Plan Constraints and Preferences in PDDL3

The Language of the Deterministic Part of the Fifth International Planning Competition

Extended Abstract

Alfonso Gerevini[+] and Derek Long[*]

[+] Department of Electronics for Automation, University of Brescia (Italy), gerevini@ing.unibs.it
[*] Department of Computer and Information Sciences, University of Strathclyde (UK), derek.long@cis.strath.ac.uk

Abstract

We propose an extension to the PDDL language, called PDDL3.0, that aims at a better characterization of plan quality by allowing the user to express strong and soft constraints about the structure of the desired plans, as well as strong and soft problem goals. PDDL3.0 was the reference language of the 5th International Planning competition (IPC-5). This paper contains most of the document about PDDL3.0 that was discussed by the Consulting Committee of IPC-5, and then distributed to the IPC-5 competitors.

Introduction

The notion of plan quality in automated planning is a practically very important issue. In many real-world planning domains, we have to address problems with a large set of solutions, or with a set of goals that cannot all be achieved. In these problems, it is important to generate plans of *good or optimal quality* achieving all problem goals (if possible) or some subset of them.

In the previous International planning competitions, the plan generation CPU-time played a central role in the evaluation of the competing planners. In the fifth International planning competition (IPC-5), while considering the CPU-time, we would like to give greater emphasis to the importance of plan quality. The versions of PDDL used in the previous two competitions (PDDL2.1 and PDDL2.2) allow us to express some criteria for plan quality, such as the number of plan actions or parallel steps, and relatively complex plan metrics involving plan makespan and numerical quantities. These are powerful and expressive in domains that include metric fluents, but plan quality can still only be measured by plan size in the case of propositional planning. We believe that these criteria are insufficient, and we propose to extend PDDL with new constructs increasing its expressive power about the plan quality specification.

The proposed extended language allows us to express *strong and soft constraints on plan trajectories* (i.e. constraints over possible actions in the plan and intermediate states reached by the plan), as well as *strong and soft problem goals* (i.e. goals that must be achieved in any valid plan, and goals that we desire to achieve, but that do not have to be necessarily achieved). Strong constraints and goals must be satisfied by any valid plan, while soft constraints and goals express desired constraints and goals, some of which may be more preferred than others. Informally, in planning with soft constraints and goals, the best quality plan should satisfy "as much as possible" the soft constraints and goals according to the specified preference relation distinguishing alternative feasible plans (satisfying all strong constraints and goals). While soft constraints have been extensively studied in the CSP literature, only very recently has the planning community started to investigate them (Brafman & Chernyavsky 2005; Briel *et al.* 2004; Delgrande, Schaub, & Tompits 2005; Miguel, Jarvis, & Shen 2001; Smith 2004; Son & Pontelli 2004), and we believe that they deserve more research efforts.

The following are some informal examples of plan trajectory constraints and soft goals. Additional formal examples will be given in the next section.

Examples in a blocksworld domain: *a fragile block can never have something above it, or it can have at most one block on it*; *we would like that the blocks forming the same tower always have the same colour*; *in some state of the plan, all blocks should be on the table*.

Examples in a transportation domain: *we would like that every airplane is used* (instead of using only a few airplanes, because it is better to distribute the workload among the available resources and limit heavy usage); *whenever a ship is ready at a port to load the containers it has to transport, all such containers should be ready at that port*; *we would like that at the end of the plan all trucks are clean and at their source location*; *we would like no truck to visit any destination more than once*.

When we have soft constraints and goals, it can be useful to give different priorities to them, and this should be taken into account in the plan quality evaluation. While there is more than one way to specify the importance of a soft constraint or goal, as a first attempt to tackle this issue, for IPC-5 we have chosen a simple quantitative approach: each soft constraint and goal is associated with a numerical weight representing the cost of its violation in a plan (and hence also its relative importance with respect the other specified soft constraints and goals). Weighted soft constraints and goals are part of the plan metric expression, and the best quality plans are those optimising such an expression (more details are given in the next sections).

Using this approach we can express that certain plans are more preferred than others. Some examples are (other formalised examples are given in the next sections):[1]

I prefer a plan where every airplane is used, rather than a plan using 100 units of fuel less, which could be expressed by weighting a failure to use all the planes by a number 100 times bigger than the weight associated with the fuel use in the plan metric; *I prefer a plan where each city is visited at most once, rather than a plan with a shorter makespan*, which could be expressed by using constraint violation costs penalising a failure to visit each city at most once very heavily; *I prefer a plan where at the end each truck is at its start location, rather than a plan where every city is visited by at most one truck*, which could be expressed by using goal costs penalising a goal failure of having every truck at its start location more heavily than a failure of having in the plan every city visited by at most one truck.

We also observe that the rich additional expressive power we propose to add for goal specifications allows the expression of constraints that are actually derivable necessary properties of optimal plans. By adding them as goal conditions, we have a way to express constraints that we know will lead to the planner finding optimal plans. Similarly, one can express constraints that prevent a planner from exploring parts of the plan space that are known to lead to inefficient performance.

In the next sections, we outline some extensions to PDDL2.2 that we propose for IPC-5. We call the extended language PDDL3.0. It should be noted that this is a preliminary version of the extended language, and that a more detailed description will be prepared in the future. Moreover, given that the proposed extensions are relatively new in the planning community, and that the teams participating in IPC-5 will have limited time to develop their systems, we impose some simplifying restrictions to make the language more accessible.

State Trajectory Constraints

Syntax and Intended Meaning

State trajectory constraints assert conditions that must be met by the entire sequence of states visited during the execution of a plan. They are expressed through temporal modal operators over first order formulae involving state predicates. We recognise that there would be value in also allowing propositions asserting the occurrence of action instances in a plan, rather than simply describing properties of the states visited during execution of the plan, but we choose to restrict ourselves to state predicates in this extension of the language. The use of the extensions described here imply a new requirements flag, `:constraints`.

The basic modal operators we propose to use in IPC-5 are: `always`, `sometime`, `at-most-once`, and `at end` (for goal state conditions). We use a special default assumption that unadorned conditions in the goal specification are automatically taken to be "at end" conditions. This assumption is made in order to preserve the standard meaning for existing goal specifications, despite the fact that in a standard semantics for an LTL formula an unadorned proposition would be interpreted according to the current state. We add `within` which can be used to express deadlines. In addition, rather than allowing arbitrary nesting of modal operators, we introduce some specific operators that offer some limited nesting. We have `sometime-before`, `sometime-after`, `always-within`. Other modalities could be added, but we believe that these are sufficiently powerful for an initial level of the sublanguage modelling constraints.

It should be noted that, by combining these modalities with timed initial literals (defined in PDDL2.2), we can express further goal constraints. In particular, one can specify the interval of time when a goal should hold, or the lower bound on the time when it should hold. Since these are interesting and useful constraints, we introduce two modal operators as "syntactic sugar" of the basic language: `hold-during` and `hold-after`.

Trajectory constraints are specified in the planning problem file in a new field, called `:constraints` that will usually appear after the goal. In addition, we allow constraints to be specified in the action domain file on the grounds that some constraints might be seen as safety conditions, or operating conditions, that are not physical limitations, but are nevertheless constraints that must always be respected in any valid plan for the domain (say legal constraints or operating procedures that must be respected). This also uses a section labelled `(:constraints ...)`. The interpretation of `(:constraints ...)` in the conjunction of a domain and a problem file is that it is equivalent to having all the constraints added to the goals. The use of trajectory constraints (in the domain file or in the goal specification) implies the need for the `:constraints` flag in the `:requirements` list.

Note that no temporal modal operator is allowed in preconditions of actions. That is, all action preconditions are with respect to a state (or time interval, in the case of `over all` action conditions).

The specific BNF grammar of PDDL3.0 is given in (Gerevini & Long 2005). The following is a fragment of the grammar concerning the new modalities of PDDL3.0 for expressing constraints (`con-GD`):

```
<con-GD> ::= (at end <GD>) | (always <GD>) |
             (sometime <GD>) | (within <num> <GD>) |
             (at-most-once <GD>) |
             (sometime-after <GD> <GD>) |
             (sometime-before <GD> <GD>) |
             (always-within <num> <GD> <GD>) |
             (hold-during <num> <num> <GD> |
             (hold-after <num> <GD> | ...
```

where <GD> is a goal description (a first order logic formula), <num> is any numeric literal (in STRIPS domains it will be restricted to integer values). There is a minor complication in the interpretation of the bound for `within` and `always-within` when considering STRIPS plans (and similarly for `hold-during` and `hold-after`): the question is whether the bound refers to sequential steps (in other words, actions) or to parallel steps. For STRIPS plans, the numeric bounds will be counted in terms of plan *happenings*. For

[1]The benchmark domains and problems of IPC-5 contain many additional examples; some samples of them are described in (Gerevini & Long 2006).

instance, `(within 10` ϕ`)` would mean that ϕ must hold within 10 happenings. These would be happenings of one action or of multiple actions, depending on whether the plan is sequential or parallel.

Notes on Semantics

The semantics of goal descriptors in PDDL2.2 evaluates them only in the context of a single state (the state of application for action preconditions or conditional effects and the final state for top level goals). In order to give meaning to temporal modalities, which assert properties of trajectories rather than individual states, it is necessary to extend the semantics to support interpretation with respect to a finite trajectory (as it is generated by a plan). We propose a semantics for the modal operators that is the same basic interpretation as is used in TLPlan (Bacchus & Kabanza 2000) for $\mathcal{LT}$ and other standard LTL treatments. Recall that a *happening* in a plan for a PDDL domain is the collection of all effects associated with the (start or end points of) actions that occur at the same time. This time is then the time of the happening and a happening can be "applied" to a state by simultaneously applying all effects in the happening (which is well defined because no pair of such effects may be mutex).

Definition 1 *Given a domain D, a plan* π *and an initial state I,* π *generates the trajectory*

$$\langle (S_0, 0), (S_1, t_1), ..., (S_n, t_n) \rangle$$

iff $S_0 = I$ *and for each happening* h *generated by* π*, with* h *at time t, there is some* i *such that* $t_i = t$ *and* S_i *is the result of applying the happening* h *to* S_{i-1}*, and for every* $j \in \{1 \dots n\}$ *there is a happening in* π *at* t_j*.*

Definition 2 *Given a domain* D*, a plan* π*, an initial state* I*, and a goal* G*,* π *is valid if the trajectory it generates,* $\langle (S_0, 0), (S_1, t_1), ..., (S_n, t_n) \rangle$*, satisfies the goal:* $\langle (S_0, 0), (S_1, t_1), ..., (S_n, t_n) \rangle \models G$*.*

This definition contrasts with the original semantics of goal satisfaction, where the requirement was that $S_n \models G$. The contrast reflects precisely this requirement that goals should now be interpreted with respect to an entire trajectory. We do not allow action preconditions to use modal operators and therefore their interpretation continues to be relative to the single state in which the action is applied. The interpretation of simple formulae, ϕ (containing no modalities), in a single state S continues to be as before and continues to be denoted $S \models \phi$. In the following definition we rely on context to make clear where we are using the interpretation of non-modal formulae in single states, and where we are interpreting modal formulae in trajectories.

Definition 3 *Let* ϕ *and* ψ *be atomic formulae over the predicates of the planning problem plus equality (between objects or numeric terms) and inequalities between numeric terms, and let* t *be any real constant value. The interpretation of the modal operators is as specified in Figure 1.*

Note that this interpretation exploits the fact that modal operators are not nested. A more general semantics for nested modalities is a straight-forward extension of this one. Note also that the last four expressions in Figure 1 are expressible in different ways if one allows nesting of modalities and use of the standard LTL modality until (more details on this in (Gerevini & Long 2005)).

The constraint `at-most-once` is satisfied if its argument becomes true and then stays true across multiple states and then (possibly) becomes false and stays false. Thus, there is only at most one *interval* in the plan over which the argument proposition is true.

For general formulae (which may or may not contain modalities):

$\langle (S_0, 0), (S_1, t_1), ..., (S_n, t_n) \rangle \models$ `(and` $\phi_1 ... \phi_n$`)` iff, for every i, $\langle (S_0, 0), (S_1, t_1), ..., (S_n, t_n) \rangle \models \phi_i$

and similarly for other connectives.

Of the constraints `hold-during` and `hold-after`, `(hold-during` t_1 t_2 ϕ`)` states that ϕ must be true during the interval $[t_1, t_2)$, while `(hold-after` t ϕ`)` states that ϕ must be true after time t. The first can be expressed by using timed initial literals to specify that a dummy timed literal `d` is true during the time window $[t_1, t_2)$ together with the goal `(always (implies d` ϕ`))`.
A variant of `hold-during` where ϕ must hold *exactly* during the specified interval could be easily obtained in a similar way. The second can be expressed by using timed initial literals to specify that `d` is true only from time t, together with the goal `(sometime-after d` ϕ`)`.

Soft Constraints and Preferences

A soft constraint is a condition on the trajectory generated by a plan that the user would prefer to see satisfied rather than not satisfied, but is prepared to accept might not be satisfied because of the cost of satisfying it, or because of conflicts with other constraints or goals. In case a user has multiple soft constraints, there is a need to determine which of the various constraints should take priority if there is a conflict between them or if it should prove costly to satisfy them. This could be expressed using a qualitative approach but, following careful deliberations, we have chosen to adopt a simple quantitative approach for this version of PDDL.

Syntax and Intended Meaning

The syntax for soft constraints falls into two parts. Firstly, there is the identification of the soft constraints, and secondly there is the description of how the satisfaction, or lack of it, of these constraints affects the quality of a plan.

Goal conditions, including action preconditions, can be labelled as preferences, meaning that they do not have to be true in order to achieve the corresponding goal or precondition. Thus, the semantics of these conditions is simple, as far as the correctness of plans is concerned: they are all trivially satisfied in any state. The role of these preferences is apparent when we consider the relative quality of different plans. In general, we consider plans better when they satisfy soft constraints and worse when they do not. A complication arises, however, when comparing two plans that satisfy different subsets of constraints (where neither set strictly contains the other). In this case, we rely on a specification of the violation costs associated with the preferences.

$\langle(S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{at end}\ \phi)$	iff	$S_n \models \phi$
$\langle(S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models \phi$	iff	$S_n \models \phi$
$\langle(S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{always}\ \phi)$	iff	$\forall i : 0 \leq i \leq n \cdot S_i \models \phi$
$\langle(S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{sometime}\ \phi)$	iff	$\exists i : 0 \leq i \leq n \cdot S_j \models \phi$
$\langle(S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{within}\ t\ \phi)$	iff	$\exists i : 0 \leq i \leq n \cdot S_i \models \phi$and $t_i \leq t$
$\langle(S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{at-most-once}\ \phi)$	iff	$\forall i : 0 \leq i \leq n \cdot$ if $S_i \models \phi$ then $\exists j : j \geq i \cdot \forall k : i \leq k \leq j \cdot S_k \models \phi$ and $\forall k : k > j \cdot S_k \models \neg\phi$
$\langle(S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{sometime-after}\ \phi\ \psi)$	iff	$\forall i \cdot$ if $S_i \models \phi$ then $\exists j : i \leq j \leq n \cdot S_j \models \psi$
$\langle(S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{sometime-before}\ \phi\ \psi)$	iff	$\forall i \cdot$ if $S_i \models \phi$ then $\exists j : 0 \leq j < i \cdot S_j \models \psi$
$\langle(S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{always-within}\ t\ \phi\ \psi)$	iff	$\forall i \cdot$ if $S_i \models \phi$ then $\exists j : i \leq j \leq n \cdot S_j \models \psi$ and $t_j - t_i \leq t$

Figure 1: Semantics of the basic modal operators in PDDL3.

The syntax for labelling preferences is simple:

```
(preference [name] <GD>).
```

The definition of a goal description can be extended to include preference expressions. However, in PDDL3.0, we reject as syntactically invalid any expression in which preferences appear nested inside any connectives, or modalities, other than conjunction and universal quantifiers. We also consider it a syntax violation if a preference appears in the condition of a conditional effect. *Note that where a named preference appears inside a universal quantifier, it is considered to be equivalent to a conjunction (over all legal instantiations of the quantified variable) of preferences all with the same name.*

Where a name is selected for a preference it can be used to refer to the preference in the construction of penalties for the violated constraint. The same name can be shared between preferences, in which case they share the same penalty.

Penalties for violation of preferences are calculated using the expression

```
(is-violated <name>)
```

where <name> is a name associated with one or more preferences. This expression takes on a value equal to the number of distinct preferences with the given name that are not satisfied in the plan. Note that in PDDL3.0 we do not attempt to distinguish degrees of satisfaction of a soft constraint — we are only concerned with whether or not the constraint is satisfied. Note, too, that the count includes each separate constraint with the same name. This means that:

```
(preference VisitParis
    (forall (?x - tourist)
            (sometime (at ?x Paris))))
```

yields a violation count of 1 for (is-violated VisitParis), if at least one tourist fails to visit Paris during a plan, while

```
(forall (?x - tourist)
    (preference VisitParis
        (sometime (at ?x Paris))))
```

yields a violation count equal to the number of people who failed to visit Paris during the plan. The intention behind this is that each preference is considered to be a distinct preference, satisfied or not independently of other preferences. The naming of preferences is a convenience to allow different penalties to be associated with violation of different constraints.

Plans are awarded a value through the plan metric, introduced in PDDL2.1 (Fox & Long 2003). The constraints can be used in weighted expressions in a metric. For example,

```
(:metric minimize
     (+ (* 10 (fuel-used))
        (is-violated VisitParis)))
```

would weight fuel use as ten times more significant than violations of the VisitParis constraint. Note that the violation of a preference in the preconditions of an action is counted multiple times, depending on the number of the action occurrences in the plan. For instance, suppose that p is a preference in the precondition of an action a, which occurs three times in plan π. If the plan metric evaluating π contains the term (* k (is-violated p)), then this is interpreted as if it were (* v (* k (is-violated p))), where v is the number of separate occurrences of a in π for which the preference is not satisfied.

Semantics

We say that

$$\langle(S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{preference}\ \Phi)$$

is always true, so this allows preference statements to be combined in formulae expressing goals. The point in making the formula always true is that the preference is a soft constraint, so failure to satisfy it is not considered to falsify the goal formula. In the context of action preconditions, we say $S_i \models (\texttt{preference}\ \Phi)$ is always true, too, for the same reasons.

We also say that a preference (preference Φ) is *satisfied* iff $\langle(S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models \Phi$ and *violated* otherwise. This means that (or Φ (preference Ψ)) is the same as (preference (or $\Phi\,\Psi$)), both in terms of the satisfaction of the formulae and also in terms of whether the preference is satisfied. The same idea is applied to action precondition preferences. Hence, a goal such as:

```
(and (at package1 london)
```

```
    (preference (clean truck1)))
```

would lead to the following interpretation:

$\langle (S_0, 0), (S_1, t_1), ..., (S_n, t_n) \rangle \models$

```
    (and (at package1 london)
         (preference (clean truck1)
```

iff

$\langle (S_0, 0), (S_1, t_1), ..., (S_n, t_n) \rangle \models$

```
    (at package1 london)
```

and

$\langle (S_0, 0), (S_1, t_1), ..., (S_n, t_n) \rangle \models$

```
    (preference (clean truck1))
```

iff $S_n \models$ `(at package1 london)`

iff `(at package1 london)` $\in S_n$, since the preference is always interpreted as true. In addition, the preference would be *satisfied* iff:

$\langle (S_0, 0), (S_1, t_1), ..., (S_n, t_n) \rangle \models$

```
    (at end (clean truck1))
```

iff `(clean truck1)` $\in S_n$.

If the preference is not satisfied, it is violated.

Now suppose that we have the following preferences and plan metric:

```
(preference p1 (always (clean truck1)))
(preference p2 (and (at end (at package2 paris))
                    (sometime (clean track1))))
(preference p3 (...))

(:metric (+ (* 10 (is-violated p1)) (* 5 (is-violated p2))
            (is-violated p3))).
```

Suppose we have two plans, π_1, π_2, and π_1 does not satisfy preferences p1 and p3 (but it satisfies preference p2) and π_2 does not satisfy preferences p2 and p3 (but it satisfies preference p1), then the metric for π_1 would yield a value (11) that is higher than that for π_2 (6) and we would say that π_2 is better than π_1.

Formally, *a preference precondition is satisfied if the state in which the corresponding action is applied satisfies the preference*. Note that the restriction on where preferences may appear in precondition formulae and goals, together with the fact that they are banned from conditional effects, means that this definition is sufficient: the context of their appearance will never make it ambiguous whether it is necessary to determine the status of a preference. Similarly, *a goal preference is satisfied if the proposition it contains is satisfied in the final state*. Finally, *an invariant (*`over all`*) condition of a durative action is satisfied if the corresponding proposition is true throughout the duration of the action*.

In some case, it can be hard to combine preferences with an appropriate weighting to achieve the intended balance between soft constraints and other factors that contribute to the value of a plan (such as plan make span, resource consumption and so on). For example, to ensure that a constraint takes priority over a plan cost associated with resource consumption (such as make span or fuel consumption) is particularly tricky: a constraint must be weighted with a value that is higher than any possible consumption cost and this might not be possible to determine. With non-linear functions it is possible to achieve a bounded behaviour for costs associated with resources. For example, if a constraint, C, is to be considered always to have greater importance than the make span for the plan then a metric could be defined as follows:

```
(:metric minimize (+ (is-violated C)
                     (- 1 (/ 1 (total-time))))).
```

This metric will always prefer a plan that satisfies C, but will use make span to break ties.

Nevertheless, for the competition, where it is important to provide an unambiguous specification by which to rank plans, the use of plan metrics in this way is clearly very straightforward and convenient. We leave for later proposals the possibilities for extending the evaluation of plans in the face of soft constraints.

Some Examples

The following state trajectory constraints could be stated either as strong constraints or soft constraints.

"A fragile block can never have something above it":

```
(always (forall (?b - block)
          (implies (fragile ?b) (clear ?b))))
```

"A fragile block can have at most one block on it":

```
(always (forall (?b1 ?b2 - block)
          (implies (and (fragile ?b1) (on ?b2 ?b1))
                   (clear ?b2))))
```

"The blocks forming the same tower always have the same color":

```
(always (forall (?b1 ?b2 - block ?c1 ?c2 - color)
          (implies (and (on ?b1 ?b2) (color ?b1 ?c1)
                        (color ?b2 ?c2))
                   (= ?c1 ?c2))))
```

"Each block should be picked up *at least* once":

```
(forall (?b - block) (sometime (holding ?b)))
```

"Each block should be picked up *at most* once":

```
(forall (?b - block) (at-most-once (holding ?b)))
```

"In some state visited by the plan all blocks should be on the table":

```
(sometime (forall (?b - block) (on-table ?b)))
```

This constraint requires all the blocks to be on the table in the *same* state. In contrast, if we only require that every block should be on the table in *some* state we can write:

```
(forall (?b - block) (sometime (on-table ?b)))
```

"Whenever I am at a restaurant, I want to have a reservation":

```
(always (forall (?r - restaurant)
        (implies (at ?r) (have-reservation ?r))))
```

"Each truck should visit each city *at most* once":

```
(forall (?t - truck ?c - city) (at-most-once (at ?t ?c)))
```

"At some point in the plan all the trucks should be at city1":

```
(sometime (forall (?t - truck) (at ?t city1)))
```

"Each truck should visit each city *exactly once*":

```
(and (forall (?t - truck ?c - city)
             (at-most-once (at ?t ?c)))
     (forall (?t - truck ?c - city)
             (sometime (at ?t ?c))))
```

"Each city is visited by at most one truck at the same time":

```
(forall (?t1 ?t2 - truck ?c1 city)
   (always (implies (and (at ?t1 ?c1)
                    (at ?t2 ?c1)) (= ?t1 ?t2))))
```

The following two examples use the IPC-3 Rovers domain involving numerical fluents. "We would like that the energy of every rover should always be above the threshold of 5 units":

```
(always (forall (?r - rover) (> (energy ?r) 5))))
```

"Whenever the energy of a rover is below 5, it should be at the recharging location within 10 time units":

```
(forall (?r - rover)
   (always-within 10 (< (energy ?r) 5)
                     (at ?r recharging-point)))
```

The next two examples illustrate the usefulness of `sometime-before` and `sometime-after`. The first one states that "a truck can visit a certain city (where initially there is no truck) only after having visited another particular one"; the second one that "if a taxi has been used and it is at the depot, then it has to be cleaned" (if a taxi is used but it does not go back to the depots, then there is no need to clean it).

```
(forall (?t - truck)
    (sometime-before (at ?t city1) (at ?t city2)))

(forall (?t - taxi)
    (sometime-after (and (at ?t depot) (used ?t))
                    (clean ?t)))
```

"We want a plan moving package1 to London such that truck1 is always maintained clean, and at some point truck2 is at Paris. Moreover, we also prefer that truck3 is always clean and that at the end of the plan package2 is at London":

```
(:goal (and (at package1 london)
            (preference (at package2 london))))

(:constraints
   (and (always (clean truck1))
        (sometime (at truck2 paris))
        (preference (always (clean truck3)))
        (preference (at end (at package2 london)))))
```

"We prefer that every fragile package to be transported is insured".

```
(forall (?p - package)
    (preference P1
        (always (implies (fragile ?) (insured ?p)))))
```

We now consider an example with a plan metric.
"We want three jobs completed. We would prefer to take a coffee-break and that we take it when everyone else takes it (at coffee-time) rather than at any time. We would also like to finish reviewing a paper, but it is less important than taking a break. Finally, we would like to be finished so that we can get home at a reasonable time, and this matters more than finishing the review or having a sociable coffee break":

```
(:goal (and (finished job1)
            (finished job2)
            (finished job3)) )

(:constraints
  (and (preference break
         (sometime (at coffee-room)))
       (preference social
         (sometime (and (at coffee-room) (coffee-time))))
       (preference reviewing (reviewed paper1))))

(:plan-metric minimize
                (+  (* 5 (total-time))
                    (* 4 (is-violated social))
                    (* 2 (is-violated break))
                    (is-violated reviewing)))
```

Now consider three plans, π_1, π_2 and π_3, such that all three plans complete the three jobs. Suppose π_1 achieves this in 4 hours, but takes no break and does not include reviewing the paper. Suppose π_2 completes the jobs in 8 hours, but takes a coffee-break at coffee-time and reviews the paper. Finally, π_3 completes the jobs in 6 hours, including reviewing the paper, but only by taking a short break when the coffee room is empty. Then the values of the plans are:

Plan	Quality
π_1	5*4 + 4*1 + 2*1 + 1 = 27
π_2	5*8 + 4*0 + 2*0 + 0 = 40
π_3	5*6 + 4*1 + 2*0 + 0 = 34

This makes π_1 the best plan and π_2 the worst.

Plan Validation and Evaluation

A plan validator will be developed as an extension of the existing validator used in the previous competitions. The two key aspects of this extension are checking state trajectory constraints in the goal, which does not complicate the execution simulation for a plan, and the checking of preferences in order to compare plans. This latter extension will involve identifying the constraint violations associated with each plan and their violation times, in order to evaluate the plan quality according to the specified metric (which may include terms for the preference violations). The organizers of IPC-5 are considering the possibility of using different variants of the test problems involving only strong constraints or soft constraints, with a possible additional distinction between simple preferences, involving only goals or action preconditions, and more complex preferences involving general soft constraints. More details about this organization of the benchmarks will be announced in the the web page of the deterministic track of IPC-5: `http://ipc5.ing.unibs.it`.

Extensions and Generalization

There is considerable scope for developing the proposed extension. First, and most obviously, modal operators could be allowed to nest. This would allow a rich expressive power in the specification of modal temporal goals. Nesting would allow constraints to be applied to parts of trajectories, as is usual in modal temporal logics. In addition, we could introduce propositions representing that an action appears in a plan.

Other modal operators could be added. We have excluded them PDDL3.0 because we have found that many interesting and challenging goals can be captured without them,

$$\langle (S_0, 0), (S_1, t_1), ..., (S_n, t_n) \rangle \models (\texttt{always-persist}\ t\ \phi) \quad \text{iff} \quad \begin{array}{l} \forall i : 0 < i \leq n \cdot \text{if } S_i \models \phi \text{ and } S_{i-1} \models \neg\phi \text{ then} \\ \exists j : j - i \geq t \cdot \forall z : i \leq z \leq j \cdot S_z \models \phi \text{ and} \\ \text{if } S_0 \models \phi \text{ then } \forall z : z \leq t \cdot S_z \models \phi \end{array}$$

$$\langle (S_0, 0), (S_1, t_1), ..., (S_n, t_n) \rangle \models (\texttt{always-persist}\ t\ \phi) \quad \text{iff} \quad \begin{array}{l} \exists i : 0 < i \leq n \cdot \text{if } S_i \models \phi \text{ and } S_{i-1} \models \neg\phi \text{ then} \\ \exists j : j - i \geq t \cdot \forall z : i \leq z \leq j \cdot S_z \models \phi\text{, or} \\ \text{if } S_0 \models \phi \text{ then } \forall z : z \leq t \cdot S_z \models \phi \end{array}$$

Figure 2: Semantics of always-persist and sometime-persist.

and we do not wish to add unnecessarily to the load on potential competitors. The modal operator until would be an obvious one to add. Without nesting, a related always-until and sometime-until would allow expression of goals such as "every time a truck arrives at the depot, it must stay there until loaded" or "when the truck arrives at the depot, it must stay there until cleaned and fully refuelled at least once in the plan". The formal semantics of always-until and sometime-until can be easily derived from the one of until in LTL. By combining always-until and other modalities we can express complex constraints such as that "whenever the energy of a rover is below 5, it should be at the recharging location within 10 time units and remain there until recharged":

```
(and (always-until (charged ?r) (at ?r rechargepoint))
     (always-within 10 (< (charge ?r) 5)
                       (at ?r rechargingpoint)))
```

Another modality that would be an useful extension of the expressive power is a complement for `within`, such as persist, with the semantics that a proposition once made true must persist for at least some minimal period of time. Without nesting, a related always-persist and sometime-persist would allow expression of goals such as "I want to spend at least 2 days in each of the cities on my tour", or "every time the taxi goes to the station it must wait for at least 10 without a passenger".

The formal semantics of always-persist and sometime-persist is given in Figure 2. A generalisation that would allow `within` and persist to be combined would be to allow the time specification to be associated with a comparison operator to indicate whether the bound is an upper or lower bound.

We have deliberately not introduced the operator next, which is common in modal temporal logics. This is because concurrent fragments of a plan might cause a state change that is not relevant to the part of the state in which the next condition is intended to apply. Furthermore, the fact that PDDL plans are embedded on a real time line means that the intention behind next is less obviously relevant. We realise that next has been particularly useful in expressing control rules for planners like TALPlanner (Kvarnström & Magnusson 2003) and TLPlan (Bacchus & Kabanza 2000), but our intention in developing this extension is to focus on providing a language that is useful for expressing constraints that govern plan quality, rather than for control knowledge. We believe that the use of `always-within` captures a much more useful concept for plan quality that is actually a far more realistic constraint in modelling planning problems.

Extensions to the use of soft constraints include the definition of more complex preferences, such as conditional preferences, and a possible qualitative method for expressing priorities over preferences. Moreover, the evaluation of the soft constraints could be extended by considering a degree of constraint violation, such as the amount of time when an `always` constraint is violated, the delay that falsifies a `within` constraint, or the number of times an `always-after` constraint is violated.

Acknowledgments

We would like to thank Y. Dimopoulos, C. Domshlak, S. Edelkamp, M. Fox, P. Haslum, J. Hoffmann, A. Jonsson, D. McDermott, A. Saetti, L. Schubert, I.Serina, D. Smith and D. Weld for some very useful discussions about PDDL3.

References

Bacchus, F., and Kabanza, F. 2000. Using temporal logic to express search control knowledge for planning. *Artificial Intelligence* 116(1-2):123–191.

Brafman, R., and Chernyavsky, Y. 2005. Planning with goal preferences and constraints. In *Proc. of ICAPS-05*.

Briel, M.; Sanchez, R.; Do, M.; and Kambhampati, S. 2004. Effective approaches for partial satisfaction (over-subscription) planning. In *Proc. of the AAAI-04*.

Delgrande, P. J.; Schaub, T.; and Tompits, H. 2005. A general framework for expressing preferences in causal reasoning and planning. In *Proc. of the* 7^{th} *Int. Symposium on Logical Formalizations of Commonsense Reasoning*.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of AI Research* 20:pp. 61–124.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences in PDDL3. Technical Report RT-2005-08-47, Dep. di Elettronica per l'Automazione, Universitá di Brescia, Italy. An extension with the BNF grammar of PDDL3.0 is available from `http://ipc5.ing.unibs.it`.

Gerevini, A., and Long, D. 2006. Preferences and soft constraints in PDDL3. In *Proc. of ICAPS Workshop on Preferences and Soft constraints in Planning*.

Kvarnström, J., and Magnusson, M. 2003. Talplanner in the 3rd international planning competition: Extensions and control rules. *Journal of AI Research* 20.

Miguel, I.; Jarvis, P.; and Shen, Q. 2001. Efficient flexible planning via dynamic flexible constraint satisfaction. *Engineering Applications of Artificial Intelligence* 14(3):301–327.

Smith, D. 2004. Choosing objectives in over-subscription planning. In *Proc. of ICAPS-04*.

Son, T., C., and Pontelli, E. 2004. Planning with preferences using logic programming. In *Proc. of LPNMR-04*. Springer-Verlag. LNAI 2923.