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 beneficial for the performance. Our experiments confirm this suggestion.

1 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 [3], the HTN planner SHOP [18], and the heuristic search planner GPT [4], to cite but a few. The original version of PDDL [17], 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 [8].

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 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 flows, as we then need to reason about how the transitive closure of a basic relation is affected.[1] There is no intuitive way to update transitive closures in the body of a PDDL action, while it is easy to axiomatize them recursively by means of PDDL axioms (see [4] for a power flow example).

The most common criticism of the original PDDL axioms was that their semantics was ill-specified, 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 [17]. In particular, we identify conditions that are sufficient to ensure that the axioms have an unambiguous meaning, and explain how these conditions can efficiently 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 efficient, simple, standard planners without specific treatment [10; 9; 7]. We bring new insight to this issue. We give evidence that axioms add significant 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 [19]. As it turns out, axioms are an essential feature because it is impossible to compile them away—provided we require 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 [10; 9; 7], works without restriction. However, the above mentioned results suggest that it might be much more efficient to deal with axioms inside the planner than to compile them away. In fact, our experiments with FF [11] suggest that adding even a simple implementation of axioms to a planner clearly outperforms the original version of the planner solving the compiled problem.

2 Syntax and Semantics

We remain in the sequential planning setting of PDDL2.1 level 1, which is essentially that of the version of PDDL with ADL actions used in the 2000 planning competition. See [2] for a description of the syntax. For clarity we do not consider types. Although we see axioms with conditions on numeric

*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 [14].

fluents, such as those featured in PDDL2.1 level 2, as very desirable, we do not consider them here for simplicity.

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 first-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 specified 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 [15]. For example, from the basic predicate on and the predicate `holding` in Blocks World, we can define 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 *stratified* axiom sets—stratified logic programs avoid unsafe use of negation and have an unambiguous, well-understood semantics [1]. The idea behind stratification is that some derived predicates should first be defined 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 defined building on the former, possibly using their negation, or in terms of themselves but without negation, and so on. Thus, a stratified axiom set is partitionable into strata, in such a way that the negation normal form[2] (NNF) of the antecedent of an axiom defining 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. Basic predicates may be used freely.

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

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

[2]In a formula in NNF, negation occurs only in literals.

Note that any stratification $\{\mathcal{D}_i, 1 \le i \le n\}$ of $\mathcal{D}$ induces a stratification $\{A_i, 1 \le i \le n\}$ 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 suffices. Several planning papers have considered this special case [10; 9; 7].

Working through the successive strata, applying axioms in any order within each stratum until a fixed point is reached and then only proceeding to the next stratum, always leads to the same final fixed point independently of the chosen stratification [1, p. 116]. It is this final fixed point which we take to be the meaning of the axiom set.

We now spell out the semantics formally. Since we have a finite domain and no functions, we identify the objects in the domain with the ground terms (constants) that denote them, and states with finite 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 define the semantics, however, we first 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 first order logic, as though there were no relationship between S and D. Where $?\vec{x}$ denotes a vector of variables and $\vec{\mathrm{t}}$ denotes a vector of ground terms, we define:

Definition 2

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

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:

Definition 3

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

Given this, we associate stratum A_i with the function $[\![A]\!]_i$ which maps a given basic state S to the least fixed 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 stratified axiom set A denotes the function $[\![A]\!] = [\![A]\!]_n$:

Definition 4 *Let $\{A_i, 1 \le i \le n\}$ be an arbitrary stratification for a stratified axiom set A. For each state S, let:*

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

Then $[\![A]\!](S)$ is defined as $[\![A]\!]_n(S)$.

Finally, given a stratified 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:

Algorithm 1 Stratification

```
1.  function STRATIFY(D, A)
2.    for each i ∈ D do
3.      for each j ∈ D do
4.        R[i, j] ← 0
5.    for each (: derived (j ?x⃗) (f ?x⃗)) ∈ A do
6.      for each i ∈ D do
7.        if i occurs negatively in NNF(f ?x⃗) then
8.          R[i, j] ← 2
9.        else if i occurs positively in NNF(f ?x⃗) then
10.           R[i, j] ← MAX(1, R[i, j])
11.   for each j ∈ D do
12.     for each i ∈ D do
13.       for each k ∈ D do
14.         if MIN(R[i, j], R[j, k]) > 0 then
15.           R[i, k] ← MAX(R[i, j], R[j, k], R[i, k])
16.   if ∀i ∈ D R[i, i] ≠ 2 then
17.     stratification ← ∅, remaining ← D, level ← 1
18.     while remaining ≠ ∅ do
19.       stratum ← ∅
20.       for each j ∈ remaining do
21.         if ∀i ∈ remaining R[i, j] ≠ 2 then
22.           stratum ← stratum ∪ {j}
23.       remaining ← remaining \ stratum
24.       stratification ← stratification ∪ {(level, stratum)}
25.       level ← level + 1
26.     return stratification
27.   else fail
```

Definition 5 $\quad S \models_A f \quad$ *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 satisfies 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 [2].

Checking that the axiom set in a domain description is stratified and computing a stratification 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 between the two strata (lines 2-15). R is first filled 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 stratification iff the strict relation encoded in R is irreflexive, that is iff $R[i, i] \neq 2$ for all $i \in \mathcal{D}$ (line 16). In that case, the stratification 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).

3 Axioms Add Significant 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* [19]. 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.

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

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 stratified axiom set as in Definition 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 defined in the usual way [2], except that the modeling relation in Definition 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* [19] 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]

Definition 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 satisfies 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]

Definition 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 significantly 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.

[4]We use here a slightly simplified definition of compilability.

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

In order to investigate the compilability between PDDL and 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 fixed polynomial) in the representation of the domain description. From the results on the computational complexity of pure DATALOG and DATALOG with stratified negation [6], the next theorem is immediate.

Theorem 1 *The 1-step planning problem for* 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 [20] result into account that first-order query evaluation over a finite 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 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 [19], 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 [13] 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 [5], we get a *polynomial advice string* [12] 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 [12] results, this implies that EXPTIME = PSPACE. ∎

4 Compilations with Exponential Results

While it is impossible to find 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 [10] and Garagnani [9] under severe restrictions on the use of negated derived predicates. Specifically, these schemes do not work if negated derived predicates appear anywhere in the planning task,[6] and the latter scheme [9] is further restricted to pure DATALOG axioms.

Figure 1 PDDL instances induced by **f**

```
(: predicates  ; all predicates in B ∪ D
               (done_1) ... (done_n)
               (fixed_0) ... (fixed_n)
               (new))
for each i ∈ {1, ..., n}
(: action stratum_i
  : parameters ()
  : precondition (and (fixed_{i-1}) (not (fixed_i)))
  : effect (and (done_i))
                (forall (x⃗_{i,1})
                   (when (and (f_{i,1} ?x⃗_{i,1}) (not (d_{i,1} ?x⃗_{i,1})))
                         (and (d_{i,1} ?x⃗_{i,1}) (new))))
                         ...
                (forall (x⃗_{i,|A_i|})
                   (when (and (f_{i,|A_i|} ?x⃗_{i,|A_i|}) (not (d_{i,|A_i|} ?x⃗_{i,|A_i|})))
                         (and (d_{i,|A_i|} ?x⃗_{i,|A_i|}) (new))))))
(: action fixpoint_i
  : parameters ()
  : precondition (done_i)
  : effect (and (when (not(new)) (fixed_i))
                (not(new))
                (not (done_i))))
for each o ∈ O
(: action NAME(o)
  : parameters PARAMETERS(o)
  : precondition (and PRECONDITION(o) (fixed_k))
  : effect (and EFFECT(o)
                (not (fixed_m)) ... (not (fixed_n))
                (not (done_m)) ... (not (done_n))
                (forall x⃗_{m,1} (not (d_{m,1} ?x⃗_{m,1})))
                         ...
                (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 modified in EFFECT(o)}∪{n+1})

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

An interesting contrasting approach is that of Davidson and Garagnani [7]. 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 stratified semantics specified in Section 2 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 4 shows the main elements of the PDDL instances induced by **f**. **f** computes a stratification $\{A_i, 1 \leq i \leq n\}$ of the set of axioms A, as explained in Section 2, 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**

[6] This remains true even if negation is compiled away as per the Gazen and Knoblock method [10].

encodes each stratum as an extra action $\mathtt{stratum}_i$ (see lines 5-15 in Figure 4) which applies all axioms $a_{i,j}$ at this stratum in parallel, records that this was done ($\mathtt{done}_i$) and whether anything new (new) was derived in doing so. Each $a_{i,j}$ is encoded as a universally quantified and conditional effect of $\mathtt{stratum}_i$—see lines 9-15. To ensure that the precedence between strata is respected, $\mathtt{stratum}_i$ is only applicable when the fixed point for the previous stratum has been reached (i.e. when $\mathtt{fixed}_{i-1}$) and the fixed point for the current stratum has not (i.e. when $(\mathtt{not}\ (\mathtt{fixed_i}))$)—see line 7. **f** encodes the fixpoint computation at each stratum i using an extra action $\mathtt{fixpoint}_i$, which is applicable after a round of one or more applications of $\mathtt{stratum}_i$ (i.e., when $\mathtt{done}_i$ is true), asserts that the fixed point has been reached (i.e. $\mathtt{fixed}_i$) whenever nothing new has been derived during this last round, and resets new and $\mathtt{done}_i$ 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 fixed point for that stratum has been computed by adding $\mathtt{fixed}_k$ 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 modified in the effect of o, or $n+1$ if there is none. After applying o, we may need to re-compute the fixed 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, $\mathtt{fixed}_0$ holds initially, and the goal requires $\mathtt{fixed}_n$ 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 $\mathtt{stratum}_i$ actions. Even if the planner is able to interleave as few $\mathtt{fixpoint}_i$ actions as possible with the $\mathtt{stratum}_i$ 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 $|\mathcal{C}|$.

5 Planning: With or Without Axioms?

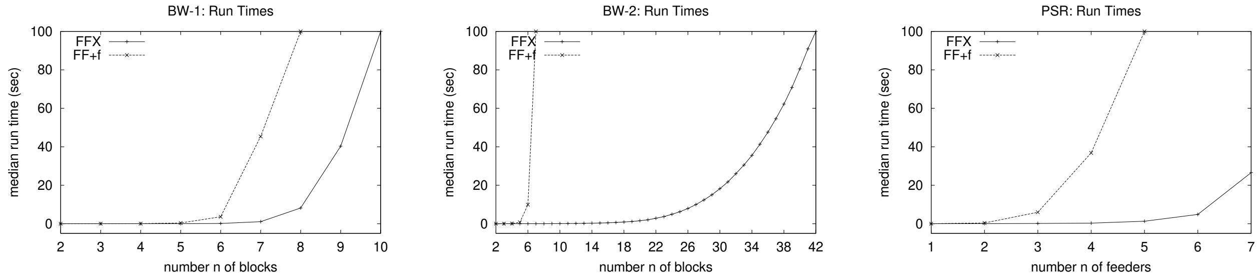
The absence of a polynomial time compilation scheme preserving plan size linearly not only indicates that axioms bring (much needed) expressive power, but it also suggests that extending a planner to explicitly deal with axioms may lead to much better performance than using a compilation scheme with the original version of the planner. To confirm this hypothesis, we extended the FF planner [11] 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 $(:\mathtt{derived}(d\ ?\vec{x})(f\ ?\vec{x}))$ into an operator with parameters $(?\vec{x})$, precondition $(f\ ?\vec{x})$ and effect $(d\ ?\vec{x})$, with a flag 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 fixed points associated with the stratification computed by Algorithm 1.

One domain we chose for our experiments is the usual Blocks World (BW) with 4 operators. In contrast to most other common benchmarks, in BW there is a natural distinction between basic and derived predicates; in particular BW is the only common benchmark domain we are aware of where the stratification of the axioms requires more than one stratum. 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. above is only used in goal descriptions. For the experiment labelled BW-1 in the figures below, we generated 30 random initial states for each size $n = 2 \ldots 10$ and took the goal that any block initially on the table had to be above all those that were initially not. Note that expressing the resulting goal using on and ontable would require exponential space, highlighting once more the utility of derived predicates. As shown in the figure, the median run-time of $\text{FF}_{\mathcal{X}}$ shows a significant improvement over that of FF+**f**. The plans found by FF+**f** in this experiment were an order of magnitude longer than those found by $\text{FF}_{\mathcal{X}}$. The experiment labelled BW-2 shows, for $n = 2 \ldots 42$, the special case of those instances for which the initial state has only one tower. Here the improvement in run time is dramatic, as $\text{FF}_{\mathcal{X}}$ finds the optimal plans whose length is only linear in n.

Another domain we ran experiments on is the challenging Power Supply Restoration (PSR) benchmark [4], 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 flow, [4]. We considered a version of the benchmark without any uncertainty for which 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 3 switches per feeder and with 30% faulty lines. The third figure above compares the median run times of $\text{FF}_{\mathcal{X}}$ and FF+**f** as a function of n. Again the improvement in performance resulting from handling axioms explicitly is undeniable. In this experiment, the plan length does not vary much with n: with our parameters for the random instances generation, it is clustered around 5 actions for $\text{PDDL}_{\mathcal{X}}$ instances, and around 50 for the compiled instances. 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.



Other published compilation schemes [10; 9] are not applicable to the above domains whose descriptions involve negated derived predicates. The exponential space transformation by Davidson and Garagnani [7] is applicable to BW, but was unable to cope with problems larger than $n = 4$ because it substitutes for their definitions all occurrences of non-recursive derived predicates until none remains. This turns preconditions into ADL constructs that quickly become too complex for FF's pre-processing step to compile them away in reasonable time, in difference to the experiments described above where pre-processing time was negligible.

6 Conclusion

As reflected 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 efficiency perspective. In addition, we have provided a clear formal semantics for PDDL axioms, identified 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 field 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 ramification problems, see e.g. [16]. 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

[1] K. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann, 1987.

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

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

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

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

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

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

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

[9] M. Garagnani. A correct algorithm for efficient planning with preprocessed domain axioms. In *Research and Development in Intelligent Systems XVII*. Springer, 2000.

[10] C. Gazen and C. Knoblock. Combining the expressivity of UCPOP with the efficiency of Graphplan. In *Proc. ECP*, 1997.

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

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

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

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

[15] J. Lloyd. *Foundations of Logic Programming*. Springer, 1993.

[16] N. McCain and H. Turner. A causal theory of ramifications and qualifications. In *Proc. IJCAI*, 1995.

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

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

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

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