Principles of AI Planning
3. PDDL

Bernhard Nebel and Robert Mattmüller

Albert-Ludwigs-Universität Freiburg

October 28th, 2011
3.1 Schematic operators

3.2 PDDL
3.1 Schematic operators

- Schematic operators
Schematic operators

- Description of state variables and operators in terms of a given finite set of objects.
- Analogy: propositional logic vs. predicate logic
- Planners take input as schematic operators and translate them into (ground) operators. This is called grounding.
Schematic operators: example

Schematic operator \( \text{drive} \_ \text{car} \_ \text{from} \_ \text{to}(x,y_1,y_2) \):

\[
x \in \{ \text{car}1, \text{car}2 \}, \\
y_1 \in \{ \text{Freiburg, Strasbourg} \}, \\
y_2 \in \{ \text{Freiburg, Strasbourg} \}
\]

\[
\langle \text{in}(x,y_1), \text{in}(x,y_2) \land \lnot \text{in}(x,y_1) \rangle
\]

corresponds to the operators

\[
\langle \text{in}(\text{car}1, \text{Freiburg}), \text{in}(\text{car}1, \text{Strasbourg}) \land \lnot \text{in}(\text{car}1, \text{Freiburg}) \rangle, \\
\langle \text{in}(\text{car}1, \text{Strasbourg}), \text{in}(\text{car}1, \text{Freiburg}) \land \lnot \text{in}(\text{car}1, \text{Strasbourg}) \rangle, \\
\langle \text{in}(\text{car}2, \text{Freiburg}), \text{in}(\text{car}2, \text{Strasbourg}) \land \lnot \text{in}(\text{car}2, \text{Freiburg}) \rangle, \\
\langle \text{in}(\text{car}2, \text{Strasbourg}), \text{in}(\text{car}2, \text{Freiburg}) \land \lnot \text{in}(\text{car}2, \text{Strasbourg}) \rangle,
\]

plus four operators that are never applicable (inconsistent change set!)
and can be ignored, like

\[
\langle \text{in}(\text{car}1, \text{Freiburg}), \text{in}(\text{car}1, \text{Freiburg}) \land \lnot \text{in}(\text{car}1, \text{Freiburg}) \rangle.
\]
**Schematic operators: quantification**

**Existential quantification (for formulae only)**

Finite disjunctions $\varphi(a_1) \lor \cdots \lor \varphi(a_n)$ represented as

$\exists x \in \{a_1, \ldots, a_n\} : \varphi(x)$.

**Universal quantification (for formulae and effects)**

Finite conjunctions $\varphi(a_1) \land \cdots \land \varphi(a_n)$ represented as

$\forall x \in \{a_1, \ldots, a_n\} : \varphi(x)$.

**Example**

$\exists x \in \{A, B, C\} : \text{in}(x, \text{Freiburg})$ is a short-hand for

$\text{in}(A, \text{Freiburg}) \lor \text{in}(B, \text{Freiburg}) \lor \text{in}(C, \text{Freiburg})$. 
3.2 PDDL

- Overview
- Domain files
- Problem files
- Example
PDDL: the Planning Domain Definition Language

- used by almost all implemented systems for deterministic planning
- supports a language comparable to what we have defined above (including schematic operators and quantification)
- syntax inspired by the Lisp programming language: e.g. prefix notation for formulae
  
  \[
  \text{(and (or (on A B) (on A C))}
  
  \text{(or (on B A) (on B C))}
  
  \text{(or (on C A) (on A B))}
  \]
PDDL: domain files

A domain file consists of

- (define (domain DOMAINNAME)
- a :requirements definition (use :strips :typing by default)
- definitions of types (each parameter has a type)
- definitions of predicates
- definitions of operators
Example: blocks world (with hand) in PDDL

Note: Unlike in the previous chapter, here we use a variant of the blocks world domain with an explicitly modeled gripper/hand.

(define (domain BLOCKS)
 (:requirements :strips :typing)
 (:types block)
 (:predicates (on ?x - block ?y - block)
 (ontable ?x - block)
 (clear ?x - block)
 (handempty)
 (holding ?x - block)
 )
PDDL: operator definition

- (:action OPERATORNAME)
- list of parameters: (?x - type1 ?y - type2 ?z - type3)
- precondition: a formula
  
  `<schematic-state-var>`
  
  (and <formula> ... <formula>)
  
  (or <formula> ... <formula>)
  
  (not <formula>)
  
  (forall (?x1 - type1 ... ?xn - typen) <formula>)
  
  (exists (?x1 - type1 ... ?xn - typen) <formula>)

Note: Pyperplan only supports atoms and conjunctions of atoms.
effect:

<schematic-state-var>
(not <schematic-state-var>)
(and <effect> ... <effect>)
(when <formula> <effect>)
(forall (?x1 - type1 ... ?xn - typen) <effect>)

Note: Pyperplan only supports literals and conjunctions of literals.
(:action stack
  :parameters (?x - block ?y - block)
  :precondition (and (holding ?x) (clear ?y))
  :effect (and (not (holding ?x))
            (not (clear ?y))
            (clear ?x)
            (handempty)
            (on ?x ?y)))
A problem file consists of

- (define (problem PROBLEMNAME))
- declaration of which domain is needed for this problem
- definitions of objects belonging to each type
- definition of the initial state (list of state variables initially true)
- definition of goal states (a formula like operator precondition)
(define (problem example)
  (:domain BLOCKS)
  (:objects a b c d - block)
  (:init (clear a) (clear b) (clear c) (clear d)
       (ontable a) (ontable b) (ontable c)
       (ontable d) (handempty))
  (:goal (and (on d c) (on c b) (on b a)))
)

Example run on the Pyperplan planner

# ./pyperplan.py blocks-dom.pddl blocks-prob.pddl

[...]
2011-10-27 22:29:21,326 INFO  Search start: example
2011-10-27 22:29:21,330 INFO  114 Nodes expanded
2011-10-27 22:29:21,330 INFO  Search end: example
[...]
[...]
Example plan found by the Pyperplan planner

# cat blocks-prob.pddl.soln
(pick-up b)
(stack b a)
(pick-up c)
(stack c b)
(pick-up d)
(stack d c)
Example: blocks world in PDDL

(define (domain BLOCKS)
  (:requirements :strips :typing)
  (:types block)
  (:predicates (on ?x - block ?y - block)
      (ontable ?x - block)
      (clear ?x - block)
      (handempty)
      (holding ?x - block)
  )
)
(:action pick-up
  :parameters (?x - block)
  :precondition (and (clear ?x) (ontable ?x)
                   (handempty))
  :effect (and (not (ontable ?x))
               (not (clear ?x))
               (not (handempty))
               (holding ?x)))
(:action put-down
   :parameters (?x - block)
   :precondition (holding ?x)
   :effect (and (not (holding ?x))
               (clear ?x)
               (handempty)
               (ontable ?x)))
(:action stack
  :parameters (?x - block ?y - block)
  :precondition (and (holding ?x) (clear ?y))
  :effect (and (not (holding ?x))
    (not (clear ?y))
    (clear ?x)
    (handempty)
    (on ?x ?y)))
(:action unstack
  :parameters (?x - block ?y - block)
  :precondition (and (on ?x ?y) (clear ?x)
                  (handempty))
  :effect (and (holding ?x)
               (clear ?y)
               (not (clear ?x))
               (not (handempty))
               (not (on ?x ?y))))
(define (problem example)
  (:domain BLOCKS)
  (:objects a b c d - block)
  (:init (clear a) (clear b) (clear c) (clear d)
         (ontable a) (ontable b) (ontable c)
         (ontable d) (handempty))
  (:goal (and (on d c) (on c b) (on b a)))
)