1 BNF definition of PDDL 3.1

Hereby a complete BNF syntax definition of the PDDL 3.1 language is presented (completely corrected) based on the originally published articles and information about PDDL 1.2, 2.1, 2.2, 3.0 and 3.1 [1-5].

1.1 Domain description
Complete BNF description of PDDL 3.1 (completely corrected)  
Daniel L. Kovacs (dkovacs@mit.edu)

```plaintext
<term> ::= "object−class" <function−term> ...
<function−term> ::= "object−class" <function−symbol> <term>*
<exp> ::= "name" <exp> ...
<binary-op> ::= (binary-op) <exp> <exp>*
<exp> ::= "name" <multi-op> <exp> <exp>*
<exp> ::= "name" <function−symbol> <term>*
<binary-op> ::= "name" <multi-op> <binary−comp> ...
<binary-comp> ::= "name" ...
<binary-comp> ::= <binary−comp> <binary−comp>*
<binary-op> ::= <binary−comp> <binary−comp>*
<name> ::= <letter> <any char>*
<letter> ::= a..z | A..Z
<number> ::= <digit>'<decimal>' ...
<decimal> ::= 0..9 ...
<effect> ::= (and <effect>)* ...
<conditional-effects> ::= (forall (typed list (variable))<effect>) ...
<effect> ::= (when <conditional−effects> <effect>) ...
<effect> ::= (when <conditional−effects> ..)
<effect> ::= (not (atomic formula(term)) ...
<effect> ::= (assign <function−term><term>) ...
<effect> ::= (assign <function−term><term> undefined)
<conditional-effect> ::= (and <effect>)* ...
<assign> ::= assign ...
<predict> ::= predict ...
<assign> ::= scale−up ...
<assign> ::= scale−down ...
<assign> ::= increase ...
<assign> ::= decrease ...
<duration−action−def> ::= :<duration−action <da−symbol> 
parameters (typed list (variable))<da−def body>
<da−symbol> ::= <name>
<da−def body> ::= !duration <duration−constraint> ...
<duration−constraint> ::= <op> <value> ...
<op> ::= − ...
<value> ::= <digit> ...
<effect> ::= (and <effect>)* ...
<assign> ::= assign ...
<assign> ::= duration−inequalities ...
<assign> ::= duration−inequalities ...
<assign> ::= duration−inequalities ...
<assign> ::= duration−inequalities ...
<assign> ::= duration−inequalities ...
<assign> ::= duration−inequalities ...
<assign> ::= duration−inequalities ...
<assign> ::= duration−inequalities ...
<derive-def> ::= :<derived <atomic formula skeleton> <GD> ...
```
1.2 Problem description

<problem> ::= (define (problem <name>)
   [domain <name>]
   [require-def]
   [object declaration]
   [init]
   [goal]
   [constraints] [metric-spec]
   [length-spec])

<object declaration> ::= (:objects <typed list (name)>)
<init> ::= (init <init-e1>*)
<init-e1> ::= (:literal (name))
<goal> ::= (:goal <pre-GD>)
<constraints> ::= (preference [pref-name] <con-GD>)
<metric-spec> ::= (:metric <optimization> metric-f-spec)
<optimization> ::= (minimize)
<metric-f-spec> ::= (function-symbol) (<name>*
<metric-f-exp> ::= (number)
<length-spec> ::= (length [] (universal integer) [parallel integer])

1.2.1 Lifting restrictions (from constraint declaration)

If we wish to embed modal operators into each other, then we should use these rules instead of those in section 1.2 respectively.

<con-GD> ::= (always <con2-GD>)
<con2-GD> ::= (after <con2-GD> before <con2-GD>)
<con3-GD> ::= (within <number> <con2-GD>)
<con4-GD> ::= (at-most-once <con2-GD>)
<con5-GD> ::= (most <con2-GD> <con2-GD>)
<con6-GD> ::= (most <con2-GD> <con2-GD>)
<con7-GD> ::= (always <con2-GD> <con2-GD>)
<con8-GD> ::= (hold-during <number> <con2-GD>)
<con9-GD> ::= (hold-after <number> <con2-GD>)
<con10-GD> ::= (GD)
1.3 Requirements

Here is a table of all requirements in PDDL3.1. Some requirements imply others; some are abbreviations for common sets of requirements. If a domain stipulates no requirements, it is assumed to declare a requirement for :strips.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description/Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>:strips</td>
<td>Basic STRIPS-style adds and deletes</td>
</tr>
<tr>
<td>:typing</td>
<td>Allow type names in declarations of variables</td>
</tr>
<tr>
<td>:negative-preconditions</td>
<td>Allow not in goal descriptions</td>
</tr>
<tr>
<td>:equality</td>
<td>Support = as built-in predicate</td>
</tr>
<tr>
<td>:existential-preconditions</td>
<td>Allow exists in goal descriptions</td>
</tr>
<tr>
<td>:universal-preconditions</td>
<td>Allow forall in goal descriptions</td>
</tr>
<tr>
<td>:conditional-effects</td>
<td>Allow when in action effects</td>
</tr>
<tr>
<td>:fluents</td>
<td>= :numeric-fluents</td>
</tr>
<tr>
<td>:object-fluents</td>
<td>+ :numeric-fluents</td>
</tr>
<tr>
<td>:quantified-preconditions</td>
<td>+ :quantified-preconditions</td>
</tr>
<tr>
<td>:conditional-effects</td>
<td>+ :conditional-effects</td>
</tr>
<tr>
<td>:numeric-fluents</td>
<td>Allow numeric function definitions and use of effects using assignment operators and arithmetic preconditions.</td>
</tr>
<tr>
<td>:adl</td>
<td>= :strips + :typing</td>
</tr>
<tr>
<td>:negative-preconditions</td>
<td>+ :disjunctive-preconditions</td>
</tr>
<tr>
<td>:equality</td>
<td>+ :equality</td>
</tr>
<tr>
<td>:quantified-preconditions</td>
<td>+ :quantified-preconditions</td>
</tr>
<tr>
<td>:conditional-effects</td>
<td>+ :conditional-effects</td>
</tr>
<tr>
<td>:duration-inequalities</td>
<td>Allows duration constraints in durative actions using inequalities.</td>
</tr>
<tr>
<td>:continuous-effects</td>
<td>Allows durative actions to affect fluents continuously over the duration of the actions.</td>
</tr>
<tr>
<td>:derived-predicates</td>
<td>Allows predicates whose truth value is defined by a formula</td>
</tr>
<tr>
<td>:timed-initial-literals</td>
<td>Allows the initial state to specify literals that will become true at a specified time point. Implies :duration-inequalities.</td>
</tr>
<tr>
<td>:preferences</td>
<td>Allows use of preferences in action preconditions and goals.</td>
</tr>
<tr>
<td>:constraints</td>
<td>Allows use of constraints fields in domain and problem files. These may contain modal operators supporting trajectory constraints.</td>
</tr>
<tr>
<td>:action-costs</td>
<td>If this requirement is included in a PDDL specification, the use of numeric fluents is enabled (similar to the :numeric-fluents requirement). However, numeric fluents may only be used in certain very limited ways:</td>
</tr>
</tbody>
</table>

1. Numeric fluents may not be used in any conditions (preconditions, goal conditions, conditions of conditional effects, etc.).
2. A numeric fluent may only be used as the target of an effect if it is 0-ary and called total-cost. If such an effect is used, then the total-cost fluent must be explicitly initialized to 0 in the initial state.
3. The only allowable use of numeric fluents in effects is in effects of the form (increase (total-cost) <numeric-term>), where the <numeric-term> is either a non-negative numeric constant or of the form <function-symbol> <term>*. (The <term> here is interpreted as shown in the PDDL grammar, i.e. it is a variable symbol or an object constant. Note that this <term> cannot be a <function-term>, even if the object fluents requirement is used.)
4. No numeric fluent may be initialized to a negative value.
5. If the problem contains a :metric specification, the objective must be (minimize (total-cost)), or - only if the :durative-actions requirement is also set - to minimize a linear combination of total-cost and total-time, with non-negative coefficients.

Note that an action can have multiple effects that increase (total-cost), which is particularly useful in the context of conditional effects. Also note that these restrictions imply that (total-cost) never decreases throughout plan execution, i.e., action costs are never negative.
References


