logic for situation calculus game structures with infinite states actually does work on many domains and verification problems. We show that in some cases we must use facts about the initial situation to get convergence in a finite number of steps. Note that we require a suitable axiomatization $\mathcal{D}_Z$ of the integers for some games that use integer arithmetic.

The Light World (LW) game is an infinite state game structure with an infinite number of different models. It is also a multi-player game so properties related to cooperative and adversarial versions were examined. The verification of the possibility to win for the cooperative version of the game converges in a finite number of steps just by first-order logic entailment using only unique name and domain closure axioms $\mathcal{D}_{ca}$ and the axiomatization of the integers $\mathcal{D}_{Z}$. From the converged formula we can deduce that it is possible to achieve the goal in all legal situations. The verification of the existence of the winning strategy for the adversarial version of the game converges in a finite number of steps just by first-order logic entailment using only unique name and domain closure axioms $\mathcal{D}_{ca}$ and the axiomatization of the integers $\mathcal{D}_{Z}$. From the converged formula we can deduce that X can ensure that she/he will achieve the goal in some specific legal situations, but not in the initial situation. So for this domain, the [DLP10] method as originally specified works.

The Oil Lamp World (OLW) game is an infinite state game structure with an infinite number of different models. It is a single player game so the possibility of winning is the property that is verified. We try to verify the possibility of winning. In this case, the [DLP10] method, which only uses the simplest part of the domain theory, the unique names and domain closure for action axioms, fails to converge in a finite number of steps. But we also show that extending the method to use some selected facts about the initial situation and some state constraints does allow us to get convergence in a finite number of steps.

The In-Line Tic-Tac-Toe (TTT1D) game is an infinite state game structure with an infinite number of different models. It is also a multi-player game so properties related to cooperative and adversarial versions were examined. Among all the games that we have developed, this is probably the most like a real game - it is a natural simplification and extension to infinite space of the classic Tic-Tac-Toe game. The verification of the possibility to win for the cooperative version of the game converges in a finite number of steps just by first-order logic entailment using unique name and domain closure axioms $\mathcal{D}_{ca}$ and a suitable axiomatization $\mathcal{D}_{Z}$ of integers without requiring any knowledge of the initial situation. However the proof is long and many cases have to be handled. From the converged formula we can deduce that the goal may be achieved in all legal situations. The verification of the existence of the winning strategy for the adversarial version of the game also converges in a finite number of steps. From the converged formula we can deduce that X can ensure that she/he will achieve the goal in some specific legal situations, but not in the initial situation.

The Mark Down (MD) game is an infinite state game structure with an infinite number of different models. It is also a multi-player game so properties related to cooperative and adversarial versions were examined. The verification of the possibility to win for the cooperative version of the game fails to converge in a finite number of steps by first-order logic entailment using only unique name and domain closure axioms $\mathcal{D}_{ca}$ and the axiomatization $\mathcal{D}_Z$ of integers. The convergence in a finite number of steps is achieved if we add some facts from the initial situation. From the converged formula we can deduce that it is possible to achieve the goal in some specific legal situations, and that there may be no possibility to achieve the goal in some situations. The verification of the existence of the winning strategy for the adversarial version of the game fails to converge in a finite number of steps by first-order logic entailment using only unique name and domain closure axioms $\mathcal{D}_{ca}$ and the axiomatization $\mathcal{D}_Z$ of integers. But the convergence is achieved in a finite number of steps if we add some facts from the initial situation. From the converged formula we can deduce that X can ensure that she/he will achieve the goal in some specific legal situations including the given initial situation.

The Mark Up (MU) game is an infinite state game structure with an infinite number of different models. It is also a multi-player game so properties related to cooperative and adversarial versions were examined. The verification of the possibility to win for the cooperative version of the game fails to converge in a finite number of steps by first-order logic entailment using only unique name and domain closure axioms $\mathcal{D}_{ca}$ and the axiomatization $\mathcal{D}_Z$ of integers. The convergence in a finite number of steps is achieved if we add some facts from the initial situation. From the converged formula we can deduce that the goal may be achieved in all legal situations. The verification of the existence of the winning strategy for the adversarial version of the game fails to converge in a finite number of steps by first-order logic entailment using only unique name and domain closure axioms $\mathcal{D}_{ca}$ and the axiomatization $\mathcal{D}_Z$ of integers. But the convergence is achieved in a finite number of steps if we add some facts from the initial situation. From the converged formula we can deduce that X can ensure that she/he will achieve the goal in all legal situations.

4 Characteristic Graph-Based Verification of Properties of GameGolog Structures

As part of this thesis research we analyzed the feasibility and effectiveness of characteristic graph construction in verification of game theories (De Giacomo, Lespérance and Pearce [DLP10]). The main point there was to extend the technique of characteristic graphs to check ATL-type game-theoretic properties as presented in the background section of this thesis. The game domains from chapter 3 have been used to evaluate the characteristic graph approach to temporal property verification. It turns that with these two-player turn-taking types of games the calculations and results were very similar to those of the symbolic manipulation-based technique presented in chapter 3. As there was not much new that would be contributed by this method based on the sample problems, our focus was put on selecting one representative problem and demonstrating how the technique is used step-by-step. The objective was to create a tutorial that can be followed in future research and used in characteristic graph calculation for other game-like problems.

4.1 Light World (LW)

The Light World (LW) example that was analyzed here is the same problem presented in chapter 3. It shows that the characteristic graph technique works for this example and produces the same results as the symbolic formula manipulation technique that was researched in section 3.1.

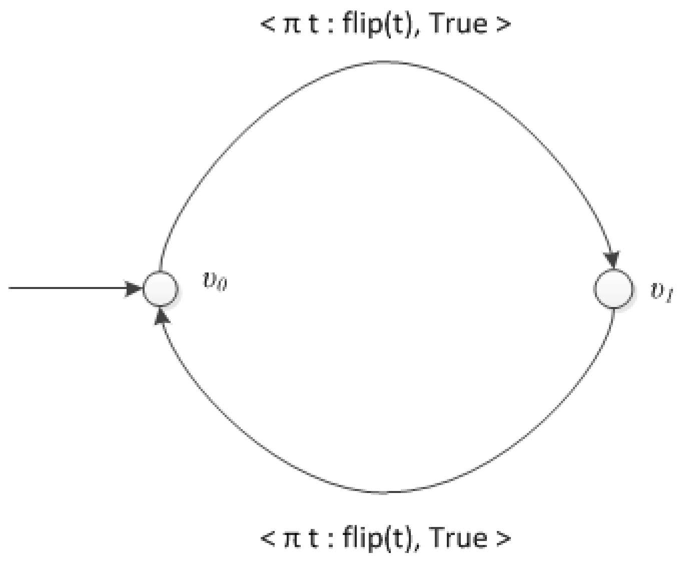
4.1.1 GameGolog Program of the LW Game

The GameGolog program for the LW (Light World) Game is given as:

```
$\rho_{LW}$ =
    while $\neg Finished()$ do (
        $[X\ \pi t.flip(t)]$;
        if $\neg Finished()$ then
            $[O\ \pi t.flip(t)]$
        else
            $True?$
    )
```

4.1.2 Characteristic Graph of the LW Game

The characteristic graph $\mathcal{G}_{LW}$ of the Light World (LW) problem for the program ρ_{LW} has two nodes as the players take turns. The first vertex (for actions of player X) has the program ρ_{LW} left to run and can terminate if the fluent *Finished* (defined as just the *Wins* fluent in the axiomatization) holds. The second vertex (for actions of player O) has the step $[O\ \pi t.flip(t)]$ and then program ρ_{LW} left to run and can also terminate if the fluent *Finished* holds. The edges from vertex to vertex are labelled with actions of the player of the vertex and are always possible. The graph is as follows:



Where:

- $agent(\upsilon_0) = X$ i.e., υ_0 is controlled by agent X that is actions from υ_0 are possible iff $turn(s) = X$
- $agent(\upsilon_1) = O$ i.e., υ_1 is controlled by agent O that is actions from υ_1 are possible iff $turn(s) = O$
- $\upsilon_0 = \langle(\rho_{LW}, Finished()\rangle$
- $\upsilon_1 = \langle[O\ \pi t.flip(t)]; \rho_{LW}, Finished()\rangle$

We also use the Situation Calculus game structure axiomatization of this problem as defined in chapter 3. Here the *Legal* predicate is replaced by the axioms that use the GameGolog program ρ_{LW} as explained in section 2.5.3. Now, *Legal* can be assumed to hold for all situations that respect the GameGolog program, that is, all situations that result from running the program are implicitly legal, and therefore *Legal* can be removed from the $Wins$ predicate in particular.

4.1.3 Possibility of Winning

The property that it is possible for X to eventually win can be represented by the following formula:

$$\exists\Diamond Wins(X) \doteq \mu Z.Wins(X) \ \vee \ \exists \bigcirc Z$$

This can be verified by performing the De Giacomo et al. [DLP10] labelling method for characteristic graphs (as explained in the background section of this thesis):

$$[\![\mu Z.Wins(X) \ \vee \ \exists \bigcirc Z]\!]$$

We proceed by using regression and the labelling fixpoint approximation operation LFP. We try to show that successive labellings are equivalent using only the unique name and domain closure axioms for actions $\mathcal{D}_{ca}^{LW}$, and a suitable axiomatization of integers $\mathcal{D}_Z$. The technique does not always converge and this needs to be checked as we proceed.

In the computations we will use the following lemma:

L 4.1.1. *Concrete case of* PRE *(labelling of $\bigcirc$, see section 2.7.3) definition for the LW domain.*

$\text{PRE}\ (\{X,O\},\{\langle v_0,\phi_0\rangle,\langle v_1,\phi_1\rangle\}) = \{$

$\langle v_0, \exists t.\ \mathcal{R}(\phi_1(do(flip(X,t),s)))\rangle,$

$\langle v_1, \exists t.\ \mathcal{R}(\phi_0(do(flip(O,t),s)))\rangle\}$

proof sketch: the lemma follows from the expansion of the definition of PRE *for $\mathcal{G}_{LW}$* □

L 4.1.2. *Concrete case of the $[\![Wins(X,s)]\!]$.*

$[\![Wins(X,s)]\!] =$

$\{\langle v_0, On(1,s) \wedge On(2,s)\rangle,$

$\langle v_1, On(1,s) \wedge On(2,s)\rangle\}$

proof sketch: the lemma follows from the definitions of the labelling operations from section 2.7.3 □

The fixpoint approximations of the operation LFP and the details of the labelling are explained next.

$\mathcal{Z}_0(s) = [\![Wins(X,s) \vee \exists \bigcirc False]\!] =$

$[\![Wins(X,s)]\!]$ OR PRE $(\{X,O\},[\![False]\!])$

Now,

$[\![False]\!] =$ by the definitions of the labelling operation from section 2.7.3

$\{\langle v_0, False\rangle,$

$\langle v_1, False\rangle\}$

$\textsc{pre}\ (\{X, O\}, [\![False]\!]) =$ by Lemma 4.1.1 and $[\![False]\!]$ above

$\{\langle v_0, False\rangle,$

$\langle v_1, False\rangle\}$

And thus,

$\mathbfcal{Z}_\mathbf{0}(\mathbf{s}) = [\![\mathbf{Wins}(\mathbf{X}, \mathbf{s}) \vee \exists \bigcirc \mathbf{False}]\!] =$

$\{\langle v_0, On(1, s) \wedge On(2, s)\rangle,$

$\langle v_1, On(1, s) \wedge On(2, s)\rangle\}$

The labels (for either node of the graph) of this approximation evaluate to true if s is such that X is winning in s already (in no steps). These are situations where light 1 and light 2 are on. □

$\mathcal{Z}_1(s) = [\![Wins(X, s) \vee \exists \bigcirc \mathcal{Z}_0]\!] =$

$[\![Wins(X, s)]\!]$ $\textsc{or}$ $\textsc{pre}\ (\{X, O\}, \mathcal{Z}_0)$

Now,

$\textsc{pre}\ (\{X, O\}, \mathcal{Z}_0) =$ by Lemma 4.1.1 and $\mathcal{Z}_0$ from the previous step

$\{\langle v_0, \exists t.\ \mathcal{R}(On(1, do(flip(X, t), s)) \wedge On(2, do(flip(X, t), s)))\rangle,$

$\langle v_1, \exists t.\ \mathcal{R}(On(1, do(flip(O, t), s)) \wedge On(2, do(flip(O, t), s)))\rangle\}$

$=$ by P3.1.2 and P3.1.3 from chapter 3 and FOL

$\{\langle v_0, \exists t.\ ($

$(\neg On(1, s) \wedge t = 1 \vee On(1, s) \wedge t \neq 1) \wedge$

$(\neg On(2,s) \wedge t = 2 \vee On(2,s) \wedge t \neq 2))\rangle,$

$\langle \upsilon_1, \exists t.\ ($

$(\neg On(1,s) \wedge t = 1 \vee On(1,s) \wedge t \neq 1) \wedge$

$(\neg On(2,s) \wedge t = 2 \vee On(2,s) \wedge t \neq 2))\rangle\}$

$=$ by FOL ($\vee$ over $\wedge$)

$\{\langle \upsilon_0, \exists t.\ ($

$\neg On(1,s) \wedge t = 1 \wedge \neg On(2,s) \wedge t = 2 \vee$

$\neg On(1,s) \wedge t = 1 \wedge On(2,s) \wedge t \neq 2 \vee$

$On(1,s) \wedge t \neq 1 \wedge \neg On(2,s) \wedge t = 2 \vee$

$On(1,s) \wedge t \neq 1 \wedge On(2,s) \wedge t \neq 2)\rangle,$

$\langle \upsilon_1, \exists t.\ ($

$\neg On(1,s) \wedge t = 1 \wedge \neg On(2,s) \wedge t = 2 \vee$

$\neg On(1,s) \wedge t = 1 \wedge On(2,s) \wedge t \neq 2 \vee$

$On(1,s) \wedge t \neq 1 \wedge \neg On(2,s) \wedge t = 2 \vee$

$On(1,s) \wedge t \neq 1 \wedge On(2,s) \wedge t \neq 2)\rangle\}$

$=$ by FOL (removal of contradiction and quantifier elimination)

$\{\langle \upsilon_0, ($

$\neg On(1,s) \wedge On(2,s) \vee$

$On(1,s) \wedge \neg On(2,s) \vee$

$On(1,s) \wedge On(2,s))\rangle,$

$\langle \upsilon_1, \exists t.\ ($

$\neg On(1,s) \wedge On(2,s) \vee$

$On(1,s) \wedge \neg On(2,s) \vee$

$On(1,s) \wedge On(2,s))\rangle\}$

$=$ by FOL (combining of line 2 and 4, 3 and 4, 6 and 8, line 7 and 8)

$\{\langle \upsilon_0, On(1,s) \vee On(2,s)\rangle,$

$\langle v_1, On(1,s) \vee On(2,s)\rangle\}$

And thus,

$\boldsymbol{\mathcal{Z}_1(s) = [\![Wins(X,s) \vee \exists \bigcirc \mathcal{Z}_0]\!] =}$

$[\![Wins(X,s)]\!]$ OR PRE $(\{X,O\}, \mathcal{Z}_0)$

$=$ by the definition of OR

$\{\langle v_0, On(1,s) \wedge On(2,s) \vee On(1,s) \vee On(2,s)\rangle,$

$\langle v_1, On(1,s) \wedge On(2,s) \vee On(1,s) \vee On(2,s)\rangle\}$

$=$ by FOL (subsumption of $On(1,s) \wedge On(2,s)$)

$\{\langle v_0, On(1,s) \vee On(2,s)\rangle,$

$\langle v_1, On(1,s) \vee On(2,s)\rangle\}$

The labels (for either node of the graph) of this approximation evaluate to true if s is such that the game can be won in at most 1 step. These are legal situations where player X wins already or one of lights 1 or 2 is on (X or O can turn the other light at the next step). $\square$

$\mathcal{Z}_2(s) = [\![Wins(X,s) \vee \exists \bigcirc \mathcal{Z}_1]\!] =$

$[\![Wins(X,s)]\!]$ OR PRE $(\{X,O\}, \mathcal{Z}_1)$

Now,

PRE $(\{X,O\}, \mathcal{Z}_1) =$ by Lemma 4.1.1 and $\mathcal{Z}_1$ from the previous step

$\{\langle v_0, \exists t.\ \mathcal{R}(On(1, do(flip(X,t),s)) \vee On(2, do(flip(X,t),s)))\rangle,$

$\langle v_1, \exists t.\ \mathcal{R}(On(1, do(flip(O,t),s)) \vee On(2, do(flip(O,t),s)))\rangle\}$

$=$ by P3.1.2 and P3.1.3 and FOL

$\{\langle v_0, \exists t.\ ((\neg On(1,s) \wedge t = 1 \vee On(1,s) \wedge t \neq 1) \vee (\neg On(2,s) \wedge t = 2 \vee On(2,s) \wedge t \neq 2))\rangle,$

$\langle v_1, \exists t.\ ((\neg On(1,s) \wedge t = 1 \vee On(1,s) \wedge t \neq 1) \vee (\neg On(2,s) \wedge t = 2 \vee On(2,s) \wedge t \neq 2))\rangle\}$

$=$ by FOL ($\vee$ over $\wedge$)

$\{\langle v_0, (\neg On(1,s) \vee On(1,s) \vee \neg On(2,s) \vee On(2,s))\rangle,$

$\{\langle v_1, (\neg On(1,s) \vee On(1,s) \vee \neg On(2,s) \vee On(2,s))\rangle\}$

$=$ by FOL (combining on line 2 and 3)

$\{\langle v_0, True\rangle,$

$\langle v_1, True\rangle\}$

And thus,

$\mathcal{Z}_{\mathbf{2}}(\mathbf{s}) = [\![\mathbf{Wins}(\mathbf{X},\mathbf{s}) \vee \exists \bigcirc \mathcal{Z}_{\mathbf{1}}]\!] =$

$[\![Wins(X,s)]\!]$ OR PRE $(\{X,O\}, \mathcal{Z}_1)$

$=$ by the definition of OR

$\{\langle v_0, On(1,s) \wedge On(2,s) \vee True\rangle,$

$\langle v_1, On(1,s) \wedge On(2,s) \vee True\rangle\}$

$=$ by FOL

$\{\langle v_0, True\rangle,$

$\langle v_1, True\rangle\}$

The labels (for either node of the graph) of this approximation evaluate to true if s is such that the game can be won in at most 2 steps. Here it is true for all situations as one step can turn light 1 on and the second step can turn light 2 on. □

$\mathcal{Z}_3(s) = [\![Wins(X,s) \vee \exists \bigcirc \mathcal{Z}_2]\!] =$

$[\![Wins(X,s)]\!]$ OR PRE $(\{X,O\}, \mathcal{Z}_2)$

Now,

PRE $(\{X,O\}, \mathcal{Z}_2) =$ by Lemma 4.1.1 and $\mathcal{Z}_1$ from the previous step

$\{\langle v_0, \exists t.\ \mathcal{R}(True)\rangle,$

$\langle v_1, \exists t.\ \mathcal{R}(True)\rangle\}$

$=$

$\{\langle v_0, True\rangle,$

$\{\langle v_1, True\rangle\}$

And thus,

$\mathcal{Z}_{\mathbf{3}}(\mathbf{s}) = [\![\mathbf{Wins}(\mathbf{X}, \mathbf{s}) \vee \exists \bigcirc \mathcal{Z}_{\mathbf{2}}]\!] =$

$[\![Wins(X, s)]\!]$ OR PRE $(\{X, O\}, \mathcal{Z}_2)$

$=$ by the definition of OR

$\{\langle v_0, On(1, s) \wedge On(2, s) \vee True\rangle,$

$\langle v_1, On(1, s) \wedge On(2, s) \vee True\rangle\}$

$=$ by FOL

$\{\langle v_0, True\rangle,$

$\langle v_1, True\rangle\}$ □

Thus the labelling fixpoint expansion procedure converges in the 4th step as we have:

$$\mathcal{D}_{ca}^{LW} \models \mathcal{Z}_2(s) = \mathcal{Z}_3(s)$$

And therefore by some reasoning about the program and Theorem 5 of [DLP10]:

$$\mathcal{D}_{GS}^{LW'} \setminus \Sigma \models \exists \Diamond Wins(X)[\rho_{LW}, s] \ \equiv \ True$$

where $\mathcal{D}_{GS}^{LW'}$ is $\mathcal{D}_{GS}^{LW}$ with Legal defined as a GameGolog program

By Theorem 6 of [DLP10] it follows by the initial state axioms that $\mathcal{D}_{GS}^{LW'} \models \exists \Diamond Wins(X)[\rho_{LW}, S_0]$, i.e., the goal may eventually be reached from the initial situation.

4.1.4 Existence of a Winning Strategy

The existence of a strategy to ensure $Wins(X)$ by group $G = \{X\}$ can be represented by the following formula:

$$\langle\langle\{X\}\rangle\rangle\Diamond Wins(X) \doteq \mu Z.\ Wins(X) \vee \langle\langle\{X\}\rangle\rangle \bigcirc Z$$

This can be verified by performing the De Giacomo et al. [DLP10] labelling method for characteristic graphs:

$$[\![\mu Z.\ Wins(X)\ \vee \langle\langle\{X\}\rangle\rangle \bigcirc Z]\!]$$

We proceed by using regression and the labelling fixpoint approximation operation LFP. We try to show that successive labellings are equivalent using only the unique name and domain closure axioms for actions $\mathcal{D}_{ca}^{LW}$, and a suitable axiomatization of integers $\mathcal{D}_Z$. The technique does not always converge and this needs to be checked as we proceed.

In the computations we will use the following lemma:

L 4.1.3. *Concrete case of the* PRE *(labelling of $\bigcirc$, see section 2.7.3) definition for the LW domain.*

$$\begin{aligned}
&\text{PRE}\ \ (\{X\}, \{\langle v_0, \phi_0\rangle, \langle v_1, \phi_1\rangle\}) = \{\\
&\qquad \langle v_0, \exists t.\ \mathcal{R}(\phi_1(do(flip(X,t),s)))\rangle,\\
&\qquad \langle v_1, \forall t.\ \mathcal{R}(\phi_0(do(flip(O,t),s)))\rangle\}
\end{aligned}$$

proof sketch: the lemma follows from the expansion of the definition of PRE *for* $\mathcal{G}_{LW}$ □

The fixpoint approximations of the operation LFP and the details of the labelling are explained next.

$$\begin{aligned}
&\mathcal{Z}_0(s) = [\![Wins(X,s) \vee \langle\langle\{X\}\rangle\rangle \bigcirc False]\!] =\\
&\qquad [\![Wins(X,s)]\!]\ \text{OR}\ \ \text{PRE}\ \ (\{X\}, [\![False]\!])
\end{aligned}$$

Now,

$[\![False]\!] =$ by the definitions of the labelling operation from section 2.7.3

$\{\langle v_0, False\rangle,$

$\langle v_1, False\rangle\}$

$\textsc{pre}\ (\{X\}, [\![False]\!]) =$ by Lemma 4.1.3 and $[\![False]\!]$ above

$\{\langle v_0, False\rangle,$

$\langle v_1, False\rangle\}$

And thus,

$\mathcal{Z}_\mathbf{0}(\mathbf{s}) = [\![\mathbf{Wins}(\mathbf{X}, \mathbf{s}) \vee \langle\langle\{\mathbf{X}\}\rangle\rangle \bigcirc \mathbf{False}]\!] =$

$\{\langle v_0, On(1, s) \wedge On(2, s)\rangle,$

$\langle v_1, On(1, s) \wedge On(2, s)\rangle\}$

The labels of this approximation evaluate to true if s is such that X is winning in s already (in no steps). These are situations where light 1 and light 2 are on. □

$\mathcal{Z}_1(s) = [\![Wins(X, s) \vee \langle\langle\{X\}\rangle\rangle \bigcirc \mathcal{Z}_0]\!] =$

$[\![Wins(X, s)]\!]\ \textsc{or}\ \textsc{pre}\ (\{X\}, \mathcal{Z}_0)$

Now,

$\textsc{pre}\ (\{X\}, \mathcal{Z}_0) =$ by Lemma 4.1.3 and $\mathcal{Z}_0$ from the previous step

$\{\langle v_0, \exists t.\ \mathcal{R}(On(1, do(flip(X, t), s)) \wedge On(2, do(flip(X, t), s)))\rangle,$

$\langle v_1, \forall t.\ \mathcal{R}(On(1, do(flip(O, t), s)) \wedge On(2, do(flip(O, t), s)))\rangle\}$

$=$ by P3.1.2 and P3.1.3 from chapter 3 and FOL

$\{\langle v_0, \exists t.\ ($

$(\neg On(1, s) \wedge t = 1 \vee On(1, s) \wedge t \neq 1) \wedge$

$(\neg On(2, s) \wedge t = 2 \vee On(2, s) \wedge t \neq 2))\rangle,$

$\langle \upsilon_1, \forall t.\ ($

$(\neg On(1,s) \wedge t = 1 \vee On(1,s) \wedge t \neq 1) \wedge$

$(\neg On(2,s) \wedge t = 2 \vee On(2,s) \wedge t \neq 2))\rangle\}$

$=$ by FOL ($\vee$ over $\wedge$)

$\{\langle \upsilon_0, \exists t.\ ($

$\neg On(1,s) \wedge t = 1 \wedge \neg On(2,s) \wedge t = 2 \vee$

$\neg On(1,s) \wedge t = 1 \wedge On(2,s) \wedge t \neq 2 \vee$

$On(1,s) \wedge t \neq 1 \wedge \neg On(2,s) \wedge t = 2 \vee$

$On(1,s) \wedge t \neq 1 \wedge On(2,s) \wedge t \neq 2)\rangle,$

$\langle \upsilon_1, \forall t.\ ($

$\neg On(1,s) \wedge t = 1 \wedge \neg On(2,s) \wedge t = 2 \vee$

$\neg On(1,s) \wedge t = 1 \wedge On(2,s) \wedge t \neq 2 \vee$

$On(1,s) \wedge t \neq 1 \wedge \neg On(2,s) \wedge t = 2 \vee$

$On(1,s) \wedge t \neq 1 \wedge On(2,s) \wedge t \neq 2)\rangle\}$

$=$ by FOL (removal of contradiction and quantifier elimination)

$\{\langle \upsilon_0, ($

$\neg On(1,s) \wedge On(2,s) \vee$

$On(1,s) \wedge \neg On(2,s) \vee$

$On(1,s) \wedge On(2,s))\rangle,$

$\langle \upsilon_1,$

$\neg On(1,s) \wedge On(2,s) \wedge$ for t=1

$On(1,s) \wedge \neg On(2,s) \wedge$ for t=2

$On(1,s) \wedge On(2,s)\rangle\}$ for $2 < t$

$=$ by FOL (contradiction in lines 6 and 7, combining of line 2 and 4, 3 and 4, 6)

$\{\langle \upsilon_0, On(1,s) \vee On(2,s)\rangle,$

$\langle \upsilon_1, False\rangle\}$

And thus,

$\mathcal{Z}_\mathbf{1}(\mathbf{s}) = [\![\mathbf{Wins}(\mathbf{X},\mathbf{s}) \vee \langle\langle\{\mathbf{X}\}\rangle\rangle \bigcirc \mathcal{Z}_\mathbf{0}]\!] =$

$[\![Wins(X,s)]\!]$ OR PRE $(\{X\},\mathcal{Z}_0)$

$=$ by the definition of OR

$\{\langle v_0, On(1,s) \wedge On(2,s) \vee On(1,s) \vee On(2,s)\rangle,$

$\langle v_1, On(1,s) \wedge On(2,s) \vee False\rangle\}$

$=$ by FOL (subsumption of $On(1,s)$ land $On(2,s)$)

$\{\langle v_0, On(1,s) \vee On(2,s)\rangle,$

$\langle v_1, On(1,s) \wedge On(2,s)\rangle\}$

The labels (for either node of the graph) of this approximation evaluate to true if s is such that the game can be won in at most 1 step. These are situations where player X wins already, or the next action is by player X and one of lights 1 or 2 is on (X can turn on the other missing light). □

$\mathcal{Z}_2(s) = [\![Wins(X,s) \vee \langle\langle\{X\}\rangle\rangle \bigcirc \mathcal{Z}_1]\!] =$

$[\![Wins(X,s)]\!]$ OR PRE $(\{X\},\mathcal{Z}_1)$

Now,

PRE $(\{X\},\mathcal{Z}_1) =$ by Lemma 4.1.3 and $\mathcal{Z}_1$ from the previous step

$\{\langle v_0, \exists t.\ \mathcal{R}(On(1,do(flip(X,t),s)) \wedge On(2,do(flip(X,t),s)))\rangle,$

$\langle v_1, \forall t.\ \mathcal{R}(On(1,do(flip(O,t),s)) \vee On(2,do(flip(O,t),s)))\rangle\}$

$=$ by P3.1.2 and P3.1.3 and FOL

$\{\langle v_0, \exists t.\ ((\neg On(1,s) \wedge t=1 \vee On(1,s) \wedge t \neq 1) \wedge (\neg On(2,s) \wedge t=2 \vee On(2,s) \wedge t \neq 2))\rangle,$

$\langle v_1, \forall t.\ (\neg On(1,s) \wedge t=1 \vee On(1,s) \wedge t \neq 1 \vee \neg On(2,s) \wedge t=2 \vee On(2,s) \wedge t \neq 2)\rangle\}$

$=$ by FOL ($\vee$ over $\wedge$ etc.)

$\{\langle v_0, ($

$\neg On(1, s) \wedge On(2, s) \vee$ $\quad t = 1$

$On(1, s) \wedge \neg On(2, s) \vee$ $\quad t = 2$

$On(1, s) \wedge On(2, s))\rangle,$ $\quad 2 < t$

$\langle \upsilon_1,$

$(\neg On(1, s) \vee On(2, s)) \wedge$ $\quad t = 1$

$(On(1, s) \vee \neg On(2, s)) \wedge$ $\quad t = 2$

$(On(1, s) \vee On(2, s))\rangle\}$ $\quad 2 < t$

$=$ by FOL (combining on line 2 and 4, 3 and 4, combining line 6 and 8, 7 and 8)

$\{\langle \upsilon_0, On(2, s) \vee On(1, s)\rangle,$

$\langle \upsilon_1, (On(2, s) \wedge On(1, s))\rangle\}$

$=$ by FOL

$\{\langle \upsilon_0, On(1, s) \vee On(2, s)\rangle,$

$\langle \upsilon_1, On(1, s) \wedge On(2, s)\rangle\}$

And thus,

$\mathscr{Z}_\mathbf{2}(\mathbf{s}) = [\![\mathbf{Wins}(\mathbf{X}, \mathbf{s}) \vee \langle\langle\{\mathbf{X}\}\rangle\rangle \bigcirc \mathscr{Z}_\mathbf{1}]\!] =$

$[\![Wins(X, s)]\!]$ OR PRE $(\{X\}, \mathscr{Z}_1)$

$=$ by the definition of OR

$\{\langle \upsilon_0, On(1, s) \wedge On(2, s) \vee On(1, s) \vee On(2, s)\rangle,$

$\langle \upsilon_1, On(1, s) \wedge On(2, s) \vee On(1, s) \wedge On(2, s)\rangle\}$

$=$ by FOL

$\{\langle \upsilon_0, On(1, s) \vee On(2, s)\rangle,$

$\langle \upsilon_1, On(1, s) \wedge On(2, s)\rangle\}$

The labels (for either node of the graph) of this approximation evaluate to true if s is such that the game can be won in at most 2 steps. These are situations where player X wins already, or the next action is by player X and one of lights 1 or 2 is on (X can turn on the other missing light). $\square$

Thus the labelling fixpoint expansion procedure converges in the 3th step as we have:

$$\mathcal{D}_{ca}^{LW} \models \mathcal{Z}_1(s) = \mathcal{Z}_2(s)$$

And therefore by reasoning about the program and Theorem 5 of [DLP10] as we get:

$$\mathcal{D}_{GS}^{LW'} \setminus \Sigma \models \langle\langle\{X\}\rangle\rangle\Diamond Wins(X)[\rho_{LW}, s] \equiv On(1, s) \vee On(2, s)$$

where $\mathcal{D}_{GS}^{LW'}$ is $\mathcal{D}_{GS}^{LW}$ with Legal defined as a GameGolog program

By the Theorem 6 of [DLP10] and by the initial state axioms it follows that:

$$\mathcal{D}_{GS}^{LW'} \models \neg\langle\langle\{X\}\rangle\rangle\Diamond Wins(X)[\rho_{LW}, S_0]$$

i.e., there is no winning strategy for X in the initial situation S_0 as $Control(S_0) = X$ and none of the lights 1 or 2 is on.

However we have that

$$\mathcal{D}_{GS}^{LW'} \models \langle\langle\{X\}\rangle\rangle\Diamond Wins(X)[\rho_{LW}, S_1]$$

where $S_1 = do(flip(O, 3), do(flip(X, 1), S_0))$

i.e., there is a winning strategy for X in the situation S_1 where X has first turned light 1 on and then O has turned light 3 on, as X can turn on light 2 next.

4.2 Discussion

The first conclusion is that the characteristic graph method does work for some infinite domains. The technique also supports incomplete specifications – the basic action theories do not need to have a single model. For the selected example Light World (LW) game the verification of the possibility of winning and the existence of the winning strategy converged in a finite number of steps just by first-order logic entailment using only unique name and domain closure axioms $\mathcal{D}_{ca}$ and the axiomatization of the integers, regardless of the state in the initial situation. We also used the technique on game domains from chapter 3 to evaluate the characteristic graph approach to temporal property verification. The usability of the method appears to be very similar to the symbolic manipulation-based technique for simple games that have only 2 configurations. For more complex games with several stages of play, GameGolog theories would likely provide substantial advantages for verification as it appears that the characteristic graph method modularizes the computations as the cases are broken up.

5 Automated Evaluation-Based Verification of Properties of Game Theories

As part of this thesis research the feasibility and effectiveness of automated evaluation-based verification of properties of game theories as per De Giacomo, Lespérance and Pearce [DLP10] was developed and applied to verification of a sample game theory for the well-known Tic-Tac-Toe game. The main point was to develop a technique that can automatically check ATL-type game-theoretic properties as presented in the background section of this thesis. We provide several variants of the technique: a method based on Basic Action Theories, a variant where constraints on actions of players can be specified, and a variant where the game structure is given in the form of game's GameGolog program.

5.1 Verifier for Basic Action Theory Game Structures

The code for the evaluation-based verifier is listed in the appendix A.1.

The technique presented in this section builds on the logic programming evaluator for SitCalc projection queries developed by Reiter [Rei01] for initial state theories with the closed world assumption (which makes them complete theories), that relies on regression. It uses Lloyd-Topor transformations to soundly reduce the evaluation of complex First-Order projection queries to that of atomic projection queries. In our implementation we handle the the basic operator $\langle\langle G\rangle\rangle \bigcirc \varphi[S]$ essentially by macro-expanding it into its situation calculus definition and evaluating the result. It has been developed and implemented in Prolog and tested on the classic Tic-Tac-Toe game theory expressed in the Situation Calculus using Prolog clauses. Here the term “evaluation-based” refers to the use of evaluation instead of entailment to check state properties under the conditions of complete theory (i.e. single model) and closed-world assumption.

In general the verifier checks if a given temporal property expressed in the Ł-Logic holds for a given situation. It is equipped with the constructs to represent Ł-Logic formulas and the parsing and evaluation mechanism to check if a formula holds. The verifier code is independent of any particular problem domain. It uses the Basic Action Theory axiomatization of the modelled domain that is kept in a separate source file. The evaluation is done according to the semantics of the Ł-Logic and is straightforward for the most part. The main element of the evaluator is the evaluation of temporal terms (according to their definition) by using the “next” operator $\langle\langle G \rangle\rangle \bigcirc \varphi$, the μ operator fixpoint procedure, and regression. Here is the code for the $\langle\langle G \rangle\rangle \bigcirc \varphi$ operator:

```
%################################################
% "can ensure next" operator
% this macro expands according to SitCalc definition
%################################################
holds(next(G,F),S) :- !, (
    incontrol(G,S), %*** group in control, any legal successor action ***
    holds(exists_successor(G,F),S)
    ;
    incontrol(-G,S), %*** anti-group in control, all legal successor actions ***
    holds(forall_successors2(-G,F),S)).
%##### F holds in do(a,s) for some legal actions of group G #####
holds(exists_successor(G,F),S) :- !,
   member(P,G), agent_action(P, A),
   S1=do(A,S), legal(S1), holds(F,S1), !.
```

```
%##### F holds in do(a,s) for all legal actions of anti-group G #####
%##### (defined as negation of existential negative goal) #####
holds(forall_successors2(-G,F),S) :- !, not(holds(exists_successor2(-G,-F),S)).
%##### F holds in do(a,s) for some legal actions of anti-group G #####
holds(exists_successor2(-G,F),S) :- !,
    agent(P), not(member(P,G)), agent_action(P, A),
    S1=do(A,S), legal(S1), holds(F,S1), !.
%################################################
```

The $[[G]] \bigcirc \varphi[S]$ case is handled as $\neg\langle\langle G\rangle\rangle \bigcirc \neg\varphi[S]$. The μ operator is handled by using the same fixpoint approximates method as used in chapter 3, except that we put a limit on the number of expansions and we do not check for convergence – we simply check if the given situation S is in the successive approximates, i.e. we check if the approximate evaluates to $True$ for such S:

```
%################################################
% mu approximation
%################################################
%#####  binding diameter reached (as in bounded model checking) #####
mu_approx(Z,F,Int,N,S) :- binding_diameter(Max), N>Max, !,
    write('binding diameter '), write(N), write(' reached - stop'), nl, !, fail.
%#####  else: substitute and chech if it holds #####
mu_approx(Z,F,Int,N,S) :- subst(Z,Int,F,Fx), holds(Fx,S), !, output1(N,Fx).
%#####  else: do next approximation macro-expansion #####
mu_approx(Z,F,Int,N,S) :- M is N+1, subst(Z,Int,F,Int2), !, mu_approx(Z,F,Int2,M,S).
%################################################
```

Checking of equivalence on arbitrary First-Order formulas could be quite complex and therefore a simpler approach is taken. Looking at the form of the approximations and the definition of the operator $\bigcirc$, it can be observed that if any approximation evaluates to $True$ for S then the next approximation will also evaluate to $True$. Therefore the evaluation-based approach simply stops as soon as any approximation evaluates to $True$. The downside is that it may never stop or will not stop even if the formulas of consecutive approximations are equivalent but do not evaluate to $True$. One way to make sure the procedure stops is to put a limit on the number of approximations that would be somehow natural for the game that is modelled. The idea is similar to the binding diameter concept in bounded model checking (Biere [Bie09]). For some modelled domains there will be no solutions beyond their binding diameter and for others it will be just a practical safety mechanism to prevent infinite execution.

For example the operator $\langle\langle G\rangle\rangle\Diamond\varphi$ employs the μ operator over the "next" operator $\bigcirc$ as follows:

$Z_1 = \mathit{goal} \vee \langle\langle G\rangle\rangle \bigcirc False$, then compute $Z_1[S]$ and return success if it is $True$

$Z_2 = \mathit{goal} \vee \langle\langle G\rangle\rangle \bigcirc Z_1$, then compute $Z_2[S]$ and return success if it is $True$

...

$Z_{limit} = \mathit{goal} \vee \langle\langle G\rangle\rangle \bigcirc Z_{limit-1}$, then compute $Z_{limit}[S]$ and return success if it is $True$

otherwise report failure

The code for $\langle\langle G\rangle\rangle\Diamond\varphi$ corresponds to definition in alternating time μ-calculus [DLP10]:

```
%##############################################
% "ensure eventually" operator (in terms of mu and next)
%##############################################
holds(ensure-eventually(G,F),S) :- !, holds(mu(z,F v next(G,z)),S).
%##############################################
```

EXAMPLE:

The complete Prolog code for the axiomatization and the results of the tests for the Tic-Tac-Toe game (which is finite state) are given in the appendix A.2 and A.3. In the examples we verify the non-trivial property of the existence of the winning strategy. The Basic Action Theory axiomatization of the game [DLP10] provides the formula for the goal, the specification of the domain actions, and some other auxiliary and optional predicates. It defines the fluents, their successor-state axioms, and also the state in the initial situation S_0. These axioms are quite simple with the possible exception of *Legal* that captures the rules of the game of the player alternating moves:

```
%################################################
%##### the goal
wins(P,S) :- inline(C1,C2,C3), cell(P,C1,S), cell(P,C2,S), cell(P,C3,S), agent(P), legal(S).
%################################################
%##### holds if the situation is LEGAL (it can this situation be arrived at)
legal(s0) :- !.
legal(S) :- S = do(A,s0), agent_action(x,A), poss(A, s0), !. %%% X must play first
% alternate the actions: X's action after O's
legal(S) :- S = do(A1,S1), S1 = do(A2,S2), agent_action(x,A1), agent_action(o,A2),
    poss(A1,S1), poss(A2,S2),not(finished(S1)), !.
% alternate the actions: O's action after X's
legal(S) :- S = do(A1,S1), S1 = do(A2,S2), agent_action(o,A1), agent_action(x,A2),
    poss(A1,S1), poss(A2,S2),not(finished(S1)), !.
%################################################
%##### poss: possible (is it possible when given legal or not situation)
poss(mark(P,C),S) :- cell(b,C,S), agent(P), cell(C).
%################################################
```

```
%##### the initial situation - all cells in domain are b (blank)

cell(b,C,s0) :- cell(C).

%###############################################

%##### successor state axiom(s)

cell(M,C,do(A,S)) :- A=mark(M,C) ; cell(M,C,S), not(A=mark(_,C)).

%###############################################
```

The properties are verified by running Ł-Logic queries against the verifier and the Basic Action Domain axiomatization of the game domain. Here is an example to check for the existence of a winning strategy from a situation where some actions have already occurred:

```
?-holds(ensure-eventually([x],wins(x)),do(mark(x,2),do(mark(o,9),do(mark(x,1),do(mark(o,4),do(mark(x,5),s0)))))).
### XX_ ###
### OX_ ###
### __O ###
trying ##### approximation 1 ---> wins(x) v next([x], false)
trying ##### approximation 2 ---> wins(x) v next([x], wins(x) v next([x], false))
trying ---> next([x],wins(x) v next([x], false))
    ---> for S = do(mark(x, 2), do(mark(o, 9), do(mark(x, 1), do(mark(o, 4), do(mark(x, 5), s0)))))
trying ##### approximation 3 ---> wins(x) v next([x], wins(x) v next([x], wins(x) v next([x], false)))
trying ---> next([x],wins(x) v next([x], wins(x) v next([x], false))) ---> S=
   do(mark(x, 2), do(mark(o, 9), do(mark(x, 1), do(mark(o, 4), do(mark(x, 5), s0)))))
trying ---> next([x],wins(x) v next([x], false)) ---> S=
   do(mark(o, 3), do(mark(x, 2), do(mark(o, 9), do(mark(x, 1), do(mark(o, 4), do(mark(x, 5), s0))))))
> successor EXISTS for  G ---> next([x],wins(x) v next([x], false)) ---> S=
   do(mark(o, 3), do(mark(x, 2), do(mark(o, 9), do(mark(x, 1), do(mark(o, 4), do(mark(x, 5), s0))))))
trying ---> next([x],wins(x) v next([x], false)) ---> S=
   do(mark(o, 6), do(mark(x, 2), do(mark(o, 9), do(mark(x, 1), do(mark(o, 4), do(mark(x, 5), s0))))))
> successor EXISTS for  G ---> next([x],wins(x) v next([x], false)) ---> S=
   do(mark(o, 6), do(mark(x, 2), do(mark(o, 9), do(mark(x, 1), do(mark(o, 4), do(mark(x, 5), s0))))))
trying ---> next([x],wins(x) v next([x], false)) ---> S=
   do(mark(o, 7), do(mark(x, 2), do(mark(o, 9), do(mark(x, 1), do(mark(o, 4), do(mark(x, 5), s0))))))
> successor EXISTS for  G ---> next([x],wins(x) v next([x], false)) ---> S=
   do(mark(o, 7), do(mark(x, 2), do(mark(o, 9), do(mark(x, 1), do(mark(o, 4), do(mark(x, 5), s0))))))
trying ---> next([x],wins(x) v next([x], false)) ---> S=
   do(mark(o, 8), do(mark(x, 2), do(mark(o, 9), do(mark(x, 1), do(mark(o, 4), do(mark(x, 5), s0))))))
> successor EXISTS for  G ---> next([x],wins(x) v next([x], false)) ---> S=
   do(mark(o, 8), do(mark(x, 2), do(mark(o, 9), do(mark(x, 1), do(mark(o, 4), do(mark(x, 5), s0))))))
> successor FORALL for -G ---> next([x],wins(x) v next([x], wins(x) v next([x], false))) ---> S=
   do(mark(x, 2), do(mark(o, 9), do(mark(x, 1), do(mark(o, 4), do(mark(x, 5), s0)))))
> ##### approximation 3 holds --> wins(x) v next([x], wins(x) v next([x], wins(x) v next([x], false)))
```

Several situations have been tested for the existence of a winning strategy. The tests were done for 2 game scenarios (s0 to s5, and sw0 to sw6) with the objective of finding out at which point the game is not settled. There are two additional tests on an illegal situation and a game that cannot be won. In general the number of approximations represents the maximum number of moves before the game can be won. A failure is reported (if the limit was reached) when there is no strategy or the game cannot be won. The running time increases non-linearly as more and more approximations need to be considered. The results (the number of iterations and the running time in seconds or minutes) are:

```
s5           s4           s3           s2           s1           s0
### XOO ###  ### XOO ###  ### XO. ###  ### XO. ###  ### X.. ###  ### ... ###
### X.. ###  ### X.. ###  ### X.. ###  ### ... ###  ### ... ###  ### ... ###
### X.. ###  ### ... ###  ### ... ###  ### ... ###  ### ... ###  ### ... ###
1 appr 0s    2 appr 0s    5 appr 3s    6 appr 6s    failed 2m    failed 15m

for s2 can ensure winning and in at most 5 steps
for s0 can not ensure winning in up to 11 steps at which point the board is full thus there is no strategy

sw6          sw5          sw4          sw3          sw2          sw1          sw0
### XXX ### ### XX. ###   ### XX. ###  ### X.. ###  ### X.. ###  ### ... ###  ### ... ###
### OX. ###  ### OX. ###  ### OX. ###  ### OX. ###  ### OX. ###  ### OX. ###  ### .X. ###
### .OO ###  ### .OO ###  ### ..O ###  ### ..O ###  ### ... ###  ### ... ###  ### ... ###
1 appr 0s    2 appr 0s    3 appr 1s    4 appr 1s    5 appr 3s    6 appr 6s    failed 1m

the tests s0 to s5 are for progressive stages of a game that starts in the corner
the tests sw0 to sw6 are for progressive stages of a game that starts in the centre
there is no strategy for X to guarantee winning in sw0 but there is one in sw1 where O made a mistake

illegal      no-winning
### XX. ###  ### XX. ###
### ... ###  ### XO. ###
### ... ###  ### O.. ###
5 appr 1s    failed 0s
```

One of the tests is for an initial situation that is illegal with respect to the common rules of the game. Still from the point of view of the situation calculus axiomatization, this is a legal game with just a particular initial situation and this game has a winning strategy for player X. Also, another test is for an initial situation where player X may not be able to win if the opponent plays it intelligently. Here no approximation will evaluate to true and the program fails quickly as there are very few legal situations that result from that initial situation. □

We believe that our implementation is sound (assuming a proper Prolog interpreter is used). It is not complete, in part for the same reasons that Prolog is not a complete reasoner for FOL. We leave the proof of soundness for future work. It remains to be seen how well the verifier handles domains where the set of actions is infinite. Also the reasoner can only reason about concrete situations. Still any starting situation can be given as the initial situation to verify the property of interest in different circumstances and not only S_0. Also it is fully automatic and just requires a Basic Action Theory axiomatization of the game domain. We leave more experimental evaluation for future work.

5.2 Verifier for Basic Action Theory Game Structures with Constraints

The code for the evaluation-based fixpoint evaluator with constraints is listed in the appendix B.1.

Quite often certain agent actions are known or preferred in particular situations. This could be due to the strategy of the opponent being apparent or some well-known heuristics for the game. Capturing such preferences into the game theory would allow to model more realistic types of agents and to gain efficiency by cutting down on search. For the Tic-Tac-Toe game one might know that the opponent always tries to mark the corners first and this knowledge might be sufficient to determine if there is a winning strategy where there is none in the the general case. These "soft constraint", that is the choices of action that are taken if the actions are possible in a given situation, can cut down on the number of alternatives when the "next" operator $\bigcirc$ is used in temporal property verification. A mechanism is needed for the game modeller to supply state constraints in advance so that the reasoner program can use them during verification. These could be in the form of rules or decision trees but since player's preferred actions really depend on the current state (i.e. the values of the fluents) and not on the history (it does not matter how we arrived at the decision point), a function from situations to a set of actions appears appropriate to represent the constraints on the behaviours of the players. During computation of the "next" operator $\bigcirc$ if the set of preferred actions for the current situation is not empty then the legal actions of that set are used in the quantifiers, otherwise the operator behaves as usual considering all legal actions. Of course if the function returns an empty set or all of the actions then the effect is as if no constraining exists.

To implement soft constraints we propose to modify the "next" operator $\bigcirc$. We test for existence of any agent constraints first and then use them if they are present otherwise quantify over the actions in the "old" way. This applies to both the cases of the agents of coalition and agents of anti-coalition. The soft constraints should be axiomatized specific to the problem being modelled as a predicate $Preferred(p, a, s)$ that holds if an action a is a preferred action for player p in situation s. Then the new "next" operator $\bigcirc_b$ is derived from the standard operator $\bigcirc$ by introducing cases for when soft constraints are present and when they are not. The new "next" operator $\bigcirc_b$ is defined as follows:

$$
\begin{aligned}
&\langle\langle G\rangle\rangle \bigcirc_b \varphi \doteq \\
&\quad \{\exists agt \in G.\ Control(agt, now)\} \wedge \{\exists a.\ Preferred(agt, a, now) \wedge agent(a) = agt \wedge Legal(do(a, now))\} \wedge \\
&\qquad (\exists agt \in G.\ Control(agt, now) \wedge \exists a.\ Preferred(agt, a, now) \wedge agent(a) = agt \\
&\qquad\quad \wedge Legal(do(a, now)) \wedge \varphi[do(a, now)]) \vee \\
&\quad \{\exists agt \in G.\ Control(agt, now)\} \wedge \neg\{\exists a.\ Preferred(agt, a, now) \wedge agent(a) = agt \wedge Legal(do(a, now))\} \wedge \\
&\qquad (\exists agt \in G.\ Control(agt, now) \wedge \exists a.\ agent(a) = agt \wedge \\
&\qquad\quad \wedge Legal(do(a, now)) \wedge \varphi[do(a, now)]) \vee \\
&\quad \{\exists agt \notin G.\ Control(agt, now)\} \wedge \{\exists a.\ Preferred(agt, a, now) \wedge agent(a) = agt \wedge Legal(do(a, now))\} \wedge \\
&\qquad (\forall agt \notin G.\ Control(agt, now) \wedge \forall a.\ Preferred(agt, a, now) \wedge agent(a) = agt \wedge \\
&\qquad\quad Legal(do(a, now)) \supset \varphi[do(a, now)]) \vee \\
&\quad \{\exists agt \notin G.\ Control(agt, now)\} \wedge \neg\{\exists a.\ Preferred(agt, a, now) \wedge agent(a) = agt \wedge Legal(do(a, now))\} \wedge \\
&\qquad (\forall agt \notin G.\ Control(agt, now) \wedge \forall a.\ agent(a) = agt \wedge \\
&\qquad\quad Legal(do(a, now)) \supset \varphi[do(a, now)])
\end{aligned}
$$

It can be noticed that for the coalition agents if any preferred and legal actions exist then only the first conjunct will be a factor, otherwise there are no preferred legal actions and only the second conjunct is a factor. If the preferred actions include all actions then there is no difference between the first and the second conjunct. The reasoning is analogous for the agents of the anti-coalition.

The only difference between the code of the constrained verifier and the generic verifier is in the predicates performing the quantification – they have been duplicated to handle the cases of when soft constraints exist and when they do not. We can switch easily between the constrained and unconstrained version - we can simply define $Preferred$ to be always $false$ and the constrained version of the *exists_successor* predicate will always be ignored as it will always fail.

```
%##### F holds in do(a,s) for some PREFERRED actions of group G #####
holds(exists_successor(G,F),S) :-
    member(P,G), preferred(P,A,S), agent_action(P, A),
    S1=do(A,S), legal(S1), !, %% --> preferred action exists so commit to this version
    member(PX,G), preferred(PX,AX,S), agent_action(PX, AX),
    SX=do(AX,S), legal(SX), holds(F,SX), !.
%##### F holds in do(a,s) for some legal actions of group G #####
holds(exists_successor(G,F),S) :-
    member(P,G), agent_action(P, A),
    S1=do(A,S), legal(S1), holds(F,S1), !.
%##### F holds in do(a,s) for PREFERRED actions of anti-group G #####
holds(exists_successor2(-G,F),S) :-
    agent(P), not(member(P,G)), preferred(P,A,S), agent_action(P, A),
    S1=do(A,S), legal(S1), !,  %% --> preferred action exists so commit to this version
    agent(PX), not(member(PX,G)),
    preferred(PX,AX,S), agent_action(PX, AX),
    SX=do(AX,S), legal(SX), holds(F,SX), !.
%#### F holds in do(a,s) for some legal actions of anti-group G ####
holds(exists_successor2(-G,F),S) :-
    agent(P), not(member(P,G)), agent_action(P, A),
    S1=do(A,S), legal(S1), holds(F,S1), !.
```

EXAMPLE:

The complete Prolog code and the test results for the constrained axiomatization of the Tic-Tac-Toe game are given in the appendix B.2 and B.3. In the examples we verify the same non-trivial property of the existence of winning strategy. We now provide the axiomatization of the *Preferred* predicate in addition to the Basic Action Theory axiomatization of the game which remains unchanged. The axiomatization for the *Preferred* predicate for player O that prefers to grab corners early is:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% SOFT CONSTRAINTS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
preferred(P,A,S) :- P=o, cell(b,1,S), A=mark(P,1). %% grab corners
preferred(P,A,S) :- P=o, cell(b,3,S), A=mark(P,3). %% grab corners
preferred(P,A,S) :- P=o, cell(b,7,S), A=mark(P,7). %% grab corners
preferred(P,A,S) :- P=o, cell(b,9,S), A=mark(P,9). %% grab corners
```

The properties are verified by running the same Ł-Logic queries as for the unconstrained version against the verifier and the extended Basic Action Domain axiomatization of the game domain. Here is an example to check for the existence of a winning strategy from a situation where some actions already occurred. Here it is X's turn and O prefers to grab available corners:

```
  ?-holds(ensure([x],wins(x)),do(mark(o,9),do(mark(x,1),do(mark(o,4),do(mark(x,5),s0))))).
  ### X.. ###
  ### OX. ###
  ### ..O ###
  trying ##### approximation 1 ---> wins(x) v next([x], false)
  trying ##### approximation 2 ---> wins(x) v next([x], wins(x) v next([x], false))
  next([x],wins(x) v next([x], false)) ---> for S =
    do(mark(o, 9), do(mark(x, 1), do(mark(o, 4), do(mark(x, 5), s0))))
  trying ##### approximation 3 --->
    wins(x) v next([x], wins(x) v next([x], wins(x) v next([x], false)))
  next([x],wins(x) v next([x], wins(x) v next([x], false))) ---> for S =
    do(mark(o, 9), do(mark(x, 1), do(mark(o, 4), do(mark(x, 5), s0))))
  next([x],wins(x) v next([x], false)) ---> for S =
    do(mark(x, 2), do(mark(o, 9), do(mark(x, 1), do(mark(o, 4), do(mark(x, 5), s0)))))
  next([x],wins(x) v next([x], false)) ---> for S =
    do(mark(x, 3), do(mark(o, 9), do(mark(x, 1), do(mark(o, 4), do(mark(x, 5), s0)))))
```

```
next([x],wins(x) v next([x], false)) ---> for S =
  do(mark(x, 6), do(mark(o, 9), do(mark(x, 1), do(mark(o, 4), do(mark(x, 5), s0)))))
next([x],wins(x) v next([x], false)) ---> for S =
  do(mark(x, 7), do(mark(o, 9), do(mark(x, 1), do(mark(o, 4), do(mark(x, 5), s0)))))
next([x],wins(x) v next([x], false)) ---> for S =
  do(mark(x, 8), do(mark(o, 9), do(mark(x, 1), do(mark(o, 4), do(mark(x, 5), s0)))))
trying ##### approximation 4 --->
  wins(x) v next([x], wins(x) v next([x], wins(x) v next([x], wins(x) v next([x], false))))
next([x],wins(x) v next([x], wins(x) v next([x], wins(x) v next([x], false)))) ---> for S =
  do(mark(o, 9), do(mark(x, 1), do(mark(o, 4), do(mark(x, 5), s0))))
next([x],wins(x) v next([x], wins(x) v next([x], false))) ---> for S =
  do(mark(x, 2), do(mark(o, 9), do(mark(x, 1), do(mark(o, 4), do(mark(x, 5), s0)))))
next([x],wins(x) v next([x], false)) ---> for S =
  do(mark(o, 3), do(mark(x, 2), do(mark(o, 9), do(mark(x, 1), do(mark(o, 4), do(mark(x, 5), s0))))))
next([x],wins(x) v next([x], false)) ---> for S =
  do(mark(o, 7), do(mark(x, 2), do(mark(o, 9), do(mark(x, 1), do(mark(o, 4), do(mark(x, 5), s0))))))
> ##### approximation 4 holds --->
  wins(x) v next([x], wins(x) v next([x], wins(x) v next([x], wins(x) v next([x], false))))
```

In the above example the verification completes after 4 expansions, that is, X can ensure winning in up to 3 moves providing O follows the "grab corners first" strategy. Indeed the next move is X that cannot win in just one move, the next move is O who will mark upper right or bottom left corner after that X can complete the game.

The same situations as for the non-constrained verifier have been tested for the existence of the winning strategy. In general the running time decreases significantly and some winning strategies now exist where they did not in the non-constrained version of the game. The results (the number of iterations and the running time in seconds or minutes) are:

```
s5            s4            s3            s2            s1            s0
### XOO ###   ### XOO ###   ### XO. ###   ### XO. ###   ### X.. ###   ### ... ###
### X.. ###   ### X.. ###   ### X.. ###   ### ... ###   ### ... ###   ### ... ###
### X.. ###   ### ... ###   ### ... ###   ### ... ###   ### ... ###   ### ... ###
1 appr 0s     2 appr 0s     5 appr 1s     4 appr 1s     5 appr 2s     6 appr 4s

sw6           sw5           sw4           sw3           sw2           sw1           sw0
### XXX ###  ### XX. ###    ### XX. ###   ### X.. ###   ### X.. ###   ### ... ###   ### ... ###
### OX. ###   ### OX. ###   ### OX. ###   ### OX. ###   ### OX. ###   ### OX. ###   ### .X. ###
### .OO ###   ### .OO ###   ### ..O ###   ### ..O ###   ### ... ###   ### ... ###   ### ... ###
1 appr 0s     2 appr 0s     3 appr 0s     4 appr 0s     5 appr 1s     4 appr 1s     5 appr 2s

illegal       no-winning
### XX. ###   ### XX. ###
### ... ###   ### XO. ###
### ... ###   ### O.. ###
5 appr 1s     failed 2s
```

Player X has strategies to win in several situations since the opponent will not be taking the required counter-moves. $\square$

We believe that the verifier with soft constraints works – it allows to verify temporal properties in new circumstances and can be also significantly faster that the generic version. We believe that the axiomatization of the user preferences is quite straightforward and the encoding should not be difficult. Still the constrained verifier inherits its other advantages and disadvantages from the generic verifier presented in the previous section. We leave more experimental evaluation for future work.

5.3 Verifier for GameGolog-Expressed Game Structures

The code for the evaluation-based fixpoint evaluator for GameGolog-expressed game structures is listed in the appendix C.1.

The technique presented in this section builds on the evaluator from section 5.1. An implementation in Prolog has been developed that operates on game structures expressed in GameGolog to prove the feasibility of such approach. Since a GameGolog program represents the structure of the game, the main idea was to employ it to define what is legal in the game. The new verifier is very similar to the one described in section 5.1 and the key difference is that now the game structure can be modelled in a more procedural and therefore more natural way. The GameGolog definition of the game is used in the *Legal* predicate that occurs in the "next" operator $\bigcirc$ when verifying properties. In essence the implementation shows the feasibility of GameGolog to express game structures in proving game properties. The developed verifier understands the syntax and semantics of GameGolog and operates on GameGolog structures to mimic their execution in order to verify properties of games. As stipulated by [DLP10] the choices in the non-deterministic actions are recorded in the situation during program execution. This way any situation during GameGolog program execution allows us to exactly recreate the execution path. In our implementation we handle the axiomatization of *Legal* as a star-composition of GameGolog single-step transitions as defined in [DLP10] and the background section of this thesis. The legality of a situation is determined by the fact that it can be reached from S_0 by execution a number of steps of a GameGolog program ρ_0:

$$Legal_{GG}(s) \equiv \exists \rho'.Trans^*(\rho_0, S_0, \rho', s)$$

The verifier has been developed in Prolog and tested on the classic Tic-Tac-Toe game theory expressed in the Situation Calculus using Prolog clauses. The code of the verifier is essentially the same as the generic verifier from section 5.1 with the addition of the GameGolog single-step semantics of $trans$ and $final$ as explained in the background section. The verifier still requires the basic action theory model for the verified game with the $Legal$ predicate being defined in terms of the GameGolog program of the game and the star-composition of $trans$. In [DLP10] $trans$ is built into the "next" operator $\bigcirc$ for improved efficiency, while here we only try to extend the basic verifier by replacing the way the legality of the situation is defined using GameGolog. In the future we plan to improve the verifier and implement the "next" operator $\bigcirc$ as defined in [DLP10] and to compare the efficiencies of the evaluation. Here is the code for the $trans$, $final$, the star-composition of $trans$:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% GAMEGOLOG SEMANTICS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% similar to Golog semantcs
%% parameter U specifies which agent controls nondeterministic actions in situation S
%% parameter UX specifies which agent controls nondeterministic actions in situation SX
:- op(200, fy, [?]).     %% operator for domain
:- op(200, xfy, [of]).   %% operator for domain
:- op(210, xfy, [:]).    %% operator for sequence of actions
:- op(220, xfy, [@@]).   %% operator for nd choice of action
trans(E,S,SX) :- trans(_U,E,S,_UX,_EX,SX).
trans(U,?P,SX,U,_EX,SX) :- !, fail. %% test
trans(U,?true : E,S,UX,EX,SX) :- !, trans(U,E,S,UX,EX,SX). %% performance improvement

trans(U,E1 : E2,S,UX,EX,SX) :- trans(U,E1,S,UX,EX1,SX), EX = (EX1 : E2). %% action sequence

trans(U,E1 : E2,S,UX,EX,SX) :- final(E1,S), trans(U,E2,S,UX,EX,SX).  %% action sequence

trans(U,if(P,E1,E2),S,UX,EX,SX) :- holds(P,S), trans(U,E1,S,UX,EX,SX). %% if then else

trans(U,if(P,E1,E2),S,UX,EX,SX) :- not(holds(P,S)), trans(U,E2,S,UX,EX,SX). %% if then else

trans(U,while(P,E),S,UX,EX,SX):-%% while

holds(P,S), trans(U,E,S,UX,EX1,SX), EX = (EX1 : while(P,E)).

trans(U,E1 @@ E2,S,U,E1,do(left(U),S)). %% nd branch

trans(U,E1 @@ E2,S,U,E2,do(right(U),S)). %% nd branch

trans(U,pi(V of D,E),S,U,EX,SX) :- DP=..[D,VX], DP, poss(pick(U,VX),S),

SX = do(pick(U,VX),S), subst(V,VX,E,EX). %% nd choice of argument

trans(U,star(E),S,U,E : star(E),do(continue(U),S)). %% star

trans(U,star(E),S,U,?true,do(stop(U),S)). %% star

trans(_U,uc(UX1,E),S,UX,EX,SX) :- trans(UX1,E,S,UX,EX,SX). %% under control

trans(U,A,S,U,?true,do(A,S)) :- primitive_action(A),poss(A,S). %% primitive action

final(E,S) :- primitive_action(E),!,fail. %% primitive action

final(?P,S) :- holds(P,S). %% test

final(E1 : E2,S) :- final(E1,S), final(E2,S). %% sequence

final(if(P,E1,E2),S) :- holds(P,S), final(E1,S). %% if

final(if(P,E1,E2),S) :- not(holds(P,S)), final(E2,S). %% if

final(while(P,E),S) :- holds(P,S), final(E,S) ; not(holds(P,S)). %% while

final(E1 @@ E2,S) :- !, fail. %% nd branch - GG version

final(pi(V of D,E),S) :- !, fail. %% nd choice of argument - GG version

final(star(E),S) :- !, fail. %% star - GG version

% trans* (star composition of trans)

transs(U,E,s0,UX,EX,do(A,s0)) :- trans(U,E,s0,UX,EX,do(A,s0)).
transs(U,E,S,UX,EX,do(A,S1)) :- transs(U,E,S,U1,E1,S1), trans(U1,E1,S1,UX,EX,do(A,S1)).
```

The game axiomatization must define its GameGolog program and the *Legal* predicate in terms of this GameGolog program and the star-composition of *trans*. It also needs to aximatize the additional actions for recording non-deterministic choices. Such actual aximatization is shown in the example below.

Please note that in the verifier the implementation of the non-deterministic choice of argument actions requires the enumeration of actions (for quantification). Our approach was to extend the definition of the *pick* action to require the specification of the domain of the choice. Also please note that the ability to constrain the actions of players exists in this approach as well by simply following the method from section 5.2. Alternatively the strategy could be given as another GameGolog program that can be run on the current situation. Once it stops we could see what the next possible actions are and use them as the preferred actions. We leave this for future work.

EXAMPLE:

The complete Prolog code and the test results for the GameGolog-expressed Tic-Tac-Toe game are given in the appendix C.2 and C.3. In the examples we verify the same non-trivial property of the existence of winning strategy. We provide the axiomatization of the *Legal* predicate in terms of star-composition of $Trans$ predicates and the GameGolog code of the game. The rest of the Basic Action Theory axiomatization of the game remains unchanged with the addition of axiomatization for the new actions required to encode the non-deterministic choices. The axiomatization for the new actions, the *Legal* predicate, and the GameGolog program for Tic-Tac-Toe are:

```
%##### domain of actions #####

primitive_action(pick(P,C)) :- agent(P), cell(C).

primitive_action(stop(P)) :- agent(P).

primitive_action(continue(P)) :- agent(P).
```

```
%##### poss: possible  #####

poss(pick(P,C),S) :- cell(b,C,S), agent(P), cell(C).

poss(stop(P),S).

poss(continue(P),S).

%##### agent_action - determine the agent of an action #####

agent_action(P,A) :-  A = pick(P,C), agent(P), cell(C).

agent_action(P,A) :-  A = stop(P), agent(P).

agent_action(P,A) :-  A = continue(P), agent(P).

%##### definition LEGAL using trans* and GameGolog program #####

legal(S) :- proc(ttt,R0), transs(_U,R0,s0,_UX,_RX,S).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% GameGolog (note: all bound variables must have unique names)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

proc(ttt,

    while(-finished,

        uc(x, pi(c1 of cell, mark(x,c1))) :

        if(-finished,

            uc(o, pi(c2 of cell, mark(o,c2))),

            ? true)

        )

    ).
```

The properties are verified by running the same Ł-Logic queries as for the general version of the verifier against the extended Basic Action Theory axiomatization of the game domain. Here is an example to check for the existence of a winning strategy from a situation where some actions already occurred:

```
  ?-holds(ensure([x],wins(x)),
     do(mark(o,3),do(pick(o,3),do(mark(x,4),do(pick(x,4),
        do(mark(o,2),do(pick(o,2),do(mark(x,1),do(pick(x,1),s0))))))))).
  ### XOO ###
  ### X.. ###
  ### ... ###
  trying ##### approximation 1 ---> wins(x) v next([x], false)
  trying ##### approximation 2 ---> wins(x) v next([x], wins(x) v next([x], false))
  trying ---> next([x],wins(x) v next([x], false)) ---> for S=
     do(mark(o, 3), do(pick(o, 3), do(mark(x, 4), do(pick(x, 4), do(mark(o, 2),
        do(pick(o, 2), do(mark(x, 1), do(pick(x, 1), s0))))))))
  trying ##### approximation 3 ---> wins(x) v next([x], wins(x) v next([x], wins(x) v next([x], false)))
  trying ---> next([x],wins(x) v next([x], wins(x) v next([x], false))) ---> for S=
     do(mark(o, 3), do(pick(o, 3), do(mark(x, 4), do(pick(x, 4), do(mark(o, 2), do(pick(o, 2),
        do(mark(x, 1), do(pick(x, 1), s0))))))))
  trying ---> next([x],wins(x) v next([x], false)) ---> for S=
     do(pick(x, 5), do(mark(o, 3), do(pick(o, 3), do(mark(x, 4), do(pick(x, 4), do(mark(o, 2),
        do(pick(o, 2), do(mark(x, 1), do(pick(x, 1), s0)))))))))
  trying ---> next([x],wins(x) v next([x], false)) ---> for S=
     do(pick(x, 6), do(mark(o, 3), do(pick(o, 3), do(mark(x, 4), do(pick(x, 4), do(mark(o, 2),
        do(pick(o, 2), do(mark(x, 1), do(pick(x, 1), s0)))))))))
  trying ---> next([x],wins(x) v next([x], false)) ---> for S=
     do(pick(x, 7), do(mark(o, 3), do(pick(o, 3), do(mark(x, 4), do(pick(x, 4), do(mark(o, 2),
        do(pick(o, 2), do(mark(x, 1), do(pick(x, 1), s0)))))))))
  > successor EXISTS for  G ---> next([x],wins(x) v next([x], false)) ---> for S=
     do(pick(x, 7), do(mark(o, 3), do(pick(o, 3), do(mark(x, 4), do(pick(x, 4), do(mark(o, 2),
        do(pick(o, 2), do(mark(x, 1), do(pick(x, 1), s0)))))))))
  > successor EXISTS for  G ---> next([x],wins(x) v next([x], wins(x) v next([x], false))) ---> for S=
     do(mark(o, 3), do(pick(o, 3), do(mark(x, 4), do(pick(x, 4), do(mark(o, 2),
        do(pick(o, 2), do(mark(x, 1), do(pick(x, 1), s0))))))))
  > ##### approximation 3 holds ---> wins(x) v next([x], wins(x) v next([x], wins(x) v next([x], false)))
```

The same situations as for the generic verifier have been tested for the existence of the winning strategy. In general the running time increases significantly and some verifications took several hours to fail. Additionally the situation terms are now double the original size as each agent action is preceded with the pick action of the non-deterministic choice of argument, as well as, test situations require explicit enumeration of the new actions that encode the non-deterministic choices. We also needed to increase the binding diameter constant as the situations are now longer. The results (the number of iterations and the running time in seconds or minutes) are:

```
  s5            s4            s3            s2            s1            s0
  ### XOO ###   ### XOO ###   ### XO. ###   ### XO. ###   ### X.. ###   ### ... ###
  ### X.. ###   ### X.. ###   ### X.. ###   ### ... ###   ### ... ###   ### ... ###
  ### X.. ###   ### ... ###   ### ... ###   ### ... ###   ### ... ###   ### ... ###
  1 appr 0s     3 appr 7s     9 appr 10m    11 appr 36m   failed v.long failed v.long

  sw6           sw5           sw4           sw3           sw2           sw1           sw0
  ### XXX ###  ### XX. ###    ### XX. ###   ### X.. ###   ### X.. ###   ### ... ###   ### ... ###
  ### OX. ###   ### OX. ###   ### OX. ###   ### OX. ###   ### OX. ###   ### OX. ###   ### .X. ###
  ### .OO ###   ### .OO ###   ### ..O ###   ### ..O ###   ### ... ###   ### ... ###   ### ... ###
  1 appr 0s     3 appr 5s     5 appr 42s    7 appr 2m     9 appr 13m    11 appr 34m   failed v.long

  illegal       no-winning
```

```
### XX. ###   ### XX. ###
### ... ###   ### XO. ###
### ... ###   ### O.. ###
failed 10s    failed 2.5m
```

□

We believe that the implementation is sound with respect to the GameGolog semantics (assuming a proper Prolog interpreter is used) but we leave the proof of soundness for future work. The GameGolog-constrained verifier works but inherits its advantages and disadvantages from the generic verifier presented in section 5.1. It appears significantly slower and the requirement to record the non-deterministic choices in the situation makes the situation terms double their size with respect to the generic versions. We believe that building *trans* into the evaluation of the "next" operator as in [DLP10] should improve efficiency. We leave more experimental evaluation (e.g. on infinite domain games) for future work.

5.4 Discussion

In this thesis we implemented and tested an automated verifier for Basic Action Theory game structures. It performs evaluation-based verification of properties of game theories as per De Giacomo, Lespérance and Pearce [DLP10]. The main point was to develop a technique that can automatically check ATL-type game-theoretic properties. The evaluation-based fixpoint approximation technique for verification has been implemented in Prolog. This implementation has been tested on the finite-state Tic-Tac-Toe game theory expressed in the situation calculus and coded in Prolog. The verification is performed for the non-trivial property of the existence of winning strategy.

Additionally a variant of the technique where constraints on actions of players can be specified has been proposed, implemented and tested. Quite often it is known that some actions are known or preferred in certain situations whether for the players of the coalition or the anti-coalition. Therefore a mechanism has been developed for the game modeller to supply strategy or action constraints that are known in advance, so that the developed reasoner program can use them where non-deterministic choices of action are made. During computation of the operator $\bigcirc$ if the set for the current situation is not empty then it is used in the quantifiers, otherwise the operator behaves as usual with no restrictions during quantification. These soft constraint, that is the choices of action that are taken if the actions are possible in a given situation, help the efficiency (for coalition players) or specify opponent's strategy (if such is known) and cut down the quantification when the operator $\bigcirc$ is used. This behaviour may represent a strategy or level of expertise of the players. The constraints are supplied as a function from the current situation to a set of actions.

Finally a variant of the technique where the game structure is given in the form of game's GameGolog program has been proposed, implemented and tested. Since a GameGolog program represents the structure of the game, it was used to define what is legal in the game. The key difference is that now the game structure can be modelled in a more procedural and therefore more natural way. In essence the implementation developed to show the feasibility of GameGolog to express game structures in proving game properties is similar to the implementation proposed at the beginning of the section with the difference that the *Legal* predicate is replaced with one that utilizes GameGolog single-step semantics as defined in the background section of this thesis. In essence the developed evaluator understands the syntax and semantics of GameGolog and operates on GameGolog structures to mimic their execution in order to verify properties of games. The verifier still requires the basic action theory model for the verified game with the exception of the *Legal* predicate. Also the ability to constrain the actions of players is also supported in this approach.

All the automatic methods follow the same principle as the symbolic manipulation method - they try to verify that a formula in L language expressing the ATL-type game property holds. They employ the same algorithm for which the key point is checking of convergence of the fixpoint, that is, if two consecutive approximations are equivalent. Such test of equivalence on arbitrary formulas could be quite complex therefore a simpler approach is taken - we check if any of the fixpoint expansions evaluate to true for the initial or a given situation. The techniques may never stop even though the two consecutive approximations are equivalent but do not evaluate to true. To deal with this, we have a limit on the number of approximations that would be somehow natural for the game that is modelled and for which properties are verified. Here the term "evaluation-based" refers to the use of evaluation instead of entailment to check state properties under the conditions of complete theory (i.e. single model) and closed-world assumption. In the future work we plan to test the technique on infinite domains and domains with infinitely many actions.

6 Conclusions and Future Work

This section provides a summary of the research conducted for this thesis. First a brief summary is provided, next a brief recapitulation of the area of work, the problem, and the approach is given. Following this, we discuss the thesis's contributions. Finally an outline of possible future work that naturally arises from the research in this thesis is given.

6.1 Conclusion

In brief, the work presented in this thesis is intended to be a demonstration that the techniques introduced in De Giacomo, Lespérance and Pearce [DLP10] do work on verification problems with infinite state space. The case studies performed indeed support the claim that this approach to verifying infinite state systems actually works. The presented example problems have varying types of properties that, although not exhaustive and complete, do allow us to evaluate if the techniques work on many interesting problems. Another part of the thesis examines the use of characteristic graphs (Claßen and Lakemeyer [CL08]) in verifying properties of games. Additionally the thesis proposes a refinement of the formalism to incorporate player strategy/behaviour constraints (for example if the strategy of some of the players is known) and to subsequently to perform temporal property verifications where such constraints may be given. Also, an implementation of an evaluation-based verifier has been developed and employed to verify some temporal properties of the classic tic-tac-toe game.

The research performed in this thesis is focused on problems that can be modelled as games. Many problems

– such as multi-agent games, contingent planning, multi-agent planning and process orchestration – can be viewed as games where agents try to achieve certain objectives or make sure certain conditions hold no matter how other agents behave. There has been some recent work on formalisms for specifying such game structures and for specifying temporal properties that coalitions of agents can ensure in games. This work is mostly based on Alternating-Time Temporal Logic (ATL), ATL* and the alternating-time mu-calculus (AMC) (Alur, Henzinger, and Kupferman [AHK02]). Although they provide elegant and expressive languages for properties that one might want to verify, such logics do not address how to specify a model of a game structure over which the property is to be verified or how to specify strategies for agents to follow. The work in De Giacomo, Lespérance and Pearce [DLP10] devises a formalism that can express the temporal properties and the game structures using the same foundation based on a situation calculus axiomatization which is very natural for this type of problems. It allows reasoning about game structures and multi-agent interaction problems that require strategic thinking, not just games in conventional sense. It puts a focus on verification and off-line synthesis, does not use probabilities and utilities, and provides a logical framework for specifying game theoretic problems in natural way in a language that combines declarative and procedural elements. This thesis is based on this research.

Expressing the ATL temporal "group G ensures next φ" property (denoted $\langle\langle G \rangle\rangle \bigcirc \varphi$) as a formula in model checking can be difficult. The results of model checking technology are quite low level and the technology is restricted to finite states structures. It does not combine declarative and procedural elements – while for many problems an action programming language would provide a natural specification language. Also it is best if the property to check can be expressed in the same formalism as the problem, as the danger of loosing something in translation and the risk of errors are reduced. De Giacomo, Lespérance and Pearce [DLP10] do just that, but emphasize that their study is essentially theoretical and they predict that effectiveness guarantees will only be available for very specific cases. They called for complementing their work with experimental studies to understand whether these techniques, especially those based on the labelling of characteristic graphs, are effective in practical cases. Also so far little work has been done for infinite domains with the exception Claßen and Lakemeyer's work [CL08].

This thesis addresses the questions mentioned above. It evaluates the new techniques introduced by De Giacomo, Lespérance and Pearce [DLP10] on infinite domains. Based on the assumptions that all agents have complete knowledge of the theory, that actions are observable by all agents, and that there are no sensing actions that allow agents to gain additional private knowledge, this thesis follows their logical framework for solving game theoretic problems. The framework supports incomplete specifications of the application domain – the basic action theories do not need to have a single model. The framework also supports infinite state settings. Their framework includes a new language GameGolog that is based on ConGolog (De Giacomo, Lespérance, and Levesque [DLL00]) to specify precisely which agent can take action in any game situation and what actions it can perform (a program in GameGolog can clearly and conveniently specify the game structure) and uses the background situation calculus action theory. Their framework also proposes a rich language for specifying temporal properties for game structures. This language uses first-order quantification, mu-calculus, and game-theoretic path quantifiers. Several techniques for verifying such temporal properties are proposed , as well. These techniques are: symbolic fixpoint approximation, labelling on characteristic graphs (Claßen and Lakemeyer [CL08]), and the suggestion of an implementation that can automatically perform a fixpoint approximation on finite state game structures. In the paper they also discuss how constraints (for example forms of fairness) can be expressed and used in verifying properties. The approach taken by this thesis was to develop some examples and test the algorithms introduced in De Giacomo, Lespérance and Pearce [DLP10] on them.

Several game domains have been developed that are quite representative of the type of problem that the technique is trying to address. Although the domains are rather simple, they have features present in practical examples, and they do allow us to evaluate the method. Our experiments do confirm that the method does work on several verification problems with infinite state space. We also identify some examples where the method, which only uses the simplest part of the domain theory, the unique names and domain closure for action axioms, fails to converge in a finite number of steps. We show that in some of these cases, extending the method to use some selected facts about the initial situation and some state constraints does allow us to get convergence in a finite number of steps. Finally, our example domains and properties should be useful for evaluating other approaches to infinite state verification and synthesis.

We also evaluated the version of the [DLP10] verification method that uses characteristic graphs. In this thesis we use graphs constructed by hand and the labellings are done by hand. It was observed that the symbolic manipulation and characteristic graph labelling gave essentially analogous results for the corresponding game structure verification.

The implementation of the evaluation-based fixpoint approximation is the most practical component of the work presented in this thesis. It has been done in Prolog and it does handle game settings. Many examples are provided and discussed. The Prolog implementation can be used to show whether a temporal property is true in the initial or any given situation. The implementation can be given an upper limit on the number of fixpoint expansions and this is for practical reasons since there is no guarantee the method will terminate in general. Unlike the symbolic manipulation technique, this method works only on domains with a finite number of actions and with complete theory for the initial situation (it checks whether the initial situation satisfies the formulas produced by fixpoint iterations). The implemented program has been refined to incorporate preferred actions. This allows us to incorporate constraints on the agent strategies in playing the game and to evaluate the temporal properties of a game structure when such constraints are present. These constraints can also be a mechanism for the modeller to supply heuristic guidance so that the reasoner can use them to prove the fixpoint convergence in a more efficient and more practical way. The constraints could be in a form of relations, decision trees, rules, or some other constructs. The work done for this thesis additionally incorporates another implementation, also done in Prolog that handles a GameGolog program specifying the game structure. GameGolog often proves convenient in modelling game problems; it is a nice tool to express complex games. In essence a game structure expressed as a GameGolog program is submitted to the property verifier.

The methods researched in this thesis work for infinite domains where very few methods are available in verifying temporal properties. Although it has been shown that model checking techniques can be used to verify that temporal properties hold in game structures and these techniques can be used to synthesize strategies for agents in a coalition to ensure that properties hold, model checking approaches only work for finite domains. Additionally, we tried to reason about all situations (without the theory for initial situations) and we think that a finite state approach to perform similar analysis does not currently exist.

6.2 Future Work

The topic of this thesis is a very interesting area and there are a lot of questions that one could ask such as what is practical and what is not, or is there a way to obtain winning strategies for temporal properties that can be verified. A few questions and areas of future continued work that arose from the research in this thesis and some very high level ideas are presented below.

One thing that could be done would be to research the possibility to automate the symbolic fixpoint approximation that for now are done manually due to the complexity of the calculus that needs to be used. This would require some automated symbolic manipulation techniques for regression, simplification of the resulting formulas, and checking if two subsequent formulas are equivalent. One more idea here would be to employ stochastic satisfiability methods to do so. Also, one could investigate if some heuristic strategy constraints and move guidance can be incorporated into the approximation computation to improve the efficiency of the computation and to make the computation more practical.

Another idea would be to implement the technique for characteristic graph labelling - for a given characteristic graph, GameGolog program of the game, the axiomatization, and the properties to be verified perform automatic iterative labelling and check for convergence. During manual computation for the characteristic graphs it was observed that the proofs in some cases were relatively straightforward and I think that it should be possible to use a theorem prover with the right kind of strategy specified to do these problems in practical way. It was also observed that for the examples chosen the fixpoint approximation formulas could be simplified significantly at each step and the resulting formula was not growing exponentially. It would be interesting to find out the types of problems that have this property as the verification of temporal properties would be faster whereas the state space grows exponentially in the number of moves. Also, for simple games where the program goes through 2 alternating moves, using symbolic or characteristic graph method did not make a big difference in volume of computation. Here the characteristic graph and symbolic approach seem to be doing the same thing. It would be interesting to look at slightly more complex problems where there are more than 2 nodes and see if there is a relation in the amount of computation and if characteristic graph approach provides some benefits during the verification. It would be interesting to see what happens on games with many nodes and a more interesting structure (for example Wumpus World or Dungeons and Dragons) and see if in some case it simplifies the proof.

Yet another set of ideas relates to the evaluation-based verification. The implementation could be reviewed to perform equivalence or implication checking between consecutive iterations of the fixpoint approximation. Currently it is just checked that the formula evaluates to true. This would truly capture the potential of the technique and produce the converged formula that could be then used to examine the conditions for the existence of a checked property. Since the implementation allows for constraints and guidance of player moves, another idea would be to see what are the particular game heuristics that can make the technique more efficient and more practical. With regards to the GameGolog version of the evaluator - the GameGolog program that alternates between players could be split into 2 concurrent GameGolog programs for each player where each is concerned with the behaviour of just that one player. On top of it the soft constraint on the player moves could be also expressed as a GameGolog program which takes priority over the regular player program. This approach could be examined if it works and if it brings any benefits.

One can also examine how to extract the conditions and the strategy to achieve the desired temporal properties based on the convergence of the techniques. It appears that the converged formulas provide the way to extract the strategy at the verification stage. Since each iteration is a regression step from the goal, the number of approximations before convergence is the most number of actions in which the property can be achieved. If there is a convergence (especially to true) then the formula gives us the conditions on all possible situations for which the verified property exists - this can give us the strategy to achieve such property. In case of winning a game, when playing the actual game if the actions are decided that the resulting situation will make the convergence formula true then it is a non-loosing strategy (there is a way to ensure the win after the move). This may still be a non-relevant action but if I can satisfy a formula from an earlier iteration of approximation then it is a true strategy - as the number of steps to achieve the property lowers. In general some future work would be to analyze if indeed one can devise general methods for conditions and strategies for achieving properties and to implement it.

For practical purposes the framework proposed in De Giacomo, Lespérance and Pearce [DLP10] could also be enhanced to allow probabilistic and utility measures to perform mini-max prioritization of actions. One could also try to employ the research techniques for on-line game playing where there is no strategy in the initial situation but as the game is played we want to make sure we make the moves or observe the opponent making mistakes that would put us in a situation for which some the fixpoint approximation holds and now we can ensure the win. There are several issues related to on-line playing that are not considered here such as making sure that we do not take an action that ensures that the opponent can win, we need to worry about efficiency and speed of the decision making, and the ability to use heuristics.

Bibliography

[AHK02] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.

[Ake78] S. B. Akers. Binary Decision Diagrams. *IEEE Trans. Comput.*, 27(6):509–516, June 1978.

[BBS09] Paul P. Boca, Jonathan P. Bowen, and Jawed I. Siddiqi. *Formal Methods: State of the Art and New Directions*. Springer, first edition, September 2009.

[Bie09] Armin Biere. Bounded Model Checking. In *Handbook of Satisfiability*, pages 457–481. 2009.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[BKM95] Robert S. Boyer, Matt Kaufmann, and J Strother Moore. The Boyer-Moore Theorem Prover and Its Interactive Enhancement, 1995.

[BS07] Julien Bradfield and Colin Stirling. Modal mu-calculi. In *Handbook of Modal Logic*, volume 3, pages 721–756. Elsevier, 2007.

[CABN97] William Chan, Richard Anderson, Paul Beame, and David Notkin. Combining constraint solving and symbolic model checking for a class of systems with non-linear constraints. In *In Computer Aided Verification*, pages 316–327. Springer-Verlag, 1997.

[CCGR00] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NUSMV: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2:2000, 2000.

[CE81] Edmund M. Clarke and E. Allen Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Logic of Programs*, pages 52–71, 1981.

[CGP01] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model checking.* MIT Press, 2001.

[CL08] Jens Claßen and Gerhard Lakemeyer. A Logic for Non-Terminating Golog Programs. In *Proc. of KR'08*, pages 589–599, 2008.

[DLL00] Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. ConGolog, A Concurrent Programming Language Based on the Situation Calculus. *AIJ*, 121(1–2):109–169, 2000.

[DLP10] G. De Giacomo, Y. Lespérance, and A. R. Pearce. Situation Calculus-based Programs for Representing and Reasoning about Game Structures. 2010.

[Eme96] E. Allen Emerson. Model Checking and the Mu-calculus. In *Descriptive Complexity and Finite Models*, pages 185–214, 1996.

[End72] Herbert B. Enderton. *A mathematical introduction to logic.* Academic Press, 1972.

[GLP05] Michael R. Genesereth, Nathaniel Love, and Barney Pell. General Game Playing: Overview of the AAAI Competition. *AI Magazine*, 26(2):62–72, 2005.

[Ham82] A.G. Hamilton. *Numbers, Sets and Axioms: The Apparatus of Mathematics.* Cambridge, 1982.

[Hav99] Klaus Havelund. Java PathFinder User Guide. *NASA Ames Research*, 1999.

[Hoa69] C. A. R. Hoare. An Axiomatic Basis for Computer Programming. *ACM*, 12(10):576–580, 1969.

[Hol03] Gerard Holzmann. *Spin model checker, the: primer and reference manual.* Addison-Wesley Professional, first edition, 2003.

[KMM00] Matt Kaufmann, J. Strother Moore, and Panagiotis Manolios. *Computer-Aided Reasoning: An Approach.* Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[LP12] Alessio Lomuscio and Wojciech Penczek. Symbolic Model Checking for Temporal-Epistemic Logic. In *Logic Programs, Norms and Action*, pages 172–195, 2012.

[LQR09] Alessio Lomuscio, Hongyang Qu, and Franco Raimondi. MCMAS: A Model Checker for the Verification of Multi-Agent Systems. In *Proc. of CAV'09*, pages 682–688, 2009.

[LRL$^+$97] Hector J. Levesque, Ray Reiter, Yves Lespérance, Fangzhen Lin, and Richard B. Scherl. GOLOG: A Logic Programming Language for Dynamic Domains. *JLP*, 31:59–84, 1997.

[McM92] Kenneth Lauchlin McMillan. *Symbolic model checking: an approach to the state explosion problem.* PhD thesis, Pittsburgh, PA, USA, 1992. UMI Order No. GAX92-24209.

[Mil06] Dale Miller. Representing and Reasoning with Operational Semantics. pages 4–20. 2006.

[Mon01] Jean Francois Monin. *Understanding Formal Methods.* Springer-Verlag Inc., 2001.

[Par76] David Park. Finiteness is Mu-Ineffable. *Theor. Comput. Sci.*, 3(2):173–181, 1976.

[PPS06] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of Reactive(1) Designs. In *Proc. of VMCAI'06*, pages 364–380, 2006.

[PR99] Fiora Pirri and Ray Reiter. Some Contributions to the Metatheory of the Situation Calculus. *J. ACM*, 46(3):261–325, 1999.

[Rei01] Ray Reiter. *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems.* The MIT Press, 2001.

[SD09] Sebastian Sardina and Giuseppe De Giacomo. Composition of ConGolog Programs. In *Proc. of IJCAI'09*, pages 904–910, 2009.

[Tar55] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. of Mathematics*, 5(2):285309, 1955.

[Ume10] Akihiro Umemura. SAT/SMT solvers and their applications. *Computer Software*, 27(3):3_24–3_35, 2010.

Appendices

A Verifier for Basic Action Theory Game Structures

A.1 Prolog Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% L-LANGUAGE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:- op(800, xfy, [&]).   /* operator for conjunction */
:- op(850, xfy, [v]).   /* operator for disjunction */
:- op(870, xfy, [=>]).  /* operator for implication */
:- op(880,xfy, [<=>]).  /* operator for equivalence */

% explicit enumeration of constructs (all others are treated as atoms)
ll_construct(Pred) :-
   Pred = -W ;    %% negation of W
   Pred = (W1 & W2) ;    %% conjunction, uses & in place of ^
   Pred = (W1 v W2) ;    %% disjunction
   Pred = (W1 => W2)  ;    %% implication
   Pred = (W1 <=> W2) ;    %% equivalence
   Pred = some(X,W) ;    %% exists X . W
   Pred = all(X,W) ;    %% forall X . W
   Pred = mu(z,F) ;    %% mu X . F
   Pred = exists_successor(G,F) ;  %% exists possible successor state for agent
   Pred = exists_successor2(G,F) ;  %% exists possible successor state for anti-group
   Pred = forall_successors2(G,F) ;  %% for all possible successor states for anti-group
   Pred = next(G,F);    %% <<G>>oF
   Pred = ensure-eventually(G,F).    %% <<G>><>F
ll_atom(Pred) :- not(ll_construct(Pred)).

% negations of HOLDS constructs
holds(false,S) :- !, fail.
holds(-false,S) :- !.
holds(-P,S) :- ll_atom(P), !, not(holds(P,S)).
holds(-(-P),S) :- !, holds(P,S).
holds(-(P & Q),S) :- !, holds(-P v -Q,S).
holds(-(P v Q),S) :- !, holds(-P & -Q,S).
holds(-(P => Q),S) :- !, holds(-(-P v Q),S).
holds(-(P <=> Q),S) :- !, holds(-((P => Q) & (Q => P)),S).
holds(-all(V,P),S) :- !, holds(some(V,-P),S).
holds(-P,S) :- not(holds(P,S)).    %% the regular way for all others

% semantics of L-Language constructs
holds(P & Q,S) :-!, holds(P,S), holds(Q,S).    %% conjunction, uses & in place of ^
holds(P v Q,S) :-!, (holds(P,S); holds(Q,S)).    %% disjunction
holds(P => Q,S) :-!, holds(-P v Q,S).    %% implication as horn disjunction
holds(P <=> Q,S) :-!, holds((P => Q) & (Q => P),S).    %% equivalence as mutual impication
holds(all(V,P),S) :-!, holds(-some(V,-P),S).    %% universal in terms of existential
holds(some(V,P),S) :-!, subst(V,_,P,P1), holds(P1,S).    %% existential
holds(mu(z,F),S) :- !, mu_approx(z,F,false,1,S).    %% mu as approximation
holds(next(G,false),S) :- !, fail.    %% ignore for performance

%################################################
% "can ensure next" operator
% this macro expands according to SitCalc definition
%################################################
holds(next(G,F),S) :- !,    %% next operator <<G>>oF
  write('trying ---> '), output3(G,F,S), (
  incontrol(G,S), % group must be in control
  holds(exists_successor(G,F),S),
  write('> successor EXISTS for  G ---> '), output3(G,F,S)
  ;
  incontrol(-G,S), % anti-group must be in control
  holds(forall_successors2(-G,F),S),
  write('> successor FORALL for -G ---> '), output3(G,F,S)
  ).
holds(exists_successor(G,F),S) :- !,    %% F holds in do(a,s) for some legal actions of group G
  member(P,G),
  agent_action(P, A),
  S1=do(A,S), legal(S1),
  holds(F,S1), !.
holds(forall_successors2(-G,F),S) :- !,    %% F holds in do(a,s) for all legal actions of anti-group G
  not(holds(exists_successor2(-G,-F),S)).
holds(exists_successor2(-G,F),S) :- !,    %% F holds in do(a,s) for some legal actions of anti-group G
  agent(P), not(member(P,G)),
  agent_action(P, A),
  S1=do(A,S), legal(S1),
  holds(F,S1), !.
incontrol(-G,S) :- !,    %% group -G is in control
  agent(P), not(member(P,G)), agent_control(P,S), !.
incontrol(G,S) :- !,    %% group G is in control
  member(P,G), agent_control(P,S), !.

%% HOLDS for situation-suppressed formulas (non fluents, system predicates)
holds(Pred,S) :- restoreSitArg(Pred,S,PredEx), !, PredEx.
holds(Pred,S) :- ll_atom(Pred), Pred.
% !!! the domain must provide how to restore situation arguments for all fluents
% !!! for example: restoreSitArg(ontable(X),S,ontable(X,S)).

%########################
%### mu approximation ###
%########################

% binding diameter reached (as for bounded model checking)
mu_approx(Z,F,Int,N,S) :- binding_diameter(Max), N>Max, !,
    write('binding diameter '), write(N), write(' reached - stop'), nl, !, fail.
% else: show progress at the start of step (and fail to continue)
mu_approx(Z,F,Int,N,S) :- subst(Z,Int,F,Fx), output2(N,Fx), fail.
% else: substitute and chech if holds
mu_approx(Z,F,Int,N,S) :- subst(Z,Int,F,Fx), holds(Fx,S), !, output1(N,Fx).
% else: next approximation
mu_approx(Z,F,Int,N,S) :- M is N+1, subst(Z,Int,F,Int2), !, mu_approx(Z,F,Int2,M,S).

%################################################
% "ensure eventually" operator (in terms of mu and next)
%################################################
holds(ensure-eventually(G,F),S) :- !,    %% diamond operator <<G>><>F
  holds(mu(z,F v next(G,z)),S).


%#########################
%### helper predicates ###
%#########################
```

```
% output(M,F) = write out approximation (M = level,  F = formula)
output1(M,F) :- output_details(verbose), !,
    write('> ##### approximation '), write(M), write(' holds ---> '), write(F), nl.
output2(M,F) :- output_details(verbose), !,
    write('trying ##### approximation '), write(M), write(' ---> '), write(F), nl.
output3(G,F,S) :- output_details(verbose), !,
    write('next('), write(G), write(','), write(F), write(') ---> for S = '), write(S), nl.

% member(X,L) = asserts that X a member of a list L
member(X,[X|T]).
member(X,[H|T]) :- member(X,T).

% subst(NamePrev, NameNew, TermPrev, TermNew)
%    = asserts that TermNew is TermPrev with NamePrev replaced by NameNew
subst(_, _, [], []) :- !.
subst(_, _, TP, TP) :- var(TP), !.
subst(NP, NN, NP, NN) :- not(var(NN)), !.
subst(NP, NN, TP, TN) :- not(NP = TP), TP =..[H1|T1], !, subst_list(NP,NN,T1,T2),TN =..[H1|T2].
subst_list(_, _, [], []) :- !.
subst_list(NP, NN, [H1|T1], [H2|T2]) :- subst(NP,NN,H1,H2), !, subst_list(NP,NN,T1,T2).
```

A.2 Tic-Tac-Toe Game Axiomatization in Prolog

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% setup
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:- consult('llanguage').
:- dynamic(proc/2).
:- dynamic(restoreSitarg/3).
:- style_check(-discontiguous).
:- set_flag(print_depth,100).

output_details(verbose).  %% just define this term  for approximations to pe printed as we go
binding_diameter(11). %% maximul level of approximation expansion

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% fluents
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% GOAL
wins(P,S) :- inline(C1,C2,C3), cell(P,C1,S), cell(P,C2,S), cell(P,C3,S), agent(P), legal(S).

% SITCALC ARGS - restore S for fluents and goal
restoreSitArg(wins(P), S, wins(P,S)).
restoreSitArg(cell(P,C), S, cell(P,C,S)).
restoreSitArg(finished, S, finished(S)).
restoreSitArg(completed, S, completed(S)).

% DERIVED FLUENTS

completed(S) :- not(cell(b,_,S)).  %% no more moves
finished(S) :- completed(S) ; wins(P,S). %% no more moves or someone wins

result(S) :- not(legal(S)), write('illegal'), nl, !.
result(S) :- wins(P,S), write('winner: '), write(P), nl, !.
result(S) :- completed(S), write('draw'), nl, !.
result(_) :- write('in-progress'), nl, !.

% does a set make a line when ordered somehow
```

```
inline(C1,C2,C3) :- line(C1,C2,C3).
inline(C1,C2,C3) :- line(C1,C3,C2).
inline(C1,C2,C3) :- line(C2,C1,C3).
inline(C1,C2,C3) :- line(C2,C3,C1).
inline(C1,C2,C3) :- line(C3,C1,C2).
inline(C1,C2,C3) :- line(C3,C2,C1).

% OUTPUTS

disp_game(S) :-
write('#####'),nl,
write('#'),disp(1,S),disp(2,S),disp(3,S),write('#'),nl,
write('#'),disp(4,S),disp(5,S),disp(6,S),write('#'),nl,
write('#'),disp(7,S),disp(8,S),disp(9,S),write('#'),nl,
write('#####'),nl.

disp(C,S) :- cell(M,C,S), (M=b, write('.') ; M=x, write('X'); M=o, write('O')).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% domain
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% ACTIONS
primitive_action(mark(P,C)) :- agent(P), cell(C).

% AGENTS
agent(o).
agent(x).

% CELLS
cell(1).
cell(2).
cell(3).
cell(4).
cell(5).
cell(6).
cell(7).
cell(8).
cell(9).

% LINES
line(1,2,3).
line(4,5,6).
line(7,8,9).
line(1,4,7).
line(2,5,8).
line(3,6,9).
line(1,5,9).
line(3,5,7).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% D_poss
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% poss: possible (is it possible when given legal or not situation)
poss(mark(P,C),S) :- cell(b,C,S), agent(P), cell(C).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% D_ssa
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

cell(M,C,do(A,S)) :- A=mark(M,C) ; cell(M,C,S), not(A=mark(_,C)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%% D_S0
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% all cells in domain are b (blank) in s0.
cell(b,C,s0) :- cell(C).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% D_legal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% control: agent is the next to action in situation
agent_control(P,S) :- agent_action(P,A), legal(do(A,S)), !.

% agent_action - determine the agent of an action
agent_action(P,A) :-  A = mark(P,C), agent(P), cell(C).

% legal: is the situation LEGAL (can this situation be arrived at)
legal(s0) :- !.
legal(S) :- %%% X must play first
S = do(A,s0),
agent_action(x,A),
poss(A, s0), !.
legal(S) :- % alternate the actions: X's action after O's
S = do(A1,S1), S1 = do(A2,S2),
agent_action(x,A1), agent_action(o,A2),
poss(A1,S1), poss(A2,S2),not(finished(S1)), !.
legal(S) :- % alternate the actions: O's action after X's
S = do(A1,S1), S1 = do(A2,S2),
agent_action(o,A1), agent_action(x,A2),
poss(A1,S1), poss(A2,S2),not(finished(S1)), !.
```

A.3 Tic-Tac-Toe Game Property Tests

```
%%% TEST SITUATIONS %%%
sit(s5,S) :- S=do(mark(x,7),do(mark(o,3),do(mark(x,4),do(mark(o,2),do(mark(x,1),s0))))).
% XOO
% X..
% X..
sit(s4,S) :- S=do(mark(o,3),do(mark(x,4),do(mark(o,2),do(mark(x,1),s0)))).
% XOO
% X..
% ...
sit(s3,S) :- S=do(mark(x,4),do(mark(o,2),do(mark(x,1),s0))).
% XO.
% X..
% ...
sit(s2,S) :- S=do(mark(o,2),do(mark(x,1),s0)).
% XO.
% ...
% ...
sit(s1,S) :- S=do(mark(x,1),s0).
% X..
% ...
% ...
sit(s0,S) :- S=s0.
% ...
% ...
% ...

%%%%%%%%%%
sit(sw6,S) :- S=do(mark(x,3), do(mark(o, 8), do(mark(x, 2), do(mark(o, 9), do(mark(x, 1),
    do(mark(o, 4), do(mark(x, 5), s0))))))).
% XXX
% OX.
% .OO
sit(sw5,S) :- S=do(mark(o,8),do(mark(x,2),do(mark(o,9),do(mark(x,1),do(mark(o,4),do(mark(x,5),s0)))))).
% XX.
% OX.
% .OO
sit(sw4,S) :- S=do(mark(x,2),do(mark(o,9),do(mark(x,1),do(mark(o,4),do(mark(x,5),s0))))).
% XX.
% OX.
% ..O
sit(sw3,S) :- S=do(mark(o,9),do(mark(x,1),do(mark(o,4),do(mark(x,5),s0)))).
% X..
% OX.
% ..O
sit(sw2,S) :- S=do(mark(x,1),do(mark(o,4),do(mark(x,5),s0))).
% X..
% OX.
% ...
sit(sw1,S) :- S=do(mark(o,4),do(mark(x,5),s0)).
% ...
% OX.
% ...
sit(sw0,S) :- S=do(mark(x,5),s0).
% ...
% .X.
% ...
%%%%%%%%%%
sit(sx1,S) :- S=do(mark(x,2),do(mark(x,1),s0)). % illegal
% XX.
% ...
% ...
sit(sx2,S) :- S=do(mark(x, 2), do(mark(o, 7), do(mark(x, 4), do(mark(o, 5), do(mark(x, 1), s0))))).
% XX.
% XO.
% O..
%%% TESTS %%%
% sit(s5,S), holds(ensure([x],wins(x)),S).
% approx 1, 0s
% sit(s4,S), holds(ensure([x],wins(x)),S).
% approx 2, 0s
% sit(s3,S), holds(ensure([x],wins(x)),S).
% approx 5, 3s
% sit(s2,S), holds(ensure([x],wins(x)),S).
% approx 6, 6s
% sit(s1,S), not(holds(ensure([x],wins(x)),S)).
% binding diameter, 2m
% sit(s0,S), not(holds(ensure([x],wins(x)),S)).
% binding diameter, 15m
%%%%%%%%%%
% sit(sw6,S), holds(ensure([x],wins(x)),S).
% approx 1, 0s
% sit(sw5,S), holds(ensure([x],wins(x)),S).
% approx 2, 0s
% sit(sw4,S), holds(ensure([x],wins(x)),S).
% approx 3, 1s
% sit(sw3,S), holds(ensure([x],wins(x)),S).
% approx 4, 1s
% sit(sw2,S), holds(ensure([x],wins(x)),S).
% approx 5, 3s
% sit(sw1,S), holds(ensure([x],wins(x)),S).
% approx 6, 6s
% sit(sw0,S), not(holds(ensure([x],wins(x)),S)).
% binding diameter, 1m
%%%%%%%%%%
% sit(sx1,S), holds(ensure([x],wins(x)),S).
% approx 5, 1s
% sit(sx2,S), not(holds(ensure([x],wins(x)),S)).
% binding diameter, 0s
```

B Verifier for Basic Action Theory Game Structures with Action/Strategy Constraints

B.1 Prolog Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% L-LANGUAGE-B
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:- op(800, xfy, [&]).   /* operator for conjunction */
:- op(850, xfy, [v]).   /* operator for disjunction */
:- op(870, xfy, [=>]).  /* operator for implication */
:- op(880,xfy, [<=>]).  /* operator for equivalence */

% explicit enumeration of constructs (all others are treated as atoms)
ll_construct(Pred) :-
   Pred = -W ;    %% negation of W
   Pred = (W1 & W2) ;    %% conjunction, uses & in place of ^
   Pred = (W1 v W2) ;    %% disjunction
   Pred = (W1 => W2)  ;    %% implication
   Pred = (W1 <=> W2) ;    %% equivalence
   Pred = some(X,W) ;    %% exists X . W
   Pred = all(X,W) ;    %% forall X . W
   Pred = mu(z,F) ;    %% mu X . F
   Pred = exists_successor(P,F) ;  %% exists possible successor state for agent
   Pred = forall_successors(P,F) ;  %% for all possible successor states for agent
   Pred = exists_successor2(P,F) ;  %% exists possible successor state for group
   Pred = forall_successors2(P,F) ;  %% for all possible successor states for group
   Pred = next(G,F);    %% <<G>>oF
   Pred = ensure(G,F).    %% <<G>><>F
ll_atom(Pred) :- not(ll_construct(Pred)).

% negations of HOLDS constructs
holds(false,S) :- !, fail.
holds(-false,S) :- !.
holds(-P,S) :- ll_atom(P), !, not(holds(P,S)).
holds(-(-P),S) :- !, holds(P,S).
holds(-(P & Q),S) :- !, holds(-P v -Q,S).
holds(-(P v Q),S) :- !, holds(-P & -Q,S).
holds(-(P => Q),S) :- !, holds(-(-P v Q),S).
holds(-(P <=> Q),S) :- !, holds(-((P => Q) & (Q => P)),S).
holds(-all(V,P),S) :- !, holds(some(V,-P),S).
holds(-P,S) :- not(holds(P,S)).    %% the regular way for all others

% semantics of L-Language constructs
holds(P & Q,S) :-!, holds(P,S), holds(Q,S).    %% conjunction, uses & in place of ^
holds(P v Q,S) :-!, (holds(P,S); holds(Q,S)).    %% disjunction
holds(P => Q,S) :-!, holds(-P v Q,S).    %% implication as horn disjunction
holds(P <=> Q,S) :-!, holds((P => Q) & (Q => P),S).    %% equivalence as mutual impication
holds(all(V,P),S) :-!, holds(-some(V,-P),S).    %% universal in terms of existential
holds(some(V,P),S) :-!, subst(V,_,P,P1), holds(P1,S).    %% existential
holds(mu(z,F),S) :- !, mu_approx(z,F,false,1,S).    %% mu as approximation
holds(next(G,false),S) :- !, fail.    %% ignore for performance
holds(next(G,F),S) :- !,    %% next operator <<G>>oF
  write('trying ---> '), output3(G,F,S), (
  incontrol(G,S), !, % group must be in control
  holds(exists_successor(G,F),S),
  write('> successor EXISTS for  G ---> '), output3(G,F,S)
  ;
  incontrol(-G,S), !, % anti-group must be in control
  holds(forall_successors2(-G,F),S),
  write('> successor FORALL for -G ---> '), output3(G,F,S)
  ).
holds(ensure(G,F),S) :- !,    %% diamond operator <<G>><>F
  holds(mu(z,F v next(G,z)),S).
holds(exists_successor(G,F),S) :-    %% F holds in do(a,s) for some PREFERRED actions of group G
  member(P,G),
  preferred(P,A,S), agent_action(P, A),
  S1=do(A,S), legal(S1), !,    %% --> preferred action exists so commit to this version of predicate
  member(PX,G),
  preferred(PX,AX,S), agent_action(PX, AX),
  SX=do(AX,S), legal(SX), holds(F,SX), !.
holds(exists_successor(G,F),S) :-    %% F holds in do(a,s) for some legal actions of group G
  member(P,G),
  agent_action(P, A),
  S1=do(A,S), legal(S1),
  holds(F,S1), !.
holds(forall_successors2(-G,F),S) :- !,    %% F holds in do(a,s) for all legal actions of anti-group G
  not(holds(exists_successor2(-G,-F),S)).
holds(exists_successor2(-G,F),S) :-    %% F holds in do(a,s) for PREFERRED actions of anti-group G
  agent(P), not(member(P,G)),
  preferred(P,A,S), agent_action(P, A),
  S1=do(A,S), legal(S1), !,    %% --> preferred action exists so commit to this version of predicate
  agent(PX), not(member(PX,G)),
  preferred(PX,AX,S), agent_action(PX, AX),
  SX=do(AX,S), legal(SX), holds(F,SX), !.
holds(exists_successor2(-G,F),S) :-    %% F holds in do(a,s) for some legal actions of anti-group G
  agent(P), not(member(P,G)),
  agent_action(P, A),
  S1=do(A,S), legal(S1),
  holds(F,S1), !.
incontrol(-G,S) :- !,    %% group -G is in control
  agent(P), not(member(P,G)), agent_control(P,S), !.
incontrol(G,S) :- !,    %% group G is in control
  member(P,G), agent_control(P,S), !.

%% HOLDS for situation-suppressed formulas (non fluents, system predicates)
holds(Pred,S) :- restoreSitArg(Pred,S,PredEx), !, PredEx.
holds(Pred,S) :- ll_atom(Pred), Pred.
% !!! the domain must provide how to restore situation arguments for all fluents
% !!! for example: restoreSitArg(ontable(X),S,ontable(X,S)).

%########################
%### mu approximation ###
%########################

% binding diameter reached (as for bounded model checking)
mu_approx(Z,F,Int,N,S) :- binding_diameter(Max), N>Max, !,
    write('binding diameter '), write(N), write(' reached - stop'), nl, !, fail.
% else: show progress at the start of step (and fail to continue)
mu_approx(Z,F,Int,N,S) :- subst(Z,Int,F,Fx), output2(N,Fx), fail.
% else: substitute and chech if holds
mu_approx(Z,F,Int,N,S) :- subst(Z,Int,F,Fx), holds(Fx,S), !, output1(N,Fx).
```

```
% else: next approximation
mu_approx(Z,F,Int,N,S) :- M is N+1, subst(Z,Int,F,Int2), !, mu_approx(Z,F,Int2,M,S).

%#########################
%### helper predicates ###
%#########################

% output(M,F) = write out approximation (M = level,  F = formula)
output1(M,F) :- output_details(verbose), !,
    write('> ##### approximation '), write(M), write(' holds ---> '), write(F), nl.
output2(M,F) :- output_details(verbose), !,
    write('trying ##### approximation '), write(M), write(' ---> '), write(F), nl.
output3(G,F,S) :- output_details(verbose), !,
    write('next('), write(G), write(','), write(F), write(') ---> for S = '), write(S), nl.

% member(X,L) = asserts that X a member of a list L
member(X,[X|T]).
member(X,[H|T]) :- member(X,T).

% subst(NamePrev, NameNew, TermPrev, TermNew)
%    = asserts that TermNew is TermPrev with NamePrev replaced by NameNew
subst(_, _, [], []) :- !.
subst(_, _, TP, TP) :- var(TP), !.
subst(NP, NN, NP, NN) :- not(var(NN)), !.
subst(NP, NN, TP, TN) :- not(NP = TP), TP =..[H1|T1], !, subst_list(NP,NN,T1,T2),TN =..[H1|T2].
subst_list(_, _, [], []) :- !.
subst_list(NP, NN, [H1|T1], [H2|T2]) :- subst(NP,NN,H1,H2), !, subst_list(NP,NN,T1,T2).
```

B.2 Tic-Tac-Toe Game Axiomatization in Prolog

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% setup
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:- consult('llanguageb').
:- dynamic(proc/2).
:- dynamic(restoreSitarg/3).
:- style_check(-discontiguous).
:- set_flag(print_depth,100).

output_details(verbose).  %% just define this term  for approximations to pe printed as we go
binding_diameter(11). %% maximul level of approximation expansion

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% fluents
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% GOAL
wins(P,S) :- inline(C1,C2,C3), cell(P,C1,S), cell(P,C2,S), cell(P,C3,S), agent(P), legal(S).

% SITCALC ARGS - restore S for fluents and goal
restoreSitArg(wins(P), S, wins(P,S)).
restoreSitArg(cell(P,C), S, cell(P,C,S)).
restoreSitArg(finished, S, finished(S)).
restoreSitArg(completed, S, completed(S)).

% DERIVED FLUENTS

completed(S) :- not(cell(b,_,S)).  %% no more moves
finished(S) :- completed(S) ; wins(P,S). %% no more moves or someone wins
```

```
result(S) :- not(legal(S)), write('illegal'), nl, !.
result(S) :- wins(P,S), write('winner: '), write(P), nl, !.
result(S) :- completed(S), write('draw'), nl, !.
result(_) :- write('in-progress'), nl, !.

% does a set make a line when ordered somehow
inline(C1,C2,C3) :- line(C1,C2,C3).
inline(C1,C2,C3) :- line(C1,C3,C2).
inline(C1,C2,C3) :- line(C2,C1,C3).
inline(C1,C2,C3) :- line(C2,C3,C1).
inline(C1,C2,C3) :- line(C3,C1,C2).
inline(C1,C2,C3) :- line(C3,C2,C1).

% OUTPUTS

disp_game(S) :-
write('#####'),nl,
write('#'),disp(1,S),disp(2,S),disp(3,S),write('#'),nl,
write('#'),disp(4,S),disp(5,S),disp(6,S),write('#'),nl,
write('#'),disp(7,S),disp(8,S),disp(9,S),write('#'),nl,
write('#####'),nl.

disp(C,S) :- cell(M,C,S), (M=b, write('.') ; M=x, write('X'); M=o, write('O')).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% domain
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% ACTIONS
primitive_action(mark(P,C)) :- agent(P), cell(C).

% AGENTS
agent(o).
agent(x).

% CELLS
cell(1).
cell(2).
cell(3).
cell(4).
cell(5).
cell(6).
cell(7).
cell(8).
cell(9).

% LINES
line(1,2,3).
line(4,5,6).
line(7,8,9).
line(1,4,7).
line(2,5,8).
line(3,6,9).
line(1,5,9).
line(3,5,7).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% D_poss
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% poss: possible (is it possible when given legal or not situation)
poss(mark(P,C),S) :- cell(b,C,S), agent(P), cell(C).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%% D_ssa
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

cell(M,C,do(A,S)) :- A=mark(M,C) ; cell(M,C,S), not(A=mark(_,C)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% D_S0
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% all cells in domain are b (blank) in s0.
cell(b,C,s0) :- cell(C).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% D_legal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% control: agent is the next to action in situation
agent_control(P,S) :- agent_action(P,A), legal(do(A,S)), !.

% agent_action - determine the agent of an action
agent_action(P,A) :-  A = mark(P,C), agent(P), cell(C).

% legal: is the situation LEGAL (can this situation be arrived at)
legal(s0) :- !.
legal(S) :-
S = do(A,s0),
agent_action(x,A),
poss(A, s0), !.
legal(S) :-
S = do(A1,S1), S1 = do(A2,S2),
agent_action(x,A1), agent_action(o,A2),
poss(A1,S1), poss(A2,S2),not(finished(S1)), !.
legal(S) :-
S = do(A1,S1), S1 = do(A2,S2),
agent_action(o,A1), agent_action(x,A2),
poss(A1,S1), poss(A2,S2),not(finished(S1)), !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% SOFT CONSTRAINTS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% preferred - soft constraint for P i.e. try to restrict agent actions if possible
preferred(P,A,S) :- P=o, cell(b,1,S), A=mark(P,1). %% grab corners
preferred(P,A,S) :- P=o, cell(b,3,S), A=mark(P,3). %% grab corners
preferred(P,A,S) :- P=o, cell(b,7,S), A=mark(P,7). %% grab corners
preferred(P,A,S) :- P=o, cell(b,9,S), A=mark(P,9). %% grab corners
```

B.3 Tic-Tac-Toe Game Property Tests

```
%%% TEST SITUATIONS %%%
sit(s5,S) :- S=do(mark(x,7),do(mark(o,3),do(mark(x,4),do(mark(o,2),do(mark(x,1),s0))))).
% XOO
% X..
% X..
sit(s4,S) :- S=do(mark(o,3),do(mark(x,4),do(mark(o,2),do(mark(x,1),s0)))).
% XOO
% X..
% ...
sit(s3,S) :- S=do(mark(x,4),do(mark(o,2),do(mark(x,1),s0))).
% XO.
% X..
% ...
```

```
sit(s2,S) :- S=do(mark(o,2),do(mark(x,1),s0)).
% XO.
% ...
% ...
sit(s1,S) :- S=do(mark(x,1),s0).
% X..
% ...
% ...
sit(s0,S) :- S=s0.
% ...
% ...
% ...
%%%%%%%%%%
sit(sw6,S) :- S=do(mark(x,3), do(mark(o, 8), do(mark(x, 2), do(mark(o, 9),
    do(mark(x, 1), do(mark(o, 4), do(mark(x, 5), s0))))))).
% XXX
% OX.
% .OO
sit(sw5,S) :- S=do(mark(o,8),do(mark(x,2),do(mark(o,9),do(mark(x,1),do(mark(o,4),do(mark(x,5),s0)))))).
% XX.
% OX.
% .OO
sit(sw4,S) :- S=do(mark(x,2),do(mark(o,9),do(mark(x,1),do(mark(o,4),do(mark(x,5),s0))))).
% XX.
% OX.
% ..O
sit(sw3,S) :- S=do(mark(o,9),do(mark(x,1),do(mark(o,4),do(mark(x,5),s0)))).
% X..
% OX.
% ..O
sit(sw2,S) :- S=do(mark(x,1),do(mark(o,4),do(mark(x,5),s0))).
% X..
% OX.
% ...
sit(sw1,S) :- S=do(mark(o,4),do(mark(x,5),s0)).
% ...
% OX.
% ...
sit(sw0,S) :- S=do(mark(x,5),s0).
% ...
% .X.
% ...
%%%%%%%%%%
sit(sx1,S) :- S=do(mark(x,2),do(mark(x,1),s0)). % illegal
% XX.
% ...
% ...
sit(sx2,S) :- S=do(mark(x, 2), do(mark(o, 7), do(mark(x, 4), do(mark(o, 5), do(mark(x, 1), s0))))).
% XX.
% XO.
% O..
%%% TESTS %%%
% sit(s5,S), holds(ensure([x],wins(x)),S).
% approx 1, 0s
% sit(s4,S), holds(ensure([x],wins(x)),S).
% approx 2, 0s
% sit(s3,S), holds(ensure([x],wins(x)),S).
% approx 5, 0s
% sit(s2,S), holds(ensure([x],wins(x)),S).
% approx 4, 1s
% sit(s1,S), holds(ensure([x],wins(x)),S).
% approx 5, 2s
% sit(s0,S), holds(ensure([x],wins(x)),S).
% approx 6, 4s
```

```
%%%%%%%%%%
% sit(sw6,S), holds(ensure([x],wins(x)),S).
% approx 1, 0s
% sit(sw5,S), holds(ensure([x],wins(x)),S).
% approx 2, 0s
% sit(sw4,S), holds(ensure([x],wins(x)),S).
% approx 3, 0s
% sit(sw3,S), holds(ensure([x],wins(x)),S).
% approx 4, 0s
% sit(sw2,S), holds(ensure([x],wins(x)),S).
% approx 5, 1s
% sit(sw1,S), holds(ensure([x],wins(x)),S).
% approx 4, 1s
% sit(sw0,S), holds(ensure([x],wins(x)),S).
% approx 5, 2s
%%%%%%%%%%
% sit(sx1,S), holds(ensure([x],wins(x)),S).
% approx 5, 2s
% sit(sx2,S), holds(ensure([x],wins(x)),S).
% failed, 2s
```

C Verifier for GameGolog-Expressed Game Structures

C.1 Prolog Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% GAMEGOLOG SEMANTICS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% similar to Golog semantcs
%% there is a new parameter U to trans and it holds if agent U is in control in situation S
%% there is a new parameter UX to trans and it holds if agent UX is in control in situation SX

:- op(200, fy, [?]).    %% operator for domain
:- op(200, xfy, [of]).    %% operator for domain
:- op(210, xfy, [:]).    %% operator for sequence of actions
:- op(220, xfy, [@@]).    %% operator for nd choice of action

trans(E,S,SX) :- trans(_U,E,S,_UX,_EX,SX).

trans(U,?P,SX,U,_EX,SX) :- !, fail.    %% test
trans(U,?true : E,S,UX,EX,SX) :- !, trans(U,E,S,UX,EX,SX).    %% performance improvement
trans(U,E1 : E2,S,UX,EX,SX) :- trans(U,E1,S,UX,EX1,SX), EX = (EX1 : E2).    %% action sequence
trans(U,E1 : E2,S,UX,EX,SX) :- final(E1,S), trans(U,E2,S,UX,EX,SX).    %% action sequence
trans(U,if(P,E1,E2),S,UX,EX,SX) :- holds(P,S), trans(U,E1,S,UX,EX,SX).    %% if then else
trans(U,if(P,E1,E2),S,UX,EX,SX) :- not(holds(P,S)), trans(U,E2,S,UX,EX,SX).  %% if then else
trans(U,while(P,E),S,UX,EX,SX):-    %% while
  holds(P,S), trans(U,E,S,UX,EX1,SX), EX = (EX1 : while(P,E)).
trans(U,E1 @@ E2,S,U,E1,do(left(U),S)).    %% nd branch
trans(U,E1 @@ E2,S,U,E2,do(right(U),S)).    %% nd branch
trans(U,pi(V of D,E),S,U,EX,SX) :- DP=..[D,VX], DP, poss(pick(U,VX),S),
  SX = do(pick(U,VX),S), subst(V,VX,E,EX).    %% nd choice of argument
trans(U,star(E),S,U,E : star(E),do(continue(U),S)).    %% star
trans(U,star(E),S,U,?true,do(stop(U),S)).    %% star
trans(_U,uc(UX1,E),S,UX,EX,SX) :- trans(UX1,E,S,UX,EX,SX).    %% under control
trans(U,A,S,U,?true,do(A,S)) :- primitive_action(A),poss(A,S).    %% primitive action

final(E,S) :- primitive_action(E),!,fail.    %% primitive action
final(?P,S) :- holds(P,S).    %% test
final(E1 : E2,S) :- final(E1,S), final(E2,S).    %% sequence
final(if(P,E1,E2),S) :- holds(P,S), final(E1,S).    %% if
final(if(P,E1,E2),S) :- not(holds(P,S)), final(E2,S).    %% if
final(while(P,E),S) :- holds(P,S), final(E,S) ; not(holds(P,S)).    %% while
final(E1 @@ E2,S) :- !, fail.    %% nd branch - GG version
final(pi(V of D,E),S) :- !, fail.    %% nd choice of argument - GG version
final(star(E),S) :- !, fail.    %% star - GG version

% trans* (star composition of trans)
transs(U,E,s0,UX,EX,do(A,s0)) :- trans(U,E,s0,UX,EX,do(A,s0)).
transs(U,E,S,UX,EX,do(A,S1)) :- transs(U,E,S,U1,E1,S1), trans(U1,E1,S1,UX,EX,do(A,S1)).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% L-LANGUAGE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:- op(800, xfy, [&]).   /* operator for conjunction */
:- op(850, xfy, [v]).   /* operator for disjunction */
:- op(870, xfy, [=>]).  /* operator for implication */
:- op(880,xfy, [<=>]).  /* operator for equivalence */

% explicit enumeration of constructs (all others are treated as atoms)
ll_construct(Pred) :-
   Pred = -W ;    %% negation of W
   Pred = (W1 & W2) ;    %% conjunction, uses & in place of ^
   Pred = (W1 v W2) ;    %% disjunction
   Pred = (W1 => W2)  ;    %% implication
   Pred = (W1 <=> W2) ;    %% equivalence
   Pred = some(X,W) ;    %% exists X . W
   Pred = all(X,W) ;    %% forall X . W
   Pred = mu(z,F) ;    %% mu X . F
   Pred = exists_successor(G,F) ;  %% exists possible successor state for group
   Pred = exists_successor2(G,F) ;  %% exists possible successor state for anti-group
   Pred = forall_successors2(G,F) ;  %% for all possible successor states for anti-group
   Pred = next(G,F);    %% <<G>>oF
   Pred = ensure(G,F).    %% <<G>><>F
ll_atom(Pred) :- not(ll_construct(Pred)).

% negations of HOLDS constructs
holds(false,S) :- !, fail.
holds(-false,S) :- !.
holds(-P,S) :- ll_atom(P), !, not(holds(P,S)).
holds(-(-P),S) :- !, holds(P,S).
holds(-(P & Q),S) :- !, holds(-P v -Q,S).
holds(-(P v Q),S) :- !, holds(-P & -Q,S).
holds(-(P => Q),S) :- !, holds(-(-P v Q),S).
holds(-(P <=> Q),S) :- !, holds(-((P => Q) & (Q => P)),S).
holds(-all(V,P),S) :- !, holds(some(V,-P),S).
holds(-P,S) :- not(holds(P,S)).    %% the regular way for all others

% semantics of L-Language constructs
holds(P & Q,S) :-!, holds(P,S), holds(Q,S).    %% conjunction, uses & in place of ^
holds(P v Q,S) :-!, (holds(P,S); holds(Q,S)).    %% disjunction
holds(P => Q,S) :-!, holds(-P v Q,S).    %% implication as horn disjunction
holds(P <=> Q,S) :-!, holds((P => Q) & (Q => P),S).    %% equivalence as mutual impication
holds(all(V,P),S) :-!, holds(-some(V,-P),S).    %% universal in terms of existential
holds(some(V,P),S) :-!, subst(V,_,P,P1), holds(P1,S).    %% existential
holds(mu(z,F),S) :- !, mu_approx(z,F,false,1,S).    %% mu as approximation
holds(next(G,false),S) :- !, fail.    %% ignore for performance
holds(next(G,F),S) :- !,    %% next operator <<G>>oF
  write('trying ---> '), output3(G,F,S), (
  incontrol(G,S), % group must be in control
  holds(exists_successor(G,F),S),
  write('> successor EXISTS for  G ---> '), output3(G,F,S)
  ;
  incontrol(-G,S), % anti-group must be in control
  holds(forall_successors2(-G,F),S),
  write('> successor FORALL for -G ---> '), output3(G,F,S)
  ).
holds(ensure(G,F),S) :- !,    %% diamond operator <<G>><>F
  holds(mu(z,F v next(G,z)),S).
holds(exists_successor(G,F),S) :- !,    %% F holds in do(a,s) for some legal actions of group G
  member(P,G),
  agent_action(P, A),
  S1=do(A,S), legal(S1),
  holds(F,S1), !.
holds(forall_successors2(-G,F),S) :- !,    %% F holds in do(a,s) for all legal actions of anti-group G
  not(holds(exists_successor2(-G,-F),S)).
holds(exists_successor2(-G,F),S) :- !,    %% F holds in do(a,s) for some legal actions of anti-group G
  agent(P), not(member(P,G)),
  agent_action(P, A),
  S1=do(A,S), legal(S1),
  holds(F,S1), !.
incontrol(-G,S) :- !,    %% group -G is in control
  agent(P), not(member(P,G)), agent_control(P,S), !.
incontrol(G,S) :- !,    %% group G is in control
  member(P,G), agent_control(P,S), !.

%% HOLDS for situation-suppressed formulas (non fluents, system predicates)
holds(Pred,S) :- restoreSitArg(Pred,S,PredEx), !, PredEx.
holds(Pred,S) :- ll_atom(Pred), Pred.
% !!! the domain must provide how to restore situation arguments for all fluents
% !!! for example: restoreSitArg(ontable(X),S,ontable(X,S)).

%########################
%### mu approximation ###
%########################

% binding diameter reached (as for bounded model checking)
mu_approx(Z,F,Int,N,S) :- binding_diameter(Max), N>Max, !,
    write('binding diameter '), write(N), write(' reached - stop'), nl, !, fail.
% else: show progress at the start of step (and fail to continue)
mu_approx(Z,F,Int,N,S) :- subst(Z,Int,F,Fx), output2(N,Fx), fail.
% else: substitute and chech if holds
mu_approx(Z,F,Int,N,S) :- subst(Z,Int,F,Fx), holds(Fx,S), !, output1(N,Fx).
% else: next approximation
mu_approx(Z,F,Int,N,S) :- M is N+1, subst(Z,Int,F,Int2), !, mu_approx(Z,F,Int2,M,S).

%#########################
%### helper predicates ###
%#########################

% output(M,F) = write out approximation (M = level,  F = formula)
output1(M,F) :- output_details(verbose), !,
    write('> ##### approximation '), write(M), write(' holds ---> '), write(F), nl.
output2(M,F) :- output_details(verbose), !,
    write('trying ##### approximation '), write(M), write(' ---> '), write(F), nl.
output3(G,F,S) :- output_details(verbose), !,
    write('next('), write(G), write(','), write(F), write(') ---> for S = '), write(S), nl.

% member(X,L) = asserts that X a member of a list L
member(X,[X|T]).
member(X,[H|T]) :- member(X,T).

% subst(NamePrev, NameNew, TermPrev, TermNew)
%    = asserts that TermNew is TermPrev with NamePrev replaced by NameNew
subst(_, _, [], []) :- !.
subst(_, _, TP, TP) :- var(TP), !.
subst(NP, NN, NP, NN) :- not(var(NN)), !.
subst(NP, NN, TP, TN) :- not(NP = TP), TP =..[H1|T1], !, subst_list(NP,NN,T1,T2),TN =..[H1|T2].
subst_list(_, _, [], []) :- !.
subst_list(NP, NN, [H1|T1], [H2|T2]) :- subst(NP,NN,H1,H2), !, subst_list(NP,NN,T1,T2).
```

C.2 Tic-Tac-Toe GameGolog Axiomatization in Prolog

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%% setup
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:- consult('gg').
:- dynamic(proc/2).
:- dynamic(restoreSitarg/3).
:- style_check(-discontiguous).
:- set_flag(print_depth,100).

output_details(verbose).  %% just define this term  for approximations to pe printed as we go
binding_diameter(22). %% maximul level of approximation expansion

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% fluents
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% GOAL
wins(P,S) :- inline(C1,C2,C3), cell(P,C1,S), cell(P,C2,S), cell(P,C3,S), agent(P), legal(S).

% SITCALC ARGS - restore S for fluents and goal
restoreSitArg(wins(P), S, wins(P,S)).
restoreSitArg(cell(P,C), S, cell(P,C,S)).
restoreSitArg(finished, S, finished(S)).
restoreSitArg(completed, S, completed(S)).

% DERIVED FLUENTS

completed(S) :- not(cell(b,_,S)).  %% no more moves
finished(S) :- completed(S) ; wins(P,S). %% no more moves or someone wins

result(S) :- not(legal(S)), write('illegal'), nl, !.
result(S) :- wins(P,S), write('winner: '), write(P), nl, !.
result(S) :- completed(S), write('draw'), nl, !.
result(_) :- write('in-progress'), nl, !.

% does a set make a line when ordered somehow
inline(C1,C2,C3) :- line(C1,C2,C3).
inline(C1,C2,C3) :- line(C1,C3,C2).
inline(C1,C2,C3) :- line(C2,C1,C3).
inline(C1,C2,C3) :- line(C2,C3,C1).
inline(C1,C2,C3) :- line(C3,C1,C2).
inline(C1,C2,C3) :- line(C3,C2,C1).

% OUTPUTS

disp_game(S) :-
write('#####'),nl,
write('#'),disp(1,S),disp(2,S),disp(3,S),write('#'),nl,
write('#'),disp(4,S),disp(5,S),disp(6,S),write('#'),nl,
write('#'),disp(7,S),disp(8,S),disp(9,S),write('#'),nl,
write('#####'),nl.

disp(C,S) :- cell(M,C,S), (M=b, write('.') ; M=x, write('X'); M=o, write('O')).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% domain
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% ACTIONS
primitive_action(mark(P,C)) :- agent(P), cell(C).
primitive_action(pick(P,C)) :- agent(P), cell(C).
primitive_action(stop(P)) :- agent(P).
primitive_action(continue(P)) :- agent(P).
```

```
% AGENTS
agent(o).
agent(x).

% CELLS
cell(1).
cell(2).
cell(3).
cell(4).
cell(5).
cell(6).
cell(7).
cell(8).
cell(9).

% LINES
line(1,2,3).
line(4,5,6).
line(7,8,9).
line(1,4,7).
line(2,5,8).
line(3,6,9).
line(1,5,9).
line(3,5,7).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% D_poss
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% poss: possible (is it possible when given legal or not situation)
poss(mark(P,C),S) :- cell(b,C,S), agent(P), cell(C).
poss(pick(P,C),S) :- cell(b,C,S), agent(P), cell(C).
poss(stop(P),S).
poss(continue(P),S).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% D_ssa
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

cell(M,C,do(A,S)) :- A=mark(M,C) ; cell(M,C,S), not(A=mark(_,C)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% D_S0
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% all cells in domain are b (blank) in s0.
cell(b,C,s0) :- cell(C).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% D_legal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% control: agent is the next to action in situation
agent_control(P,S) :- agent_action(P,A), legal(do(A,S)), !.

% agent_action - determine the agent of an action
agent_action(P,A) :-  A = mark(P,C), agent(P), cell(C).
agent_action(P,A) :-  A = pick(P,C), agent(P), cell(C).
agent_action(P,A) :-  A = stop(P), agent(P).
agent_action(P,A) :-  A = continue(P), agent(P).

% legal: is the situation LEGAL (can this situation be arrived at) using trans*
legal(S) :- proc(ttt,R0), transs(_U,R0,s0,_UX,_RX,S).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% GameGolog
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% program
% all bound variables must have unique names
proc(ttt,
while(-finished,
uc(x, pi(c1 of cell, mark(x,c1))) :
if(-finished,
uc(o, pi(c2 of cell, mark(o,c2))),
? true)
)
).
```

C.3 Tic-Tac-Toe Game Property Tests

```
%%% TEST SITUATIONS %%%
sit(s5,S) :- S=do(mark(x,7),do(pick(x,7),do(mark(o,3),do(pick(o,3),do(mark(x,4),do(pick(x,4),
    do(mark(o,2),do(pick(o,2),do(mark(x,1),do(pick(x,1),s0)))))))))).
% XOO
% X..
% X..
sit(s4,S) :- S=do(mark(o,3),do(pick(o,3),do(mark(x,4),do(pick(x,4),do(mark(o,2),do(pick(o,2),
    do(mark(x,1),do(pick(x,1),s0)))))))).
% XOO
% X..
% ...
sit(s3,S) :- S=do(mark(x,4),do(pick(x,4),do(mark(o,2),do(pick(o,2),do(mark(x,1),do(pick(x,1),s0)))))).
% XO.
% X..
% ...
sit(s2,S) :- S=do(mark(o,2),do(pick(o,2),do(mark(x,1),do(pick(x,1),s0)))).
% XO.
% ...
% ...
sit(s1,S) :- S=do(mark(x,1),do(pick(x,1),s0)).
% X..
% ...
% ...
sit(s0,S) :- S=s0.
% ...
% ...
% ...
%%%%%%%%%%
sit(sw6,S) :- S=do(mark(x,3), do(pick(x,3),do(mark(o, 8), do(pick(o,8),do(mark(x, 2),
    do(pick(x,2), do(mark(o, 9), do(pick(o,9),do(mark(x, 1), do(pick(x,1),do(mark(o, 4), do(pick(o,4),
        do(mark(x, 5), do(pick(x,5),s0)))))))))))))).
% XXX
% OX.
% .OO
sit(sw5,S) :- S=do(mark(o,8),do(pick(o,8),do(mark(x,2),do(pick(x,2),do(mark(o,9),do(pick(o,9),
    do(mark(x,1),do(pick(x,1),do(mark(o,4),do(pick(o,4),do(mark(x,5),do(pick(x,5),s0)))))))))))).
% XX.
% OX.
% .OO
sit(sw4,S) :- S=do(mark(x,2),do(pick(x,2),do(mark(o,9),do(pick(o,9),do(mark(x,1),do(pick(x,1),
    do(mark(o,4),do(pick(o,4),do(mark(x,5),do(pick(x,5),s0)))))))))).
% XX.
% OX.
% ..O
```

```
sit(sw3,S) :- S=do(mark(o,9),do(pick(o,9),do(mark(x,1),do(pick(x,1),
    do(mark(o,4),do(pick(o,4),do(mark(x,5),do(pick(x,5),s0)))))))).
% X..
% OX.
% ..O
sit(sw2,S) :- S=do(mark(x,1),do(pick(x,1),do(mark(o,4),do(pick(o,4),
    do(mark(x,5),do(pick(x,5),s0)))))).
% X..
% OX.
% ...
sit(sw1,S) :- S=do(mark(o,4),do(pick(o,4),do(mark(x,5),do(pick(x,5),s0)))).
% ...
% OX.
% ...
sit(sw0,S) :- S=do(mark(x,5),do(pick(x,5),s0)).
% ...
% .X.
% ...
%%%%%%%%%%
sit(sx1,S) :- S=do(mark(x,2),do(pick(x,2),do(mark(x,1),do(pick(x,1),s0)))). % illegal
% XX.
% ...
% ...
sit(sx2,S) :- S=do(mark(x, 2), do(pick(x,2), do(mark(o, 7), do(pick(o,7), do(mark(x, 4),
    do(pick(x,4), do(mark(o, 5), do(pick(o,5), do(mark(x, 1), do(pick(x,1), s0)))))))))).
% XX.
% XO.
% O..

%%% TESTS %%%
% sit(s5,S), holds(ensure([x],wins(x)),S).
% approx 1, 0s
% sit(s4,S), holds(ensure([x],wins(x)),S).
% approx 3, 7s
% sit(s3,S), holds(ensure([x],wins(x)),S).
% approx 9, 10m
% sit(s2,S), holds(ensure([x],wins(x)),S).
% approx 11, 36m
% sit(s1,S), not(holds(ensure([x],wins(x)),S)).
% binding diameter, ???
% sit(s0,S), not(holds(ensure([x],wins(x)),S)).
% binding diameter, ???
%%%%%%%%%%
% sit(sw6,S), holds(ensure([x],wins(x)),S).
% approx 1, 0s
% sit(sw5,S), holds(ensure([x],wins(x)),S).
% approx 3, 5s
% sit(sw4,S), holds(ensure([x],wins(x)),S).
% approx 5, 42s
% sit(sw3,S), holds(ensure([x],wins(x)),S).
% approx 7, 2m
% sit(sw2,S), holds(ensure([x],wins(x)),S).
% approx 9, 13.5m
% sit(sw1,S), holds(ensure([x],wins(x)),S).
% approx 11, 34m
% sit(sw0,S), not(holds(ensure([x],wins(x)),S)).
% binding diameter, ???
%%%%%%%%%%
% sit(sx1,S), not(holds(ensure([x],wins(x)),S)).
% binding diameter, 10s
% sit(sx2,S), not(holds(ensure([x],wins(x)),S)).
% binding diameter, 2.5m
```