In Defense of PDDL Axioms[*]

Sylvie Thiébaux
Computer Sciences Laboratory
The Australian National University
Canberra, ACT 0200, Australia
Sylvie.Thiebaux@anu.edu.au

Jörg Hoffmann and **Bernhard Nebel**
Institut für Informatik
Universität Freiburg
D-79110 Freiburg, Germany
<last-name>@informatik.uni-freiburg.de

Abstract

There is controversy as to whether explicit support for PDDL-like axioms and derived predicates is needed for planners to handle real-world domains effectively. Many researchers have deplored the lack of precise semantics for such axioms, while others have argued that it might be best to compile them away. We propose an adequate semantics for PDDL axioms and show that they are an essential feature by proving that it is impossible to compile them away if we restrict the growth of plans and domain descriptions to be polynomial. These results suggest that adding a reasonable implementation to handle axioms inside the planner is bene£cial for the performance. Our experiments con£rm this suggestion.

Motivation

It is not uncommon for planners to support *derived* predicates, whose truth in the current state is inferred from that of some *basic* predicates via some *axioms* under the closed world assumption. While basic predicates may appear as effects of actions, derived ones may only be used in preconditions, effect contexts and goals. Planners in this family include the partial order planner UCPOP (Barrett *et al.* 1995), the HTN planner SHOP (Nau *et al.* 1999), and the heuristic search planner GPT (Bonet & Thiébaux 2003), to cite but a few. The original version of PDDL (McDermott 1998), the International Planning Competition language, also featured such axioms and derived predicates. However, these were never used in competition events, and did not survive PDDL2.1, the extension of the language to temporal planning (Fox & Long 2002).

This is unfortunate, as the lack of axioms impedes the ability to elegantly and concisely represent real-world domains. Such domains typically require checking complex conditions which are best built hierarchically, from elementary conditions on the state variables to increasingly abstract ones. Without axioms, preconditions and effect contexts quickly become unreadable, or postconditions are forced to include supervenient properties which are just logical consequences of the basic ones—that is when we are lucky and extra actions do not need to be introduced or action descriptions customised.

Moreover, axioms provide a natural way of capturing the effects of actions on common real world structures such as paths or ¤ows, e.g. electricity ¤ows, chemical ¤ows, traf£c ¤ows, etc. For instance, a serious benchmark contender for the 2004 competition is a deterministic version of the power supply restoration problem described by Thiébaux and Cordier (2001). Given a network consisting of power sources, electric lines and switches, an important aspect of the problem is to determine which are the lines currently fed by the various sources, and how feeding is affected when opening or closing switches. Computing and updating "fed" following a switching operation requires traversing the possible paths of network. There is no intuitive way to do this in the body of a PDDL action, while a recursive axiomatisation of "fed" from the current positions (open or closed) of the switches is relatively straightforward (Bonet & Thiébaux 2003).[1]

The most common criticism of the original PDDL axioms was that their semantics was ill-speci£ed, and that the conditions under which the truth of the derived predicates could be uniquely determined were unclear. We remedy this by providing a clear semantics for PDDL axioms while remaining consistent with the original description in (McDermott 1998). In particular, we identify conditions that are suf£cient to ensure that the axioms have an unambiguous meaning, and explain how these conditions can ef£ciently be checked.

Another common view is that axioms are a non-essential language feature which it might be better to compile away than to deal with explicitly, compilation offering the advantage of enabling the use of more ef£cient, simple, standard planners without speci£c treatment (Gazen & Knoblock 1997; Garagnani 2000; Davidson & Garagnani 2002). We bring new insight to this issue. We give evidence that axioms add signi£cant expressive power to PDDL. We take "expressive power" to be a measure of how concisely domains and plans can be expressed in a formalism and use the notion of compilability to analyse that (Nebel 2000). As it turns out, axioms are an essential feature because it is impossible to compile them away—provided we require

[*]A short version of this paper will appear in the proceedings of IJCAI-03. The full proofs of the theorems appear in the techreport: http : //csl.anu.edu.au/~thiebaux/papers/trarp0103.pdf.

[1]In that respect, PDDL axioms offer advantages over the use of purely logical axioms as in the original version of STRIPS (Lifschitz 1987).

Figure 1 Syntax of PDDL with axioms

```
<domain>                  ::= (define (domain <name>)
                                [<constant-def>]
                                [<predicates-def>]
                                [<axiom-def>*]
                                <action-def>*)
<constants-def>           ::= (:constants <name>+)
<predicate-def>           ::= (:predicates <skeleton>+)
<skeleton>                ::= (<predicate> <variable>*)
<predicate>               ::= <name>
<variable>                ::= ?<name>
<axiom-def>               ::= (:derived <skeleton> formula)
<formula>                 ::= <atomic-formula>
<formula>                 ::= (not <formula>)
<formula>                 ::= (and <formula> <formula>+)
<formula>                 ::= (or <formula> <formula>+)
<formula>                 ::= (imply <formula> <formula>)
<formula>                 ::= (exists (<variable>+) formula)
<formula>                 ::= (forall (<variable>+) formula)
<atomic-formula>          ::= (<predicate> <term>*)
<ground-atomic-formula>   ::= (<predicate> <name>*)
<term>                    ::= <name>
<term>                    ::= <variable>
<action-def>              ::= (:action <name>
                               :parameters (<variable>*)
                               <action-def body>)
<action-def body>         ::= [:precondition <formula>]
                               :effect <eff-formula>
<eff-formula>             ::= <one-eff-formula>
<eff-formula>             ::= (and <one-eff-formula>
                                   <one-eff-formula>+)
<one-eff-formula>         ::= <atomic-effs>
<one-eff-formula>         ::= (when formula <atomic-effs>)
<one-eff-formula>         ::= (forall (<variable>+) <atomic-effs>)
<one-eff-formula>         ::= (forall (<variable>+)
                                (when formula <atomic-effs>))
<atomic-effs>             ::= <literal>
<atomic-effs>             ::= (and <literal> <literal>+)
<literal>                 ::= <atomic-formula>
<literal>                 ::= (not <atomic-formula>)
<task>                    ::= (define (task <name>)
                                (:domain <name>)
                                <object declaration>
                                <init>
                                <goal>)
<object declaration>      ::= (:objects <name>*)
<init>                    ::= (:init <ground-atomic-formula>*)
<goal>                    ::= (:goal <formula>)
```

the domain descriptions to grow only polynomially and the plans to grow only polynomially in the size of the original plans and domain descriptions. Of course, if we allow for exponential growth, then compilations become possible and we specify one such transformation, which, unlike those previously published (Gazen & Knoblock 1997; Garagnani 2000; Davidson & Garagnani 2002), works without restriction. However, the above mentioned results suggest that it might be much more ef£cient to deal with axioms inside the planner than to compile them away. In fact, our experiments with FF (Hoffmann & Nebel 2001) suggest that adding even a simple implementation of axioms to a planner clearly outperforms the original version of the planner solving the compiled problem.

Syntax and Semantics

This paper remains in the sequential planning setting. We therefore start from the syntax of PDDL2.1 level 1, which is essentially that of the version of PDDL with ADL actions used in the 2000 planning competition (Bacchus 2000). For clarity, we omit types. Although we see axioms with conditions on numeric ¤uents, such as those featured in PDDL2.1 level 2, as very desirable, we do not consider them here for simplicity.

The syntax of PDDL with axioms is given in Figure 1. `<axiom-def>` is the only addition to the original syntax. Let $\mathcal{B}$ and $\mathcal{D}$ be two sets of predicate symbols with $\mathcal{B} \cap \mathcal{D} = \emptyset$, called the set of basic and derived predicates, respectively. Symbols in $\mathcal{D}$ are not allowed to appear in the initial state description and in atomic effects of actions, but may appear in preconditions, effect contexts, and goals. The domain description features a set of axioms A. These have the form $(:\texttt{derived}\ (d\ ?\vec{x})\ (f\ ?\vec{x}))$, where $d \in \mathcal{D}$, and where f is a £rst-order formula built from predicate symbols in $\mathcal{B} \cup \mathcal{D}$ and whose free variables are those in the vector $\vec{x}$.

Intuitively, an axiom $(:\texttt{derived}\ (d\ ?\vec{x})\ (f\ ?\vec{x}))$ means that when $(f\ ?\vec{x})$ is true at the speci£ed arguments in a given state, we should *derive* that $(d\ ?\vec{x})$ is true at those arguments in that same state. Unlike traditional implications, these derivations are not to be contraposed (the negation of f is not derived from the negation of d), and what cannot be derived as true is false (closed world assumption). Because of the closed world assumption, there is never any need to explicitly derive negative literals, so the constraint that the consequent of axioms be *positive* literals does not make us lose generality.

In sum, axioms are essentially (function free) logic program statements (Lloyd 1993). For example, from the basic predicate `on` and the predicate `holding` in Blocks World, we can de£ne the predicate `clear`, as follows:

```
(:derived (clear ?x)
  (and (not (holding ?x))
       (forall (?y) (not (on ?y ?x))))))
```

Another classic is `above`, the transitive closure of `on`, e.g.:

```
(:derived (above ?x ?y)
  (or (on ?x ?y)
      (exists (?z) (and (above ?x ?z)
                        (above ?z ?y)))))
```

In a planning context, it is natural and convenient to restrict attention to so-called *strati£ed* axiom sets—strati£ed logic programs avoid unsafe use of negation and have an unambiguous, well-understood semantics (Apt *et al.* 1987). The idea behind strati£cation is that some derived predicates should £rst be de£ned in terms of the basic ones possibly using negation, or in terms of themselves (allowing for recursion) but *without* using negation. Next, more abstract predicates can be de£ned building on the former, possibly using their negation, or in terms of themselves but without negation, and so on. Thus, a strati£ed axiom set is partitionable into strata, in such a way that the negation normal form[2] (NNF) of the antecedent of an axiom de£ning a predicate belonging to a given stratum uses arbitrary occurrences of predicates belonging to strictly lower strata and *positive* occurrences of predicates belonging to the same stratum. There is no restriction on the use of basic predicates.

De£nition 1 *An axiom set A is strati£ed iff there exists a partition (strati£cation) of the set of derived predicates $\mathcal{D}$ into (non-empty) subsets $\{\mathcal{D}_i, 1 \leq i \leq n\}$ such that for all $(:\texttt{derived}\ (d_i\ ?\vec{x})\ (f\ ?\vec{x})) \in A$:*

[2] In a formula in NNF, negation occurs only in literals. NNF is computable in linear time in the length of the formula.

1. *if d_j appears in* $\mathrm{NNF}(f\ ?\vec{x})$*, then $d_i \in \mathcal{D}_i$ and $d_j \in \mathcal{D}_j$ such that $j \leq i$,*
2. *if d_j appears negated in* $\mathrm{NNF}(f\ ?\vec{x})$*, then $d_i \in \mathcal{D}_i$ and $d_j \in \mathcal{D}_j$ such that $j < i$.*

Note that any strati£cation $\{\mathcal{D}_i, 1 \leq i \leq n\}$ of $\mathcal{D}$ induces a strati£cation $\{A_i, 1 \leq i \leq n\}$ of A in the obvious way: $A_i = \{(:\texttt{derived}\ (d_i\ ?\vec{x})\ (f_i\ ?\vec{x})) \in A \mid d_i \in \mathcal{D}_i\}$. Note also that when no derived predicate occurs negated in the NNF of the antecedent of any axiom, a single stratum suf£ces. Several planning papers have considered this interesting special case (Gazen & Knoblock 1997; Garagnani 2000; Davidson & Garagnani 2002). .

Working through the successive strata, applying axioms in any order within each stratum until a £xed point is reached and then only proceeding to the next stratum, always leads to the same £nal £xed point independently of the chosen strati£cation (Apt *et al.* 1987, p. 116). It is this £nal £xed point which we take to be the meaning of the axiom set.

We now spell out the semantics formally. Since we have a £nite domain and no functions, we identify the objects in the domain with the ground terms (constants) that denote them, and states with £nite sets of ground atoms. More precisely, a state is taken to be a set of ground *basic* atoms: the derived ones will be treated as elaborate descriptions of the basic state. In order to de£ne the semantics, however, we £rst need to consider an extended notion of "state" consisting of a set S of basic atoms and an arbitrary set D of atoms in the derived vocabulary. The modeling conditions for extended states are just the ordinary ones of £rst order logic, as though there were no relationship between S and D. Where $?\vec{x}$ denotes a vector of variables and $\vec{\mathtt{t}}$ denotes a vector of ground terms, we de£ne:

De£nition 2

$\langle S, D\rangle \models (b\ \vec{\mathtt{t}})$ *for* $b \in \mathcal{B}$ *iff* $(b\ \vec{\mathtt{t}}) \in S$
$\langle S, D\rangle \models (d\ \vec{\mathtt{t}})$ *for* $d \in \mathcal{D}$ *iff* $(d\ \vec{\mathtt{t}}) \in D$
$\langle S, D\rangle \models (\texttt{not}\ f)$ *iff* $\langle S, D\rangle \not\models f$
$\langle S, D\rangle \models (\texttt{and}\ f_1\ f_2)$ *iff* $\langle S, D\rangle \models f_1$ *and* $\langle S, D\rangle \models f_2$
$\langle S, D\rangle \models (\texttt{or}\ f_1\ f_2)$ *iff* $\langle S, D\rangle \models f_1$ *or* $\langle S, D\rangle \models f_2$
$\langle S, D\rangle \models (\texttt{forall}\ (?\vec{x})\ (f\ ?\vec{x}))$ *iff* $\langle S, D\rangle \models (f\ \vec{\mathtt{t}})$ *for all* $\vec{\mathtt{t}}$
$\langle S, D\rangle \models (\texttt{exists}\ (?\vec{x})\ (f\ ?\vec{x}))$ *iff* $\langle S, D\rangle \models (f\ \vec{\mathtt{t}})$ *for some* $\vec{\mathtt{t}}$

Applying axiom $a \equiv (:\texttt{derived}\ (d\ ?\vec{x})\ (f\ ?\vec{x}))$ in a state S augmented with derived atoms D, results in the set $[\![a]\!](S, D)$ of further derived atoms:

De£nition 3

$$[\![a]\!](S, D) = \left\{(d\ \vec{\mathtt{t}}) \mid \langle S, D\rangle \models (f\ \vec{\mathtt{t}}),\ \vec{\mathtt{t}} \textit{ is ground}\right\}$$

Given this, we associate stratum A_i with the function $[\![A]\!]_i$ which maps a given basic state S to the least £xed point attainable by applying the axioms in A_i starting from the extended state consisting of S and of the set of ground derived atoms returned at the previous stratum by $[\![A]\!]_{i-1}$. The strati£ed axiom set A denotes the function $[\![A]\!] = [\![A]\!]_n$:

Algorithm 1 Strati£cation

```
function STRATIFY(D, A)
  for each i ∈ D do
    for each j ∈ D do
      R[i, j] ← 0
  for each (: derived (j ?x) (f ?x)) ∈ A do
    for each i ∈ D do
      if i occurs negatively in NNF(f ?x) then
        R[i, j] ← 2
      else if i occurs positively in NNF(f ?x) then
          R[i, j] ← MAX(1, R[i, j])
  for each j ∈ D do
    for each i ∈ D do
      for each k ∈ D do
        if MIN(R[i, j], R[j, k]) > 0 then
          R[i, k] ← MAX(R[i, j], R[j, k], R[i, k])
  if ∀i ∈ D R[i, i] ≠ 2 then
    stratification ← ∅, remaining ← D, level ← 1
    while remaining ≠ ∅ do
      stratum ← ∅
      for each j ∈ remaining do
        if ∀i ∈ remaining R[i, j] ≠ 2 then
          stratum ← stratum ∪ {j}
      remaining ← remaining \ stratum
      stratification ← stratification ∪ {(level, stratum)}
      level ← level + 1
    return stratification
  else fail
```

De£nition 4 *Let $\{A_i, 1 \leq i \leq n\}$ be an arbitrary strati£cation for a strati£ed axiom set A. For each state S, let:*

$$[\![A]\!]_0(S) = \emptyset, \quad \textit{and for all } 1 \leq i \leq n$$
$$[\![A]\!]_i(S) = \bigcap\Big\{D \mid \bigcup_{a \in A_i} [\![a]\!](S, D) \cup [\![A]\!]_{i-1}(S) \subseteq D\Big\}$$

Then $[\![A]\!](S)$ is de£ned as $[\![A]\!]_n(S)$.

Finally, given a strati£ed axiom set A, we write $S \models_A f$ to indicate that a formula f composed of both basic and derived predicates holds in state S:

De£nition 5 $\quad S \models_A f \quad \textit{iff} \quad \langle S, [\![A]\!](S)\rangle \models f$

This modeling relation is used when applying an action in state S to check preconditions and effect contexts, and to determine whether S satis£es the goal. This is the only change introduced by the axioms into the semantics of PDDL and completes our statement of the semantics. The rest carries over verbatim from (Bacchus 2000).

From a practical point of view, checking that the axiom set in a domain description is strati£ed and computing a strati£cation can be done in polynomial time in the size of the domain description, using Algorithm 1. The algorithm starts by building a $|\mathcal{D}| \times |\mathcal{D}|$ matrix[3] R such that $R[i, j] = 2$ when it follows from the axioms that predicate i's stratum must be strictly lower than predicate j's stratum, $R[i, j] = 1$ when i's stratum must be lower than j's stratum but not necessarily strictly, and $R[i, j] = 0$ when there is no constraint

[3] By $|\cdot|$ we denote the cardinality of a set.

between the two strata (lines 2-15). R is £rst £lled with the values encoding the status (strict or not) of the base constraints obtained by direct examination of the axioms (lines 5-10). Then, the consequences of the base constraints are computed, similarly as one would compute the transitive closure of a relation (lines 11-15). There exists a strati£cation iff the strict relation encoded in R is irre¤exive, that is iff $R[i, i] \neq 2$ for all $i \in \mathcal{D}$ (line 16). In that case, the strati£cation corresponding to the smallest pre-order consistent with R is extracted, i.e. predicates are put in the lowest stratum consistent with R (lines 17-26).

Axioms Add Signi£cant Expressive Power

It is clear that axioms add something to the expressive power of PDDL. In order to determine how much power is added, we will use the *compilability approach* (Nebel 2000). Basically, what we want to determine is how concisely a planning task can be represented if we compile the axioms away. Furthermore, we want to know how long the corresponding plans in the compiled planning task will become.

In the following, we take a $\text{PDDL}_\mathcal{X}$ planning domain description to be a tuple $\Delta = \langle \mathcal{C}, \mathcal{B}, \mathcal{D}, A, O \rangle$, where $\mathcal{C}$ is the set of constant symbols, $\mathcal{B}$ is the set of basic predicates, $\mathcal{D}$ is the set of derived predicates, A is a strati£ed axiom set as in De£nition 1, and O is a set of action descriptions (with the mentioned restriction on the appearance in atomic effects of the symbols in $\mathcal{D}$). A $\text{PDDL}_\mathcal{X}$ *planning instance* or *task* is a tuple $\Pi = \langle \Delta, \mathcal{I}, \mathcal{G} \rangle$, where Δ is the domain description, and $\mathcal{I}$ and $\mathcal{G}$ are the initial state (a set of ground basic atoms) and goal descriptions (a formula), respectively. The result of applying an action in a (basic) state and what constitutes a valid plan (sequence of actions) for a given planning task are de£ned in the usual way (Bacchus 2000), except that the modeling relation in De£nition 5 is used in place of the usual one. By a PDDL domain description and planning instances we mean those without any axioms and derived predicates, i.e., a PDDL domain description has the form $\langle \mathcal{C}, \mathcal{B}, \emptyset, \emptyset, O \rangle$.

We now use *compilation schemes* (Nebel 2000) to translate $\text{PDDL}_\mathcal{X}$ domain descriptions to PDDL domain descriptions. Such schemes are functions that translate domain descriptions between planning formalisms without any restriction on their computational resources but the constraint that the target domain should be only polynomially larger than the original.[4]

De£nition 6 *A* compilation scheme from $\mathcal{X}$ to $\mathcal{Y}$ *is a tuple of functions* $\mathbf{f} = \langle f_\delta, f_i, f_g \rangle$ *that induces a function F from $\mathcal{X}$-instances $\Pi = \langle \Delta, \mathcal{I}, \mathcal{G} \rangle$ to $\mathcal{Y}$-instances $F(\Pi)$ as follows:*

$$F(\Pi) = \big\langle f_\delta(\Delta), \mathcal{I} \cup f_i(\Delta), \mathcal{G} \wedge f_g(\Delta) \big\rangle$$

and satis£es the following conditions:

1. *there exists a plan for Π iff there exists a plan for $F(\Pi)$,*
2. *and the size of the results of f_δ, f_i, and f_g is polynomial in the size of their argument Δ.*

In addition, we measure the size of the corresponding plans in the target formalism.[5]

[4]We use here a slightly simpli£ed de£nition of compilability.

[5]The size of an instance, domain description, plan, etc. is denoted by $|| \cdot ||$.

De£nition 7 *If a compilation scheme* $\mathbf{f}$ *has the property that for every plan P solving an instance Π, there exists a plan P' solving $F(\Pi)$ such that $||P'|| \leq c \times ||P|| + k$ for positive integer constants c and k, then* $\mathbf{f}$ *is a* compilation scheme preserving plan size linearly, *and if $||P'|| \leq p(||P||, ||\Pi||)$ for some polynomial p, then* $\mathbf{f}$ *is a* compilation scheme preserving plan size polynomially.

From a practical point of view, one can regard compilability preserving *plan size linearly* as an indication that the target formalism is *at least as expressive* as the source formalism. Conversely, if a *super-linear* blowup of the plans in the target formalism is required, this indicates that the source formalism *is more expressive* than the target formalism—a planning algorithm for the target formalism would be forced to generate signi£cantly longer plans for compiled instances, making it probably infeasible to solve such instances. If plans are required to grow even *super-polynomially*, then the increase of expressive power must be dramatic. Incidentally, exponential growth of plan size is necessary to compile axioms away.

In order to investigate the compilability between PDDL and $\text{PDDL}_\mathcal{X}$, we will analyze restricted planning problems such as the *1-step planning problem* and the *polynomial step planning problem*. The former is the problem of whether there exists a 1-step plan to solve a planning task, the latter is the problem whether there exists a plan polynomially sized (for some £xed polynomial) in the representation of the domain description. From the results on the computational complexity of pure DATALOG and DATALOG with strati£ed negation (Dantsin *et al.* 2001), the next theorem is immediate.

Theorem 1 *The 1-step planning problem for $\text{PDDL}_\mathcal{X}$ is EXPTIME-complete, even if all axioms are in pure* DATALOG.

If we now consider PDDL planning tasks, it turns out that the planning problem is considerably easier, even if we allow for polynomial length plans. Since guessing a plan of polynomial size and verifying it can easily be done in polynomial space, the polynomial step PDDL planning problem is obviously in PSPACE. Taking in addition Vardi's (1982) result into account that £rst-order query evaluation over a £nite database is PSPACE-complete, hardness follows as well.

Theorem 2 *The polynomial step planning problem for* PDDL *is PSPACE-complete.*

From these two statements it follows immediately that it is very unlikely that there exists a *polynomial time* compilation scheme from $\text{PDDL}_\mathcal{X}$ to PDDL preserving plan size polynomially. Otherwise, it would be possible to solve all problems requiring exponential time in polynomial space, which is considered as quite unlikely. As argued, however, by Nebel (2000), if we want to make claims about *expressiveness*, then we should not take the computational resources of the compilation scheme into account but allow for computationally unconstrained transformations. Interestingly, even allowing for such unconstrained compilation schemes changes nothing.

Theorem 3 *Unless EXPTIME = PSPACE, there is no compilation scheme from* PDDL$_{\mathcal{X}}$ *(even restricted to pure* DATALOG *axioms) to* PDDL *preserving plan size polynomially.*

Proof Sketch. We use a proof idea similar to the one Kautz and Selman (1992) used to prove that approximations of logical theories of a certain size are not very likely to exist. By using a DATALOG theory in order to describe all instances of the *linearly bounded alternating Turing machine* acceptance problem up to a certain size, which in its general form is EXPTIME-complete (Chandra *et al.* 1981), we get a *polynomial advice string* (Karp & Lipton 1982) if a compilation scheme from PDDL$_{\mathcal{X}}$ to PDDL preserving plan size polynomially exists. This would imply that EXPTIME $\subseteq$ PSPACE/poly. However, by Karp and Lipton's (1982) results, this implies that EXPTIME = PSPACE. ■

Compilations with Exponential Results

While it is impossible to £nd a concise equivalent PDDL planning instance that guarantees short plans, it is possible to come up with a poly-size instance which may have exponentially longer plans in the worst case. Such compilation schemes have been described by e.g. Gazen and Knoblock (1997) and Garagnani (2000) under severe restrictions on the use of negated derived predicates. Speci£cally, these schemes do not work if negated derived predicates appear anywhere in the planning task,[6] and the latter scheme (Garagnani 2000) is further restricted to pure DATALOG axioms.

An interesting contrasting approach is that of Davidson and Garagnani (2002). They propose to compile pure DATALOG axioms solely into conditional effects, which means that the resulting plans will have exactly the same length. However, as is implied by Theorem 3, their domain description suffers a super-polynomial growth.

We now specify a generally applicable compilation scheme producing poly-size instances, which we will use as a baseline in our performance evaluation. In contrast to the schemes mentioned above, it complies with the strati£ed semantics speci£ed while dealing with negated occurrences of derived predicates anywhere in the planning task.

Theorem 4 *There exists a polynomial time compilation scheme* $\mathbf{f} = \langle f_\delta, f_i, f_g \rangle$*, such that for every* PDDL$_{\mathcal{X}}$ *domain description* $\Delta = \langle \mathcal{C}, \mathcal{B}, \mathcal{D}, A, O \rangle$*:* $||f_i(\Delta)|| = c_1$ *and* $||f_g(\Delta)|| = c_2$ *for some constants* c_1 *and* c_2*, and* $f_\delta(\Delta) = \langle \mathcal{C}, \mathcal{B}', \emptyset, \emptyset, O' \rangle$ *is a* PDDL *domain with* $|\mathcal{B}'| \leq |\mathcal{B}| + 3\ |\mathcal{D}| + 2$ *and with* $||O'|| \leq p(||O||, ||A||)$ *for some polynomial* p*.*

Proof Sketch. Figure shows the main elements of the PDDL instances induced by **f**. **f** computes a strati£cation $\{A_i, 1 \leq i \leq n\}$ of the set of axioms A, as previously explained, where in stratum i, each axiom $a_{i,j}$ is of the form (: `derived` $(d_{i,j}\ ?\vec{x}_{i,j})$ $(f_{i,j}\ ?\vec{x}_{i,j})$) for $1 \leq j \leq |A_i|$. **f** encodes each stratum as an extra action `stratum`$_i$ (see lines 5-15 in Figure) which applies all axioms $a_{i,j}$ at this stratum in parallel, records that this was done (`done`$_i$) and whether anything new (`new`) was derived in doing so. Each $a_{i,j}$ is

[6] This remains true even if negation is compiled away as per the Gazen and Knoblock method (Gazen & Knoblock 1997).

Figure 2 PDDL instances induced by **f**

```
1. (: predicates  ; all predicates in B ∪ D
2.                (done_1) ... (done_n)
3.                (fixed_0) ... (fixed_n)
4.                (new))
 for each i ∈ {1, ..., n}
5. (: action stratum_i
6.    : parameters ()
7.    : precondition (and (fixed_{i-1}) (not (fixed_i)))
8.    : effect (and (done_i))
9.                (forall (x⃗_{i,1})
10.                  (when (and (f_{i,1} ?x⃗_{i,1}) (not (d_{i,1} ?x⃗_{i,1})))
11.                        (and (d_{i,1} ?x⃗_{i,1}) (new))))
12.                            ...
13.               (forall (x⃗_{i,|A_i|})
14.                  (when (and (f_{i,|A_i|} ?x⃗_{i,|A_i|}) (not (d_{i,|A_i|} ?x⃗_{i,|A_i|})))
15.                        (and (d_{i,|A_i|} ?x⃗_{i,|A_i|}) (new))))))
16. (: action fixpoint_i
17.    : parameters ()
18.    : precondition (done_i)
19.    : effect (and (when (not(new)) (fixed_i))
20.                (not(new))
21.                (not (done_i))))
 for each o ∈ O
22. (: action NAME(o)
23.    : parameters PARAMETERS(o)
24.    : precondition (and PRECONDITION(o) (fixed_k))
25.    : effect (and EFFECT(o)
26.                (not (fixed_m)) ... (not (fixed_n))
27.                (not (done_m)) ... (not (done_n))
28.                (forall x⃗_{m,1} (not (d_{m,1} ?x⃗_{m,1})))
29.                            ...
30.                (forall x⃗_{n,|A_n|} (not (d_{n,|A_n|} ?x⃗_{n,|A_n|})))))
 Where k = max({i | some d_{i,j} occurs in PRECONDITION(o)}∪{0}) and
 m = min({i | a predicate in some f_{i,j} is modi£ed in EFFECT(o)}∪{n+1})

31. (: init I ∪ (fixed_0))
32. (: goal (and G (fixed_n)))
```

encoded as a universally quanti£ed and conditional effect of `stratum`$_i$—see lines 9-15. To ensure that the precedence between strata is respected, `stratum`$_i$ is only applicable when the £xed point for the previous stratum has been reached (i.e. when `fixed`$_{i-1}$) and the £xed point for the current stratum has not (i.e. when (`not` (`fixed`$_i$)))—see line 7. **f** encodes the £xpoint computation at each stratum i using an extra action `fixpoint`$_i$, which is applicable after a round of one or more applications of `stratum`$_i$ (i.e., when `done`$_i$ is true), asserts that the £xed point has been reached (i.e. `fixed`$_i$) whenever nothing new has been derived during this last round, and resets `new` and `done`$_i$ for the next round—see lines 16-21. Next, the precondition and effect of each action description $o \in O$ are augmented as follows (see lines 22-30). Let $0 \leq k \leq n$ be the highest stratum of any derived predicate appearing in the precondition of o, or 0 if there is no such predicate. Before applying o, we must make sure that the £xed point for that stratum has been computed by adding `fixed`$_k$ to the precondition. Similarly, let $1 \leq m \leq n+1$ be the lowest stratum such that some predicate in the antecedent of some axiom in A_m is modi£ed in the effect of o, or $n+1$ if there is none. After applying o, we may need to re-compute the £xed points for the strata above m, that is, the effect must reset `fixed`, `done`, and the value of all derived propositions, at strata m and above. Finally, $\texttt{fixed}_0$ holds initially, and the goal requires $\texttt{fixed}_n$ to be true. The fact that **f** preserves domain description size polynomially, and the bounds given in theorem 4, follow directly from the construction. ■

It is obvious that a plan P for a planning task Π can be recovered from a plan P' for the compiled planning task $F(\Pi)$, by simply stripping all occurrences of `stratum` and `fixpoint` actions. In the worst case of course, there is no polynomial p such that $||P'|| \leq p(||P||, ||\Pi||)$. Indeed, the worst-case is obtained when, initially and after each action from P, all derived predicates need to be (re)computed and only one proposition is ever derived per application of $\texttt{stratum}_i$ actions. Even if the planner is able to interleave as few $\texttt{fixpoint}_i$ actions as possible with the $\texttt{stratum}_i$ actions, this still leads to a plan of length $||P'|| = ||P|| + (||P||+1)(\sum_{i=1}^{n}(|\overline{\mathcal{D}_i}| + 3)) = ||P|| + (||P||+1)(3n + |\overline{\mathcal{D}}|)$, where $\overline{\mathcal{D}}$ denotes the set of all instances of predicates in $\mathcal{D}$. Observe that $\overline{\mathcal{D}}$ is not polynomially bounded in $|\mathcal{D}|$ and $|\mathcal{C}|$.

Planning: With or Without Axioms?

The absence of a polynomial time compilation scheme preserving plan size linearly not only indicates that axioms bring (much needed) expressive power, but it also suggests that extending a planner to explicitly deal with axioms may lead to much better performance than using a compilation scheme with the original version of the planner. To con£rm this, we extended the FF planner (Hoffmann & Nebel 2001) with a straightforward implementation of axioms—we call this extension $\text{FF}_\mathcal{X}$—and compared results obtained by $\text{FF}_\mathcal{X}$ on $\text{PDDL}_\mathcal{X}$ instances with those obtained by FF on the PDDL instances produced via compilation with **f**.

$\text{FF}_\mathcal{X}$ transforms each axiom $(:\texttt{derived}(d\ ?\vec{x})(f\ ?\vec{x}))$ into an operator with parameters $(?\vec{x})$, precondition $(f\ ?\vec{x})$ and effect $(d\ ?\vec{x})$, with a ¤ag set to distinguish it from a "normal" operator. During the relaxed planning process that FF performs to obtain its heuristic function, the axiom actions are treated as normal actions and can be chosen for inclusion in a relaxed plan. However, the heuristic value only counts the number of *normal* actions in the relaxed plan. During the forward search FF performs, only normal actions are considered; after each application of such an action, the axiom actions are applied so as to obtain the successive £xed points associated with the strati£cation computed by Algorithm 1.

One domain we chose for our experiments is good old Blocks World (BW). In contrast to most other common benchmarks, in BW there is a natural distinction between basic and derived predicates; in particular BW with 4 operators is the only common benchmark domain we are aware of where the strati£cation of the axioms requires more than one stratum. We experimented with two versions of BW:

1op: The version with a single `move` operator. `on` is the only basic predicate, the table being treated as a block. There is a single stratum consisting of `clear` and `above`. Note that `above` is only used in goal descriptions.

4ops: The version with the 4 operators `pickup`, `putdown`, `stack` and `unstack`. The basic predicates are `on` and `ontable`, and the derived ones are `above` and `holding` (stratum 1), as well as `clear` and `handempty` (stratum 2) whose axiomatisations use the negation of `holding`.

For each of those versions, we considered 3 types of planning tasks:

strict: A $\text{PDDL}_\mathcal{X}$ task is built from a given pair of BW states as follows. The £rst state is taken to be the initial one, and the second is converted into an incompletely speci£ed goal description by writing "`above`" whereas one would normally have written "`on`" and omitting the mention of those blocks that would normally have been on the table. Note that expressing the resulting goal using `on` and `ontable` would require exponential space, highlighting once more the utility of derived predicates.

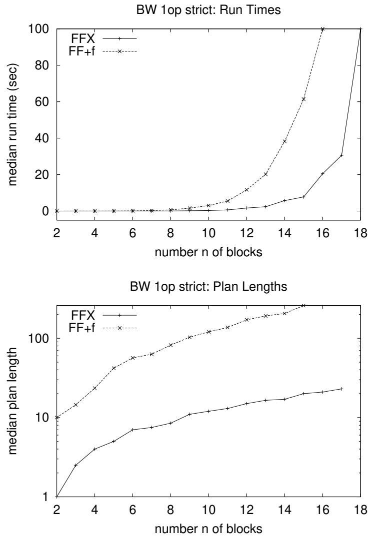
loose: A $\text{PDDL}_\mathcal{X}$ task is built from a single BW state by taking it to be the initial state and asking that any block which is on the table initially end up above all those that were initially not.

one tower: This is the special case of those **loose** tasks for which the initial state has only one tower. In their **1op** versions, those tasks are one of the examples considered by Davidson and Garagnani (2002).

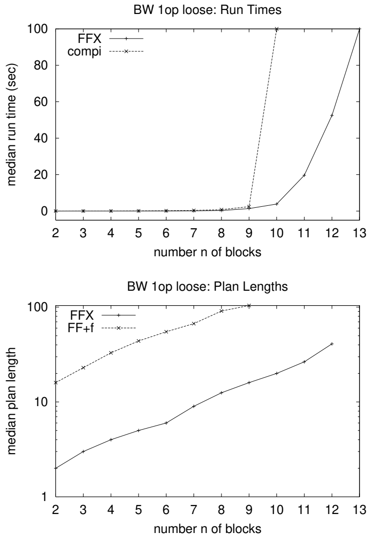
For each combination $\{\textbf{1op}, \textbf{4ops}\} \times \{\textbf{strict}, \textbf{loose}\}$, we generated 30 random instances of each size n (number of blocks), using the random BW states generator provided by Slaney and Thiébaux (2001). For **one tower** tasks of a given size, which are all identical up to a permutation of the blocks, a single instance suf£ces per value of n. The £gures below show the median run-time and median plan length obtained by $\text{FF}_\mathcal{X}$ and FF+**f** and as a function of n for each of the 6 combinations. In all cases but **4ops strict**, the median run-time of $\text{FF}_\mathcal{X}$ shows a signi£cant improvement over that of FF+**f**. For **one tower** tasks, the improvement is dramatic, as $\text{FF}_\mathcal{X}$ £nds the optimal plans whose length is linear in n. With the **strict** and **loose** tasks in contrast, the plans found by $\text{FF}_\mathcal{X}$ are only an order of magnitude larger than those found by FF+**f**. Note that FF's goal-ordering techniques were not used in either versions of the program. Although extending these techniques to deal with axioms is relatively straightforward, we have not invested any time yet in doing so. Goal ordering has been shown to greatly improve the performance of FF on BW, and due to the lack of it, FF's behavior in the above experiments is signi£cantly worse than reported in the literature (Hoffmann & Nebel 2001).

Another domain we ran experiments on is the challenging Power Supply Restoration (PSR) benchmark (Thiébaux & Cordier 2001), which is derived from a real-world problem in the area of power distribution. The domain description requires a number of complex, recursive, derived predicates to axiomatize the power ¤ow, see (Bonet & Thiébaux 2003). We considered a version of the benchmark in which the locations of the faults and the current network con£guration are completely known, and the goal is to resupply all resuppliable lines. For each number $n = 1$ to 7 feeders, we generated 100 random networks with a maximum of

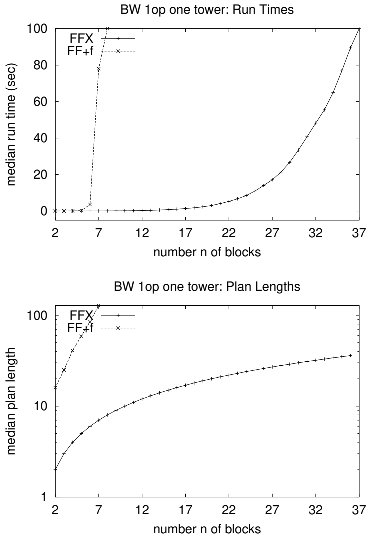
BW: 1op, strict



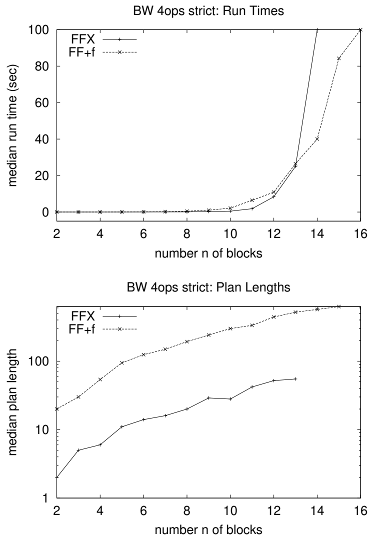
BW: 1op, loose



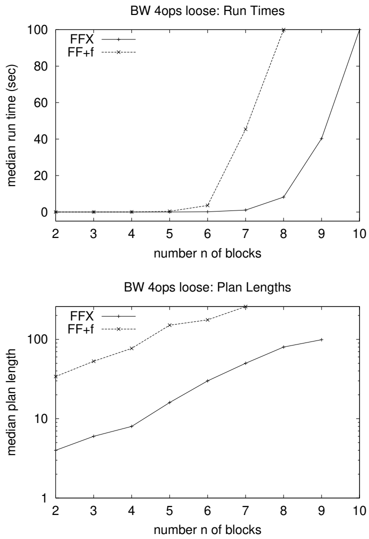
BW: 1op, one tower



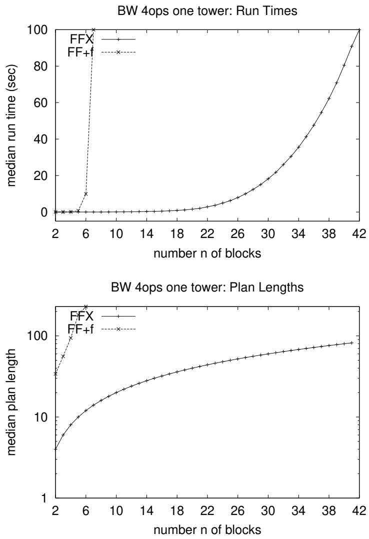
BW: 4ops, strict



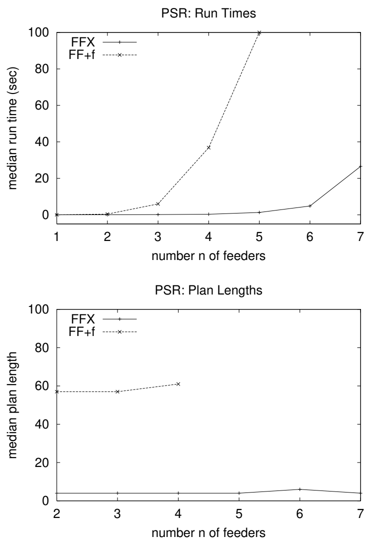
BW: 4ops, loose



BW: 4ops, one tower



PSR: random instances



PSR: known instances

instance				run-time (sec)		plan length	
network	feeders	lines	faults	FF$_\mathcal{X}$	FF+**f**	FF$_\mathcal{X}$	FF+**f**
basic	2	5	1	0.08	2.48	5	70
small-rural	3	7	2	0.79	194.22	6	76
simple	3	7	2	1.17	244.31	7	82
random	3	9	3	8.06	2073.47	7	82
simplified-rural	7	11	2	467.90	-	6	-

3 switches per feeder and with 30% faulty lines. We also considered the networks of increasing dif£culty described in (Bertoli *et al.* 2002; Bonet & Thiébaux 2003): **basic**, **small-rural**, **simple**, **random**, and **simpli£ed-rural**. The left-hand side £gures compare the median run times and plan length for $\mathrm{FF}_{\mathcal{X}}$ and FF+**f** as a function of n on the random instances, while the right-hand one reports run times and plan length on the known instances. A dash (-) indicates that the instance could not be solved within 5000 secs. Again the improvement in performance resulting from handling axioms explicitly is undeniable. In these experiments, the plan length does not vary much with n: with our parameters for the random instances generation, it is clustered around 5 actions for $\mathrm{PDDL}_{\mathcal{X}}$ instances, and around 50 for the compiled instances (the known instances exhibit similar £gures). Yet this makes all the difference between what is solvable in reasonable time and what is not.

Although the domains in these experiments are by no means chosen to show off the worst-case for the compilation scheme, they nevertheless illustrate its drawbacks. The difference of performance we observe is due to the facts that compilation increases the branching factor, increases the plan length, and obscures the computation of the heuristic.

In our BW experiments, we also considered the possibility of compiling the *non-recursive* derived predicates away as suggested by Davidson and Garagnani (2002), by simply substituting their de£nition for them wherever they occur until no occurrence remains. We did not experiment with compiling recursive derived predicate as per their method because this requires signi£cant implementation effort and the authors were unable to provide us with an implementation at the time of writing this paper. Instead, we considered two treatments of the recursive predicates: one using axioms and running $\mathrm{FF}_{\mathcal{X}}$ and the other using compilation via **f** and running the original FF. In **1op** domains, the run-times obtained with the former, resp. the latter, treatment are similar to those obtained by $\mathrm{FF}_{\mathcal{X}}$, resp. by FF+**f** in the £gures above. To be precise, the run-times are slighly larger than those in the £gure for **1op loose** and **1 op one tower**, and slightly lower for **1op strict**. On the other hand, in **4ops** domains, both variants (i.e. regardless of whether the recursive predicates were axiomatized or compiled away) were unable to cope with problems larger than $n = 4$. This is due to the fact that substituting for the non-recursive derived predicates results in operator descriptions with quite complex ADL constructs. These make FF's pre-processing infeasible, as it compiles the ADL constructs away following Gazen & Knoblock [2001] (instantiating the operators, and expanding all quanti£ers in the formulae), and needs to create and simplify thousands of £rst-order formulae even in comparatively small planning tasks. In the **1op** case, preconditions are kept manageable because `clear` is the only derived predicate and its de£nition in terms of `on` is relatively simple, while the **4ops** case suf£ces to make preconditions too challenging. We did not experiment with the other published compilation schemes (Gazen & Knoblock 1997; Garagnani 2000), as they are not applicable to the above domains whose descriptions involve negated derived predicates.

Conclusion

As re¤ected by recent endeavours in the international planning competitions, there is a growing (and, in our opinion, desirable) trend towards more realistic planning languages and benchmark domains. In that context, it is crucial to determine which additional language features are particularly relevant. The main contribution of this paper is to give theoretical and empirical evidence of the fact that axioms *are* important, from both an expressivity and ef£ciency perspective. In addition, we have provided a clear formal semantics for PDDL axioms, identi£ed a general and easily testable criterion for axiom sets to have an unambiguous meaning, and given a compilation scheme which is more generally applicable than those previously published (and also more effective in conjunction with forward heuristic search planners like FF).

Future work will include more extensive empirical studies involving a more elaborate treatment of axioms within FF and planners of different types, as well as the extension of derived predicates and axioms to the context of the numerical and temporal language features recently introduced with PDDL 2.1. Axioms have long been an integral part of action formalisms in the £eld of reasoning about action and change where, much beyond the inference of derived predicate considered here, they form the basis for elegant solutions to the frame and rami£cation problems, see e.g. (McCain & Turner 1995). It is our hope that the adoption of PDDL axioms will eventually encourage the planning community to make greater use of these formalisms.

Acknowledgements

Thanks to Blai Bonet, Marina Davidson, Stefan Edelkamp, Maria Fox, John Lloyd, and John Slaney for fruitful discussions which helped to improve this paper.

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.

Bacchus, F. 2000. Subset of PDDL for the AIPS2000 planning competition. `www.cs.toronto.edu/aips2000`.

Barrett, A.; Christianson, D.; Friedman, M.; Kwok, C.; Golden, K.; Penberthy, S.; Sun, Y.; and Weld, D. 1995. UCPOP User's Manual. Technical Report 93-09-06d, The University of Washington, Computer Science Department.

Bertoli, P.; Cimatti, A.; Slaney, J.; and Thiébaux, S. 2002. Solving power supply restoration problems with planning via symbolic model checking. In *Proc. ECAI*, 576–580.

Bonet, B., and Thiébaux, S. 2003. GPT Meets PSR. In *Proc. ICAPS, to appear*.

Chandra, A. K.; Kozen, D. C.; and Stockmeyer, L. J. 1981. Alternation. *Journal of the ACM* 28(1):114–133.

Dantsin, E.; Eiter, T.; Gottlob, G.; and Voronkov, A. 2001. Complexity and expressive power of logic programming. *ACM Computing Surveys* 33(3):374–425.

Davidson, M., and Garagnani, M. 2002. Pre-processing planning domains containing language axioms. In *Proc. UK PlanSIG workshop*.

Fox, M., and Long, D. 2002. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. `www.dur.ac.uk/d.p.long/IPC/pddl.html`.

Garagnani, M. 2000. A correct algorithm for ef£cient planning with preprocessed domain axioms. In *Research and Development in Intelligent Systems XVII*. Springer.

Gazen, C., and Knoblock, C. 1997. Combining the expressivity of UCPOP with the ef£ciency of Graphplan. In *Proc. ECP*.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Karp, R. M., and Lipton, R. J. 1982. Turing machines that take advice. *L' Enseignement Mathématique* 28:191–210.

Kautz, H. A., and Selman, B. 1992. Forming concepts for fast inference. In *Proc. AAAI*.

Lifschitz, V. 1987. On the semantics of STRIPS. In *Proc. 1986 Workshop on Reasoning about Actions and Plans*, 1–9.

Lloyd, J. 1993. *Foundations of Logic Programming*. Springer.

McCain, N., and Turner, H. 1995. A causal theory of rami£cations and quali£cations. In *Proc. IJCAI-95*.

McDermott, D. 1998. PDDL – The Planning Domain Definition Language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

Nau, D.; Cao, Y.; Lotem, A.; and Munoz-Avila, H. 1999. SHOP: simple hierarchical ordered planner. In *Proc. IJCAI*.

Nebel, B. 2000. On the compilability and expressive power of propositional planning formalisms. *JAIR* 12:271–315.

Slaney, J., and Thiébaux, S. 2001. Blocks world revisited. *Arti£cial Intelligence* 125(1-2):119–153.

Thiébaux, S., and Cordier, M.-O. 2001. Supply restoration in power distribution systems — a benchmark for planning under uncertainty. In *ECP*, 85–95.

Vardi, M. Y. 1982. The complexity of relational query languages. In *Proc. STOC*.