The STRIPS Subset of PDDL for the Learning Track of IPC-08

Alan Fern
School of Electrical Engineering and Computer Science
Oregon State University

April 9, 2008

This document defines two subsets of PDDL (planning domain definition language) that will be used in the learning track of the 2008 International Planning Competition. Both languages are fully compatible with standard PDDL. The first PDDL fragment corresponds to the un-typed STRIPS action language and was selected for its simplicity in order to facilitate wider participation. The second language is simply the STRIPS language extended with explicit PDDL typing constructs. The competition will contain both typed and un-typed encodings of each domain and problem instance. These encodings will be semantically equivalent and the participants will have the choice of which to use.

This document draws largely on the one written by Fahiem Bacchus for the 2000 International Planning Competition.

Syntactic Notation

We follow the notation used in the original PDDL definition. That is, we use an extended BNF (EBNF) with the following conventions:

1. Each rule is of the form <syntactic element> ::= expansion.
2. Angle brackets delimit names of syntactic elements.
3. Square brackets ([and]) surround optional material.

4. An asterisk (*) means zero or more of; a plus (+) means one or more of.

5. Ordinary parenthesis are an essential part of the syntax we are defining and have no semantics in the EBNF meta language.

6. We use a single EBNF to describe both typed and un-typed STRIPS. Rules or elements that have the superscript *:typing* are only allowed to be used when the (:requirements :typing) tag is included, indicating that typed STRIPS is being used. To further clarify we have also used boldface to indicate items and rules that are only available for typed STRIPS.

Domains

We now describe the languages formally. The EBNF for defining a typed or un-typed STRIPS domain is given in Figure 1.

Names. In EBNF the category <name> consists of strings of characters that beginning with a letter and contain only letters, digits, hyphens, and underscores. Case is not significant. <name> non-terminals are required to be unique. That is, one cannot use the same name in two different definitions.

Requirements. The tag (:requirements :typing) will be included directly after the domain name if and only if the domain is a typed STRIPS domain.

Typing. For typed STRIPS domains the :types field will immediately follow the requirements tag. This field provides a type declaration, which for our PDDL fragment will simply be a list of names, where each name in the list becomes a distinct type. Note that the more general PDDL typing language allows for some amount of type hierarchy. For simplicity we will not include hierarchical type structures in our domains.

Constants. The optional :constants field is either a lit of names for un-typed STRIPS or a typed list of names for typed STRIPS. A typed list of names is simply a list of names, where each name specifies a constant, which may also include types that are preceded by minus signs. The type of a constant is taken to be that of the first type that follows the constant. For example, (:constants h1 h2 - hand b1 b2 - block) defines four constants h1 and h2 of type hand and b1 and b2 of type block.

Predicates. The required :predicates field consists of a list of declarations of predicates. For untyped STRIPS each declaration gives the predicate

<domain>	::=	(define (domain <name>)
		[(:requirements :typing)]
		[(:types <name>$^+$)]$^{\text{:typing}}$
		[<constants-def>]
		<predicates-def>
		<action-def>*)
<constants-def>	::=	(:constants <typed?-list-of (names)>)
<typed?-list-of (x)>	::=	x*
<typed?-list-of (x)>	**::=$^{\text{:typing}}$**	**x$^+$ - <name> <typed-list-of (x)>**
<predicates-def>	::=	(:predicates <atom skeleton>$^+$)
<atom skeleton>	::=	(<predicate> <typed?-list-of (variable)>)
<predicate>	::=	<name>
<variable>	::=	?<name>
<action-def>	::=	(:action <name>
		:parameters <action-parameters>
		<action-def body>)
<action-parameters>	::=	(<typed?-list-of (variable)>)
<action-def body>	::=	[:precondition <conjunction (atom)>]
		:effect <conjunction (literal)>
<conjunction (x)>	::=	x
<conjunction (x)>	::=	(and x$^+$)
<atom>	::=	(<predicate> <term>*)
<literal>	::=	<atom>
<literal>	::=	(not <atom>)
<term>	::=	<name>
<term>	::=	<variable>

Figure 1: EBNF for STRIPS domain definitions. Typed STRIPS domains will always have the (:requirements :typing) tag immediately after the domain name which makes all of the rules in the EBNF available for use. Un-typed STRIPS domains will not include the (:requirements :typing) tag and are not allowed to make use of lines in the EBNF that are shown in bold.

name followed by a list of variables to declare the arity of the predicate. For typed STRIPS we specify a typed list of variables, which has the same form as the typed lists used for constant declarations described above. Equality = is a predefined predicate taking two arguments of any type.

Actions. For un-typed STRIPS the :parameters list is simply the list of variables on which the particular rule operates, i.e., its arguments. For typed STRIPS the :parameter list is a typed list of variables. :precondition is a conjunction of atoms, while :effect is a conjunction of literals. In both cases, all of the free variables in these components must appear among the :parameters. These actions have standard STRIPS semantics. In particular, every type abiding binding of the :parameters, σ, generates a particular instance of the action. In the case of un-typed STRIPS this means that every possible combination of constant bindings to variables is a potential action. An action instance is applicable to a state S if and only if the :precondition is true in S under the bindings σ. The instance will then map to a new state S' generated by adding to all positive atoms appearing in :effect (after applying the binding σ), deleting from S all negative atomic formulas appearing in :effect (after applying σ), and leaving unchanged all other ground atomic formulas true in S.

Types Versus Un-Typed STRIPS Domains

Each domain in the competition will have two equivalent descriptions: one typed and one un-typed. The typed description will always include the (:requires :typing) flag and is allowed to use the entire EBNF grammar provided above. The un-typed description will never include the flag and is not allowed to use any of the typing constructs. This means that un-typed STRIPS will not include any type definitions and will not include types in the parameter lists of actions, constant declarations, or predicate declarations. When creating domains for the competition, the typed version will be created first and then an un-typed version will be created by explicitly defining monadic predicates for each type and including them in the preconditions of actions to enforce the type constraints.

<problem>	::=	(define (problem <name>)
		(:domain <name>)
		(:objects <object declaration>)
		[<init>]
		<goal>)
<object declaration>	::=	<typed?-list-of (name)>
<init>	::=	(:init <atom>*)
<goal>	::=	(:goal <conjunction (ground-atom)>)
<ground-atom>	::=	(<predicate> <name>*)

Figure 2: EBNF for STRIPS problem definition. The non-terminals <conjunction (x)> and <typed?-list-of (x)> are defined in Figure 1.

Problems

A problem is what a planner tries to solve. It is defined with respect to a domain. A problem specifies two things: an initial situation, and a goal to be achieved. The syntax for problem specifications is given in Figure 2. A problem definition must specify an initial situation by a list of initially true ground atoms. The :objects field is required to be present and lists objects that exist in this problem (which might be a superset of those appearing it the initial situation). For un-typed STRIPS this is simply a list of object names. For typed STRIPS this is a typed list of object names, where an object is assigned to the first type that follows is, where the types are immediately preceded by minus signs. The :goal of a problem definition is a conjunction of ground atoms. Note that the goals will not involve variables in the competition. A solution to a problem is a series of actions such that (a) the action sequence is feasible starting in the given initial situation; (b) the :goal, if any, is true in the situation resulting from executing the action sequence.

Example Domain and Problem

Below is an example domain definitions and problems from the blocks world for typed and un-typed STRIPS.

```
(define (domain typed-blocksworld)
(:requirements :typing)
(:typing block hand)
(:predicates (clear ?b - block)
             (on-table ?b - block)
             (empty ?h - hand)
             (holding ?h - hand ?b - block)
             (on ?b1 ?b2 - block))
(:action pickup
  :parameters (?h - hand ?b - block)
  :precondition (and (clear ?b) (on-table ?b) (empty ?h))
  :effect (and (holding ?h ?b) (not (clear ?b)) (not (on-table ?b))
               (not (empty ?h))))
(:action putdown
  :parameters  (?h - hand ?b - block)
  :precondition (holding ?h ?b)
  :effect (and (clear ?b) (empty ?h) (on-table ?b)
               (not (holding ?h ?b))))
(:action stack
  :parameters  (?h - hand ?b ?underb - block)
  :precondition (and (clear ?underb) (holding ?h ?b))
  :effect (and (empty ?h) (clear ?b) (on ?b ?underb)
               (not (clear ?underb)) (not (holding ?h ?b))))
(:action unstack
  :parameters  (?h - hand ?b ?underb - block)
  :precondition (and (on ?b ?underb) (clear ?b) (empty ?h))
  :effect (and (holding ?h ?b) (clear ?underb)
               (not (on ?b ?underb)) (not (clear ?b)) (not (empty ?h)))))

(define (problem typed-blocks1)
         (:domain typed-blocksworld)
         (:requirements :typing)
         (:objects H - hand A B C - block)
         (:init (clear A) (on A B) (on B C) (on-table C) (empty H))
         (:goal (and (on C B) (on B A))))
```

Figure 3: Example blocksworld domain and problem for Typed STRIPS.

```
(define (domain untyped-blocksworld)
(:predicates (clear ?b)
             (on-table ?b)
             (empty ?h)
             (holding ?h ?b)
             (on ?b1 ?b2)
             (hand ?h)
             (block ?b))
(:action pickup
  :parameters (?h ?b)
  :precondition (and (hand ?h) (block ?b) (clear ?b) (on-table ?b) (empty ?h))
  :effect (and (holding ?h ?b) (not (clear ?b)) (not (on-table ?b))
               (not (empty ?h))))
(:action putdown
  :parameters  (?h ?b)
  :precondition (and (hand ?h) (block ?b) (holding ?h ?b))
  :effect (and (clear ?b) (empty ?h) (on-table ?b)
               (not (holding ?h ?b))))
(:action stack
  :parameters  (?h ?b ?underb)
  :precondition (and (hand ?h) (block ?b) (block ?underb)
                     (clear ?underb) (holding ?h ?b))
  :effect (and (empty ?h) (clear ?b) (on ?b ?underb)
               (not (clear ?underb)) (not (holding ?h ?b))))
(:action unstack
  :parameters  (?h ?b ?underb)
  :precondition (and (hand ?h) (block ?b) (block ?underb)
                     (on ?b ?underb) (clear ?b) (empty ?h))
  :effect (and (holding ?h ?b) (clear ?underb)
               (not (on ?b ?underb)) (not (clear ?b)) (not (empty ?h)))))

(define (problem untyped-blocks1)
         (:domain untyped-blocksworld)
         (:objects H A B C)
         (:init (hand H) (block A) (block B) (block C)
                (clear A) (on A B) (on B C) (on-table C) (empty H))
         (:goal (and (on C B) (on B A))))
```

Figure 4: Example blocksworld domain and problem for un-typed STRIPS.