Solving the Inferential Frame Problem in the General Game Description Language

Javier Romero
Universität Potsdam
javier@cs.uni-potsdam.de

Abdallah Saffidine
University of New South Wales
abdallahs@cse.unsw.edu.au

Michael Thielscher
University of New South Wales
mit@cse.unsw.edu.au

Abstract

The Game Description Language GDL is the standard input language for general game-playing systems. While players can gain a lot of traction by an efficient inference algorithm for GDL, state-of-the-art reasoners suffer from a variant of a classical KR problem, the inferential frame problem. We present a method by which general game players can transform any given game description into a representation that solves this problem. Our experimental results demonstrate that with the help of automatically generated domain knowledge, a significant speedup can thus be obtained for the majority of the game descriptions from the AAAI competition.

Introduction

General Game Playing (GGP) research seeks to design systems able to understand new games and use such descriptions to play those games effectively. These systems cannot be endowed with game-specific algorithms because only at runtime they will be told the rules of a game. The general Game Description Language (GDL) (Love et al. 2006; Thielscher 2010) has been developed for this purpose and has become the standard at international GGP competition since 2005 (Genesereth, Love, and Pell 2005; Genesereth and Björnsson 2013). GDL uses formal logic to provide players with a specification of the initial game state, the possible moves and how they change the position, and the winning conditions. Players also are informed about how much time they have—the so-called STARTCLOCK—to analyse the rules and reason their way to a strategy; and how much time then to decide on every move—the PLAYCLOCK—once the game has started.

For the selection of moves, two major approaches have been applied successfully in general game playing. These are depth-limited game tree search along with solution concepts like such as Minimax (Kuhlmann, Dresner, and Stone 2006; Clune 2007; Schiffel and Thielscher 2007); and guided random simulations along with statistical analysis (Finnsson and Björnsson 2008; Méhat and Cazenave 2011). For both methods the quality of play depends crucially on the number of nodes in the game tree that can be visited under the strict time constraints of the given PLAYCLOCK. While ultimately a good strategy makes the difference, a lot of traction can be gained by an efficient computation of search nodes (Genesereth and Björnsson 2013).

Game trees in GGP are built by inferring the possible moves and successor states from the logical descriptions of preconditions and effects in GDL. Efficient methods exist that use tailored data structures for this purpose (Schkufza, Love, and Genesereth 2008; Waugh 2009; Kissmann and Edelkamp 2010; Saffidine and Cazenave 2011). But a recent comparative assessment of state-of-the-art reasoners has shown that still they are significantly slower than handmade, game-specific algorithms for state update, so bridging this gap is an open problem (Schiffel and Björnsson 2013).

This is a quintessentially KRR problem: A given representation of knowledge needs to be automatically turned into one that is more efficient to reason with. Specifically, reasoners for GDL face the classical *inferential frame problem* (McCarthy and Hayes 1969; Reiter 1991; Thielscher 1999): If used directly, the logical game description requires inferring all components of a successor state anew, rather than updating the current state description only by the components that change (Schiffel and Björnsson 2013).

We address this problem by developing a method for automatically transforming any given GDL description into a representation that solves the inferential frame problem. General game-playing systems apply this translation during the STARTCLOCK in order to boost the efficiency of the state update computation for their game tree search when they play the game later. The transformation consists of several steps. First, the given GDL axioms for state update are rewritten into syntactically equivalent formulas that describe the changes between successor states. Second, the resulting formulas are subjected to automatic logical simplification. Finally, and crucial for matching the efficiency of a game-specific implementation, the representation is simplified further based on domain-specific knowledge. The latter needs to be established from the game rules through automated theorem proving (Haufe, Schiffel, and Thielscher 2012).

Our results in this paper are three-fold. We present a method by which any GDL game description is automatically turned into a representation that can be used directly to infer state updates without suffering from the inferential frame problem. We provide a formal proof of the correct-

```
1  role(x).
2  role(o).
3  init(cell(1,1,b)).
4  ...
5  init(cell(3,3,b)).
6  init(control(x)).
7
8  legal(P,mark(M,N)) :- true(control(P)), true(cell(M,N,b)).
9  legal(x,noop) :- true(control(o)).
10 legal(o,noop) :- true(control(x)).
11 next(cell(M,N,x)) :- does(x,mark(M,N)).          %r1
12 next(cell(M,N,o)) :- does(o,mark(M,N)).          %r2
13
14 next(cell(M,N,O)) :-                             %r3
15     does(P,mark(J,K)),
16     true(cell(M,N,O)),
17     ( distinct(M,J) or distinct(N,K) ).
18
19 next(control(x)) :- true(control(o)).            %r4
20 next(control(o)) :- true(control(x)).            %r5
```

Figure 1: A GDL description of Tic-Tac-Toe (without the definition of termination and goalhood). Positions are encoded using the features *control*(P), where $P \in \{x, o\}$; and *cell*(X, Y, Z), where $X, Y \in \{1, 2, 3\}$ and $Z \in \{x, o, b\}$ (with $b \mathrel{\hat{=}}$ "blank").

ness of this translation, and we present experimental results obtained with a state-of-the-art reasoner, Fluxplayer (Schiffel and Thielscher 2007; Schiffel and Björnsson 2013), that demonstrate a significant efficiency gain for the majority of the GDL game descriptions from the AAAI GGP competition (Genesereth and Björnsson 2013).

Our result has implications beyond the specific game description language used for GGP. The update axioms in GDL can be viewed (Schiffel and Thielscher 2011) as general successor state axioms (SSAs), which provide the standard solution (Reiter 1991) to the frame problem—the representational aspect thereof, to be more specific—in the classical KR formalism for reasoning about actions, the Situation Calculus (McCarthy and Hayes 1969). Any standard reasoner for actions based on some form of SSAs can therefore profit from our techniques if a straightforward implementation requires to compute unchanged fluents individually.

The remainder of the paper proceeds as follows. After a brief introduction to GDL, we show how game descriptions can be rewritten into a direct representation of the changes effected by moves. We prove the correctness of the overall translation. We show how the result can be further simplified using automatically derived, game-specific knowledge. We present our experimental results and conclude.

Game Description Language

GDL has been developed to formalise the rules of any finite game with complete information in such a way that the description can be automatically processed by a GGP system.

Game descriptions in GDL are sets of *normal logic program clauses* (Lloyd 1987) that may include negation and disjunction in the bodies. Especially designed for game descriptions, the language uses a few pre-defined predicates:

role(R)	*R* is a player
init(F)	*F* holds in the initial position
true(F)	*F* holds in the current position
legal(R,M)	player *R* has legal move *M*
does(R,M)	player *R* does move *M*
next(F)	*F* holds in the next position
terminal	the current position is terminal
goal(R,N)	player *R* gets goal value *N*
distinct(X,Y)	*X*,*Y* syntactically different terms

In GDL it is assumed that gameplay happens synchronously, that is, all players move simultaneously and the world changes only in response to moves.

As an example, Figure 1 shows an excerpt of a GDL description for the simple game of Tic-Tac-Toe.[1] GDL imposes some syntactic restrictions on a set of clauses with the intention to ensure an unambiguous interpretation of the rules and finiteness of the set of derivable predicate instances (Love et al. 2006). Let it suffice to say that any valid GDL description

- must be stratified (Apt, Blair, and Walker 1987) and allowed (Lloyd and Topor 1986);
- has predicate *next* only in the head of clauses while *true* and *does* can only appear in clause bodies.

Stratified logic programs are known to admit a unique answer set (Gelfond 2008).

The semantics of a set of game rules has been informally described by a state transition system by Genesereth, Love, and Pell (2005) and later formalised as follows (Schiffel and Thielscher 2010). Let $G \models A$ denote that atom A is true in the unique answer set of a stratified set of rules G. The *players* in game G are $R = \{r : G \models \mathit{role}(r)\}$. The *initial state* is $\{f : G \models \mathit{init}(f)\}$. A move m is *legal in state* $S = \{f_1, \ldots, f_k\}$ if $G \cup S^{\mathtt{true}} \models \mathit{legal}(r, m)$, where

$$S^{\mathtt{true}} \stackrel{\text{def}}{=} \{\mathit{true}(f_1)., \mathit{true}(f_2)., \ldots, \mathit{true}(f_k).\}$$

A *joint move* M in state S assigns each role $r \in R$ a legal move. The *state transition* from S via M is defined by

$$\{f : G \cup S^{\mathtt{true}} \cup M^{\mathtt{does}} \models \mathit{next}(f)\} \qquad (1)$$

Here, $S^{\mathtt{true}}$ is as above and

$$M^{\mathtt{does}} \stackrel{\text{def}}{=} \{\mathit{does}(r_1, m_1)., \ldots, \mathit{does}(r_n, m_n).\}$$

assuming that joint move M assigns m_i to player r_i. Finally, a *terminal state* is any S such that $G \cup S^{\mathtt{true}} \models \mathit{terminal}$; and a *goal value* for player $r \in R$ in state S is any v for which $G \cup S^{\mathtt{true}} \models \mathit{goal}(r, v)$ holds.

Solving the Inferential Frame Problem in GDL

The inferential frame problem in GDL is the problem of having to compute anew all the elements of a state that persist

[1] A word on the syntax: We use a Prolog-style notation for GDL rules as we find this more readable than the usual prefix notation used at competitions. Variables are denoted by uppercase letters.

after a move. For example, in Tic-Tac-Toe when a player marks a cell then all the other cells remain unchanged. In GDL this persistence has to be represented explicitly by a frame rule, as $r3$ in Fig. 1. Hence, when a reasoner updates a state it has to recompute the values of all the cells that persist. In contrast, consider the following rule:

```
fnext(cell(M,N,O)):- does(P,mark(M,N)),true(cell(M,N,O)).
```

where *fnext*$/1$ is a new predicate to represent that a true element changes to false in the next state. Intuitively, this *fnext rule* says that a cell loses its previous mark when it gets marked by a player. In contrast to frame rule $r3$, which has to be executed once for every individual cell that persists, the *fnext* rule only has to be executed once per turn.

Our solution to the inferential frame problem in GDL is to translate frame rules such as $r3$ into *fnext* rules and then use the latter instead. In this way, rather than having to compute everything that persists, the reasoner will compute everything that changes to false. In most of the domains there are far less changes to false than persisting state features, so we expect an improvement in the performance of the reasoner.

Preliminaries

Let $\phi = \{v_1/t_1, \ldots, v_n/t_n\}$. ϕ is said to be a substitution when each v_i is a variable and each t_i is a term, and for no distinct i and j is v_i the same as v_j. We denote by $Eq(\phi)$ the formula $\bigwedge_{v_i/t_i \in \phi} v_i = t_i$. Let l be a literal and let $\phi = \{v_1/t_1, \ldots, v_n/t_n\}$ be a substitution. The instantiation of l by ϕ, written $l\phi$, is formed by replacing every occurrence of v_i in l by t_i. The instantiation of a set of literals L by ϕ, written $L\phi$, is the set of the instantiations of the literals of L by ϕ. Atom a subsumes atom b, or $a \succeq b$, if there is a substitution ϕ such that $a\phi = b$. For sets of atoms A and B, we say that A subsumes B, or $A \succeq B$, if there is a substitution ϕ such that $A\phi \supseteq B$.

We consider logic programming notation as an alternative to first order logic notation, interpreting the symbol $\leftarrow$ as logical implication, every comma as $\wedge$, every *or* as $\vee$ and every *not* as $\neg$, and assuming all free variables are universally quantified from the outside. We identify a set of formulas with the conjunction of its elements, and sometimes we omit the curly braces among the elements of a set. For a rule r, by $head_r$ we refer to the atom in the head, by $body_r$ we refer to the set of conjuncts in the body, and by $does_r$ we refer to the set of positive literals of predicate *does*$/2$ in the body. A *next rule* is a rule whose head is an atom of predicate *next*$/1$, and by G_{next} we refer to the next rules of a GDL description G. An *fnext rule* is a rule whose head is an atom of predicate *fnext*$/1$. We assume that *fnext*$/1$ does not appear anywhere in a given GDL description. Let r be a next rule with head *next*(f), then we say that r is a *frame rule* if *true*(f) belongs to the body of the rule. In this case we denote by $cond_r$ the condition $body_r \setminus (\{\textit{true}(f)\} \cup does_r)$ and we may write r as:

$$\textit{next}(f) \leftarrow \textit{true}(f) \wedge does_r \wedge cond_r$$

For example, in the Tic-Tac-Toe description, the next rules are $r_1, r_2, \ldots, r_5$ (cf. the labels in Fig. 1); $r3$ is the unique frame rule; $does_{r_3}$ is $\{does(P, mark(J, K))\}$; and $cond_{r_3}$ is $\{M \neq J \vee N \neq K\}$. Let P be a set of predicates and A a set of ground atoms of predicates from P, the *closed world assumption* (Reiter 1978) over predicates P in A, written *CWA*$_P(A)$, is the set of formulas $p(\vec{x}) \leftrightarrow \bigvee_{p(\vec{t}) \in A} \vec{x} = \vec{t}$ for every $p \in P$, where $\vec{x}$ is a tuple of distinct variables, $\vec{t}$ is a tuple of ground terms and $\vec{x} = \vec{t}$ is interpreted as usual. We may define the state transition from a state S via a joint move M as:

$$\{f : \textit{CWA}_P(A) \cup G_{next} \models_{FOL} \textit{next}(f)\} \qquad (2)$$

where A is the unique answer set of the program $G' = (G \setminus G_{next}) \cup S^{\texttt{true}} \cup M^{\texttt{does}}$, P are the predicates that appear in G', and $\models_{FOL}$ is the entailment relationship in first-order logic under the unique names assumption. Using the fact that predicate *next*$/1$ does not appear in G', the following theorem is easy to prove.

Theorem 1 *The definitions of the state transitions (1) and (2) are equivalent.*

The Translation

We describe the translation for a GDL description G with a set of players R. We say that f is a fluent of G if $f \in S$ for some state S of G. For the simple case where G is ground and there is a single player $p \in R$, for every fluent f and every move m we have the *fnext* rule:

$$\textit{fnext}(f) \leftarrow true(f) \wedge does(p, m) \wedge \bigwedge_{r \in \mathcal{F}r(f,m)} \neg cond_r$$

where $\mathcal{F}r(f, m)$ is the set of frame rules r with head *next*(f) and $does_r \subseteq \{does(p, m)\}$. Intuitively, the rule says that f changes to false by move m if none of the conditions hold for f to persist. For this simple case the translation is straightforward: For every fluent f and every move m we replace the frame rules from G by the new *fnext* rules, and we modify the state transition definition so that f belongs to a successor state if either *next*(f) is entailed or f is true in the previous state and *fnext*(f) is not entailed.

The general case requires two generalizations. First, the translation is defined for possibly non ground atoms a of predicate *next*$/1$. For any such atom a the frame rules in G, written $\mathcal{F}r_G(a)$, are the set of frame rules $r \in G$ such that $a \succeq head_r$. For example, r_3 is a frame rule for atom *next*$(cell(X, Y, Z))$ but not for *next*$(cell(X, Y, b))$.

Second, the translation is defined for a set of possibly non ground atoms of predicate $does/2$. For this we introduce the concept of a general joint move. We say that M is a *joint move* of G if it is a joint move in some state S of G. A *general joint move* $\mathcal{M}$ of G is a set of $|R|$ atoms of predicate *does*$/2$ such that the first terms of two distinct elements of $\mathcal{M}$ are different. Intuitively, a general joint move is a set of $does/2$ atoms that can be instantiated to a joint move. Every joint move M of G is also a general joint move of G, and the set $\bigcup_{1 \leq i \leq |R|} \{does(P_i, A_i)\}$ is a general joint move of any G with $|R|$ players. For the Tic-Tac-Toe description,

$$\mathcal{M}_1 = \{does(x, noop), does(o, noop)\}$$
$$\mathcal{M}_2 = \{does(x, noop), does(o, mark(A, B))\}$$
$$\mathcal{M}_3 = \{does(P_1, noop), does(P_2, mark(A, B))\}$$

are some other general joint moves. For a general joint move $\mathcal{M}$, by $\mathcal{I}_{\mathcal{M}}$ we denote the set of inequalities $p_i \neq p_j$ for every two distinct atoms $does(p_i, m_i)$, $does(p_j, m_j) \in \mathcal{M}$. For example, $\mathcal{I}_{\mathcal{M}_1}$ and $\mathcal{I}_{\mathcal{M}_2}$ are $\{x \neq o\}$ and $\mathcal{I}_{\mathcal{M}_3}$ is $\{P_1 \neq P_2\}$.

Translation for an atom and a general joint move

We first define the translation for an atom a of predicate *next*/1 and a general joint move $\mathcal{M}$ of G. We assume that a and $\mathcal{M}$ have different variables that do not appear in G.

The set of unifiers of a rule $r \in \mathcal{F}r_G(a)$ with a and $\mathcal{M}$, written $\mathcal{U}_r(a, \mathcal{M})$, is the set of substitutions ϕ such that $(head_r \cup does_r)\phi \subseteq (a \cup \mathcal{M})\phi$.

For example, $\mathcal{U}_{r_3}(\textit{next}(cell(X, Y, Z)), \mathcal{M}_1) = \emptyset$, while $\mathcal{U}_{r_3}(\textit{next}(cell(X, Y, Z)), \mathcal{M}_2)$ has many elements, including $\{X/M, Y/N, Z/O, A/J, B/K, P/o\}$.

For a rule $r \in \mathcal{F}r_G(a)$, if $\mathcal{M}$ subsumes $M^{\texttt{does}}$ for a joint move M, then we may rewrite r as $r(a, \mathcal{M})$:

$$\textit{next}(f) \leftarrow \textit{true}(f) \wedge \mathcal{M} \wedge \mathcal{I}_{\mathcal{M}} \wedge cond_r \wedge \bigvee_{\phi \in \mathcal{U}_r(a,\mathcal{M})} Eq(\phi)$$

The set $\mathcal{I}_{\mathcal{M}}$ is added to guarantee that if the body holds then $\mathcal{M}$ is instantiated to a joint move. Formally, we have the following proposition.

Proposition 1 *Let a, $\mathcal{M}$ and r be defined as above. For any joint move M in a state S such that $\mathcal{M} \succeq M^{\texttt{does}}$, the sets*

$$\{f : \textit{CWA}_P(A) \cup r \models_{FOL} \textit{next}(f)\} \quad (3)$$

and

$$\{f : \textit{CWA}_P(A) \cup r(a, \mathcal{M}) \models_{FOL} \textit{next}(f)\} \quad (4)$$

are the same (A and P are as in (2)).

Proof: ($\Rightarrow$) Let $g \in (3)$, we prove that $g \in (4)$. If $g \in (3)$ then there is a substitution α such that α only mentions variables in r, $head_r\alpha$ and $body_r\alpha$ are ground, $\textit{CWA}_P(A) \models_{FOL} body_r\alpha$ and $head_r\alpha$ is $\textit{next}(g)$. Given that $r \in \mathcal{F}r_G(a)$ there is a substitution β that only mentions variables in a such that $a\beta = head_r$. Also, given that $\mathcal{M} \succeq M^{\texttt{does}}$ there is a substitution γ that only mentions variables in $\mathcal{M}$ such that $\mathcal{M}\gamma = M^{\texttt{does}}$. Let δ be the composition $\beta\gamma\alpha$ and let $head_r$ be of the form $\textit{next}(h)$. We show that all the elements of $body_{r(a,\mathcal{M})}\delta$ are entailed by $\textit{CWA}_P(A)$: $true(f)\delta$ is entailed because $true(f)\beta = true(h)$ and $true(h)\alpha$ is entailed by $\textit{CWA}_P(A)$, $(\mathcal{M} \wedge \mathcal{I}_{\mathcal{M}})\delta$ is entailed because $\mathcal{M}\gamma$ is equal to $M^{\texttt{does}}$, which is entailed by $\textit{CWA}_P(A)$, $cond_r\delta$ is entailed because $cond_r\alpha$ is entailed by $\textit{CWA}_P(A)$, and for the disjunction $\bigvee_{\phi \in \mathcal{U}_r(a,\mathcal{M})} Eq(\phi)$ we have that $\delta \in \mathcal{U}_r(a, \mathcal{M})$ and $Eq(\delta)\delta$ is a tautology. Finally, given that $a\beta = head_r$ and $head_r\alpha$ is $\textit{next}(g)$ we have that $a\delta$ is $\textit{next}(g)$, and given that the elements of $body_{r(a,\mathcal{M})}\delta$ are entailed by $\textit{CWA}_P(A)$ we have that $\textit{next}(g)$ is entailed by $\textit{CWA}_P(A) \cup r(a, \mathcal{M})$ and $g \in (4)$.

($\Leftarrow$) Similar to the other direction. □

In $r(a, \mathcal{M})$ we are considering all substitutions $\phi \in \mathcal{U}_r(a, \mathcal{M})$, but we only need to consider the most general ones. For this, we define a selection of unifiers of r with a and $\mathcal{M}$ as a set $S \subseteq \mathcal{U}_r(a, \mathcal{M})$ such that: (1) for every $\alpha \in \mathcal{U}_r(a, \mathcal{M})$, there exists $\beta \in S$ such that $(head_r \cup does_r)\beta \succeq (head_r \cup does_r)\alpha$, and (2) for every $\alpha, \beta \in S$, if $\alpha \neq \beta$, then $(head_r \cup does_r)\alpha \not\succeq (head_r \cup does_r)\beta$. The selections of most general unifiers are unique up to variable renaming, so to simplify the presentation we abuse notation by referring to *the* selection of most general unifiers, denoted by $\mathcal{U}^*_r(a, \mathcal{M})$. Then we may define $r^*(a, \mathcal{M})$ as we did for $r(a, \mathcal{M})$ but using $\mathcal{U}^*_r(a, \mathcal{M})$ instead of $\mathcal{U}_r(a, \mathcal{M})$, and Proposition 1 still holds if we use $r^*(a, \mathcal{M})$ in (4) instead of $r(a, \mathcal{M})$. In the Tic-Tac-Toe description, $r^*_3(\textit{next}(cell(X, Y, Z)), \mathcal{M}_2)$ is:

$$\begin{aligned}&\textit{next}(cell(X, Y, Z)) \leftarrow \textit{true}(cell(X, Y, Z)) \wedge \\ &\quad does(x, noop) \wedge does(o, mark(A, B)) \wedge x \neq o \wedge \\ &\quad (M \neq J \vee N \neq K) \wedge \\ &\quad (X = M \wedge Y = N \wedge Z = O \wedge A = J \wedge B = K \wedge P = o)\end{aligned}$$

The set of rules $\{r^*(a, \mathcal{M}) \mid r \in \mathcal{F}r_G(a)\}$ represents the persistence of the fluents g such that $a \succeq \textit{next}(g)$ when a joint move M with $\mathcal{M} \succeq M^{\texttt{does}}$ is performed. Then we may define the translation of a for $\mathcal{M}$ in G, written $\mathcal{T}_G(a, \mathcal{M})$, as the first-order *fnext* rule:

$$\begin{aligned}\textit{fnext}(f) \leftarrow \textit{true}(f) \wedge \mathcal{M} \wedge \mathcal{I}_{\mathcal{M}} \wedge \\ \Big(\bigwedge_{r \in \mathcal{F}r_G(a)} \bigwedge_{\phi \in \mathcal{U}^*_r(a,\mathcal{M})} \neg\exists\vec{x}(cond_r \wedge Eq(\phi))\Big)\end{aligned}$$

where a is $\textit{next}(f)$ and $\vec{x}$ are the variables appearing in r. We add the existential quantifier inside the rule because we have to make sure that for no possible substitution of the variables the conditions hold. Intuitively, the rule says that an instance of f changes to false after doing an instance of $\mathcal{M}$ if none of the conditions hold for f to persist. This generalizes the simple case, where all the elements are ground and $\mathcal{M}$ is a singleton set. In the Tic-Tac-Toe description, $\mathcal{T}_G(\textit{next}(cell(X, Y, Z)), \mathcal{M}_2)$ is:

$$\begin{aligned}&\textit{fnext}(cell(X, Y, Z)) \leftarrow \textit{true}(cell(X, Y, Z)) \wedge \\ &\quad does(x, noop) \wedge does(o, mark(A, B)) \wedge x \neq o \wedge \\ &\quad \neg\exists MJNKOP \\ &\qquad \big((M \neq J \vee N \neq K) \wedge \\ &\qquad\quad X = M \wedge Y = N \wedge Z = O \wedge A = J \wedge B = K \wedge P = o\big)\end{aligned}$$

The following proposition states formally the role of $\mathcal{T}_G(a, \mathcal{M})$.

Proposition 2 *Let a and $\mathcal{M}$ be defined as above. For any joint move M in a state S such that $\mathcal{M} \succeq M^{\texttt{does}}$, the sets*

$$\{f : \textit{CWA}_P(A) \cup \mathcal{F}r_G(a) \models_{FOL} \textit{next}(f)\} \quad (5)$$

and

$$\begin{aligned}\{f : a \succeq \textit{next}(f), f \in S^{\texttt{true}} \textit{ and} \\ \textit{CWA}_P(A) \cup \mathcal{T}_G(a, \mathcal{M}) \not\models_{FOL} \textit{fnext}(f)\}\end{aligned} \quad (6)$$

are the same (A and P are as in (2)).

Proof Sketch: It holds that $f \in (5)$ iff there is a rule $r \in \mathcal{F}r_G(a)$ such that $\textit{CWA}_P(A) \cup r \models_{FOL} \textit{next}(f)$ iff (by Proposition 1) there is a rule $r \in \mathcal{F}r_G(a)$ such that $\textit{CWA}_P(A) \cup r(a, \mathcal{M}) \models_{FOL} \textit{next}(f)$ iff $a \succeq \textit{next}(f)$, $f \in S^{\texttt{true}}$ and $\textit{CWA}_P(A) \cup \mathcal{T}_G(a, \mathcal{M}) \not\models_{FOL} \textit{fnext}(f)$ iff $f \in (6)$.

□

The complete translation

To replace the frame rules from the GDL description we have to make sure that the new *fnext* rules cover any possible move of the game. For this we introduce the concept of a *move covering*, a set of general joint moves such that for every joint move M in a state that is reachable from the initial state there is an element of the set that subsumes $M^{\texttt{does}}$. For example, the singleton set $\{\bigcup_{1\leq i\leq |R|}\{does(P_i, A_i)\}\}$ is a generic move covering for any game G with $|R|$ players. Another move covering for the Tic-Tac-Toe description is

$$\begin{aligned}\mathcal{C}_1 = \{&\{does(x, noop), does(o, mark(A, B))\},\\ &\{does(o, noop), does(x, mark(A, B))\}\}\end{aligned} \tag{7}$$

Now given an atom a of predicate *next*/1 and a move covering $\mathcal{C}$ we can replace the frame rules $\mathcal{F}r_G(a)$ by the *fnext* rules $\bigcup_{\mathcal{M}\in\mathcal{C}} \mathcal{T}_G(a, \mathcal{M})$. For every fluent f such that $a \succeq$ *next*(f) and every joint move M in a reachable state S there is an $\mathcal{M} \in \mathcal{C}$ that subsumes $M^{\texttt{does}}$. Then by Proposition 2, *next*(f) is entailed by a frame rule if and only if *fnext*(f) is not entailed by the *fnext* rules, and we can redefine the state transition just like in the simple case.

In order to translate many atoms at the same time we define a *translation basis* as a set $\mathcal{X} = \{\langle a_1, \mathcal{C}_1\rangle, \ldots, \langle a_n, \mathcal{C}_n\rangle\}$ where each a_i is an atom of predicate *next*/1 and each $\mathcal{C}_i$ is a move covering. In the Tic-Tac-Toe description, for example,

$$\mathcal{X}_1 = \{\langle next(cell(X, Y, Z)), \mathcal{C}_1\rangle, \langle next(control(X)), \mathcal{C}_1\rangle\}$$

is a translation basis, where $\mathcal{C}_1$ is as in (7). The translation of $\mathcal{X}$ in G, written $G^{\mathcal{X}}$, is

$$\bigcup_{\langle a,\mathcal{C}\rangle\in\mathcal{X}} \Big(\bigcup_{\mathcal{M}\in\mathcal{C}} \mathcal{T}_G(a, \mathcal{M})\Big)$$

In the Tic-Tac-Toe example, after some logical simplifications we obtain that G^{χ_1} is the following set of rules:

$$\begin{aligned}fnext(cell(X, Y, Z)) &\leftarrow does(o, mark(X, Y)) \wedge\\ &\quad\ does(x, noop) \wedge true(cell(X, Y, Z))\\ fnext(cell(X, Y, Z)) &\leftarrow does(x, mark(X, Y)) \wedge\\ &\quad\ does(o, noop) \wedge true(cell(X, Y, Z))\\ fnext(control(X)) &\leftarrow true(control(X))\end{aligned}$$

Let $\mathcal{F}^{\mathcal{X}}$ be the set $\{f \mid f$ is a fluent and for some $\langle a, \mathcal{C}\rangle \in \mathcal{X}, a \succeq$ *next*$(f)\}$ of fluents that are translated in $G^{\mathcal{X}}$, and let $\mathcal{F}r_G^{\mathcal{X}}$ be the set $\bigcup_{\langle a,\mathcal{C}\rangle\in\mathcal{X}} \mathcal{F}r_G(a)$ of the frame rules in G that have been translated in $G^{\mathcal{X}}$. We define the *$\mathcal{X}$-translation state transition* from a state S via a joint move M as:

$$\begin{aligned}&\{f : \mathit{CWA}_P(A) \cup (G_{next} \setminus \mathcal{F}r_G^{\mathcal{X}}) \models_{FOL} \mathit{next}(f)\} \cup\\ &\{f \in \mathcal{F}^{\mathcal{X}} \cap S^{\texttt{true}} : \mathit{CWA}_P(A) \cup G^{\mathcal{X}} \not\models_{FOL} \mathit{fnext}(f)\}\end{aligned} \tag{8}$$

Theorem 2 *The definitions of the state transitions (1), (2) and (8) are equivalent.*

Proof Sketch: Given Theorem 1 it suffices to prove that the update transitions (2) and (8) are equivalent. It holds that $f \in$ (2) iff either (i) $CWA_P(A) \cup (G_{next} \setminus \mathcal{F}r_G^{\mathcal{X}}) \models_{FOL}$ *next*(f) or $CWA_P(A) \cup \mathcal{F}r_G^{\mathcal{X}} \models_{FOL}$ *next*(f) iff either (i) or for some $\langle a, \mathcal{C}\rangle \in \mathcal{X}$, $CWA_P(A) \cup \mathcal{F}r_G(a) \models_{FOL}$ *next*(f) iff either (i) or (by Proposition 2) for some $\langle a, \mathcal{C}\rangle \in \mathcal{X}$ it holds that $a \succeq$ *next*(f), and also $f \in S^{\texttt{true}}$ and for every $\langle a, \mathcal{C}\rangle \in \mathcal{X}$ such that $a \succeq$ *next*(f) and every $\mathcal{M} \in \mathcal{C}$ such that $\mathcal{M} \succeq M^{\texttt{does}}$ it is the case that that $CWA_P(A) \cup \mathcal{T}_G(a, \mathcal{M}) \not\models$ *fnext*(f) iff either (i) or $f \in \mathcal{F}^{\mathcal{X}}$, $f \in S^{\texttt{true}}$ and $CWA_P(A) \cup G^{\mathcal{X}} \not\models_{FOL}$ *fnext*(f) iff $f \in$ (8). □

Empirical Evaluation

Implementing the Translation

The translation described in the previous section is parametric over a translation basis $\mathcal{X}$. Early experimentation indicated that the generic singleton move covering could be improved by partially grounding the role and action terms.

In a preliminary step, we perform a simple reachability analysis to compute a superset of the domain of the *next* and *does* predicates. This allows us to identify the roles R in the game, as well as the fluent names $\mathcal{F}$ and for each role r, the action names $\mathcal{A}_r$ for that role. We can then define the *direct* move covering $\mathcal{C}_D = \{\bigcup_{r\in R}\{$*does(r, m$_r$)*$\}$: $m_1 \in A_1, \ldots, m_{|R|} \in A_{|R|}\}$. In Tic-Tac-Toe, we obtain the roles $\{x, o\}$, the fluent names $\{$*control*, *cell*$\}$, and the same action names $\{$*mark*, *noop*$\}$ for both roles.

$$\begin{aligned}\mathcal{C}_D = \{&\{does(x, noop), does(o, noop)\},\\ &\{does(x, noop), does(o, mark(A, B))\},\\ &\{does(x, mark(A, B)), does(o, noop)\},\\ &\{does(x, mark(A, B)), does(o, mark(C, D))\}\}\end{aligned}$$

The direct move covering sometimes contains general joint moves that can never appear in the reachable states of the game. It is possible to use the theorem proving technique described by Haufe, Schiffel, and Thielscher (2012) to derive automatically whether an instance of a given general joint moves can ever occur in a reachable state. Never occurring general joint moves can safely be removed from the covering. We call $\mathcal{C}_P$ the *pruned* move covering resulting from $\mathcal{C}_D$ after pruning provably impossible general joint moves. In Tic-Tac-Toe, we obtain $\mathcal{C}_P = \mathcal{C}_1$ as in (7).

The corresponding *direct* translation basis is $\mathcal{X}_D = \{\langle$*next(f)*$, \mathcal{C}_D\rangle : f \in \mathcal{F}\}$, and the *pruned* translation basis is $\mathcal{X}_P = \{\langle$*next(f)*$, \mathcal{C}_P\rangle, f \in \mathcal{F}\}$.

The rest of the implementation follows the first order logic description provided in the previous section to obtain an encoding in a GDL-like form. The only difference is that since GDL does not allow direct universal quantification, we need to introduce auxiliary predicates to capture the universally quantified variables.

Experimental Results

We implemented the two described translations to generate alternative game rule encodings and we used the FluxPlayer system to process them. The FluxPlayer system is based on ECLiPSe Prolog and has been participating in GGP competitions since 2006.

We can now proceed to measure whether the proposed translations improve the performance of a game engine based on the FluxPlayer system. In its standard operational

Table 1: Time needed for a minimax search up to depth d using various encodings, in seconds.

Game G	d	k-states	$\mathcal{O}$	$\mathcal{X}_D$	$\mathcal{X}_P$
amazons	1	2.13	0.10	0.10	**0.08**
battle	1	2.31	**0.82**	0.98	
beatmania	6	598	**20.5**	21.3	
blocks	14	1240	30.3	**24.7**	
brawl	3	55.5	36.2	15.1	**13.9**
breakthrough	4	268	66.4	26.2	**21.5**
bunk_t	3	259	22.3	39.8	**15.1**
buttons	13	2390	**56.9**	88.9	
checkers	5	26.9	21.0	23.2	**13.7**
chinesecheckers1	6	134	33.2	**7.97**	
chinesecheckers2	6	139	42.0	11.7	**10.7**
chinesecheckers3	6	139	50.5	21.9	**12.6**
chinesecheckers4	6	53.2	22.3	30.1	**11.7**
chinesecheckers6	6	128	78.0	519	**23.5**
chinesecheckers6 -simultaneous	1	118	**37.2**	193	
chomp	3	65.6	16.2	9.50	**9.19**
crisscross	9	139	25.5	18.6	**15.8**
doubletictactoe	3	259	**22.3**	41.0	
dup-statelarge	9	350	**38.0**	38.6	
dup-statemedium	10	1400	76.9	**70.5**	
firefighter	5	1190	**10.8**	12.9	
hanoi	13	1040	40.5	**38.0**	
knightstour	10	2690	**72.5**	101	
knightthrough	3	65.2	13.9	4.56	**4.10**
max_knights	2	4.10	**2.18**	7.59	
minichess	8	188	**20.4**	40.5	24.7
nim1	8	740	47.5	50.9	**46.9**
nim2	5	997	**28.0**	35.7	29.7
nim3	3	158	**3.15**	4.51	3.54
nim4	3	169	**3.44**	4.81	3.85
numbertictactoe	2	104	3.68	3.08	**2.50**
othello-comp2007	6	20.3	57.9	**57.8**	
othellosuicide	6	20.3	58.0	**57.7**	
pancakes6	8	2020	**24.8**	41.7	
pawn_whopping	5	234	**24.0**	25.7	
pawntoqueen	7	5.28	58.4	34.9	**34.7**
pentago_2008	5	2830	**62.8**	125	81.8
point_grab	3	1010	**18.0**	193	
racer	1	2.71	**0.26**	0.36	
racetrackcorridor	3	350	**84.8**	295	
roshambo2	5	1120	**18.6**	235	
ruledepthexp	13	16.4	**37.6**	39.1	
ruledepthlinear	20	2100	**33.0**	35.0	
sheep_and_wolf	7	31.1	19.5	7.21	**5.16**
skirmish	5	42.4	26.6	10.5	**9.58**
statespacelarge	9	350	26.6	**23.9**	
statespacemedium	10	1400	89.2	**74.3**	
sum15	8	624	36.2	29.8	**26.1**
tictactoe	9	986	56.6	48.9	**43.7**
tictactoeserial	6	86.4	54.4	47.8	**40.4**
tictictoe	3	143	18.5	**12.6**	
wallmaze	6	289	**88.6**	144	

mode, FluxPlayer uses the underlying Prolog implementation and Prolog's *assert* and *retract* mechanism to maintain the set of fluent instances holding in the current game state. Measuring engine speed in GGP is typically done by selecting games from previous GGP competitions and running the game engine related part of standard search algorithms. Following recent work by Schiffel and Björnsson (2013), we use the time needed for a naive minimax search to estimate the raw performance. The benchmark domains have been used in international or local GGP competitions and can be found online.[2]

We translate each game under consideration using the direct and the pruned translation bases. The time needed for the transformation between the classical and proposed encodings falls under ten second. The translation is performed once per game and all GGP competitions so far use much longer STARTCLOCK and PLAYCLOCK, so we can disregard the transformation time from our considerations.

Table 1 aggregates the results. For each game G under consideration, d is the maximal depth reachable in an iterative deepening minimax search within 3 minutes using the classical state transition. We then measure the time (in seconds) needed to perform a full minimax search up to d with the proposed encodings: the original state transition $\mathcal{O}$, the direct translation $\mathcal{X}_D$, and the pruned translation $\mathcal{X}_P$.

As an additional precaution, we output with each encoding the number of visited states in the minimax search. Asserting that all encodings induce the same number of visited states for a given search depth increases the confidence in the correctness of the implementation. The *k-states* column indicates the number of states (in thousands) involved in the search.

Discussion and Conclusion

When the theorem proving approach cannot rule out any general joint move, the $\mathcal{X}_D$ and $\mathcal{X}_P$ encodings are identical. This happens mostly in games with simultaneous moves, in particular in single-agent problems.

We can see that in an important fraction of the domains, the pruned frame encoding $\mathcal{X}_P$ leads to an engine at least twice as fast as the original encoding. Expectedly, using a pruned action covering always improves the performance over the naive action covering. Nevertheless, the naive move covering encoding already provides a notable improvement over the original encoding in a majority of the games.

The existence of domains for which our translation provides no improvement should not deter from its adoption, in particular from a portfolio perspective. Indeed, a general game player can benchmark the original and the translated encodings during the set-up time, and use the faster one in the PLAYCLOCK.

We have shown that our translation could indeed improve the performance of reasoners in General Game Playing. While our presentation and evaluation were focussed on the Game Description Language, our approach can be applied to other successor state axioms-based action formalisms.

[2] http://games.ggp.org

Acknowledgments

This research was supported under the Go8-DAAD Australia-Germany Joint Research Cooperation scheme, by the DFG grant SCHA 550/9-1 and by the Australian Research Council (project DP 120102023). The third author is also affiliated with the University of Western Sydney.

References

Apt, K.; Blair, H.; and Walker, A. 1987. Towards a theory of declarative knowledge. In Minker, J., ed., *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann. chapter 2, 89–148.

Clune, J. 2007. Heuristic evaluation functions for general game playing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 1134–1139. Vancouver: AAAI Press.

Finnsson, H., and Björnsson, Y. 2008. Simulation-based approach to general game playing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 259–264. Chicago: AAAI Press.

Gelfond, M. 2008. Answer sets. In van Harmelen, F.; Lifschitz, V.; and Porter, B., eds., *Handbook of Knowledge Representation*, 285–316. Elsevier.

Genesereth, M., and Björnsson, Y. 2013. The international general game playing competition. *AI Magazine* 34(2):107–111.

Genesereth, M.; Love, N.; and Pell, B. 2005. General game playing: Overview of the AAAI competition. *AI Magazine* 26(2):62–72.

Haufe, S.; Schiffel, S.; and Thielscher, M. 2012. Automated verification of state sequence invariants in general game playing. *Artificial Intelligence* 187–188:1–30.

Kissmann, P., and Edelkamp, S. 2010. Instantiating general games using Prolog or dependency graphs. In Dillmann, R.; Beyerer, J.; Hanebeck, U.; and Schultz, T., eds., *Proceedings of the German Annual Conference on Artificial Intelligence (KI)*, volume 6359 of *LNCS*, 255–262. Karlsruhe, Germany: Springer.

Kuhlmann, G.; Dresner, K.; and Stone, P. 2006. Automatic heuristic construction in a complete general game player. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 1457–1462. Boston: AAAI Press.

Lloyd, J., and Topor, R. 1986. A basis for deductive database systems II. *Journal of Logic Programming* 3(1):55–67.

Lloyd, J. 1987. *Foundations of Logic Programming*. Series Symbolic Computation. Springer, second, extended edition.

Love, N.; Hinrichs, T.; Haley, D.; Schkufza, E.; and Genesereth, M. 2006. General Game Playing: Game Description Language Specification. Technical Report LG–2006–01, Stanford Logic Group, Computer Science Department, Stanford University, 353 Serra Mall, Stanford, CA 94305. Available at: `games.stanford.edu`.

McCarthy, J., and Hayes, P. 1969. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence* 4:463–502.

Méhat, J., and Cazenave, T. 2011. A parallel general game player. *KI—Künstliche Intelligenz* 25:43–47. Springer.

Reiter, R. 1978. On closed world data bases. In Gallaire, H., and Minker, J., eds., *Logic and Data Bases*. Plenum Press, New York. 55–76.

Reiter, R. 1991. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Lifschitz, V., ed., *Artificial Intelligence and Mathematical Theory of Computation*. Academic Press. 359–380.

Saffidine, A., and Cazenave, T. 2011. A forward chaining based game description language compiler. In *Proceedings of the IJCAI Workshop on General Intelligence in Game-Playing Agents (GIGA)*, 69–75.

Schiffel, S., and Björnsson, Y. 2013. Efficiency of gdl reasoners. *IEEE Transactions on Computational Intelligence and AI in Games*.

Schiffel, S., and Thielscher, M. 2007. Fluxplayer: A successful general game player. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 1191–1196. Vancouver: AAAI Press.

Schiffel, S., and Thielscher, M. 2010. A multiagent semantics for the Game Description Language. In Filipe, J.; Fred, A.; and Sharp, B., eds., *Agents and Artificial Intelligence*, volume 67 of *Communications in Computer and Information Science*, 44–55. Springer.

Schiffel, S., and Thielscher, M. 2011. Reasoning about general games described in GDL-II. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 846–851.

Schkufza, E.; Love, N.; and Genesereth, M. 2008. Propositional automata and cell automata: Representational frameworks for discrete dynamic systems. In *Proceedings of the Australasian Joint Conference on Artificial Intelligence*, volume 5360 of *LNCS*, 56–66. Auckland: Springer.

Thielscher, M. 1999. From situation calculus to fluent calculus: State update axioms as a solution to the inferential frame problem. *Artificial Intelligence* 111(1–2):277–299.

Thielscher, M. 2010. A general game description language for incomplete information games. In Fox, M., and Poole, D., eds., *Proceedings of the AAAI Conference on Artificial Intelligence*, 994–999.

Waugh, K. 2009. Faster state manipulation in general games using generated code. In *Proceedings of the IJCAI Workshop on General Intelligence in Game-Playing Agents (GIGA)*, 91–97.