Complete BNF definition of MA-PDDL with privacy

Hereby a complete BNF syntax definition of MA-PDDL (Multi-Agent PDDL) extended with privacy is presented based on the complete and corrected BNF syntax definition of PDDL3.1 [1], the original definition of MA-PDDL [2] and privacy as defined in MA-STRIPS [3]. The minimalistic additions to PDDL3.1 syntax are highlighted separately each time. The additions to PDDL3.1 enabling description of multiple agents are highlighted with yellow, while the additions enabling the description of privacy are highlighted with light blue.

1. Domain description

```plaintext
<domain> ::= (define (domain <name>)
  [<require-def>]
  [<types-def>]
  [<constants-def>]
  [<predicates-def>]
  [<functions-def>]
  [<constraints>]
  [<structure-def>*])

<require-def> ::= (:requirements <require-key>*)
<require-key> ::= See Section 7.2.3

<types-def> ::= (:types <typed list (name)>)
<constants-def> ::= (:constants <typed list (name)>)
<constants-def> ::= (:private-privacy (:constants <typed list (name)>)
<constants-def> ::= (:multi-agent (:constants <typed list (name)>)
<private-objects> ::= (:private-privacy (:private <typed list (name)>)
<private-objects> ::= (:multi-agent (:private <agent-def> <typed list (name)>)
<predicates-def> ::= (:predicates <atomic formula skeleton>*)
<predicates-def> ::= (:private-privacy (:predicates <atomic formula skeleton>*)
<predicates-def> ::= (:multi-agent (:predicates <atomic formula skeleton>*))
/private-predicates> ::= (:private-privacy (:private-predicates <atomic formula skeleton>*)
/private-predicates> ::= (:multi-agent (:private-predicates <agent-def> <atomic formula skeleton>*)
<atomic formula skeleton> ::= (<predicate> <typed list (variable)>)
<atomic formula skeleton> ::= (:private <predicate> <typed list (variable)>)
<atomic function skeleton> ::= (<function> <typed list (variable)>)
<atomic function skeleton> ::= (:private <function> <typed list (variable)>)
(functions-def) ::= (:functions <function typed list (atomic function skeleton)>
(functions-def) ::= (:private-privacy (:functions <function typed list (atomic function skeleton)>)
(functions-def) ::= (:multi-agent (:functions <function typed list (atomic function skeleton)>)
<function typed list (x)> ::= x*
<function typed list (x)> ::= numeric-fluents x

This is deprecated since PDDL3.1, where the default fluent type is number.
<function type> ::= numeric-fluents number
<function type> ::= typing + object-fluents <type>
<constraint> ::= (:constraints <constraints <con-GD>)
<structure-def> ::= (:structure-def <structure-def>)
<structure-def> ::= (:derived-predicate <derived-def>
<structure-def> ::= (:typed list (x))
<structure-def> ::= (:typing <typing >)
<primitive-type> ::= :either <primitive-type>
<type> ::= :primitive-type
<emptyOr (x)> ::= ()
<emptyOr (x)> ::= x
<action-def> ::= (:action <action-symbol>
<action-def> ::= :primitive-type (<action <action-symbol>
<agent-def> ::= (:agent <name>)
<agent-def> ::= (:variable)
<agent-def> ::= (:type)
<agent-def> ::= (:variable - <type>)
<action-def-body> ::= (:precondition <emptyOr (pre-GD)>)
<action-def-body> ::= ([precondition <emptyOr (pre-GD)>]
<pre-GD> ::= (:pref-GD)
<pre-GD> ::= (:pref-GD)
<pre-GD> ::= (:pref-GD)
<pref-name> ::= (:name)
<GD> ::= (:GD)
<GD> ::= (:GD)
<GD> ::= (:GD)
```
Complete BNF definition of MA-PDDL with privacy

D. L. Kovacs

<GD> ::= (and <GD>*
<GD> ::= (not <GD>)
<GD> ::= (exists (existential-preconditions) <GD>)
<GD> ::= (forall (universal-preconditions) <GD>)
<GD> ::= (multi-agent <comp>)
<GD> ::= (binary-comp <f-exp> <f-exp>)
<GD> ::= (atomic formula(term))
<GD> ::= (equality (= t t))
<GD> ::= (num (<typed list (variable)>) <emptyOr (GD) > <action formula(term)>)
<GD> ::= (- <f-exp>)
<GD> ::= =<f-head>
<GD> ::= (assign-op <function-term> <term>)
<GD> ::= (assign-op undefined)
<GD> ::= (at <time-specifier> <GD>)
<GD> ::= (over <interval> <GD>)
<GD> ::= (time-specifier) (= start)
<GD> ::= (time-specifier) (= end)
<GD> ::= (all) (duration-inequalities (and <simple-duration-inequality>))
<GD> ::= (simple-duration-inequalities (at <time-specifier> <simple-duration-inequality>))
<GD> ::= (d-op) (duration-inequalities <=)
<GD> ::= (d-op) (duration-inequalities >=)
<GD> ::= (d-op) (d-value)
<GD> ::= (d-value) (number)
<GD> ::= (numeric-fluents <f-comp>)
<GD> ::= (numeric-fluents <number>)
<GD> ::= (numeric-fluents (<binary-op> <f-exp> <f-exp>))
<GD> ::= (numeric-fluents (<multi-op> <f-exp> <f-exp>))
<GD> ::= (numeric-fluents assign-op <function-term> <term>)
<GD> ::= (numeric-fluents assign-op undefined)
2. Problem description

<problem> ::= (define (problem <name>)
   [:domain <name>]
<object declaration> ::= (:objects <typed list (name)>)
<object declaration> ::= (:objects <typed list (name)> [:private-objects])
<object declaration> ::= (:objects <typed list (name)> <private-objects>*)
<init-el> ::= (:init (:init-el)*)
<init-el> ::= (:literal (name))
<init-el> ::= (:timed-initial-literals (at <number> <literal(name)>))
<init-el> ::= (:object-fluents (= <basic-function-term> <name>))
<init-el> ::= (:duration-inequalities ?duration)
<goal> ::= (:goal (:goal <pre-GD>))
<goal> ::= (:goal [:agent <agent-def>] :condition <emptyOr (pre-GD)>)
<pref-con-GD> ::= (and (prop-con-GD)*)
<pref-con-GD> ::= (relative-preconditions [:forall (typed list (variable)> <pref-con-GD>]
<pref-con-GD> ::= (preference [:preference] <name> <con-GD>)
<pref-con-GD> ::= (con-GD)
<con-GD> ::= (:all <GD>)
<con-GD> ::= (:all <GD>)
<con-GD> ::= (:at (time-specifier) <GD>)
<con-GD> ::= (:at-most-once <GD>)
<con-GD> ::= (:sometime-after <GD> <GD>)
<con-GD> ::= (:sometime-before <GD> <GD>)
<con-GD> ::= (:within <number> <GD> <GD>)
<con-GD> ::= (:hold-after <number> <GD>)
<metric-spec> ::= (:metric optimization <metric-f-exp>)
<metric-spec> ::= (:metric optimization <metric-f-exp>)
<metric-spec> ::= (:metric optimization <metric-f-exp>)
<metric-spec> ::= (:metric optimization <metric-f-exp>)
<metric-f-exp> ::= (metric <function-symbol> term*)
<metric-f-exp> ::= (total-time)
<metric-f-exp> ::= (is-violated <pref-name>)
<length-spec> ::= (length [:parallel <integer>])

The length-spec is deprecated since PDDL 2.1.
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 2 respectively.

\[
\begin{align*}
<\text{con-GD}> &::= (\text{always} \ <\text{con2-GD}> ) \\
<\text{con-GD}> &::= (\text{sometime} \ <\text{con2-GD}> ) \\
<\text{con-GD}> &::= (\text{within} <\text{number}> \ <\text{con2-GD}> ) \\
<\text{con-GD}> &::= (\text{at-most-once} \ <\text{con2-GD}> ) \\
<\text{con-GD}> &::= (\text{always-within} <\text{number}> \ <\text{con2-GD}> <\text{con2-GD}> ) \\
<\text{con-GD}> &::= (\text{sometime-before} <\text{con2-GD}> <\text{con2-GD}> ) \\
<\text{con-GD}> &::= (\text{sometime-after} <\text{con2-GD}> <\text{con2-GD}> ) \\
<\text{con-GD}> &::= (\text{at-most-once} <\text{con2-GD}> ) \\
<\text{con-GD}> &::= (\text{within} <\text{number}> <\text{con2-GD}> <\text{con2-GD}> ) \\
<\text{con-GD}> &::= (\text{hold-during} <\text{number}> <\text{number}> <\text{con2-GD}> ) \\
<\text{con-GD}> &::= (\text{hold-after} <\text{number}> <\text{con2-GD}> ) \\
<\text{con-GD}> &::= (\text{hold}}

2.2 Requirements

Here is a table of all requirements in MA-PDDL extended with privacy. Some requirements imply others; some are abbreviations for common sets of requirements. If a domain stipulates no requirements, it is assumed to declare :strips.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>:strips</td>
<td>Basic STRIPS-style adds and deletes</td>
</tr>
<tr>
<td>:typing</td>
<td>Allows type names in declarations of variables</td>
</tr>
<tr>
<td>:negative-preconditions</td>
<td>Allows not in goal descriptions</td>
</tr>
<tr>
<td>:disjunctive-preconditions</td>
<td>Allows or in goal descriptions</td>
</tr>
<tr>
<td>:equality</td>
<td>Support = as built-in predicate</td>
</tr>
<tr>
<td>:existential-preconditions</td>
<td>Allows exists in goal descriptions</td>
</tr>
<tr>
<td>:universal-preconditions</td>
<td>Allows forall in goal descriptions</td>
</tr>
<tr>
<td>:quantified-preconditions</td>
<td>=:existential-preconditions</td>
</tr>
<tr>
<td>:universal-preconditions</td>
<td>=:universal-preconditions</td>
</tr>
<tr>
<td>:conditional-effects</td>
<td>Allows when in action effects</td>
</tr>
<tr>
<td>:object-fluents</td>
<td>Allows object function definitions similarly to :numeric-fluents.</td>
</tr>
<tr>
<td>:numeric-fluents</td>
<td>Allows numeric function definitions and use of effects using assignment operators and arithmetic conditions.</td>
</tr>
<tr>
<td>:features</td>
<td>Allows use of features in action preconditions and goals.</td>
</tr>
<tr>
<td>:preferences</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>
<tr>
<td>:multi-agent</td>
<td>Allows declaration of multi-agent planning domains and problems with different agents having possibly different actions, goals and metrics, where concurrent/joint actions may interact.</td>
</tr>
<tr>
<td>:unfactored-privacy</td>
<td>Allows the declaration of private (or internal) propositions (or atoms, grounded predicates) along the lines of MA-STRIPS [3] The idea is to allow the declaration of private predicates/constants/objects on a per agent basis. So the multi-agent requirement should also be declared for this requirement to have an effect. A non-private constant/object is public. A proposition (or atom) is private if and only if it refers to a non-private predicate and each argument of the proposition is public. Otherwise the proposition (or atom) is private. The arguments of a private proposition (or atom) of an agent can include only either public constants/objects or private constants/objects of the same agent. :unfactored-privacy is mutually exclusive with :factored-privacy.</td>
</tr>
<tr>
<td>:factored-privacy</td>
<td>Allows the declaration of private (or internal) propositions (or atoms, grounded predicates) along the lines of MA-STRIPS [3] for MA-PDDL descriptions factored from unfactored MA-PDDL descriptions, on a per agent basis. Thus the difference compared to :unfactored-privacy is that in this case the MA-PDDL description does not contain the private predicates/constants/objects of other agents (it is assumed that they are not observable). A factored description is single-agent. Only the private predicates/constants/objects of the agent for which the MA-PDDL description was factored are indicated. A non-private constant/object is public. A proposition (or atom) is public if and only if it refers to a non-private predicate and each argument of the proposition is public, otherwise it is private. This requirement is exclusive with :unfactored-privacy.</td>
</tr>
</tbody>
</table>

3. Complete BNF definition of MA-PDDL with privacy

D. L. Kovacs

4
Complete BNF definition of MA-PDDL with privacy  
D. L. Kovacs

3. References