Infinite States Verification in Game-Theoretic Logics: Case Studies and Implementation

Slawomir Kmiec and Yves Lespérance

Dept. of Electrical Engineering and Computer Science, York University, Toronto, ON, Canada
skmiec@cse.yorku.ca lesperan@cse.yorku.ca

Abstract. Many practical problems where the environment is not in the system's control can be modelled in game-theoretic logics (e.g., ATL). But most work on verification methods for such logics is restricted to finite state cases. De Giacomo, Lespérance, and Pearce have proposed a situation calculus-based logical framework for representing such infinite state game-type problems together with a verification method based on fixpoint approximates and regression. Here, we extend this line of work. Firstly, we describe some case studies to evaluate the method. We specify some example domains and show that the method does allow us to verify various properties. We also find some examples where the method must be extended to exploit information about the initial state and state constraints in order to work. Secondly, we describe an evaluation-based Prolog implementation of a version of the method for complete initial state theories with the closed world assumption. It generates successive approximates and checks if they hold in the situation of interest. We describe some preliminary experiments with this tool and discuss its limitations.

1 Introduction

Many practical problems where the environment is not completely under the system's control, such as service orchestration, contingent planning, and multi-agent planning, can be modeled as games and specified in game-theoretic logics. There has been much work to define such logics (e.g., Alternating-Time Temporal Logic (ATL)) and develop verification methods for them, mainly model checking techniques [1]. However, most such work is restricted to finite state settings. De Giacomo, Lespérance, and Pearce [8] (hereafter DLP) have developed an expressive logical framework for specifying such problems within the situation calculus [16]. In their approach, a game-like problem/setting is represented as a *situation calculus game structure*, a special kind of action theory that specifies who the players are, what the legal moves are, etc. They also define a logic that combines the μ-calculus, game-theoretic path quantifiers (as in ATL), and first-order quantification, for specifying properties about such game settings. Additionally, they propose a procedural language for defining game settings, GameGolog, which is based on ConGolog [9]. Finally, they propose a method for verifying temporal properties over infinite state game structures that is based on fixpoint approximates and regression.

While DLP give examples to illustrate the expressiveness and convenience of their formalism, they recognize that their work is essentially theoretical and call for experimental studies to understand whether these techniques actually work in practice. This is

what we begin to address in this paper. We develop several example problems involving infinite state domains and represent them as situation calculus game structures. We then examine whether the DLP fixpoint approximates verification method works to verify common temporal properties. In many cases, it does indeed work. So to some extent, our work validates the DLP proposal.

We do however find other examples where the DLP method does not converge in a finite number of steps. We note that the method uses only the simplest part of the action theory, the unique name and domain closure axioms, to try to show that successive approximates are equivalent (after performing regression). Clearly, using the whole action theory is problematic as it includes a second-order axiom to specify the domain of situations. We show that in some cases, adding a few key facts that are entailed by the entire theory (from simple axioms about the initial state to state constraints proven by induction) is sufficient to get convergence in a finite number of steps. This means that the method can be used successfully in a wider range of problems if we can rely on the modeler to identify such facts. Thus, our case studies show that the kind of method proposed by DLP (and related approaches like [5, 6]) often does work for infinite domains, where very few verification methods are available, and allows reasoning about a range of game problems.

Note that in our case studies, the fixpoint approximation method was performed manually. We also describe an evaluation-based Prolog implementation of a version of the method for complete initial state theories with the closed world assumption. It generates successive approximates and checks if they hold in the situation of interest. We describe some experiments with this tool and discuss its limitations.

The paper is organized as follows. In the sext section, we review the situation calculus and the DLP framework for representing infinite state game problems and their verification method. In Section 3, we present our three case studies and discuss the results. In Section 4, we discuss our implementation of the method and how it handles the problems in two of our case studies. In the last section, we review the contributions of this work, discuss related work, and mention some issues for future work.

2 Situation Calculus Game Structures

The situation calculus (SitCalc) is a many-sorted predicate logic language for representing dynamically changing worlds in which all changes are the result of named actions [16, 18]. Actions are terms in the language, e.g., $pickup(R, X)$ could represent an action where a robot R picks up an object X. Action terms are denoted by α possibly with subscripts to differentiate different action terms. Action variables are denoted by lower case letters *a* possibly with subscripts. Action types, i.e., actions functions, which may require parameters, are denoted by upper case letters *A* possibly with subscripts. Situations represent possible world histories and are terms in the language. The distinguished constant S_0 denotes the initial situation where no action has yet been performed. The distinguished function symbol do is used to build sequences of actions such that $do(a, s)$ denotes the successor situation that results from performing action a in situation s. Fluents are predicates or functions whose values may vary from situation to situation. They 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 is physically possible (i.e. executable) in a situation s.

Given this language, one can specify action theories that describe how the world changes as the result of the available actions. We focus on *basic action theories* as proposed in [18]. We assume that there is a *finite number of action types* in the domains we consider. Thus, a basic action theory $\mathcal{D}$ is the union of the following disjoint sets: the foundational, domain independent axioms of the situation calculus (Σ); precondition axioms stating when actions are executable ($\mathcal{D}_{poss}$); successor state axioms describing how fluents change between situations ($\mathcal{D}_{ssa}$); unique name axioms for actions and domain closure on action types ($\mathcal{D}_{ca}$); and axioms describing the initial configuration of the world ($\mathcal{D}_{S_0}$). Successor state axioms specify the value of fluents in situation $do(a, s)$ in terms of the action a and the value of fluents in situation s; they encode the causal laws of the world and provide a solution to the frame problem.

Situation calculus game structures, proposed by DLP, are a specialization of basic action theories that allow multi-agent game-like settings to be modeled. In SitCalc game structures, every action a has an agent parameter and the distinguished function $agent(a)$ returns the agent of the action. Axioms for the $agent$ function are specified 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 are denoted by unique names. 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 $Control$. Standard actions are the other non-choice actions. Choice actions are always physically possible, i.e., for all choice actions a and situations s, $Poss(a, s)$. DLP introduce a distinguished predicate $Legal(s)$ that is a stronger version of possibility/legality and models the game structure of interest. It specifies what actions an agent may execute and what choices can be made according to the rules of the game. The axioms provided for $Legal$ specify the game of interest. It is required that the axioms for $Legal$ entail three properties: (1) $Legal$ implies physically possible ($Poss$), (2) legal situations are the result of an action performed in legal situations, and (3) only one agent can act in a legal situation, i.e., the game is a turn-taking game. $Control(agt, s)$ holds if agent agt is the one that is in control and can act in a legal situation s; it is defined as follows:

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

As a result of the above constraints on $Legal$, it follows that the predicate $Control$ holds for only one agent in any given legal situation. As explained in DLP, games where several agents act simultaneously can often be modeled using a round-robin of choice actions. If the result of such simultaneous choices is non-deterministic, a "game master" agent that makes the decision can be introduced. Note however that the framework assumes that the agents all have complete information and that actions are fully observable. Note also that the state of the game in situation s is captured by the fluents. Finally, DLP define a SitCalc game structure to be an action theory $\mathcal{D}_{GS} = \Sigma \cup \mathcal{D}_{poss} \cup \mathcal{D}_{ssa} \cup \mathcal{D}_{ca} \cup \mathcal{D}_{S_0} \cup \mathcal{D}_{legal}$, where $\mathcal{D}_{legal}$ contains the axioms for $Legal$ and $Control$ and for the function $agent()$, and the other components are as for standard basic action theories [18]. Note that here, a game structure is a type of situation calculus theory and not a single game model as is often the case.

DLP introduce a logical language $\mathcal{L}$ for expressing temporal properties of game structures. It is inspired by ATL [1] and based on the μ-calculus [17], as used over game structures as in [4]. The key element of the $\mathcal{L}$-logic is the $\langle\langle G\rangle\rangle \bigcirc \varphi$ operator defined as follows:

$$\begin{array}{l}\langle\langle G\rangle\rangle \bigcirc \varphi \doteq \\ \quad (\exists agt \in G.\, Control(agt, now) \wedge \\ \qquad \exists a.\, agent(a) = agt \wedge Legal(do(a, now)) \wedge \varphi[do(a, now)]) \vee \\ \quad (\exists agt \notin G.\, Control(agt, now) \wedge \\ \qquad \forall a.\, agent(a) = agt \wedge Legal(do(a, now)) \supset \varphi[do(a, now)])\end{array}$$

This operator, in essence, specifies that a coalition G of agents can ensure that φ holds next, i.e., after one more action, as follows. If an agent from the coalition G is in control in the current situation, then all we need is that there be some legal action that this agent can perform to make the formula φ hold. If the agent in control is not in coalition G, then what we need is that regardless of the action taken by the in-control agent (for all) the formula φ holds after the action. The whole logic $\mathcal{L}$ is defined as follows:

$$\begin{array}{l}\Psi ::= \varphi \mid Z(\boldsymbol{x}) \mid \Psi_1 \wedge \Psi_2 \mid \Psi_1 \vee \Psi_2 \mid \exists x.\Psi \mid \forall x.\Psi \mid \\ \qquad \langle\langle G\rangle\rangle \bigcirc \Psi \mid [[G]] \bigcirc \Psi \mid \mu Z(\boldsymbol{x}).\Psi(Z(\boldsymbol{x})) \mid \nu Z(\boldsymbol{x}).\Psi(Z(\boldsymbol{x})).\end{array}$$

In the above, φ is an arbitrary, possibly open, situation-suppressed situation calculus uniform formula, Z is a predicate variable of a given arity, $\langle\langle G\rangle\rangle \bigcirc \Psi$ is as defined above, $[[G]] \bigcirc \Psi$ is the dual of $\langle\langle G\rangle\rangle \bigcirc \Psi$ (i.e., $[[G]] \bigcirc \Psi \equiv \neg\langle\langle G\rangle\rangle \bigcirc \neg\Psi$[1]), and μ (resp. ν) is the least (resp. greatest) fixpoint operator from the μ-calculus, where the argument is written as $\Psi(Z(\boldsymbol{x}))$ to emphasize that $Z(\boldsymbol{x})$ may occur free, i.e., not quantified by μ or ν, in Ψ.

The language $\mathcal{L}$ allows one to express arbitrary temporal/dynamic properties. For example, the property that group G can ensure that eventually $\varphi(\boldsymbol{x})$ (or has a strategy to achieve $\varphi(\boldsymbol{x})$), where $\varphi(\boldsymbol{x})$ is a situation suppressed formula with free variables $\boldsymbol{x}$, may be expressed by the following least fixpoint construction:

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

Similarly, group G's ability to maintain a property $\varphi(\boldsymbol{x})$ can be expressed by the following greatest fixpoint construction:

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

We say that there is a path where $\varphi(\boldsymbol{x})$ holds next if the set of all agents can ensure that $\varphi(\boldsymbol{x})$ holds next: $\exists \bigcirc \varphi(\boldsymbol{x}) \doteq \langle\langle Agents\rangle\rangle \bigcirc \varphi(\boldsymbol{x})$. Similarly there is a path where $\varphi(\boldsymbol{x})$ eventually holds if the set of all agents has a strategy to achieve $\varphi(\boldsymbol{x})$: $\exists \Diamond \varphi(\boldsymbol{x}) \doteq \langle\langle Agents\rangle\rangle \Diamond \varphi(\boldsymbol{x})$.

DLP propose a procedure based on regression and fixpoint approximation to verify formulas of logic $\mathcal{L}$ given a SitCalc game structure theory. This recursive procedure

[1] Although $\neg\langle\langle G\rangle\rangle \bigcirc \neg\Psi$ is not in $\mathcal{L}$ according to the syntax, the equivalent formula in negation normal form is.

$\tau(\Psi)$ tries to compute a first-order formula uniform in current situation now that is equivalent to Ψ:

$$\begin{array}{l}
\tau(\varphi) = \varphi \\
\tau(Z) = Z \\
\tau(\Psi_1 \wedge \Psi_2) = \tau(\Psi_1) \wedge \ \tau(\Psi_2) \\
\tau(\Psi_1 \vee \Psi_2) = \tau(\Psi_1) \vee \tau(\Psi_2) \\
\tau(\exists x.\Psi) = \exists x.\tau(\Psi) \\
\tau(\forall x.\Psi) = \forall x.\tau(\Psi) \\
\tau(\langle\langle G\rangle\rangle \bigcirc \Psi) = \mathcal{R}(\langle\langle G\rangle\rangle \bigcirc \tau(\Psi)) \\
\tau([[G]] \bigcirc \Psi) = \neg\mathcal{R}(\langle\langle G\rangle\rangle \bigcirc \tau(\mathrm{NNF}(\neg\Psi))) \\
\tau(\mu Z.\Psi) = \mathit{lfp}Z.\tau(\Psi) \\
\tau(\nu Z.\Psi) = \mathit{gfp}Z.\tau(\Psi)
\end{array}$$

In the above, $\mathcal{R}$ represents the regression operator and $\langle\langle G\rangle\rangle \bigcirc \Psi$ is regressable if Ψ is regressable, $\mathrm{NNF}(\neg\Psi)$ denotes the negation normal form of $\neg\Psi$, and $\mathit{lfp}Z.\Psi$ and $\mathit{gfp}Z.\Psi$ are formulas resulting from the following least and greatest fixpoint procedures:

$\mathit{lfp}Z.\Psi =$
$R := False;$
$R_{new} := \Psi(False);$
while $(\mathcal{D}_{ca} \not\models R \equiv R_{new})\{$
$R := R_{new};$
$R_{new} := \Psi(R)\ \}$

$\mathit{gfp}Z.\Psi =$
$R := True;$
$R_{new} := \Psi(True);$
while $(\mathcal{D}_{ca} \not\models R \equiv R_{new})\{$
$R := R_{new};$
$R_{new} := \Psi(R)\ \}$

The fixpoint procedures test if $R \equiv R_{new}$ is entailed given only the unique name and domain closure for actions axioms $\mathcal{D}_{ca}$. In general, there is no guarantee that such procedures will ever terminate i.e., that for some i $\mathcal{D}_{ca} \models R_i \equiv R_{i+1}$. But if the *lfp* procedure does terminate, then $\mathcal{D}_{GS} \models R_i[S] \equiv \mu Z.\Psi(Z)[S]$ and R_i is first-order and uniform in now (and similarly for *gfp*). In such cases, the task of verifying a fixpoint formula in the situation calculus is reduced to that of verifying a first-order formula. We have the following result:

Theorem 1. of DLP [8]: *Let $\mathcal{D}_{GS}$ be a situation calculus game structure and let Ψ be an $\mathcal{L}$-formula. If the algorithm above terminates, then $\mathcal{D}_{GS} \models \Psi[S_0]$ if and only if $\mathcal{D}_{S_o} \cup \mathcal{D}_{ca} \models \tau(\Psi)[S_0]$.*

3 Case Studies

3.1 Light World (LW)

Our first example domain is the Light World (LW), a simple game we designed that involves an infinite row of lights, one for each integer. A light can be on or off. Each light has a switch that can be flipped, which will turn the light on (resp., off) if it was off (resp., on). There are 2 players, X and O. Players take turns and initially it is X's turn. The goal of player X is to have lights 1 and 2 on in which case player X wins the game. Initially, lights 1 and 2 are known to be off and light 5 is known to be on. Note that this is clearly an infinite state domain as the set of lights that can be turned on or off

is infinite. Note also that the game may go on forever without the goal being reached (e.g., if player O keeps turning light 1 or 2 off whenever X turns them on).

We will show that the DLP method can be used to verify some interesting properties in this domain. We apply the method with one small modification: when checking whether the two successive approximates are equivalent, we use an axiomatization of the integers D_Z in addition to the unique names and domain closure axioms for actions D_{ca}^{LW}, as our game domain involves one light for every integer.[2] The game structure axiomatization for this domain is:

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

We have only one action $flip(p,t)$, meaning that player p flips light t, with the precondition axiom (in $\mathcal{D}_{poss}^{LW}$): $Poss(flip(p,t),s) \equiv Agent(p)$. We have the fluents $On(t,s)$, meaning that light t is on in situation s, and $turn(s)$, a function that denotes the agent whose turn it is in s. The successor state axioms (in $\mathcal{D}_{ssa}^{LW}$) are as follows:

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

The rules of the game are specified using the $Legal$ predicate. We have the following axioms in $\mathcal{D}_{legal}^{LW}$:

$$Legal(do(a,s)) \equiv Legal(s) \wedge \exists p,t.\ Agent(p) \wedge turn(s) = p \wedge a = flip(p,t)$$
$$Control(p,s) \doteq \exists a.Legal(do(a,s)) \wedge agent(a) = p$$
$$agent(flip(p,t)) = p, \qquad \forall p.\{Agent(p) \equiv (p = X \vee p = O)\}, \qquad X \neq O$$

Thus legal moves involve the player whose turn it is flipping any switch. We have the following unique name and domain closure axioms for actions in $\mathcal{D}_{ca}^{LW}$:

$$\forall a.\ \{\ \exists p,t.\ a = flip(p,t)\}$$
$$\forall p,p',t,t'.\ \{\ flip(p,t) = flip(p',t') \supset p = p' \wedge t = t'\ \}$$

Finally, the initial state axioms in $\mathcal{D}_{S_0}^{LW}$ are: $turn(S_0) = X$, $\neg On(1,S_0)$, $\neg On(2,S_0)$, $On(5,S_0)$, and $Legal(S_0)$.

For our first verification example, we consider the property that it is possible for X to eventually win assuming O cooperates, which can be represented by the following formula:

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

where $Wins(X,s) \doteq Legal(s) \wedge On(1,s) \wedge On(2,s)$. We apply the DLP method to this example. We can show that the regressed approximations simplify as follows (see

[2] Our axioms and the properties we attempt to verify only use a very simple part of integer arithmetic. It should be possible to generate the proofs using the decidable theory of Presburger arithmetic [11] after encoding integers as pairs of natural numbers in the standard way [12]. 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.

[14] for more detailed versions of all proofs in this paper):

$\mathcal{D}_{ca}^{LW} \models R_0(s) \doteq Wins(X,s) \vee \mathcal{R}(\exists \bigcirc False) \equiv$
$Legal(s) \wedge On(1,s) \wedge On(2,s)$
This approximation evaluates to true if s is such that X is winning in s already (in no steps), i.e., if light 1 and light 2 are on in s.

$\mathcal{D}_{ca}^{LW} \cup D_Z \models R_1(s) \doteq Wins(X,s) \vee \mathcal{R}(\exists \bigcirc R_0) \equiv$
$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$
$Legal(s) \wedge (turn(s) = X \vee turn(s) = O) \wedge On(1,s) \vee$
$Legal(s) \wedge (turn(s) = X \vee turn(s) = O) \wedge On(2,s)$
This approximation evaluates to true if s is such that X can win in at most 1 step; these are legal situations where player X is already winning or where one of lights 1 or 2 is on, as X or O can turn the other light on at the next step.

$\mathcal{D}_{ca}^{LW} \cup D_Z \models R_2(s) \doteq Wins(X,s) \vee \mathcal{R}(\exists \bigcirc R_1) \equiv$
$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$
$Legal(s) \wedge (turn(s) = X \vee turn(s) = O)$
This approximation evaluates to true if s is such that X can win in at most 2 steps; this is the case if X is winning already or if s is any legal situation where it is X or O's turn, as the controlling player can turn light 1 on at the next step and the other player can and light 2 on at the second step.

$\mathcal{D}_{ca}^{LW} \cup D_Z \models R_3(s) \equiv Wins(X,s) \vee \mathcal{R}(\exists \bigcirc R_2) \equiv$
$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$
$Legal(s) \wedge (turn(s) = X \vee turn(s) = O)$
The fixpoint iteration procedure converges at the 4^{th} step as we have: $\mathcal{D}_{ca}^{LW} \cup D_Z \models R_2(s) \equiv R_3(s)$. Note that it can be shown using the entire theory (by induction on situations) that $\mathcal{D}_{GS}^{LW} \models R_2(s) \equiv Legal(s)$, as it is always either X's or O's turn. Thus, it is possible for X to eventually win in any legal situation. It then follows by Theorem 1 of DLP that: $\mathcal{D}_{GS}^{LW} \models \exists \Diamond Wins(X)[S_0]$ if and only if $\mathcal{D}_{GS}^{LW} \models Legal(S_0) \wedge \{On(1,S_0) \wedge On(2,S_0) \vee turn(S_0) = X \vee turn(S_0) = O\}$. By the initial state axioms, the latter holds so $\mathcal{D}_{GS}^{LW} \models \exists \Diamond Wins(X)[S_0]$, i.e., player X can eventually win in the initial situation.

For our second example, we look at the property that X can ensure that he/she eventually wins no matter what O does, i.e., the existence of a strategy that ensures $Wins(X)$. This 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 apply the DLP method to try to verify this property. We can show that the regressed approximations simplify as follows:

$\mathcal{D}_{ca}^{LW} \cup \mathcal{D}_Z \models R_0(s) \doteq Wins(X,s) \vee \mathcal{R}(\langle\langle\{X\}\rangle\rangle \bigcirc False) \equiv$
$Legal(s) \wedge On(1,s) \wedge On(2,s)$
This approximation evaluates to true if s is such that X is already winning in s (in no steps); these are situations where lights 1 and 2 are already on.

$\mathcal{D}_{ca}^{LW} \cup \mathcal{D}_Z \models R_1(s) \doteq Wins(X,s) \vee \mathcal{R}(\langle\langle\{X\}\rangle\rangle \bigcirc R_0) \equiv$
$Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$

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

This approximation evaluates to true if s is such that X can ensure it wins in at most 1 step. This holds if lights 1 and 2 are already on or if either light 1 or 2 is on and it is X's turn, as X can then turn the other light on at the next step.

The next approximate R_2 simplifies to the same formula as R_1 and $\mathcal{D}_{ca}^{LW} \cup \mathcal{D}_Z \models R_1(s) \equiv R_2(s)$, so the fixpoint iteration procedure converges in the 3^{rd} step. Therefore by Theorem 1 of DLP: $\mathcal{D}_{GS}^{LW} \models \langle\langle\{X\}\rangle\rangle\Diamond Wins(X)[S_0] \equiv R_1(S_0)$ Since both lights 1 and 2 are off initially, it follows by the initial state axioms that $\mathcal{D}_{GS}^{LW} \models \neg\langle\langle\{X\}\rangle\rangle\Diamond Wins(X)[S_0]$, i.e., there is no winning strategy for X in S_0. However, we also have that $\mathcal{D}_{GS}^{LW} \models \langle\langle\{X\}\rangle\rangle\Diamond Wins(X)[S_1]$, where $S_1 = do(flip(O, 3), do(flip(X, 1), S_0))$, i.e., X has a winning strategy in the situation S_1 where X first turned light 1 on and then O flipped light 3, as X can turn on light 2 next.

Note that when the fixpoint approximation method is able to show that a coalition can ensure that a property holds eventually, the theory is complete, and we have domain closure, we can always extract a strategy that the coalition can follow to achieve the property: a strategy works if it always selects actions for the coalition that get it from one approximate to a lower approximate (R_i to R_{i-1}).

3.2 Oil Lamp World (OLW)

The DLP 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 D_{ca} (to which we have added the axiomatization of the integers). We now 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 entire theory D_{GS}^{OLW}, including the initial state axioms, when checking if successive approximates are equivalent, then we do get convergence in a finite number of steps.

Consider the Oil Lamp World (OLW), a variant of the Light World (LW) domain discussed earlier. It also involves an infinite row of lamps, one for each integer, which can be on or off. A lamp has an igniter that can be flipped. When this happens, the lamp will go on provided that the lamp immediately to the right is already on, i.e., flipping the igniter for lamp t will turn it on if lamp $t + 1$ is already on. There is only one agent, X. The goal of X is to have lamp 1 on, in which case X wins. Observe that the game may go on indefinitely without the goal being reached, e.g., if X keeps flipping a lamp other than lamp 1 repeatedly.

The game structure axiomatization for this domain is: $\mathcal{D}_{GS}^{OLW} = \Sigma \cup \mathcal{D}_{poss}^{OLW} \cup \mathcal{D}_{ssa}^{OLW} \cup \mathcal{D}_{ca}^{OLW} \cup \mathcal{D}_{S_0}^{OLW} \cup \mathcal{D}_{Legal}^{OLW} \cup \mathcal{D}_Z$. As in the previous example, we have only one action, $flip(p, t)$, meaning that p flips the igniter on light t, with the following precondition axiom (in $\mathcal{D}_{poss}^{OLW}$): $Poss(flip(p, t), s) \equiv Agent(p)$. But there is no turn taking in this game as there is only one agent X. We have the successor state axiom (in $\mathcal{D}_{ssa}^{OLW}$):

$$On(t, do(a, s)) \equiv \exists p\, a = flip(p, t) \wedge On(t + 1, s) \vee On(t, s).$$

Note that once a lamp is turned on it remains on. The axioms in $\mathcal{D}_{legal}^{OLW}$ specifying the rules of the game are similar to the ones given earlier for the Light World domain, and

include:

$$Legal(do(a,s)) \equiv Legal(s) \wedge \exists p,t.\ Agent(p) \wedge a = flip(p,t).$$

Thus legal moves involve X flipping any igniter. The unique name and domain closure axioms for actions and the initial state axioms are exactly as in the Light World example.

We are interested in verifying the property that it is possible for X to eventually win, which is represented by the following formula:

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

where $Wins(X,s) \doteq Legal(s) \wedge On(1,s)$. We begin by applying the DLP 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 $\mathcal{D}_Z$. We can show that the regressed approximations simplify as follows:

$\mathcal{D}_{ca}^{OLW} \cup \mathcal{D}_Z \models R_0(s) \doteq Wins(X,s) \vee \mathcal{R}(\exists \bigcirc False) \equiv Legal(s) \wedge On(1,s)$
This approximation evaluates to true if s is such that X is already winning (in no steps); these are situations where lamp 1 is on.

$\mathcal{D}_{ca}^{OLW} \cup \mathcal{D}_Z \models R_1(s) \doteq Wins(X,s) \vee \mathcal{R}(\exists \bigcirc R_0) \equiv$
$\quad Legal(s) \wedge (On(1,s) \vee On(2,s))$
This approximation evaluates to true if s is such that X can win in at most 1 step; these are legal situations where either lamp 1 is on or where lamp 2 is on, and then X can turn lamp 1 on at the next step.

$\mathcal{D}_{ca}^{OLW} \cup \mathcal{D}_Z \models R_2(s) \doteq Wins(X,s) \vee \mathcal{R}(\exists \bigcirc R_1) \equiv$
$\quad Legal(s) \wedge (On(1,s) \vee On(2,s) \vee On(3,s))$
This approximation evaluates to true if s is such that X can win in at most 2 steps; these are legal situations where either lamp 1 is on, or where lamp 2 is on (and then X can turn lamp 1 on at the next step), or where lamp 3 is on (and then X can turn on lamps 2 and 1 at the next steps).

We can generalize and show that for all natural numbers i,

$$D_{ca}^{OLW} \cup D_Z \models R_i \equiv Legal(s) \wedge \bigvee_{1 \le j \le i+1} On(j,s).$$

That is, X can win in at most i steps if some lamp between 1 and $i+1$ is on. It follows that for all 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 except $i+2$ is off. Thus, the plain DLP method fails to converge in a finite number of steps.

Nonetheless, there is a way to strengthen the DLP method to get convergence in a finite number of steps. The idea is to use some facts that are entailed by the entire theory in addition to the unique name and domain closure axioms for actions D_{ca}^{OLW} and the integer axioms D_Z. First, we can show by induction on situations that any lamp that is on in the initial situation will remain on forever, i.e.,

$$\mathcal{D}_{GS}^{OLW} \models \forall k(On(k,S_0) \supset \forall s\, On(k,s)).$$

Then, it follows that for any natural numbers $i, j, i \leq j$,

$$\mathcal{D}_{ca}^{OLW} \cup \mathcal{D}_Z \cup \{On(i+1, S_0), \forall k(On(k, S_0) \supset \forall s\, On(k, s))\} \models R_j \equiv Legal(s).$$

In essence, X can eventually win in any legal situation where some lamp n is known to be on. It follows that:

$$\mathcal{D}_{ca}^{OLW} \cup \mathcal{D}_Z \cup \{On(i+1, S_0), \forall k(On(k, S_0) \supset \forall s\, On(k, s))\} \models R_i \equiv R_{i+1}.$$

Thus, the 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. Moreover, our initial state axioms include $On(5, S_0)$. Thus, $\mathcal{D}_{GS}^{OLW} \models \exists \Diamond Wins(X)[S_0]$, i.e., X can eventually win in the initial situation, as it is legal and lamp 5 is on.

We can also show by induction on situations that if all lamps are off initially, they will remain so forever:

$$\mathcal{D}_{GS}^{OLW} \setminus \mathcal{D}_{S_0}^{OLW} \models (\forall k\, \neg On(k, S_0)) \supset (\forall s \forall k\, \neg On(k, s)).$$

Then, we can show by a similar argument as above that the fixpoint approximation method converges in a finite number of steps if we use the facts that all lamp are off initially and that if all lamps are off initially, they remain off forever.

3.3 In-Line Tic-Tac-Toe (TTT1D)

Our final example domain is more like a traditional game. It involves a one-dimensional version of the well-known Tic-Tac-Toe game that is played on an infinite vector of cells, one for each integer. We show that the DLP method does work to verify both the possibility of winning and the existence of a winning strategy in this game, although in the former case the proof is long and tedious. There are two players, X and O, that take turns, with X playing first. All cells are initially blank, i.e., marked B. Players can only put their mark at the left or right edge of the already marked area. The functional fluent $curn$ denotes the marking position on the left (negative) side of the marked area and similarly $curp$ denotes the marking position on the right (positive) side of the marked area. Initially, $curn$ refers to cell 0 and $curp$ to cell 1. Player p can put its mark in the cell on the left (negative) side of the marked area, i.e., the cell referred to by $curn$, by doing the action $markn(p)$. This also decreases the value $curn$ by 1 so that afterwards, it points to the next cell on the left. There is an analogous action $markp(p)$ for marking the cell on the right (positive) side of the marked area denoted by $curp$. A player wins if it succeeds in putting its mark in 3 consecutive cells. E.g., if initially we have the following sequence of moves: $[markp(X), markn(O), markp(X), markn(O), markp(X)]$, then in the resulting situation the board is as follows:

$$\ldots, B_{-3}, B_{-2}, O_{-1}, O_0, X_1, X_2, X_3, B_4, B_5, \ldots$$

(with the subscript indicating the cell number) and X wins. Note that the game may go on indefinitely without either player winning, for instance if player O always mimics the last move of player X.

The game structure axiomatization for this domain is: $\mathcal{D}_{GS}^{T^3 1D} = \Sigma \cup \mathcal{D}_{poss}^{T^3 1D} \cup \mathcal{D}_{ssa}^{T^3 1D} \cup \mathcal{D}_{ca}^{T^3 1D} \cup \mathcal{D}_{S_0}^{T^3 1D} \cup \mathcal{D}_{Legal}^{T^3 1D} \cup \mathcal{D}_Z$. The precondition axioms (in $\mathcal{D}_{poss}^{T^3 1D}$) state that the actions $markn(p)$ and $markp(p)$ are always possible if p is an agent. The successor state axioms (in $\mathcal{D}_{ssa}^{LW}$) are as follows:

$$curn(do(a,s)) = k \equiv$$
$$\exists p.\{a = markn(p)\} \wedge curn(s) = k+1 \vee curn(s) = k \wedge \forall p.\{a \neq markn(p)\}$$

$$curp(do(a,s)) = k \equiv$$
$$\exists p.\{a = markp(p)\} \wedge curp(s) = k-1 \vee curp(s) = k \wedge \forall p.\{a \neq markn(p)\}$$

$$cell(k, do(a,s)) = p \equiv$$
$$a = markp(p) \wedge curp(s) = k \ \vee a = markn(p) \wedge curn(s) = k \vee$$
$$cell(k,s) = p \wedge \neg\exists p'.\{a = markp(p') \wedge curp(s) = k\}$$
$$\wedge \neg\exists p'.\{a = markn(p') \wedge curn(s) = k\}$$

$$turn(do(a,s)) = p \equiv agent(a) = X \wedge p = O \wedge turn(s) = X$$
$$\vee\, agent(a) = O \wedge p = X \wedge turn(s) = O$$

The rules of the game are specified (in $\mathcal{D}_{legal}^{T^3 1D}$) as follows:

$$Legal(do(a,s)) \equiv Legal(s) \wedge$$
$$\exists p.\{\, turn(s) = p \wedge agent(a) = p \wedge (a = markn(p) \ \vee a = markp(p))\,\}$$

$$Control(p,s) \doteq \exists a.Legal(do(a,s)) \wedge agent(a) = p$$
$$agent(markn(p)) = p, \qquad agent(markp(p)) = p$$
$$\forall p.\ \{Agent(p) \equiv (p = X \vee p = O)\}, \qquad X \neq O$$

The unique name and domain closure axioms for actions are specified in the usual way. Finally, we have the following initial state axioms in $\mathcal{D}_{S_0}^{T^3 1D}$: $curn(S_0) = 0$, $curp(S_0) = 1$, $turn(S_0) = X$, and $Legal(S_0)$.

We first consider whether it is possible for X to eventually win $\exists\Diamond Wins(X)$, where

$$Wins(p,s) \doteq \exists k(Legal(s) \wedge$$
$$((curn(s) = k-2 \wedge cell(k-1,s) = p \wedge cell(k,s) = p \wedge cell(k+1,s) = p) \vee$$
$$(curp(s) = k+2 \wedge cell(k+1,s) = p \wedge cell(k,s) = p \wedge cell(k-1,s) = p)))$$

(Note that this simple definition allows both players to win.) If we apply the original DLP method to this property (using only the unique name and domain closure axioms for actions $\mathcal{D}_{ca}^{T^3 1D}$ and the axiomatization of the integers $\mathcal{D}_Z$ to show that successive approximates are equivalent), the fixpoint approximation procedure does eventually converge, but only after 11 steps. 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; these state constraints are consequences of the initial state axioms and successor state axioms. So our proof has to deal with numerous cases where there are non-blank cells to the left of $curn$ or to the right of $curp$ (if we use these state constraints, the proof becomes much simpler). We omit the detailed proof (which appears in [14]). But we have that:

$$\mathcal{D}_{ca}^{T^3 1D} \cup \mathcal{D}_Z \models R_{10}(s) \doteq Wins(X,s) \vee \mathcal{R}(\exists \bigcirc R_9) \equiv Legal(s)$$

Thus, it is possible for X to win in at most 10 steps in all legal situations. Moreover we have that $\mathcal{D}_{ca}^{T^3 1D} \cup \mathcal{D}_Z \models R_{10}(s) \equiv R_{11}(s)$, and thus the fixpoint approximation procedure converges in the 11^{th} step. There are situations where it does take at least 10 steps/moves for X to win, for instance if we have $curp < curn$ with two blank cells in between, i.e., $\uparrow_p BB \uparrow_n$, where $\uparrow_n$ represents the position of $curn$ and similarly for $\uparrow_p$ and $curp$, and it is O's turn. The fact that $curp < curn$ means that the initial marks that are made will later be overwritten. It is straightforward to check that it takes at least 10 moves for X to have 3 X's in a row and win (O wins as well), for instance if O keeps playing $markn$ and X keeps playing $markp$. It follows from our convergence result by Theorem 1 of DLP that: $\mathcal{D}_{GS}^{T^3 1D} \models \exists\Diamond Wins(X)[S_0] \equiv R_{10}(S_0) \equiv Legal(S_0)$. Since we have $Legal(S_0)$ in the initial state axioms, it follows that $\mathcal{D}_{GS}^{T^3 1D} \models \exists\Diamond Wins(X)[S_0]$, i.e., it is possible for X to win in the initial situation.

Finally, we consider the property that X can ensure that it eventually wins $\langle\langle\{X\}\rangle\rangle\Diamond Wins(X)$. We can apply the original DLP method to this property (using only the unique name and domain closure axioms for actions $\mathcal{D}_{ca}^{T^3 1D}$ and the axiomatization of the integers $\mathcal{D}_Z$ to show that successive approximates are equivalent). We can show that the regressed approximations simplify as follows:

$\mathcal{D}_{ca}^{T^3 1D} \cup \mathcal{D}_Z \models R_0(s) \doteq Wins(X,s) \vee \mathcal{R}(\langle\langle\{X\}\rangle\rangle \bigcirc False) \equiv Wins(X,s)$

$\mathcal{D}_{ca}^{T^3 1D} \cup \mathcal{D}_Z \models R_1(s) \doteq Wins(X,s) \vee \mathcal{R}(\langle\langle\{X\}\rangle\rangle \bigcirc R_0)$
$\equiv R_0(s) \vee XCanPlayToWinNext(s)$

where $XCanPlayToWinNext(s) \doteq Legal(s) \wedge turn(s) = X \wedge$
$(\exists k.(curn(s) = k-1 \wedge cell(k,s) = X \wedge cell(k+1,s) = X) \vee$
$\exists k.(cell(k-2,s) = X \wedge cell(k-1,s) = X \wedge curp(s) = k) \vee$
$\exists k.(cell(k-2,s) = X \wedge cell(k-1,s) = X \wedge curn(s) = k \wedge curp(s) = k+1) \vee$
$\exists k.(curn(s) = k-2 \wedge curp(s) = k-1 \wedge cell(k,s) = X \wedge cell(k+1,s) = X) \vee$
$\exists k.(cell(k-2,s) = X \wedge curn(s) = k-1 \wedge cell(k,s) = X \wedge curp(s) = k+1) \vee$
$\exists k.(curn(s) = k-2 \wedge cell(k-1,s) = X) \wedge curp(s) = k \wedge cell(k+1,s) = X))$

This approximation evaluates to true if s is such that X can ensure to win in at most 1 step. These are legal situations where there are 3 X marks in a row on either side, i.e. $\uparrow_n XXX$ or $XXX \uparrow_p$, or where it is X's turn and there are 2 X marks already and X can fill in the missing cell to get 3 in a row, i.e. $\uparrow_n XX$ or $XX \uparrow_p$ or $\uparrow_n \uparrow_p XX$ or $XX \uparrow_n \uparrow_p$ or $\uparrow_n X \uparrow_p X$ or $X \uparrow_n X \uparrow_p$.

$\mathcal{D}_{ca}^{T^3 1D} \cup \mathcal{D}_Z \models R_2(s) \doteq Wins(X,s) \vee \mathcal{R}(\langle\langle\{X\}\rangle\rangle \bigcirc R_1) \equiv$
$R_1(s) \vee Legal(s) \wedge turn(s) = O \wedge$
$\exists m.(curn(s) < m-2 \wedge cell(m-2,s) = X \wedge cell(m-1,s) = X \wedge curp(s) = m) \wedge$
$\exists n.(curn(s) = n-1 \wedge cell(n,s) = X \wedge cell(n+1,s) = X \wedge n+1 < curp(s))$

This approximation evaluates to true if s is such that X can ensure to win in at most 2 steps. These are legal situations where X can ensure to win in at most 1 step as above, or where it is O's turn and we have both $X_k X \uparrow_p$ with $\uparrow_n < k$ and $\uparrow_n XX_k$ with $\uparrow_p > k$; then if O plays $markn$ then X can play $markp$ to win afterwards, and if O plays $markp$ then X can play $markn$ to win afterwards. The next approximation $R_3(s)$ simplifies to exactly the same formula as $R_2(s)$. Thus the procedure converges in the 4^{th} step as we have: $\mathcal{D}_{GS}^{T^3 1D} \cup \mathcal{D}_Z \models R_2(s) \equiv R_3(s)$. Therefore by Theorem 1 of DLP: $\mathcal{D}_{GS}^{T^3 1D} \models \langle\langle\{X\}\rangle\rangle\Diamond Wins(X)[S_0] \equiv R_2(S_0)$. It follows by

the initial state axioms that $\mathcal{D}_{GS}^{T^3{}_{1D}} \models \neg\langle\langle\{X\}\rangle\rangle\Diamond Wins(X)[S_0]$ i.e., there is no winning strategy for X in S_0. But $\mathcal{D}_{GS}^{T^3{}_{1D}} \models \langle\langle\{X\}\rangle\rangle\Diamond Wins(X)[S_1]$, where $S_1 = do($ $[markp(X), markn(O), markp(X), markn(O)], S_0)$, i.e., there is a winning strategy for X in a situation where X has marked twice on the right and O has marked twice on the left. We have also developed two other examples of games played on an infinite vector of cells to evaluate the DLP method; see [14] for details.

4 An Evaluation-Based Verification Tool

To further examine the feasibility of automating the DLP method, we have developed an evaluation-based Prolog implementation of a version of the method for complete initial state theories with the closed world assumption. The system can correctly verify many properties in infinite state game structures. The method is completely automated, unlike most theorem proving-based approaches. Here "evaluation-based" refers to the use of evaluation instead of entailment to check state properties under the condition of complete information (i.e., single model) and the closed-world assumption. The verifier is domain-independent. One major limitation of the current prototype is that it does not actually check for convergence of the fixpoint approximation, and thus may not terminate when the DLP method does, as we discuss later.

Our verifier builds on the logic programming evaluator for situation calculus projection queries developed by Reiter [18] for complete initial state theories with the closed world assumption. That approach uses a Prolog encoding of the domain's basic action theory as defined in [18]. For example, for the In-Line Tic-Tac-Toe domain, we have:

```
% Precondition Axioms
poss(markn(P),S) :- agent(P).
poss(markp(P),S) :- agent(P).
% Successor State Axioms
curn(K,do(A,S)) :- A=markn(_), curn(KX,S), K is KX - 1;
   not(A=markn(_)), curn(K,S).
curp(K,do(A,S)) :- A=markp(_), curp(KX,S), K is KX + 1;
   not(A=markp(_)), curp(K,S).
cell(K,M,do(A,S)) :- A=markp(M), curp(K,S); A=markn(M), curn(K,S);
   (not(A=markn(M)); not(curn(K,S))),
   (not(A=markp(M)); not(curp(K,S))), cell(K,M,S).
turn(P,do(A,S)) :- turn(x,S), P = o; turn(o,S), P = x.
legal(do(A,S)) :- turn(P,S), (A=markn(P) ; A=markp(P)), legal(S).
% Initial State Axioms
cell(_,b,s0). % all cells are initially blank
curn(0,s0). curp(1,s0). turn(x,s0). legal(s0).
```

One can evaluate projection queries using such a program, e.g., check whether `cell(2,b,do(markp(x),s0))`, i.e., that cell 2 is still blank after agent X marks right in the initial situation. The program works essentially by regressing the query to the initial situation and evaluating it against the initial state axioms. Regression involves replacing fluent atoms by the instantiated right-hand side of their successor state axiom, thus transforming a query about a situation into an equivalent one about the previous situation. For example, `cell(2,b,do(markp(x),s0))` is regressed into

```
markp(x)=markp(b), curp(2,s0);
markp(x)=markn(b), curn(2,s0);
(not(markp(x)=markn(b)); not(curn(2,s0))),
(not(markp(x)=markp(b)); not(curp(2,s0))), cell(2,b,s0)
```

which succeeds because the last disjunct holds according to the encoded initial state axioms and the unique name assumption.

Reiter [18] shows how to define an evaluator for a rich set of first-order queries on top of such an encoding of the basic action theory. Here is some of the evaluator code:

```
holds(P & Q,S) :-!, holds(P,S), holds(Q,S). % conjunction
holds(P v Q,S) :-!, (holds(P,S); holds(Q,S)). % disjunction
holds(some(V,P),S) :-!, subst(V,_,P,P1), holds(P1,S). %existential
% handled by replacing the variable by a fresh Prolog variable
holds(all(V,P),S) :-!, holds(-some(V,-P),S). % universal
...
% handling negation
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(-all(V,P),S) :- !, holds(some(V,-P),S).
holds(-P,S) :- not(holds(P,S)).
% handling atoms
holds(Pred,S) :- restoreSitArg(Pred,S,PredEx), !, PredEx.
```

The evaluator recursively evaluates the arguments of conjunctions and disjunctions. Existential quantification is left for Prolog to handle. Universal quantification is rewritten using negation and existential quantification. Negation is distributed over conjunction and disjunction. Finally, atomic fluents are regressed and evaluated using the Prolog encoding of the basic action theory.

Our verifier checks if a given temporal property expressed in the $\mathcal{L}$-Logic holds for a given situation. It is defined by extending Reiter's evaluator. We handle the key temporal operator $\langle\langle G\rangle\rangle \bigcirc \Psi[S]$ essentially by translating it into its situation calculus definition, and then checking the result in the usual way using a combination of regression and evaluation:

```
holds(canEnsureNext(G,F),S) :- !, (
   incontrol(G,S), holds(exists_successor(G,F),S);
   incontrol(-G,S), holds(forall_successors2(-G,F),S)).
holds(exists_successor(G,F),S) :- !, member(P,G),
   agent_action(P, A), S1=do(A,S), legal(S1), holds(F,S1), !.
holds(forall_successors2(-G,F),S) :- !,
   not(holds(exists_successor2(-G,-F),S)).
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 $[[G]] \bigcirc \Psi[S]$ case is handled as $\neg\langle\langle G\rangle\rangle \bigcirc \neg\Psi[S]$.

The μ and ν operators are handled by generating successive fixpoint approximates R_i as in the DLP method, except that we bound the number of approximates generated and we do not check for convergence, we simply check if the successive approximates hold in the situation of interest S:

```
holds(mu(z,F),S) :- !, mu_approx(z,F,false,1,S).
holds(nu(z,F),S) :- !, mu_approx(z,F,true,1,S).
mu_approx(Z,F,Int,N,S) :- binding_diameter(Max), N>Max, !,
   write('binding diameter '), write(N),
   write(' reached - stop'), nl, !, fail.
mu_approx(Z,F,Int,N,S) :- subst(Z,Int,F,Fx), holds(Fx,S), !,
   output1(N,Fx).
mu_approx(Z,F,Int,N,S) :- M is N+1, subst(Z,Int,F,Int2), !,
   mu_approx(Z,F,Int2,M,S).
```

By not checking for convergence, i.e. whether $\mathcal{D} \models R_{i+1} \equiv R_i$, we avoid the need for complex logical reasoning requiring theorem proving techniques. The downside is that the verifier will never terminate on $\mu Z.\Psi$ queries that are false even if the fixpoint approximation converges, as it does not detect this. To ensure termination, the user may impose a bound on the number of approximates that are generated and evaluated. The idea is similar to the binding diameter concept in bounded model checking [3]. In some cases, the bound can be a number of moves that is reasonable in the game modeled. The formula $\langle\langle G\rangle\rangle\Diamond\Psi$ is defined in terms of the μ operator as $\mu Z.\Psi \vee \langle\langle G\rangle\rangle \bigcirc Z$:

```
holds(canEnsureEventually(G,F),S):-
   !,holds(mu(z,F v canEnsureNext(G,z)),S).
```

For this, our verifier generates fixpoint approximates and evaluates them in the given situation S, stopping as soon as one of the approximates evaluates to true:

let $R_0 := \Psi \vee \langle\langle G\rangle\rangle \bigcirc False$ and evaluate $R_0[S]$; if it succeeds, return success;
else let $R_1 := \Psi \vee \langle\langle G\rangle\rangle \bigcirc R_0$ and evaluate $R_1[S]$; if it succeeds, return success;
...
else let $R_{limit} := \Psi \vee \langle\langle G\rangle\rangle \bigcirc R_{limit-1}$ and evaluate $R_{limit}[S]$; if it succeeds, return success;
else return failure.

We have tested our verifier on some of our infinite state game structure examples. On the In-Line Tic-Tac-Toe domain, the verifier can confirm that both agents can cooperate to ensure that X wins (in 5 steps) in the initial situation, i.e., the following query succeeds after generating and evaluating 6 approximates:

```
?- holds(canEnsureEventually([x,o],wins(x)),s0).
trying ##### approximation 1 ---> wins(x) v next([x, o], false)
[...]
trying ##### approximation 6 ---> wins(x) v
   next([x, o], wins(x) v
      next([x, o], wins(x) v
         next([x, o], wins(x) v
```

```
            next([x, o], wins(x) v
               next([x, o], wins(x) v next([x, o], false))))))
[...]
> successor EXISTS for G --->
     next([x, o], wins(x) v next([x, o], false)) ---> for
     do(markn(o), do(markp(x), do(markn(o), do(markn(x), s0))))
[...]
> ##### approximation 6 holds --->
[...]
wins(x) v
   next([x, o], wins(x) v
      next([x, o], wins(x) v
         next([x, o], wins(x) v
            next([x, o], wins(x) v
               next([x, o], wins(x) v next([x, o], false))))))
yes
```

As part of doing the verification, the system finds a sequence of actions by the two cooperating agents that allows X to win.

The verifier can also confirm that agent X can ensure that it wins (in 1 step) in the situation $do(markn(o), do(markp(x), do(markn(o), do(markp(x), s0))))$, where X has already put 2 marks on the right and O had already put 2 marks on the left. However, if we try to check if X can ensure that it wins in the situation $do(markp(x), do(markn(o), do(markp(x), s0)))$, where X has already put 2 marks on the right and O had already put 1 mark on the left, the verifier cannot confirm that the query is in fact false; it keeps generating successive approximates and eventually gives up after reaching the binding diameter. The problem is that O can always prevent X from winning at the next step and the verifier is not checking whether it has converged to a fixpoint in the approximation.

We have also tested our verifier on the Light World domain. This is more challenging because there are infinitely many legal actions at every state, as any switch can be flipped. The verifier succeeds in confirming that X can ensure that it wins in the situation where it flipped light 2 on initially and then O flipped light 4 on, as X can win in one step by flipping light 1 on next. But it cannot confirm that the two agents can cooperate to ensure that X eventually wins in the initial situation S_0. The problem is that this requires two steps (where X first flips light 1 or 2 on and then O flips the other one on) and there is an infinite number of flipping actions that can be performed at the first step, all of which must be considered before concluding that X cannot win in one step in S_0. If we bound the set of switches that are considered (e.g., only allow flipping the first 10 switches), then the verifier will be able to successfully verify that the query holds. It first establishes that X cannot win in one step (e.g., by flipping any of the 10 available switches) and then succeeds in finding a sequence of two actions that allows X to win. However, bounding the set of switches essentially makes the game finite state and changes what temporal properties hold. A better approach would be to modify the game to allow the set of switches considered to be progressively expanded, perhaps by a neutral agent. The important thing is to allow more actions/branches in a state only as longer sequences of actions are considered.

Additionally, the verifier cannot show that X cannot ensure that it eventually wins in the situation where it has already flipped light 2 on (as O can flip it off next and continue undoing any progress that X makes towards the goal). The verifier succeeds in showing that O can prevent X from winning at the next step (O can flip any switch except 1). It then generates the third approximate and tries to show that X can win in one step after every action that O makes next. If we bound the set of switches that are considered, the verifier can confirm that X cannot win in two steps as O can flip light 2 off next. The verifier keeps generating and evaluating successive approximates and eventually gives up after reaching the binding diameter. It does not check whether successive approximates are equivalent, and thus fails to detect that the fixpoint approximation has converged after generating the fourth approximate.

We have also tested our verifier on a formalization of the standard 2D Tic-Tac-Toe game (used as an example in [8]), a finite state domain. In this case the verifier can do a complete search and correctly answers queries about the existence of a winning strategy. For example, it can confirm that X cannot ensure that it eventually wins in the initial situation with a blank board; it can also confirm that X can ensure that it eventually wins in a situation where X has marked the center square and O has then marked a non-corner square.

To summarize, in finite state domains the verifier correctly answers queries as it can do a complete search. In infinite state domains, our verifier can often show that least fixpoint queries are true but cannot show that least fixpoint queries are false (and greatest fixpoint queries are true), because it does not check whether successive approximates are equivalent. We hope to address this in future work.

In many cases, we would like to verify properties assuming that agents are following certain strategies, or have certain strategic preferences. For example, in standard 2D Tic-Tac-Toe, one might know that a player always tries to mark corners first. This would allow modelling more realistic types of agents. It can also cut down significantly on the number of alternative actions that must be considered and speed up verification. Knowing that the opponent follows certain strategic preferences may provide the player with a way to ensure it eventually wins when it could not otherwise.

We have extended the DLP formalization to support this. There are many ways to model strategic preferences. A simple approach is to assume that the modeler defines a predicate $Preferred(p, a, s)$ that holds if and only if action a is a preferred action for player p in situation s. Note that there may be several alternative preferred actions in a situation. Other specifications of strategic preferences can be mapped to this form.

It is straightforward to modify the logic to only consider paths where all players select actions according to their preferences. We change the semantics of the $\langle\langle G\rangle\rangle \bigcirc \Psi$ operator as follows. If a player in G is in control in the current situation, Ψ must hold after some preferred action for him if there is one; if there is no preferred action, Ψ must hold after some legal action. If a player not in G is in control, Ψ must hold after all preferred actions for him if there is some preferred action, and after all legal actions if there is none. This means that $Preferred(p, a, s)$ represents soft constraints. If there are no preferred actions in a situation, we revert to considering all legal actions. Our implementation supports this type of specification of player action preferences and we have tested it on some standard 2D Tic-Tac-Toe examples.

Our verifier also supports the use of the GameGolog language proposed in DLP to specify the game structure procedurally. See [14] for more details. The current prototype implemented in SWI Prolog (`www.swi-prolog.org`), together with some examples, is available at `www.cse.yorku.ca/~skmiec/SCGSverifier/`. We believe that our verifier implementation is sound (assuming a "proper" Prolog interpreter is used, i.e., one that flounders on negative queries with free variables). It is not complete, in part for the same reasons that Prolog is not a complete reasoner for first order logic. We leave the proof of soundness for future work.

5 Discussion and Related and Future Work

In this paper, we described the results of some case studies to evaluate whether the DLP verification method actually works. We developed various infinite state game-type domains and applied the method to them. Our example domains are rather simple, but have features present in practical examples (e.g., the $T^3 1D$ domain is 1D version of Tic-Tac-Toe on an infinite board). Our experiments do confirm that the method does work on several non-trivial 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. Our example domains and properties should be useful for evaluating other approaches to infinite state verification and synthesis.

We also described an evaluation-based Prolog implementation of a version of the DLP method for complete initial state theories with the closed world assumption. It generates successive approximates and checks if they hold in the situation of interest, but does not check if the sequence of approximates converges. Our verifier is fully automatic, unlike most theorem proving-based tools. We have also extended the framework to allow agents' strategic preferences to be represented and used in verification. See [14] for more details about our verification experiments, proofs, and implemented verifier.

Among related work that deals with verification in infinite-states domains, let us mention [5, 6], which also uses methods based on fixpoint approximation. There, characteristic graphs are introduced to finitely represent the possible configurations that a Golog program representing a multi-agent interaction may visit. Their specification language is rich modal variant of the situation calculus with first and second order quantifiers, temporal operators and path quantifiers as in CTL*, and dynamic logic operators labeled with Golog programs. However, the language does not include fixpoint operators or alternating-time quantifiers, and is not a game structure logic. In their verification procedure, like DLP, they check for convergence using only the unique name axioms for actions part of the action theory. Also closely related is [19], which uses a fixpoint approximation method to compose a target process expressed as a ConGolog program out of a library of available ConGolog programs. Earlier, [13] proposed a fixpoint approximation method to verify a class of temporal properties in the situation calculus, called property persistence formulas. [20] shows how a theorem proving tool can be used to verify properties of multi-agent systems specified in ConGolog and an extended

situation calculus with mental states. A leading example of a symbolic model checker for multi-agent systems is MCMAS [15]. [2] shows that model checking of an expressive temporal language on infinite state systems is decidable if the active domain in all states remains bounded. As well, [10] shows that verification of temporal properties in bounded situation calculus theories where there is a bound on the number of fluent atoms that are true in any situation is decidable. [7] identifies an interesting class of Golog programs and action theories for which verification is decidable.

In future work, we would like to further develop our evaluation-based verifier. We plan to extend it to perform limited symbolic reasoning to detect if successive approximates are equivalent. We will also do more experimental evaluation. We would also like to implement an open-world symbolic version of the DLP method, perhaps by writing proof tactics in a theorem proving environment. It would also be desirable to develop techniques for identifying initial state properties and state constraints that can be used to show finite convergence in cases where these are needed. More generally, we need a better characterization of when this kind of method can be used successfully. The DLP framework assumes that only one agent can act in any situation, and that all agents have complete knowledge of the situation and that actions are fully observable. As a first step, it would be interesting to extend it to support synchronous moves by multiple agents. Going further, the framework should be generalized to deal with private knowledge and partial observability. Finally, the approach should be evaluated on real practical problems.

References

1. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. J. ACM 49(5), 672–713 (2002)
2. Belardinelli, F., Lomuscio, A., Patrizi, F.: An abstraction technique for the verification of artifact-centric systems. In: Brewka, G., Eiter, T., McIlraith, S.A. (eds.) Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012. pp. 319–328. AAAI Press (2012)
3. Biere, A.: Bounded model checking. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability, pp. 457–481. IOS Press (2009)
4. Bradfield, J., Stirling, C.: Modal mu-calculi. In: Handbook of Modal Logic, vol. 3, pp. 721–756. Elsevier (2007)
5. Claßen, J., Lakemeyer, G.: A logic for non-terminating Golog programs. In: Brewka, G., Lang, J. (eds.) Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference, KR 2008, Sydney, Australia, September 16-19, 2008. pp. 589–599. AAAI Press (2008)
6. Claßen, J., Lakemeyer, G.: On the verification of very expressive temporal properties of non-terminating Golog programs. In: Coelho, H., Studer, R., Wooldridge, M. (eds.) ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings. pp. 887–892. IOS Press (2010)
7. Claßen, J., Liebenberg, M., Lakemeyer, G.: On decidable verification of non-terminating Golog programs. In: Proceedings of the 10th International Workshop on Nonmonotonic Reasoning, Action and Change (NRAC 2013), Beijing, China. pp. 13–20 (2013)
8. De Giacomo, G., Lesperance, Y., Pearce, A.R.: Situation calculus-based programs for representing and reasoning about game structures. In: Lin, F., Sattler, U., Truszczynski, M. (eds.)

Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference, KR 2010, Toronto, Ontario, Canada, May 9-13, 2010. pp. 445–455 (2010)
9. De Giacomo, G., Lespérance, Y., Levesque, H.J.: ConGolog, a concurrent programming language based on the situation calculus. Artificial Intelligence 121(1–2), 109–169 (2000)
10. De Giacomo, G., Lespérance, Y., Patrizi, F.: Bounded Situation Calculus Action Theories and Decidable Verification. In: Brewka, G., Eiter, T., McIlraith, S.A. (eds.) Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012. pp. 467–477. AAAI Press (2012)
11. Enderton, H.B.: A mathematical introduction to logic. Academic Press (1972)
12. Hamilton, A.G.: Numbers, Sets and Axioms: The Apparatus of Mathematics. Cambridge University Press (1982)
13. Kelly, R.F., Pearce, A.R.: Property persistence in the situation calculus. Artif. Intell. 174(12-13), 865–888 (2010)
14. Kmiec, S.: Infinite States Verification in Game-Theoretic Logics. Master's thesis, Dept. of Electrical Engineering and Computer Science, York University, Toronto, Canada (2013)
15. Lomuscio, A., Qu, H., Raimondi, F.: MCMAS: A model checker for the verification of multi-agent systems. In: Bouajjani, A., Maler, O. (eds.) Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5643, pp. 682–688. Springer (2009)
16. McCarthy, J., Hayes, P.: Some philosophical problems from the standpoint of artificial intelligence. In: Machine Intelligence, vol. 4, pp. 463–502. Edinburgh University Press (1969)
17. Park, D.: Finiteness is mu-ineffable. Theor. Comput. Sci. 3(2), 173–181 (1976)
18. Reiter, R.: Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press (2001)
19. Sardina, S., De Giacomo, G.: Composition of ConGolog programs. In: Boutilier, C. (ed.) IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009. pp. 904–910 (2009)
20. Shapiro, S., Lespérance, Y., Levesque, H.J.: The cognitive agents specification language and verification environment for multiagent systems. In: The First International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2002, July 15-19, 2002, Bologna, Italy, Proceedings, pp. 19–26. ACM Press (2002)