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 \( \neg a \) to \( \bar{a} \).

The domain description language of \( \mathcal{A} \) consists of “ef-propositions” and “ex-propositions”. An ef-proposition is an expression of the form

\[
a \text{causes} f \text{ if } p_1, \ldots, p_m, \neg q_{m+1}, \ldots, \neg q_n \tag{2.1}
\]

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

\[
\text{executable } a \text{ if } p_1, \ldots, p_m, \neg q_{m+1}, \ldots, \neg q_n \tag{2.2}
\]

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:

$$f \text{ after } a_1, \ldots, a_m$$  \hspace{1cm} (2.3)

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

$$\text{initially } f$$  \hspace{1cm} (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: $\Pi_1(D, O) = \Pi_1^{\text{exe}} \cup \Pi_1^{\text{fl}} \cup \Pi_1^{\text{goal}} \cup \Pi_1^{\text{aux}} \cup \Pi_1^{\text{obs}} \cup \Pi_1^{\text{ch}}$, where $\Pi_1^{\text{exe}}$, $\Pi_1^{\text{fl}}$, $\Pi_1^{\text{goal}}$, $\Pi_1^{\text{aux}}$, $\Pi_1^{\text{obs}}$, $\Pi_1^{\text{ch}}$ are domain dependent and $\Pi_1^{\text{exe}}$, $\Pi_1^{\text{fl}}$, $\Pi_1^{\text{obs}}$, $\Pi_1^{\text{ch}}$ are domain independent.

1. Translating ex-propositions ($\Pi_1^{\text{exe}}$): Every ex-proposition of the form (2.2) where $n > 0$ is translated to the following rules:

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

2. Translating ef-propositions ($\Pi_1^{\text{fl}}$): Every ef-proposition of the form (2.1) is translated as follows:

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

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

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

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

3. Translating the goal literal ($\Pi_1^{\text{goal}}$): If the goal $h$ is a fluent, it is represented by the fact $\text{finally}(g)$. If $h$ is a negative fluent literal $h = \neg g$ it is represented by the fact $\text{finally}(\neg g)$. We also add the following into $\Pi_1^{\text{goal}}$, where $F$ represents $f$ or $\neg g$ depending on $h$:

   $\text{not}\_\text{goal}\(T\) \leftarrow \text{finally}\(F\), \text{not holds}(F, T)$. \\
   $\text{goal}\(T\) \leftarrow \text{time}(T), \text{not goal}(T)$. \\
   $\leftarrow \text{not goal}(l + 1)$.

4. Auxiliary rules ($\Pi_1^{\text{aux}}$): For each fluent $f$, $\Pi_1^{\text{aux}}$ contains the following rules:

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

5. Translating observations ($\Pi_1^{\text{obs}}$): For every v-proposition of the form (2.4) where $f$ is a fluent we have the rule holds$(f, T)$.

6. Inertia rules ($\Pi_1^{\text{in}}$): $\Pi_1^{\text{in}}$ contains the following rule:

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

7. Choice rules ($\Pi_1^{\text{ch}}$): 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.

   other\_occurs$(A, T) \leftarrow \text{occurs}(B, T), A \neq B$. \\
   occurs$(A, T) \leftarrow T < l$, not other\_occurs$(A, T)$, not goal$(T)$.

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 $a_1, \ldots, a_j$ if $\Pi_1(D, O)$ has an answer set $M$ with $\{\text{occurs}(a_1, 1), \ldots, \text{occurs}(a_n, n)\}$ 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 $f_1$ \ldots a causes $f_n$.)

pick\_up$(X)$ causes $\neg\text{ontable}(X), \neg\text{clear}(X), \text{holding}(X)$, \neg\text{handempty}$

executable pick\_up$(X)$ if clear$(X)$, ontatable$(X)$, handempty

put\_down$(X)$ causes ontatable$(X)$, clear$(X)$, $\neg\text{holding}(X)$, handempty

executable put\_down$(X)$ if $\text{holding}(X)$
stack \((X, Y)\) causes \(-\text{holding}(X), -\text{clear}(Y), \text{clear}(X)\)

**executable** \(\text{stack}(X, Y)\) if \(\text{holding}(X), \text{clear}(Y)\)

unstack \((X, Y)\) causes \(\text{holding}(X), \text{clear}(Y), -\text{clear}(X), -\text{handempty}, -\text{on}(X, Y)\)

**executable** unstack \((X, Y)\) if \(\text{clear}(X), \text{on}(X, Y), \text{handempty}\)

We now list the translation \(\Pi^{\text{Block}}\) presented to Smodels for a particular problem domain. \(\Pi^{\text{Block}}\) consisted of the following parts: \(\Pi^{\text{just}}\) consisted of the following parts: \(\Pi^{\text{just}} = 1(a) \cup \ldots \cup 1(f), \Pi^{\text{aux}} = 1(g), \Pi^{\text{final}} = 1(h), \Pi^{\text{goal}} = 2(a) \cup 2(b), \Pi^{\text{aux}} = 2(c), \Pi^{\text{aux}} = 2(d), \Pi^{\text{final}} = 2(e), \Pi^{\text{goal}} = 2(f)\), in Smodels syntax.

1. The domain-dependent problem description module.

(a) Defining an upper bound on the number of steps in a plan:

\[\text{const} \text{length} = 7.\]

(b) Defining objects in the problem domain:

\(\text{block}(a), \text{block}(b), \text{block}(c), \text{block}(d)\).

(c) Defining initial state for the problem domain:

\(\text{initially}(\text{clear}(c)), \text{initially}(\text{clear}(a)).\)

\(\text{initially}(\text{clear}(b)), \text{initially}(\text{clear}(d)).\)

\(\text{initially}(\text{ontable}(c)), \text{initially}(\text{ontable}(a)).\)

\(\text{initially}(\text{ontable}(b)), \text{initially}(\text{ontable}(d)).\)

\(\text{initially}(\text{handempty}).\)

(d) Defining goal state for the problem domain:

\(\text{finally}(\text{on}(d, c)), \text{finally}(\text{on}(c, b)).\)

\(\text{finally}(\text{on}(b, a)).\)

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

\(\text{fluent}(\text{on}(X, Y)) \leftarrow \text{neg}(X, Y), \text{block}(X), \text{block}(Y).\)

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

\(\text{action}(\text{pick\,up}(X)) \leftarrow \text{block}(X).\)

(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 \(\text{put\,down}(X)\). The rules for the other actions are similar.

\(\leftarrow \text{block}(X), \text{time}(T), T < \text{length}, \not\text{holds}(\text{holding}(X), T), \text{occurs}(\text{put\,down}(X), T).\)

(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 \(\text{goal}(T)\) 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 \(\text{initially}\) predicate using the following rules.

\(\text{holds}(F, 1) \leftarrow \text{literal}(F), \text{initially}(F).\)

\(\text{holds}(\neg F, 1) \leftarrow \text{l fluent}(F), \not\text{initially}(F).\)

(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.

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

4 Representing temporal constraints

In this section, we show that Bacchus and Kabanza’s style temporal constraints [BK001] (control knowledge) can be easily encoded. The temporal constraints are specified as separate rules and constraints and are simply added to the program \(\Pi^{\text{Block}}\).

1. Defining control knowledge axioms:

(a) Defining bad tower below a block \(X\):

\(\text{holds}(\text{bad\,tower\,below}(X), T) \leftarrow \text{time}(T), \text{block}(X), \text{block}(Y), \text{neg}(X, Y), \text{holds}(\text{ontable}(X), T), \text{finally}(\text{on}(X, Y)).\)

\(\text{holds}(\text{bad\,tower\,below}(X), T) \leftarrow \text{time}(T), \text{block}(X), \text{block}(Y), \text{neg}(X, Y), \text{holds}(\text{on}(X, Y), T), \text{finally}(\text{ontable}(X)).\)

\(\text{holds}(\text{bad\,tower\,below}(X), T) \leftarrow \text{time}(T), \text{block}(X), \text{block}(Y), \text{neg}(X, Y), \text{holds}(\text{on}(X, Y), T), \text{finally}(\text{holding}(Y)).\)

\(\text{holds}(\text{bad\,tower\,below}(X), T) \leftarrow \text{time}(T), \text{block}(X), \text{block}(Y), \text{neg}(X, Y), \text{holds}(\text{on}(X, Y), T), \text{finally}(\text{clear}(Y)).\)

\(\text{holds}(\text{bad\,tower\,below}(X), T) \leftarrow \text{time}(T), \text{block}(X), \text{block}(Y), \text{block}(Z), \text{neg}(X, Y), \text{neg}(Z, Y), \text{neg}(Z, X), \text{holds}(\text{on}(X, Y), T), \text{finally}(\text{on}(X, Z)).\)

\(\text{holds}(\text{bad\,tower\,below}(X), T) \leftarrow \text{time}(T), \text{block}(X), \text{block}(Y), \text{block}(Z), \text{neg}(X, Z), \text{neg}(X, Y), \text{neg}(Z, Y), \text{neg}(Z, X), \text{holds}(\text{on}(X, Y), T), \text{finally}(\text{on}(Z, Y)).\)

\(\text{holds}(\text{bad\,tower\,below}(X), T) \leftarrow \text{time}(T), \text{block}(X), \text{block}(Y), \text{holds}(\text{on}(X, Y), T), \text{neg}(X, Y), \text{holds}(\text{bad\,tower\,below}(Y), T).\)

(b) Defining good tower:

\(\text{holds}(\text{good\,tower}(X), T) \leftarrow \text{time}(T),\)
block(X), holds(clear(X), T), not finally(holding(X)), not holds(bad_tower_below(X), T).

(c) Defining bad tower:
holds(bad_tower_below(X), T) ← time(T), block(X), holds(clear(X), T), not holds(good_tower(X), T).

2. Defining temporal constraints:
(a) Do not destroy good towers:
← 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:
← 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:
← 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 AER, an extended version of A that allows ramification constraints. AER has a new type of proposition called static causal propositions (sc-propositions) of the following form:

\[ p_1, \ldots, p_m, \neg q_{m+1}, \ldots, \neg q_n \text{ causes } f \] (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 \( \Pi^{Block\_rich}_1 \), 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 \( \Pi^{Block}_1 \).
(e) Defining fluents:
   i. The basic fluents are defined as the fluents are defined in \( \Pi^{Block}_1 \).
   ii. The defined fluents are declared using the following rules.
   \[ \text{de fluent}(clear(X)) \leftarrow block(X). \]
   \[ \text{de fluent}(handempty). \]
(f-g) (f) and (g) are the same as in \( \Pi^{Block}_1 \).
(h) Effects of actions: The effects of actions on basic fluents are defined as in \( \Pi^{Block}_1 \).
   i. The relation between defined fluents and basic fluents are expressed using the following rules:
   \[ holds(clear(X), T) \leftarrow block(X), time(T), \]
   \[ holds(on(X, Y), T), not holds(neg(clear(X)), T). \]
   \[ holds(neg(clear(X)), T) \leftarrow block(X), \]
   \[ block(Y), neq(X, Y), time(T), \]
   \[ holds(on(Y, X), T). \]
   \[ holds(neg(handempty), T) \leftarrow time(T), \]
   \[ not holds(neg(handempty), T). \]
   \[ holds(handempty, T) \leftarrow time(T), \]
   \[ holds(handempty, T). \]
   \[ holds(neg(clear(X)), T) \leftarrow block(X), \]
   \[ block(Y), neq(X, Y), time(T), \]
   \[ holds(on(Y, X), T). \]
   \[ \text{Following is the qualification constraint:} \]
   \[ \leftarrow block(X), block(Y), block(Z), time(T), \]
   \[ neq(X, Z), neq(X, Y), neq(Z, Y), \]
   \[ holds(on(X, Y), T), holds(on(Z, Y), T). \]
   \[ \text{k) The ramification constraints are based on the following causal relationship between basic fluents:} \]
   \[ static\_causes(on(X, Z), neg(on(X, Y))) \leftarrow block(X), block(Y), block(Z), time(T), \]
   \[ neq(X, Z), neq(X, Y), neq(Z, Y). \]
   \[ static\_causes(on(X, Y), neg(holding(X))) \leftarrow block(X), block(Y), neg(X, Y). \]
   \[ static\_causes(on(X, Y), neg(ontable(X))) \leftarrow block(X), block(Y), neg(X, Y). \]
   \[ static\_causes(ontable(X), neg(holding(X))) \leftarrow block(X), block(Y), neg(X, Y). \]
   \[ static\_causes(ontable(X), neg(on(X, Y))) \leftarrow block(X), block(Y), neg(X, Y). \]
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 of a blocks world problem in AIPS98 is described as follows:

\[
\text{puton}(X, Y) \quad \text{causes} \; \text{on}(X, Y) \wedge \neg \text{clear}(Y) \quad \text{if} \; X \neq \text{table}
\]

\[
\text{puton}(X, Y) \quad \text{causes} \; \neg \text{on}(X, Z) \quad \text{if} \; \text{on}(X, Z)
\]

\[
\text{puton}(X, Y) \quad \text{causes} \; \text{clear}(Z) \quad \text{if} \; X \neq \text{table}, \quad Z \neq \text{table}, \text{on}(X, Z)
\]

\[
\text{executable} \quad \text{puton}(X, Y) \quad \text{if} \; X \neq \text{table}, Y \neq \text{table}, \quad \text{clear}(X), \text{clear}(Y)
\]

\[
\text{executable} \quad \text{puton}(X, Y) \quad \text{if} \; Y = \text{table}, \text{clear}(X)
\]

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 \(\Pi_1^\text{Block}\), with the exception that we need to add \(\text{block}(\text{table}) \leftarrow \).

(g) We now give a rule corresponding to the executability of the action \(\text{puton}\). (We need two other rules, which we do not present here.)

\[
\leftarrow \text{time}(T), T < \text{length}, \text{block}(X), \text{block}(Y), \quad \text{not} \; \text{holds}(\text{clear}(X), T), \text{occurs}(\text{puton}(X, Y), T).
\]

(h) Effects of actions: Due to lack of space we only give one of the four rules that encode the effect of the action \(\text{puton}(X)\).

\[
\text{holds}(\text{clear}(Z), T + 1) \leftarrow \text{time}(T), T < \text{length}, \quad \text{block}(X), \text{block}(Y), \text{block}(Z), \quad \text{not} \; \text{holds}(\text{clear}(X), T), \text{occurs}(\text{puton}(X, Y), T).
\]

2. The domain-independent module is same as those of \(\Pi_1^\text{Block}\).

5.3 Planning in ADL domains together with ramification and qualification

In this subsection, we further modify the encoding \(\Pi_2^\text{Block}\) to allow qualification and ramification constraints. We refer to the new encoding as \(\Pi_2^\text{Block,rich}\). The process of modification is similar to the modification from \(\Pi_1^\text{Block}\) to \(\Pi_2^\text{Block,rich}\).

We have only \(\text{on}(X, Y)\) as the basic fluent in this case. The only defined fluent is \(\text{clear}(X)\). We now describe the major differences between \(\Pi_2^\text{Block}\) and \(\Pi_2^\text{Block,rich}\).

1. The domain-dependent problem description module:

(a-g) Same as those of \(\Pi_2^\text{Block}\) except that we need to define \(\text{on}(X, Y)\) and \(\text{clear}(X)\) as defined fluents.

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

\[
\text{holds}(\text{on}(X, Y), T + 1) \leftarrow \text{time}(T), \quad T < \text{length}, \text{block}(X), \text{block}(Y), X \neq Y, \quad \text{not} \; \text{holds}(\text{on}(X, Y), T), \text{occurs}(\text{puton}(X, Y), T).
\]

(i) The following rules express the relation between defined fluents and basic fluents:

\[
\text{holds}(\text{clear}(X), T) \leftarrow \text{block}(X), \text{block}(Y), \quad \text{not} \; \text{holds}(\text{on}(X, Y), T), \\
\text{not} \; \text{holds}(\text{not}(\text{clear}(X)), T).
\]

\[
\text{holds}(\text{clear}(X), T) \leftarrow \text{block}(X), \text{block}(Y), \quad \text{not} \; \text{holds}(\text{not}(\text{clear}(X)), T), \\
\text{holds}(\text{on}(X, Y), T), \\
\text{not} \; \text{holds}(\text{not}(\text{clear}(X)), T).
\]

(j) Following is the qualification constraint:

\[
\leftarrow \text{block}(X), \text{block}(Y), \text{block}(Z), \text{time}(T), \quad \text{not} \; \text{holds}(\text{on}(X, Y), T), \\
\text{not} \; \text{holds}(\text{on}(Z, Y), T).
\]

(k) The ramification is based on the following:

\[
\text{static.causes}(\text{on}(X, Y), \text{neg}(\text{on}(X, Z))) \leftarrow \text{block}(X), \text{block}(Y), \text{block}(Z), \text{neg}(X, Y), \\
\text{neg}(X, Z), \text{neg}(Y, Z), \text{neg}(X, \text{table}).
\]

2. The domain-independent module:

(a-f) Same as those of \(\Pi_2^\text{Block,rich}\).

(g) The ramification constraints are accounted for by the following rule:

\[
\text{holds}(F, T) \leftarrow \text{literal}(F), \text{literal}(G), \text{neg}(F, G), \text{time}(T), \\
\text{holds}(G, T), \text{static.causes}(G, F).
\]

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 \(\Pi_1^\text{Block}\), \(\Pi_1^\text{Block,rich}\) and \(\Pi_2^\text{Block}\), \(\Pi_2^\text{Block,rich}\) 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). $\Pi_{1}^{Block, rich}$ runs much faster and scales well when number of blocks and the plan length increase. However, the encoding $\Pi_{1}^{Block}$, $\Pi_{2}^{Block, rich}$ and $\Pi_{2}^{Block}$ 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 $\Pi_{1}^{Block}$ 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


<table>
<thead>
<tr>
<th>Problems (Names / length)</th>
<th>$\Pi_{1}^{Block}$</th>
<th>$\Pi_{1}^{Block, rich}$</th>
<th>$\Pi_{2}^{Block}$</th>
<th>$\Pi_{2}^{Block, rich}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-0 (6)</td>
<td>0.11</td>
<td>0.10</td>
<td>0.14</td>
<td>0.05</td>
</tr>
<tr>
<td>4-1 (10)</td>
<td>0.14</td>
<td>0.08</td>
<td>0.10</td>
<td>0.11</td>
</tr>
<tr>
<td>4-2 (6)</td>
<td>0.03</td>
<td>0.04</td>
<td>0.06</td>
<td>0.04</td>
</tr>
<tr>
<td>5-0 (12)</td>
<td>0.15</td>
<td>0.05</td>
<td>0.28</td>
<td>0.55</td>
</tr>
<tr>
<td>5-1 (10)</td>
<td>0.54</td>
<td>0.46</td>
<td>0.94</td>
<td>0.25</td>
</tr>
<tr>
<td>5-2 (16)</td>
<td>1.09</td>
<td>1.14</td>
<td>0.95</td>
<td>1.28</td>
</tr>
<tr>
<td>6-0 (12)</td>
<td>3.00</td>
<td>3.95</td>
<td>0.96</td>
<td>0.97</td>
</tr>
<tr>
<td>6-1 (10)</td>
<td>1.12</td>
<td>2.28</td>
<td>1.73</td>
<td>0.58</td>
</tr>
<tr>
<td>6-2 (20)</td>
<td>86.59</td>
<td>10.69</td>
<td>53.10</td>
<td>0.00</td>
</tr>
<tr>
<td>7-0 (24)</td>
<td>6650.12</td>
<td>12.06</td>
<td>10.89</td>
<td>17.71</td>
</tr>
<tr>
<td>7-1 (22)</td>
<td>1022.00</td>
<td>1201.33</td>
<td>1801.16</td>
<td>11.01</td>
</tr>
<tr>
<td>7-2 (20)</td>
<td>147.93</td>
<td>448.78</td>
<td>2431.50</td>
<td>108.62</td>
</tr>
<tr>
<td>8-0 (18)</td>
<td>7958.56</td>
<td>1457.89</td>
<td>1401.33</td>
<td>10.08</td>
</tr>
<tr>
<td>8-1 (20)</td>
<td>14.19</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8-2 (16)</td>
<td>8.29</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9-0 (29)</td>
<td>150.81</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9-1 (32)</td>
<td>94.05</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10-0 (34)</td>
<td>148.87</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10-1 (34)</td>
<td>278.82</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problems (Names / length)</th>
<th>$\Pi_{1}^{Block}$</th>
<th>$\Pi_{1}^{Block, rich}$</th>
<th>$\Pi_{2}^{Block}$</th>
<th>$\Pi_{2}^{Block, rich}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-0 (6)</td>
<td>0.02</td>
<td>0.03</td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td>4-1 (10)</td>
<td>0.12</td>
<td>0.09</td>
<td>0.11</td>
<td>0.03</td>
</tr>
<tr>
<td>4-2 (6)</td>
<td>0.05</td>
<td>0.14</td>
<td>0.08</td>
<td>0.05</td>
</tr>
<tr>
<td>5-0 (12)</td>
<td>0.45</td>
<td>0.34</td>
<td>0.33</td>
<td>0.23</td>
</tr>
<tr>
<td>5-1 (10)</td>
<td>0.57</td>
<td>0.41</td>
<td>0.43</td>
<td>0.15</td>
</tr>
<tr>
<td>5-2 (16)</td>
<td>0.14</td>
<td>0.31</td>
<td>0.56</td>
<td>0.15</td>
</tr>
<tr>
<td>6-0 (12)</td>
<td>0.74</td>
<td>0.35</td>
<td>0.60</td>
<td>0.43</td>
</tr>
<tr>
<td>6-1 (10)</td>
<td>0.47</td>
<td>0.23</td>
<td>0.35</td>
<td>0.28</td>
</tr>
<tr>
<td>6-2 (20)</td>
<td>3.82</td>
<td>0.99</td>
<td>1.34</td>
<td>1.24</td>
</tr>
<tr>
<td>7-0 (24)</td>
<td>2.78</td>
<td>3.34</td>
<td>2.81</td>
<td>3.44</td>
</tr>
<tr>
<td>7-1 (22)</td>
<td>34.39</td>
<td>4.30</td>
<td>2.39</td>
<td>2.59</td>
</tr>
<tr>
<td>7-2 (20)</td>
<td>6.87</td>
<td>3.80</td>
<td>2.14</td>
<td>1.95</td>
</tr>
<tr>
<td>8-0 (18)</td>
<td>9.58</td>
<td>4.41</td>
<td>2.44</td>
<td>2.75</td>
</tr>
<tr>
<td>8-1 (20)</td>
<td>23.97</td>
<td>8.31</td>
<td>2.33</td>
<td>2.15</td>
</tr>
<tr>
<td>8-2 (16)</td>
<td>3.53</td>
<td>4.72</td>
<td>1.77</td>
<td>1.67</td>
</tr>
<tr>
<td>9-0 (29)</td>
<td>115.67</td>
<td>8.9</td>
<td>7.22</td>
<td>6.68</td>
</tr>
<tr>
<td>9-1 (32)</td>
<td>170.54</td>
<td>18.16</td>
<td>8.58</td>
<td>14.28</td>
</tr>
<tr>
<td>10-0 (34)</td>
<td>107.84</td>
<td>19.12</td>
<td>18.19</td>
<td>21.80</td>
</tr>
<tr>
<td>10-1 (34)</td>
<td>109.84</td>
<td>39.25</td>
<td>16.12</td>
<td>18.19</td>
</tr>
<tr>
<td>10-2 (34)</td>
<td>99.68</td>
<td>21.28</td>
<td>14.87</td>
<td>20.05</td>
</tr>
<tr>
<td>11-0 (36)</td>
<td>31.81</td>
<td>68.56</td>
<td>19.28</td>
<td>24.10</td>
</tr>
<tr>
<td>11-1 (32)</td>
<td>42.27</td>
<td>41.50</td>
<td>21.50</td>
<td>24.33</td>
</tr>
<tr>
<td>11-2 (38)</td>
<td>44.32</td>
<td>64.35</td>
<td>22.55</td>
<td>37.63</td>
</tr>
<tr>
<td>12-0 (36)</td>
<td>280.43</td>
<td>322.32</td>
<td>26.08</td>
<td>38.24</td>
</tr>
<tr>
<td>12-1 (42)</td>
<td>199.58</td>
<td>107.96</td>
<td>45.80</td>
<td>86.45</td>
</tr>
<tr>
<td>13-0 (44)</td>
<td>68.57</td>
<td>112.21</td>
<td>49.53</td>
<td>97.00</td>
</tr>
<tr>
<td>14-0 (48)</td>
<td>85.89</td>
<td>127.73</td>
<td>63.08</td>
<td>63.11</td>
</tr>
<tr>
<td>14-1 (46)</td>
<td>1406.47</td>
<td>1246.96</td>
<td>77.14</td>
<td>129.36</td>
</tr>
<tr>
<td>14-2 (44)</td>
<td>173.71</td>
<td>621.33</td>
<td>75.44</td>
<td>101.20</td>
</tr>
</tbody>
</table>