

The Formal Syntax and Semantics of Web-PDDL

Dejing Dou

Computer and Information Science
University of Oregon
Eugene, OR 97403, USA
dou@cs.uoregon.edu

Abstract. This white paper formally defines the syntax and semantics of Web-PDDL, a strongly typed first order logic ontology language used by OntoEngine [2,1]. The ability to express XML namespaces makes Web-PDDL well compatible with other web (ontology) languages, such as RDF and OWL. The presence of types makes it possible to do meaningful and efficient type checking during logic reasoning. The type equality and multi inheritance make Web-PDDL more expressive for describing semantic mapping rules between different ontologies.

1 Introduction

Web-PDDL is a strongly typed first order language with Lisp like syntax. The ability to express namespaces with URIs and prefixes makes Web-PDDL well compatible with other web languages. The presence of types makes it possible to do meaningful type checking of Web-PDDL documents. The type equality and multi inheritance make Web-PDDL more expressive for describing the relationship of concepts from different ontologies in the Semantic Web model.

Web-PDDL can be used to represent ontologies, datasets and queries. Here is an example, part of the `yale.bib` ontology written in Web-PDDL:

```
(define (domain yale_bib-ont)
  (:extends (uri "http://www.w3.org/2000/01/rdf-schema#" :prefix rdfs))
  (:types Publication - Obj
         Article Book Incollection Inproceedings - Publication
         Literal - @rdfs:Literal)
  (:predicates (author p - Publication a - Literal)
               .....))
```

The `:extends` declaration expresses that this domain (i.e., ontology) is extended from one or more other ontologies identified by the URIs. To avoid symbol clashes, symbols imported from other ontologies are given prefixes, such as `@rdfs:Literal`. These correspond to XML namespaces, and when Web-PDDL is translated to RDF, that's exactly what they become. Types start with capital letters. A constant or variable is declared to be of a type T by writing "`x - T`".

There was a formal definition for the syntax and less formally, the semantics of PDDL in its manual [3]. PDDL is intended to describe planning domains and

problems. It can describe sets of *actions* an agent can take, and what their *pre-conditions* and *effects* are. There is intense interest in development of notations for describing web services, and Web-PDDL, which is extended from PDDL, is ideal for that purpose [4]. Besides web services, Web-PDDL also can describe other kinds of ontologies, datasets and queries on the Semantic Web model. This dissertation defines the formal syntax and semantics of Web-PDDL in a general way and describes its differences from PDDL.

2 The Syntax of Web-PDDL

In order to define the syntax of Web-PDDL, we first list the *Symbols*:

- Parentheses: (,).
- Negation: not.
- Binary Connectives: if, iff.
- n-ary Connectives: and, or.
- Types (Classes): $t_1, t_2, t_3 - T_0...$
- Type inheritance (subClassOf): - .
- Type Equal(sameClassAs): T->
- Quantifiers: forall, exists.
- Variables: $x - t_1, y - t_2, z - t_3...$
- n-ary predicates (properties): $(P a_1 a_2...a_n)$
- n-ary functions: $(f a_1 a_2...a_n) - t$
- Constants: $c_1 - t_1, c_2 - t_2, c_3 - t_3...$
- Query variables: $?x - t_1, ?y - t_2, ?z - t_3...$
- Prefix expression for namespace: @ns₁;, @ns₂;, @ns₃;...
- Comments: begin with a semicolon(“;”) and end with the next new line.

We use Web-PDDL mainly to describe ontologies, dataset and queries on the Semantic Web. Before formally defining the syntax of ontology, dataset and queries in Web-PDDL, we define the notation for the syntax definition. Our notation is an BNF with following conventions.

- Each rule is of the form $\langle syntactic\ element \rangle ::= expansion.$
- Angle brackets delimit names of syntactic elements.
- Square brackets surround optional material.
- A plus (+) means "one or more of".
- $\langle typed\ list\ (symbol) \rangle$ means that a list of symbols with type definition. The symbol can be type, variable, query variable, constant, or function.

Then we can define the syntax of an *Ontology* in Web-PDDL:

```

<ontology> ::= (define (domain <local name>)
                [<namespaces-def>]
                [<extension-def>]
                [<types-def>]
                [<constants-def>]
                [<predicates-def>]
                [<functions-def>]
                [<axioms-def>]
                [<facts-def>])
<namespaces-def> ::= (:namespaces (:uri <URI> :prefix <prefix>)+)
<extension-def> ::= (:extends (:uri <URI> :prefix <prefix>)+)
<types-def> ::= (:types <typed list (type name)>)
<constants-def> ::= (:constants <typed list (constant name)>)

<predicates-def> ::= (:predicates <atomic formula skeleton>+)
<atomic formula skeleton> ::= (<predicate name> <typed list (variable)>)
<functions-def> ::= (:functions <typed list <function skeleton>+)
<function skeleton> ::= (<function name> <typed list (variable)>)
<axioms-def> ::= (:axioms <axiom>+)
<facts-def> ::= (:facts <atomic formula>+)
<axiom> ::= (T-> <type> <type>)
<axiom> ::= (<quantifier> (<typed list (variable)>
                        <formula>))

<formula> ::= <atomic formula>
<formula> ::= (not <formula>)
<formula> ::= (<n-ary connectives> <formula>+)
<formula> ::= (<binary connectives> <formula> <formula>)
<atomic formula> ::= (is <type> <term>)
<atomic formula> ::= (<predicate name> <term>+)
<term> ::= <string>
<term> ::= <number>
<term> ::= <variable>
<term> ::= <constant>
<term> ::= <function term>
<function term> ::= (<function name> <term>+)

```

The syntax of a *Dataset* in Web-PDDL is defined as:

```

<dataset> ::= (define (dataset <local name>)
                <ontologies-def>
                [<objects-def>])

```

```

                                <facts-def>)
<ontologies-def> ::= (:domain (:uri <URI> :prefix <prefix>)+)
<objects-def> ::= (:objects <typed list (constant name)>)
<facts-def> ::= (:facts <atomic formula>+)
<atomic formula> ::= (<predicate name> <term>+)
<term> ::= <string>
<term> ::= <number>
<term> ::= <constant>
<term> ::= <function term>
<function term> ::= (<function name> <term>+)

```

The syntax of a *Query* in Web-PDDL is defined as:

```

<query> ::= (define (query <local name>)
              <ontologies-def>
              [<objects-def>]
              [<predicates-def>]
              <query-def>
              [<answer-def>])
<ontologies-def> ::= (:domain (:uri <URI> :prefix <prefix>)+)
<objects-def> ::= (:objects <typed list (constant name)>)
<predicates-def> ::= (:predicates <atomic formula skeleton>+)
<atomic formula skeleton> ::= (<predicate name> <typed list (variable)>)
<query-def> ::= (:query
                 (queryvars <typed list (query variable)>)
                 <query formula>)

<answer-def> ::= (:output
                 (queryvars <typed list (query variable)>)
                 <query formula>)

<query formula> ::= <atomic query formula>
<query formula> ::= (not <query formula>)
<query formula> ::= (<n-ary connectives> <query formula>+)
<atomic query formula> ::= (<predicate name> <term>+)
<term> ::= <string>
<term> ::= <number>
<term> ::= <query variable>
<term> ::= <constant>

```

Note, the syntax of *Bridging Axiom* (Semantic Mapping rules) in Web-PDDL is same as the syntax of general Web-PDDL axioms. However, the types and predicates in same bridging axiom may have different namespaces between they may come from different ontologies. Readers can check [2] for more detail ex-

amples for bridging axioms. More bridging axioms with merged ontologies can be found in OntoMerge web site¹ and OntoGrate web site².

The Changes of Syntax from PDDL

Web-PDDL is mainly designed for web applications. There are some changes of syntax from PDDL:

1. We add namespaces in Web-PDDL to distinguish the symbols with same name but from different ontologies or different data resources on the Web. For example, both the `yale_bib` and `cmu_bib` ontologies have a class (type) as `Article`. In Web-PDDL, we use `@yale_bib:Article` and `@cmu_bib:Article` to represent these two types. To represent namespaces, the keywords about URIs and prefixes are added into Web-PDDL.
2. With the help of namespaces, we can represent the types from different ontologies in one Web-PDDL document. We may need to say two types are same or equal. There is no any symbol in PDDL to express this “type-equal” (sameClassAs) relationship. In Web-PDDL, we use “T->” to express it. For example, `(T-> @yale_bib:Techreport @cmu_bib:TechReport)` means that the `Techreport` type in the `yale_bib` ontology is same as the `TechReport` type in the `cmu_bib` ontology. We can not use “=” for type-equal because a type in Web-PDDL or PDDL is not an object.
3. Besides the built-in functions, such as `+`, `-`, `*`, and `/`, we can use Web-PDDL to declare any other functions.
4. In Web-PDDL, we use a simpler and more flexible way to express axioms than PDDL. For example, an axiom in blocks world is expressed in PDDL:

```
(:axiom
  :vars (x y - physob)
  :context (on x y)
  :implies (above x y))
```

In Web-PDDL, it can be expressed like:

```
(forall (x y - physob)
  (if (on x y) (above x y)))
```

5. In order to process queries, we use symbols with question mark, such as `?x`, to represent query variables in Web-PDDL. There is no query processing in PDDL, both `?x` and `x` are used for variables in PDDL.

3 The Semantics of Web-PDDL

Since Web-PDDL is a first order logic language, in order to define its semantics, we must say what *domain* is involved for the quantifiers to quantify for. We must

¹ <http://aimlab.cs.uoregon.edu/ontomerge/ontoMerge.html>

² <http://aimlab.cs.uoregon.edu/ontograte/ontoGrate.jsp>

say how we are interpreting the constant, function and predicate with respect to that domain, an *interpretation*. These two items specify a *model*. Since the formulas may contain variables, we also need to give an *assignment* of values to them when we try to define the truth and satisfaction of formulas. Therefore, we formally define the semantics of Web-PDDL as following:

Definition 3.1 A *model* for the Web-PDDL is a pair $\mathbf{M} = \langle \mathbf{D}, \mathbf{I} \rangle$ where:

\mathbf{D} is a set of sets related to types, called the *domain* of \mathbf{M} .

$\mathbf{D}_0 \in \mathbf{D}$ s.t. $\mathbf{D}_0 = \bigcup \mathbf{D}$.

\mathbf{I} is a mapping, called an *interpretation* that associates expressions with:

\mathbf{I} assigns to primitive type τ a set \mathbf{D}_τ : $\tau^{\mathbf{I}} = \mathbf{D}_\tau$.

$Object^{\mathbf{I}} = \mathbf{D}_{Object} = \mathbf{D}_0$ because *Object* is the built-in super type of all other types in a domain.

To constant c , c of type τ_1 , $c^{\mathbf{I}} \in \mathbf{D}_{\tau_1}$.

To n-ary function f with result type τ_0 and the arguments of types $\tau_1, \tau_2 \dots \tau_n$, $f^{\mathbf{I}} \subseteq \langle \mathbf{D}_{\tau_1}, \mathbf{D}_{\tau_2} \dots \mathbf{D}_{\tau_n}, \mathbf{D}_{\tau_0} \rangle$. Here, $\langle \mathbf{D}_{\tau_1}, \mathbf{D}_{\tau_2} \dots \mathbf{D}_{\tau_n}, \mathbf{D}_{\tau_0} \rangle$ is a set of n+1-tuples from the domain.

To n-ary predicate P , the arguments of types $\tau_1, \tau_2 \dots \tau_n$, $P^{\mathbf{I}} \subseteq \langle \mathbf{D}_{\tau_1}, \mathbf{D}_{\tau_2} \dots \mathbf{D}_{\tau_n} \rangle$.

Definition 3.2 An *assignment* in a model $\mathbf{M} = \langle \mathbf{D}, \mathbf{I} \rangle$ is a mapping \mathbf{A} from the set of variables to the sets of \mathbf{D} . For a variable v of type τ_1 , $v^{\mathbf{A}} \in \mathbf{D}_{\tau_1}$.

Definition 3.3 Let $\mathbf{M} = \langle \mathbf{D}, \mathbf{I} \rangle$ be a model for Web-PDDL, and let \mathbf{A} be an assignment in this model. To each term t (a constant, variable or function) of Web-PDDL, we associate a value $t^{\mathbf{I}, \mathbf{A}}$ in the sets of \mathbf{D} as follows:

1. For a constant symbol c , $c^{\mathbf{I}, \mathbf{A}} = c^{\mathbf{I}}$.
2. For a variable symbol v , $v^{\mathbf{I}, \mathbf{A}} = v^{\mathbf{A}}$.
3. For a function symbol f , $(f \ t_1 \ t_2 \dots t_n)^{\mathbf{I}, \mathbf{A}} = v$, where v is the unique object s.t. $\langle t_1^{\mathbf{I}, \mathbf{A}} \ t_2^{\mathbf{I}, \mathbf{A}} \dots t_n^{\mathbf{I}, \mathbf{A}}, v \rangle \in f^{\mathbf{I}}$

Definition 3.4 Let x be a variable of type τ . The assignment \mathbf{B} in the model \mathbf{M} is an *x-variant* of the assignment \mathbf{A} , provided \mathbf{A} and \mathbf{B} assign the same values to every variable except possibly x .

Definition 3.5 Let $\mathbf{M} = \langle \mathbf{D}, \mathbf{I} \rangle$ be a model for Web-PDDL, and let \mathbf{A} be an assignment in this model. To each formula Φ of Web-PDDL, we associate a *truth value* $\Phi^{\mathbf{I}, \mathbf{A}}$ (**t** or **f**) as follows:

1. For an atomic formula P :
 $(P \ a_1 \ a_2 \dots a_n)^{\mathbf{I}, \mathbf{A}} = \mathbf{t} \Leftrightarrow \langle a_1^{\mathbf{I}, \mathbf{A}} \ a_2^{\mathbf{I}, \mathbf{A}} \dots a_n^{\mathbf{I}, \mathbf{A}} \rangle \in P^{\mathbf{I}}$.
2. For the negation of a formula Φ , $(\text{not } \Phi)^{\mathbf{I}, \mathbf{A}} = (\text{not } \Phi^{\mathbf{I}, \mathbf{A}})$.
3. For the formulas connected by the connective *conn*:
 $(\text{conn } \Phi_1 \ \Phi_2 \dots \Phi_n)^{\mathbf{I}, \mathbf{A}} = (\text{conn } \Phi_1^{\mathbf{I}, \mathbf{A}} \ \Phi_2^{\mathbf{I}, \mathbf{A}} \dots \Phi_n^{\mathbf{I}, \mathbf{A}})$.
4. (forall $(x - t)$ Φ) $^{\mathbf{I}, \mathbf{A}} = \mathbf{t} \Leftrightarrow \Phi^{\mathbf{I}, \mathbf{B}} = \mathbf{t}$ for every assignment \mathbf{B} in \mathbf{M} that is an *x-variant* of \mathbf{A} .
5. (exists $(x - t)$ Φ) $^{\mathbf{I}, \mathbf{A}} = \mathbf{t} \Leftrightarrow \Phi^{\mathbf{I}, \mathbf{B}} = \mathbf{t}$ for some assignment \mathbf{B} in \mathbf{M} that is an *x-variant* of \mathbf{A} .

Definition 3.6 A formula Φ of Web-PDDL is *true in the model* $\mathbf{M} = \langle \mathbf{D}, \mathbf{I} \rangle$ if $\Phi^{\mathbf{I}, \mathbf{A}} = \mathbf{t}$ for all assignments \mathbf{A} . A formula Φ is *valid* if Φ is true in all models.

A set \mathcal{S} of formulas is *satisfiable* in the model $\mathbf{M} = \langle \mathbf{D}, \mathbf{I} \rangle$, provided there is some assignment \mathbf{A} such that $\Phi^{\mathbf{I}, \mathbf{A}} = \mathbf{t}$ for all $\Phi \in \mathcal{S}$. \mathcal{S} is satisfiable if it is satisfiable in some model.

The Changes of Semantics from PDDL

The main changes of semantics from PDDL to Web-PDDL are related to its type system:

1. In a model of PDDL, if we still define it as a pair $\mathbf{M} = \langle \mathbf{D}, \mathbf{I} \rangle$ where domain \mathbf{D} is a set of sets related to types. $\mathbf{D}_0 \in \mathbf{D}$ s.t. $\mathbf{D}_0 = \bigcup \mathbf{D}$. The interpretation \mathbf{I} assigns to primitive type τ a set \mathbf{D}_τ : $\tau^{\mathbf{I}} = \mathbf{D}_\tau$.
 In PDDL, any two types either are disjoint or have subsumption relationship: if $\tau_1 \neq \tau_2$, then $\mathbf{D}_{\tau_1} \cap \mathbf{D}_{\tau_2} = \emptyset$ or $\mathbf{D}_{\tau_1} \subseteq \mathbf{D}_{\tau_2}$ or $\mathbf{D}_{\tau_2} \subseteq \mathbf{D}_{\tau_1}$.
 In Web-PDDL, there is no such constraint because we have to describe relationships of types from different ontologies on Web. Two types can be equal, can be disjoint or can have subsumption relationship when we define them in ontologies:
 “(T-> τ_1 τ_2)” means $\mathbf{D}_{\tau_1} = \mathbf{D}_{\tau_2}$.
 “ τ_1 - τ_2 ” means $\mathbf{D}_{\tau_1} \subseteq \mathbf{D}_{\tau_2}$.
 “ τ_1 τ_2 - τ_0 ” means $\mathbf{D}_{\tau_1} \cap \mathbf{D}_{\tau_2} = \emptyset$ and $\mathbf{D}_{\tau_1} \subseteq \mathbf{D}_{\tau_0}$ and $\mathbf{D}_{\tau_2} \subseteq \mathbf{D}_{\tau_0}$.
 Otherwise, the relationship of two types is implicit.
2. In PDDL, one type only directly inherit from one super type. It is called *single inheritance*. In Web-PDDL, one type can directly inherit from multiple super types. It is called *multi inheritance*:
 “ τ_1 - τ_2 ” and “ τ_1 - τ_3 ” mean that $\mathbf{D}_{\tau_1} \subseteq \mathbf{D}_{\tau_2} \cap \mathbf{D}_{\tau_3}$.

References

1. OntoEngine Source Code.
<http://projects.semwebcentral.org/projects/ontoengine/>.
2. D. Dou, D. V. McDermott, and P. Qi. Ontology Translation on the Semantic Web. *Journal of Data Semantics*, 2:35–57, 2005.
3. D. V. McDermott. The Planning Domain Definition Language Manual. Technical Report 98-003, Department of Computer Science, Yale University, 1998.
4. D. V. McDermott and D. Dou. Representing Disjunction and Quantifiers in RDF. In *International Semantic Web Conference*, pages 250–263, 2002.