IJCAI–09 Formatting Instructions*

Craig Boutilier
Department of Computer Science
University of Toronto
pcchair09@ijcai.org

Abstract

The *IJCAI–09 Proceedings* will be printed from electronic manuscripts submitted by the authors. The electronic manuscript will also be included in a CD-ROM version of the proceedings. This paper provides the style instructions.

1 Translation

We show that the translation from GDL to PDDL can done, by interpreting GDL rules as situation-calculus successor-state axioms and then utilizing results from [Röger *et al.*, 2008]. Certain features special to GDL such as multiple goals are discussed separately later.

1.1 GDL to Situation Calculus

A Basic Action Theory corresponding to the given game description can be generated as described below.

BAT=ΣUT_{SSA}U T_{APA} .. Explain. Cite Reiter

$next$ **rules**

1. 1. Form the Clark completition of the next rules for each predicate
2. 2. Add existential quantifiers to free variables in GDL rule bodies
3. 3. Replace the $distinct$ predicate with negated equality
4. 4. Replace *true(function($x_1, .., x_n$))* with a new predicate *function_predicate($x_1, .., x_n$)* and *(does ?player action($\vec{X}$)* with $a = action(\vec{X})$

We get successor-state axiom versions of the next rules and they constitute T_{SSA}

For example, consider these two rules:
(¡= (next (cell ?x ?y nolight)) (true (cell ?x ?y light)) (affected ?x ?y))

(¡= (next (cell ?x ?y ?state)) (true (cell ?x ?y ?state)) (not (affected ?x ?y)))

They can be replaced by the axiom

$Cell(X, Y, Z, do(a, s)) \equiv ((Z = nolight \wedge cell(X, Y, light, s) \wedge affected(X, Y, s)) \vee (cell(X, Y, Z, s) \wedge \neg affected(X, Y, s)))$

$legal$ **rules**

The GDL $legal$ rules can similarly be transformed into the $Poss$ predicates of Basic Action Theories, with one extension: $terminal$ should be appended to the body of the legal rules. This ensures that no actions are applicable when the $terminal$ predicate is true, as per GDL semantics.

$init$ **rules**

For every predicate, for every $init$ rule of the form $(init(predicatep_1..q_1)), predicate(x_1, .., x_n) \equiv (x_1 = p_1 \wedge .. \wedge x_n = q_1) \vee .. \vee (x_1 = p_n \wedge .. \wedge x_n = q_n)$ is added to the initial database, or $\neg predicate(\vec{X})$ if no such rule exists.

Other rules

Other rules excluding $goal$, and $role$ rules are form part of the *foundational axioms*(Σ) of the Basic Action Theory.

1.2 Situation Calculus to PDDL/ADL:

[Röger *et al.*, 2008] identify certain restrictions on the BAT so that it can be translated to ADL efficiently, where efficiency is in the context of the compilability framework of [Nebel, 2000].

Keeping in mind that GDL does not have functional fluents, these are the relevant restrictions:

1. 1. The set of true situation-independent predicates and fluent predicates in the initial state have to be enumerated. That is, there are rules of the form $F(x_1, .., x_n) \equiv (x_1 = p_1 \wedge .. \wedge x_n = q_1) \vee .. \vee (x_1 = p_n \wedge .. \wedge x_n = q_n)$ or $\neg F(\vec{X})$ (1)
2. 2. Unique name axioms $c_i \neq c_j$ for every pair of different constants c_i, c_j.

The GDL specification implicitly incorporates the unique name axioms. Hence results of [Röger *et al.*, 2008] directly carry over to a restricted subset of GDL:

Theorem 1 All function-free GDL single-player game descriptions with no rules apart from next, legal,terminal, init and goal can be translated to ADL preserving plan length exactly.

*These match the formatting instructions of IJCAI-07. The support of IJCAI, Inc. is acknowledged.

The requirements in Theorem 1 are overtly restrictive.Restriction 1 implies that general rules are not allowed on the right hand side of equation(1).Also, relaxing 1 means we have to disallow functions which refer to unnamed constants to stay within the expressive power of ADL[Röger and Nebel, 2007].

We discuss how recursive stratified rules and situation-independent functions can be handled in some cases in the following sections.

1.3 Recursive Rules

GDL allows fairly unrestricted rules to be part of the foundational axioms(Σ). These correspond to the 'derived predicates' which are part of PDDL2.2. As discussed in [Thiebaux *et al.*, 2003], they add significant expressive power in the sense that they cannot be compiled to ADL without blowing up domain definitions or plan length.

The GDL specification requires that the dependency graph(the directed graph with predicates as nodes and edges from predicate p to predicate q if there is a rule with q as the head and contains p in the body) not contain cycles with negations. There is a direct correspondence between the kind of rules permitted by GDL and the requirement in [Thiebaux *et al.*, 2003] that the rules be stratified; stratification can be achieved for example by the following algorithm:

- In the GDL rule dependency graph, collapse every maximal cycle into a single vertex(all nodes in the cycle belong to the same stratum; all cycles are negation free).
- Topologically sort the resulting directed acyclic graph to get a stratification.

1.4 Function symbols

GDL allows function symbols that are situation independent. We identify a restriction on GDL rules which allow all possible function objects to be enumerated and named. It is convenient to define a domain dependency graph as in [Schiffel and Thielscher, 2007], where it is used for computing supersets of domains of predicates and functions.

Restriction R There are no two functions or predicates P,Q such that P_i and Q_j that belong to the same component and there is a rule in which P occurs as an argument of Q or Q occurs as an argument in P.

This in effect ensures that nested function terms cannot occur, allowing enumeration of all possible instantiations of function symbols starting from the constants in the game description. Thus functions can be compiled away as described below, which is similar to the procedure described in [Röger and Nebel, 2007] for compiling away functional fluents:

For each function symbol f in the game description, introduce a new predicate $predicate_f$.

- Repeatedly do the following until the rules are function-free:

  - In each rule, replace every occurance of $predicate(t_1, t_2, .., f(q_1, q2, .., q_n), .., t_m)$*with*

  $predicate(t_1, t_2, .., v, .., t_m) \wedge predicate_f(q_1, q2, .., q_n, v)$
- Next add the facts corresponding to these newly introduced predicates to the initial state:

  For each function f, and each possible instantiation $\vec{X}$ *of it's parameters(parameter domains have been identified), add the fact* $predicate_f(\vec{X}, v)$*, where* v *is a new symbol, to the initial state.*

With all previously 'unnamed' objects enumerated this way, we can add the Domain Closure Axiom to the BAT.

1.5 Multiple Goals

Another feature of GDL that has no direct analogue in ADL/PDDL is that of multiple goals with rewards.

GDL allows multiple goal rules are of the form
(goal player N)$\equiv expression$

And the semantics is that a player scores N points if the game terminates in a state in which '*expression*' is true, i.e, when $terminal \wedge (goalplayerN)$ is true.

Multiple goals are described in the ADL problem file using a modification of the ADL goal clause:

(:goal N1 expression1)
(:goal N2 expression2)
etc.

A simple extension to FF to handle multiple goals is described in the implementation section.

The following Corollary to Theorem 1 formally summarizes the above discussion:

Corollary All GDL single-player game descriptions satisfying restriction R can be compiled to ADL extended with recursive rules and multiple goals preserving plan size exactly.

2 Implementation

The system comprises of a translator and a planner. The translator is an implementation of the procedure described above. It is written in Prolog on top of the Fluxplayer system[Schiffel and Thielscher, 2007].

2.1 Planner

Our planner is built upon the FF planner[Hoffmann, 2001] with two extensions:

Axioms
In our planning system, these rules are handled within FF with the extensions described in [Thiebaux *et al.*, 2003]

Multiple Goals
The planner maintains a list of goal-reward pairs $L = (G_1, r_1), (G_2, r_2), .., (G_n, r_n)$, and a list of plans for goals that have already been achieved once.

The relaxed plan heuristic returns an estimate of the distance between a given state *s* and goal *g*, $h(s, g)$. The planner searches using the following function of the $h(s, G_i), r_i$ values for the current list of goals:

$H(s) = \sum_i h(s, G_i)^k r_i$ for $k = 1$ or $k = 2$

When the planner reaches a state satisfying a goal G_i, the path to this state, plan P_i, is saved. G_i and all goals with rewards lesser than r_i are removed from L and search is continued.

References

[Hoffmann, 2001] Jorg Hoffmann. Ff: The fast-forward planning system. *The AI Magazine*, 22:57–62, 2001.

[Nebel, 2000] Bernhard Nebel. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research*, 12:271–315, 2000.

[Röger and Nebel, 2007] Gabriele Röger and Bernhard Nebel. Expressiveness of adl and golog: Functions make a difference. In *AAAI*, pages 1051–1056, 2007.

[Röger *et al.*, 2008] Gabriele Röger, Malte Helmert, and Bernhard Nebel. On the relative expressiveness of adl and golog: The last piece in the puzzle. In *KR*, pages 544–551, 2008.

[Schiffel and Thielscher, 2007] Stephan Schiffel and Michael Thielscher. Fluxplayer: A successful general game player. In *AAAI '07: Proc. 22nd AAAI Conf. Artificial Intelligence*, pages 1191–1196, Vancouver, Canada, July 2007. AAAI Press.

[Thiebaux *et al.*, 2003] Sylvie Thiebaux, Joerg Hoffmann, and Bernhard Nebel. In defense of pddl axioms. In *In Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03*, pages 961–966. Morgan Kaufmann, 2003.