Effect of knowledge representation on model based planning : experiments using logic programming encodings

Le-chi Tuan and **Chitta Baral**
Department of Computer Science and Engineering
Arizona State University
Tempe, Arizona 85287
$\{$*lctuan,chitta*$\}$@*asu.edu*

Abstract

In this paper we implement planning using answer set programming. We consider the action language $\mathcal{A}$ and its extensions. We show that when the domain is described using richer features such as qualification, ramification and conditional effects not only the encoding is smaller, but also it takes less time to find a plan. We also show that encoding of Bacchus and Kabanza's style temporal constraints is fairly straightforward in answer set planning. Finally, unlike other model enumeration planning encodings, in our encoding we can just give an upper bound of the length of the plan, instead of the exact length. We illustrate the above features using the blocks world example from the AIPS planning contests.

1 Introduction and motivation

Recently there has been a lot of success in planning using the model enumeration approach. In this approach the planning problem is encoded in a logical language in such a way that each model of the encoded theory corresponds to a plan. This approach was first studied by Kautz and Selman [KS92] where they encoded planning problems using propositional logic. Recently, they also considered domain constraints [HSK99].

Following Subrahmanian and Zaniolo [SZ95], Dimopoulos et. al. [DNK97] and Lifschitz [L99] have worked on planning using logic programming encodings. In this approach, also called answer set planning [L99], each answer set of an encoded logic program corresponds to a plan. In this paper we follow this approach and present several translations (to logic programs) from domain descriptions in the extended language $\mathcal{A}$ and prove their correctness. The main contributions of our paper are:

- We consider features such as defined fluents, qualification, ramification constraints and give encodings of the blocks world problem – perhaps for the first time – using these features. We then experiment with these encodings and show that the richer features not only result in more compact encodings but also improve planning performance. (It should be noted that, as described by Reiter and Lin in their paper "State Constraints revisited", qualification and ramifications can not be easily expressed in propositional logic.)
- We show how domain specific temporal constraints can be easily and compactly encoded in logic programming. We report results that show that, as expected, adding temporal constraints significantly improves planning performance.
- Finally, most of the existing encodings assume a fixed plan length. Our formulations allow users to give an upper bound of plan length. The encoding ensures that once the goal has been achieved, action occurrences after that point is prohibited.

2 Background

We consider several STRIPS and ADL domains taken from the AIPS contests as our illustrative examples. STRIPS [FN71] is a simple action description language where operators are specified by a precondition list, an add-list and a delete-list. ADL is an extension of STRIPS which allows amongst other things conditional effects. $\mathcal{A}$ [GL93] is a high level action description language with a simple English-like syntax and an automata-based semantics. In this paper we use an extension of $\mathcal{A}$. This extension of $\mathcal{A}$ captures STRIPS and the ADL features. Our formal results will be with respect to this extension. We now briefly present $\mathcal{A}$.

The alphabet of the language $\mathcal{A}$ has two non-empty disjoint sets of symbols, called *fluent names* F and *action names* A. A *fluent literal* is either a fluent name or a fluent name preceded by $\neg$. For a fluent literal l, we shorten $\neg\neg l$ to l.

The domain description language of $\mathcal{A}$ consists of "ef-propositions" and "ex-propositions". An *ef-proposition* is an expression of the form

$$a \ \textbf{causes} \ f \ \textbf{if} \ p_1, \ldots, p_m, \neg q_{m+1}, \ldots, \neg q_n \qquad (2.1)$$

where a is an action name, $p_1,\ldots,p_m,q_{m+1},\ldots,q_n$, $(n \geq 0)$ are fluents.

An *ex-proposition* is an expression of the form

$$\textbf{executable} \ a \ \textbf{if} \ p_1, \ldots, p_m, \neg q_{m+1}, \ldots, \neg q_n \qquad (2.2)$$

where a is an action name, $p_1,\ldots,p_m,q_{m+1},\ldots,q_n$, $(n \geq 0)$ are fluents.

A *domain description* D is a collection of "ef-propositions" and "ex-propositions".

The query language of $\mathcal{A}$ consists of "v-propositions" (queries) of the form:

$$f \textbf{ after } a_1, \ldots, a_m \tag{2.3}$$

where f is a fluent literal, and a_1,. . .,a_m ($m \geq 0$) are action names. If $m = 0$, we write (2.3) as

$$\textbf{initially } f \tag{2.4}$$

A *set of observations* (or observation set) O is a collection of **initially** f axioms.

The semantics of $\mathcal{A}$, described in more detail in [GL93; GL98], defines an entailment relation between domain description and queries, with respect to an observation set. Intuitively, $D \models_O f$ **after** $a_1, \ldots, a_m$ means that based on the information in D and O the fluent f will be true in the situation reached after executing $a_1, \ldots, a_m$ in the initial situation. This entailment is useful in that it defines correctness of plans.

3 Encodings for planning in STRIPS

In this section we present a general encoding of planning problems in STRIPS domains described in our extension of $\mathcal{A}$. Note that in STRIPS the initial state is always complete. In our translation to logic programs, we divide the logic programs into two main modules: a problem description module specific for each initial, goal state and domain description and an answer-set generation/testing module which is domain independent.

Given a domain description D and a complete observation set O, a plan length l and a goal literal h, we construct a logic program $\Pi_1(D, O)$ consisting of the following parts: $\Pi_1(D, O) = \Pi_1^{exe} \cup \Pi_1^{ef} \cup \Pi_1^{goal} \cup \Pi_1^{aux} \cup \Pi_1^{obs} \cup \Pi_1^{in} \cup \Pi_1^{choice}$, where Π_1^{exe}, Π_1^{ef} are domain dependent and Π_1^{goal}, Π_1^{aux}, Π_1^{obs}, Π_1^{in}, Π_1^{choice} are domain independent.

1. Translating ex-propositions (Π_1^{exe}): Every ex-proposition of the form (2.2) where $n > 0$ is translated to the following rules:

   $$\begin{aligned} &executable(a, T) \leftarrow T < l, \\ &\qquad holds(p_1, T), \ldots, holds(p_m, T), \\ &\qquad not\ holds(q_{m+1}, T), not\ holds(q_n, T). \\ &\leftarrow T < l, occurs(a, T), not\ executable(a, T). \end{aligned}$$

2. Translating ef-propositions (Π_1^{ef}): Every ef-proposition of the form (2.1) is translated as follows:

   (a) if f is a fluent then we have the rule:

   $$\begin{aligned} holds(f, T+1) \ \leftarrow\ & occurs(a, T), \\ & holds(p_1, T), \ldots, holds(p_m, T), \\ & not\ holds(q_{m+1}, T), \ldots, not\ holds(q_n, T). \end{aligned}$$

   (b) if $f = \neg g$ for a fluent g, then we have the rule:

   $$\begin{aligned} holds(neg(g), T+1) \ \leftarrow\ & occurs(a, T), \\ & holds(p_1, T), \ldots, holds(p_m, T), \\ & not\ holds(q_{m+1}, T), \ldots, not\ holds(q_n, T). \end{aligned}$$

3. Translating the goal literal (Π_1^{goal}): If the goal h is a fluent, it is represented by the fact $finally(h)$. If h is a negative fluent literal $h = \neg g$ it is represented by the fact $finally(neg(g))$. We also add the following into Π_1^{goal}, where F represents f or $neg(g)$ depending on h:

   $$\begin{aligned} &not\_goal(T) \leftarrow finally(F), not\ holds(F, T). \\ &goal(T) \leftarrow time(T), not\ not\_goal(T). \\ &\leftarrow not\ goal(l+1). \end{aligned}$$

4. Auxiliary rules (Π_1^{aux}): For each fluent f, Π_1^{aux} contains the following rules:

   $contrary(f, neg(f)).\ contrary(neg(f), f).$

5. Translating observations (Π_1^{obs}): For every v-proposition of the form (2.4) where f is a fluent we have the rule $holds(f, 1)$. If f is a negative fluent literal $f = \neg g$, we have the rule $holds(neg(g), 1)$.

6. Inertia rules (Π_1^{in}): Π_1^{in} contains the following rule:

   $$\begin{aligned} holds(F, T+1) \leftarrow\ & contrary(F, G), T < l, \\ & holds(F, T), not\ holds(G, T+1). \end{aligned}$$

7. Choice rules (Π_1^{choice}): The choice rule generates action occurrences but makes sure that there is only one action that can occur at a time point T up to l, and *actions occur at time points only when the goal has not been achieved. The second condition allows us to specify an upper bound on the plan length instead of an exact plan length.*

   $$\begin{aligned} &other\_occurs(A, T) \leftarrow occurs(B, T), A \neq B. \\ &occurs(A, T) \leftarrow T < l, not\ other\_occurs(A, T), \\ &\qquad not\ goal(T). \end{aligned}$$

3.1 Main Proposition

The following proposition relates answer sets of our encoding with plans for a goal w.r.t. a given D and O.

Proposition 1 *Let D be a consistent domain description in $\mathcal{A}$ and O be a complete set of observations, l be the length of the plan that we are looking for, $n < l$ and h be a goal fluent literal. $D \models_O h$* **after** *$a_1, \ldots, a_n$ and for all $j < n$, $D \not\models_O h$* **after** *$a_1, \ldots, a_j$ iff $\Pi_1(D, O)$ has an answer set M with $\{occurs(a_1, 1), \ldots, occurs(a_n, n)\}$ as the set of facts about $occurs$ in M.*

3.2 Example of planning in STRIPS domains with complete information (Blocks-World domain)

We translated PDDL domain of a block world problem taken from AIPS2000 into $\mathcal{A}$ domain. The domain description D consists of the following propositions. (Here we use a **causes** $f_1, \ldots, f_n$ as a short hand for n propositions a **causes** f_1 ... a **causes** f_n.)

$pick\_up(X)$ **causes** $\neg ontable(X), \neg clear(X), holding(X)$
$\neg handempty$
executable $pick\_up(X)$ **if** $clear(X), ontable(X),$
$handempty$

$put\_down(X)$ **causes** $ontable(X), clear(X),$
$\neg holding(X), handempty$
executable $put\_down(X)$ **if** $holding(X)$

$stack(X,Y)$ **causes** $\neg holding(X), \neg clear(Y), clear(X)$
$handempty, on(X,Y)$
executable $stack(X,Y)$ **if** $holding(X), clear(Y)$

$unstack(X,Y)$ **causes** $holding(X), clear(Y), \neg clear(X),$
$\neg handempty, \neg on(X,Y)$
executable $unstack(X,Y)$ **if** $clear(X), on(X,Y),$
$handempty$

We now list the translation Π_1^{Block} presented to Smodels for a particular problem domain. Π_1^{Block} consisted of the following parts: $\Pi_1^{fact} = 1(a) \cup \ldots \cup 1(f)$, $\Pi_1^{exe} = 1(g)$, $\Pi_1^{ef} = 1(h)$, $\Pi_1^{goal} = 2(a) \cup 2(b)$, $\Pi_1^{aux} = 2(c)$, $\Pi_1^{obs} = 2(d)$, $\Pi_1^{in} = 2(e)$, and $\Pi_1^{choice} = 2(f)$, in Smodels syntax.

1. The domain-dependent problem description module.
   (a) Defining an upper bound on the number of steps in a plan:

   $const\ length = 7.$

   (b) Defining objects in the problem domain:

   $block(a).\ block(b).\ block(c).\ block(d).$

   (c) Defining initial state for the problem domain:

   $initially(clear(c)).\ initially(clear(a)).$
   $initially(clear(b)).\ initially(clear(d)).$
   $initially(ontable(c)).\ initially(ontable(a)).$
   $initially(ontable(b)).\ initially(ontable(d)).$
   $initially(handempty).$

   (d) Defining goal state for the problem domain:

   $finally(on(d,c)).\ finally(on(c,b)).$
   $finally(on(b,a)).$

   (e) Defining fluents: The fluents are defined by rules of the following form.

   $fluent(on(X,Y)) \leftarrow neq(X,Y), block(X),$
   $block(Y).$

   (f) Defining actions: The actions are defined by rules of the following form.

   $action(pick\_up(X)) \leftarrow block(X).$

   (g) The executability conditions were defined in a slightly different way from part (1.) of Section 3. The simplification below was possible because in the blocks world domain each action has only one executability condition. We now give the rules corresponding to the executability condition of the action $put\_down(X)$. The rules for the other actions are similar.
   $\leftarrow block(X), time(T), T < length,$
   $not\ holds(holding(X),T),$
   $occurs(put\_down(X),T).$

   (h) Defining effects of actions: The rules specifying the effect of actions were defined as described in part(2) of Section 3.

2. Domain-independent module.
   (a) We define $goal(T)$ using the first two rules of part (3) of Section 3.

   (b) The third rule of part (3) of Section 3 is used to eliminate possible answer sets which do not encode a plan of the given length.

   (c) We define *time points, literals* and *contrary fluents*.

   (d) What holds and does not hold in the initial state is defined in terms of the $initially$ predicate using the following rules.

   $holds(F,1) \leftarrow literal(F), initially(F).$
   $holds(neg(F),1) \leftarrow fluent(F),$
   $not\ initially(F).$

   (e) We use the inertia rule from part (6) of Section 3.

   (f) For enumerating action occurrences we use the following rule consisting of the cardinality construct from Smodels (instead of the rules in part (7) of Section 3) which reduces the planning time drastically.

   $1\{occurs(A,T) : action(A)\}1 \leftarrow time(T),$
   $T < length, not\ goal(T).$

4 Representing temporal constraints

In this section, we show that Bacchus and Kabanza's style temporal constraints [BK00] (control knowledge) can be easily encoded. The temporal constraints are specified as separate rules and constraints and are simply added to the program Π_1^{Block}.

1. Defining control knowledge axioms:
   (a) Defining bad tower below a block X:

   $holds(bad\_tower\_below(X),T) \leftarrow$
   $time(T), block(X), block(Y), neq(X,Y),$
   $holds(ontable(X),T), finally(on(X,Y)).$
   $holds(bad\_tower\_below(X),T) \leftarrow$
   $time(T), block(X), block(Y), neq(X,Y),$
   $holds(on(X,Y),T), finally(ontable(X)).$
   $holds(bad\_tower\_below(X),T) \leftarrow$
   $time(T), block(X), block(Y), neq(X,Y),$
   $holds(on(X,Y),T), finally(holding(Y)).$
   $holds(bad\_tower\_below(X),T) \leftarrow$
   $time(T), block(X), block(Y), neq(X,Y),$
   $holds(on(X,Y),T), finally(clear(Y)).$
   $holds(bad\_tower\_below(X),T) \leftarrow$
   $time(T), block(X), block(Y), block(Z),$
   $neq(X,Y), neq(Z,Y), neq(Z,X),$
   $holds(on(X,Y),T), finally(on(X,Z)).$
   $holds(bad\_tower\_below(X),T) \leftarrow$
   $time(T), block(X), block(Y), block(Z),$
   $neq(X,Z), neq(X,Y), neq(Z,Y),$
   $holds(on(X,Y),T), finally(on(Z,Y)).$
   $holds(bad\_tower\_below(X),T) \leftarrow time(T),$
   $block(X), block(Y), holds(on(X,Y),T),$
   $neq(X,Y), holds(bad\_tower\_below(Y),T).$

   (b) Defining good tower:

   $holds(good\_tower(X),T) \leftarrow time(T),$

$block(X), holds(clear(X), T),$
$not\ finally(holding(X)),$
$not\ holds(bad\_tower\_below(X), T).$

(c) Defining bad tower:

$$holds(bad\_tower(X), T) \leftarrow time(T), block(X), holds(clear(X), T), not\ holds(good\_tower(X), T).$$

2. Defining temporal constraints:

(a) Do not destroy good towers:

$$\leftarrow block(X), block(Y), neq(X, Y), time(T), T < length, holds(good\_tower(X), T), holds(on(Y, X), T + 1), not\ holds(clear(X), T + 1), not\ holds(good\_tower(Y), T + 1).$$

(b) Do not stack on bad towers:

$$\leftarrow block(X), block(Y), neq(X, Y), time(T), T < length, holds(bad\_tower(X), T), holds(on(Y, X), T + 1).$$

(c) Do not pick up singleton bad tower blocks unless their final positions are ready:

$$\leftarrow block(X), block(Y), neq(X, Y), time(T), T < length, holds(ontable(X), T), finally(on(X, Y)), not\ holds(good\_tower(Y), T), holds(holding(X), T + 1).$$

5 Encodings corresponding to richer features

5.1 Adding defined fluents, qualification and ramification to STRIPS

In this section, we show how ramification constraints can be encoded in answer set planning. For this, we use $\mathcal{A}_{ex}$, an extended version of $\mathcal{A}$ that allows ramification constraints. $\mathcal{A}_{ex}$ has a new type of proposition called static causal propositions (sc-propositions) of the following form:

$$p_1, \ldots, p_m, \neg q_{m+1}, \ldots, \neg q_n \textbf{ causes } f \qquad (5.5)$$

where $p_1, \ldots, p_m, q_{m+1}, \ldots, q_n$ are fluents and f is a fluent literal. The static causal propositions define the interaction among the basic fluents. For the semantics of this extended language, the readers are referred to [GL98].

We follow the following principles in the new encoding:

1. We divide the fluents into two categories: basic fluents and defined fluents.
2. The effect axioms only change the truth values of the basic fluents.
3. The defined fluents are defined from the basic fluents.
4. We add ramification axioms which further simplify the effect axioms and also simplify executability conditions.

We now consider a (different) representation of the blocks world problem that has qualification and ramification axioms, and has basic and defined fluents. For lack of space, we do not show this representation, but rather show its encoding in logic programming. In this encoding, which we will refer to as $\Pi_1^{Block.rich}$, the basic fluents are $on(X, Y)$, $ontable(X)$, $holding(X)$ and the defined fluents are $clear(X)$, $handempty$.

1. The domain-dependent problem description module:

(a-d) Parts (a) to (d) are the same as in Π_1^{Block}.

(e) Defining fluents:

i. The basic fluents are defined as the fluents are defined in Π_1^{Block}.

ii. The defined fluents are declared using the following rules.

$$def\_fluent(clear(X)) \leftarrow block(X).$$
$$def\_fluent(handempty).$$

(f-g) (f) and (g) are the same as in Π_1^{Block}.

(h) Effects of actions: The effects of actions on basic fluents are defined as in Π_1^{Block}.

(i) The relation between defined fluents and basic fluents are expressed using the following rules:

$$holds(clear(X), T) \leftarrow block(X), time(T), holds(ontable(X), T), not\ holds(neg(clear(X)), T).$$
$$holds(clear(X), T) \leftarrow block(X), block(Y), neq(X, Y), time(T), holds(on(X, Y), T), not\ holds(neg(clear(X)), T).$$
$$holds(neg(clear(X)), T) \leftarrow block(X), time(T), holds(holding(X), T).$$
$$holds(neg(clear(X)), T) \leftarrow block(X), block(Y), neq(X, Y), time(T), holds(on(Y, X), T).$$
$$holds(handempty, T) \leftarrow time(T), not\ holds(neg(handempty), T).$$
$$holds(neg(handempty), T) \leftarrow block(X), time(T), holds(holding(X), T).$$

(j) Following is the qualification constraint:

$$\leftarrow block(X), block(Y), block(Z), time(T), neq(X, Z), neq(X, Y), neq(Z, Y), holds(on(X, Y), T), holds(on(Z, Y), T).$$

(k) The ramification constraints are based on the following causal relationship between basic fluents:

$$static\_causes(on(X, Z), neg(on(X, Y))) \leftarrow block(X), block(Y), block(Z), time(T), neq(X, Z), neq(X, Y), neq(Z, Y).$$
$$static\_causes(on(X, Y), neg(holding(X))) \leftarrow block(X), block(Y), neq(X, Y).$$
$$static\_causes(on(X, Y), neg(ontable(X))) \leftarrow block(X), block(Y), neq(X, Y).$$
$$static\_causes(ontable(X), neg(holding(X))) \leftarrow block(X).$$
$$static\_causes(ontable(X), neg(on(X, Y))) \leftarrow block(X), block(Y), neq(X, Y).$$

$static\_causes(holding(X), neg(ontable(X))) \leftarrow$
$\quad block(X).$
$static\_causes(holding(X), neg(on(X,Y))) \leftarrow$
$\quad block(X), block(Y), neq(X,Y).$

2. Domain-independent module

(a-b) Parts (a) and (b) are the same as in Π_1^{Block}.

(c) We add the the definition of literal and contrary for the defined fluents.

(d-f) Parts (d) to (f) are the same as in Π_1^{Block}.

(g) The ramification constraints are defined as follows:
$holds(F,T) \leftarrow literal(F), literal(G),$
$\quad neq(F,G), time(T),$
$\quad holds(G,T), static\_causes(G,F).$

5.2 Planning in ADL domains

ADL is a language richer than STRIPS where, amongst other things, conditional effects are allowed. We take an example of a blocks world problem in AIPS98 where conditional effects are present. The domain description D is described as follows:

$puton(X,Y)$ **causes** $on(X,Y), \neg clear(Y)$ **if** $X \neq table$
$puton(X,Y)$ **causes** $\neg on(X,Z)$ **if** $on(X,Z)$
$puton(X,Y)$ **causes** $clear(Z)$ **if** $X \neq table,$
$\quad Z \neq table, on(X,Z)$
executable $puton(X,Y)$ **if** $X \neq table, Y \neq table,$
$\quad clear(X), clear(Y)$
executable $puton(X,Y)$ **if** $Y = table, clear(X)$

We now show the major differences between $\Pi_1(D,O)$ and $\Pi_2(D,O)$, where the later takes into account ADL description of the Blocks world domain. The main differences are with respect to encoding effects of actions and executability conditions.

1. The domain-dependent problem description module

(a-f) The translation is the same as in Π_1^{Block}, with the exception that we need to add $block(table) \leftarrow$.

(g) We now give a rule corresponding to the executability of the action $puton$. (We need two other rules, which we do not present here.)
$\leftarrow time(T), T < length, block(X), block(Y),$
$\quad neq(X,Y), neq(X, table), neq(Y, table),$
$\quad not\ holds(clear(X),T), occurs(puton(X,Y),T).$

(h) Effects of actions: Due to lack of space we only give one of the four rules that encode the effect of the action $puton(X)$.
$holds(clear(Z), T+1) \leftarrow$
$\quad time(T), T < length,$
$\quad block(X), block(Y), block(Z),$
$\quad neq(X,Y), neq(X,Z), neq(Z,Y),$
$\quad neq(X, table), neq(Z, table),$
$\quad holds(on(X,Z),T), occurs(puton(X,Y),T).$

2. The domain-independent module is same as those of Π_1^{Block}.

5.3 Planning in ADL domains together with ramification and qualification

In this subsection, we further modify the encoding Π_2^{Block} to allow qualification and ramification constraints. We refer to the new encoding as $\Pi_2^{Block.rich}$. The process of modification is similar to the modification from Π_1^{Block} to $\Pi_1^{Block.rich}$. We have only $on(X,Y)$ as the basic fluent in this case. The only defined fluent is $clear(X)$. We now describe the major differences between Π_2^{Block} and $\Pi_2^{Block.rich}$.

1. The domain-dependent problem description module:

(a-g) Same as those of Π_2^{Block} except that we need to define $on(X,Y)$ and $clear(X)$ as defined fluents.

(h) We now have a single rule to encode the effect of the action $puton(X,Y)$ on the single basic fluent $on(X,Y)$.

$holds(on(X,Y), T+1) \leftarrow time(T),$
$\quad T < length, block(X), block(Y), X \neq Y,$
$\quad neq(X, table), occurs(puton(X,Y),T).$

(i) The following rules express the relation between defined fluents and basic fluents:
$holds(clear(X),T) \leftarrow block(X), block(Y),$
$\quad neq(X,Y), neq(X, table), time(T),$
$\quad not\ holds(on(Y,X),T),$
$\quad not\ holds(neg(clear(X)),T).$
$holds(clear(X),T) \leftarrow block(X), block(Y),$
$\quad neq(X,Y), neq(X, table), time(T),$
$\quad holds(on(X,Y),T),$
$\quad not\ holds(neg(clear(X)),T).$
$holds(neg(clear(Y)),T) \leftarrow block(X),$
$\quad block(Y), neq(X,Y), neq(X, table),$
$\quad neq(Y, table), time(T), holds(on(X,Y),T).$

(j) Following is the qualification constraint:
$\leftarrow block(X), block(Y), block(Z), time(T),$
$\quad neq(X,Z), neq(X,Y), neq(Z,Y),$
$\quad neq(X, table), neq(Z, table), neq(Y, table),$
$\quad holds(on(X,Y),T), holds(on(Z,Y),T).$

(k) The ramification is based on the following:
$static\_causes(on(X,Y), neg(on(X,Z))) \leftarrow$
$\quad block(X), block(Y), block(Z), neq(X,Y),$
$\quad neq(Y,Z), neq(X,Z), neq(X, table).$

2. The domain-independent module:

(a-f) Same as those of $\Pi_1^{Block.rich}$.

(g) The ramification constraints are accounted for by the following rule:
$holds(F,T) \leftarrow$
$\quad literal(F), literal(G), neq(F,G), time(T),$
$\quad holds(G,T), static\_causes(G,F).$

6 Experiments and a brief analysis

We took all STRIPS and ADL versions of the blocks world problems in AIPS98 and AIPS2000 to run on a pentium III PC with 700MHz CPU and 128MB of RAM running Windows 2000. Table 1 shows the results of running Π_1^{Block}, $\Pi_1^{Block.rich}$ and Π_2^{Block}, $\Pi_2^{Block.rich}$ in Smodels [NS97] without using control knowledge. Table 2 shows the results of running those encodings with control knowledge. Blanks in the tables mean either parsing time or running time greater than one hour.

When no control knowledge is used, the encodings corresponding to $\mathcal{A}_{ex}$ (richer encodings) in both STRIPS and ADL domains perform much better than encodings corresponding to $\mathcal{A}$ ("normal" encodings). $\Pi_2^{Block.rich}$ runs much faster and scales well when number of blocks and the plan length increase. However, the encoding Π_1^{Block}, $\Pi_1^{Block.rich}$ and Π_2^{Block} take substantial amount of time when the number of blocks is greater than 8 and the plan length is over 18. $\Pi_2^{Block.rich}$ can only solve up to the problem 10-1 where there are 10 blocks with the plan length of 34. Also, use of the upper bound does not significantly affect the planning time, although too high an upper bound would lead to blow up in the size of the ground program. For example, for problem 6-2 whose minimal plan length is 20, an upper bound of 20 took 87 secs with Π_1^{Block} while an upper bound of 54 took 288 secs.

When control knowledge is used by adding additional rules and constraints, all four encodings solve all the problems of blocks world with reasonable amount of time. Table 2 shows that the normal encodings and the richer encodings are comparable, although the richer encodings perform slightly better when the number of blocks is less than 11 and the plan length is less than 34. The normal encodings perform slightly better than the richer encodings in the remaining problems.

7 Conclusion

We consider the language $\mathcal{A}$ and its extensions and present a family of encodings by translating domain descriptions in these languages into logic programs. We prove the correctness of the first translation. We demonstrate that encoding of temporal constraints is fairly straightforward in answer set planning and it really improves planning performance. We also give the encoding w.r.t description in richer languages and also show that the encoding is not only smaller but it also takes less time (in the absence of the temporal constraints) to find a plan. Finally, although in this paper we only present the blocks world problem, we did encode and experiment with the other AIPS contest problems with similar results. Due to lack of space we do not discuss them here.

References

[BK00] Bacchus F. and Kabanza F.: Using Temporal Logics to Express Search Control Knowledge for Planning. *Artificial Intelligence* volume 16, pages 123–191, 2000.

[DNK97] Dimopoulos Y., Nebel B. and Koehler J.: Encoding Planning Problems in Nonmonotonic Logic Programs. *ECP97*, 169-181

[FN71] Fikes R. and Nilsson N.: STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 1:27-120, 1971.

[GL93] Gelfond, M. and Lifschitz, V.: Representing Actions and Change by Logic Programs, *J. of Logic Programming* **17** (1993) 301–323.

[GL98] Gelfond, M. and Lifschitz, V.: Action languages, *Electronic Transactions on AI*, Vol. 3, No. 16, 1998.

[HSK99] Huang Y.C, Kautz H. and Selman B.: Control Knowledge in Planning: Benefits and Tradeoffs, *Proc. AAAI-99*, Orlando, FL, 1999

[KS92] Kautz H. and Selman B.: Planning as Satisfiability, *Proc. ECAI-92*.

[L99] Lifschitz V.: Answer set planning, *in Proceedings of the 1999 International Conference on Logic Programming*, 1999, pp. 23-37.

[NS97] Niemela, I. and Simons, P.: Smodels – an implementation of the stable model and well-founded semantics for normal logic programs, *in Proc. of LPNMR 97*, 1997, pp. 420-429.

[SZ95] Subrahmanian, V. S. and Zaniolo, C.: Relating stable models and AI planning domains, *in Proceedings of ICLP 1995*, 1995, pp. 233-247.

Table 1: Blocks world with no control knowledge

Problems (Names /length)	$\Pi_1 Block$	$\Pi_1 Block.rich$	$\Pi_2 Block$	$\Pi_2 Block.rich$
4-0 (6)	0.11	0.10	0.14	0.05
4-1 (10)	0.14	0.08	0.10	0.11
4-2 (6)	0.03	0.04	0.06	0.04
5-0 (12)	2.18	0.65	0.28	0.55
5-1 (10)	0.54	0.46	0.41	0.35
5-2 (16)	1.09	1.14	0.95	1.28
6-0 (12)	3.60	4.96	0.96	0.97
6-1 (10)	2.12	2.28	1.73	0.58
6-2 (20)	86.59	10.69	33.10	3.00
7-0 (24)	6650.5	12.06	10.89	17.77
7-1 (22)	1052.00	1027.33	1401.96	11.07
7-2 (20)	142.93	448.78	2131.50	104.62
8-0 (18)	7958.56	1457.89		10.08
8-1 (20)				14.19
8-2 (16)				8.30
9-0 (29)				94.05
9-1 (32)				150.91
10-0(34)				146.87
10-1(34)				276.85

Table 2: Blocks world with control knowledge

Problems (Names /length)	$\Pi_1 Block$	$\Pi_1 Block.rich$	$\Pi_2 Block$	$\Pi_2 Block.rich$
4-0 (6)	0.02	0.03	0.03	0.03
4-1 (10)	0.12	0.09	0.11	0.08
4-2 (6)	0.05	0.04	0.08	0.05
5-0 (12)	0.45	0.34	0.33	0.23
5-1 (10)	0.27	0.21	0.23	0.15
5-2 (16)	1.14	0.47	0.56	0.43
6-0 (12)	0.74	0.55	0.60	0.43
6-1 (10)	0.17	0.23	0.35	0.25
6-2 (20)	3.82	0.99	1.34	1.24
7-0 (24)	2.78	3.34	2.81	3.44
7-1 (22)	34.39	4.30	2.39	2.59
7-2 (20)	6.87	3.50	2.14	1.95
8-0 (18)	9.58	4.41	2.44	2.75
8-1 (20)	23.97	8.31	2.31	2.15
8-2 (16)	3.33	1.72	1.77	1.67
9-0 (29)	115.67	8.9	7.22	6.68
9-1 (32)	170.54	18.16	8.58	14.28
10-0(34)	1077.84	19.12	14.19	21.80
10-1(34)		39.25	16.12	18.19
10-2(34)	99.68	21.28	14.87	20.05
11-0(36)	31.81	68.56	19.28	24.10
11-1(32)	43.97		21.26	24.43
11-2(38)	44.32	64.35	22.55	37.63
12-0(36)	280.43	322.32	26.08	38.24
12-1(42)	199.58	107.96	43.80	86.45
13-0(44)	64.87	114.71	49.53	89.60
13-1(46)	85.89	127.73	63.08	63.11
14-0(48)	3486.27	12746.96	77.14	129.56
14-1(44)	173.71	621.33	73.44	101.20