Principles of AI Planning
3. PDDL

Malte Helmert and Bernhard Nebel

Albert-Ludwigs-Universität Freiburg

April 29th, 2010
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

\[ x \in \{\text{car1, car2}\}, \]
\[ y_1 \in \{\text{Freiburg, Strasbourg}\}, y_1 \neq y_2 \]
\[ \langle \text{in}(x, y_1), \text{in}(x, y_2) \land \neg \text{in}(x, y_1) \rangle \]

corresponds to the operators

\[ \langle \text{in}(\text{car1}, \text{Freiburg}), \text{in}(\text{car1}, \text{Strasbourg}) \land \neg \text{in}(\text{car1}, \text{Freiburg}) \rangle, \]
\[ \langle \text{in}(\text{car1}, \text{Strasbourg}), \text{in}(\text{car1}, \text{Freiburg}) \land \neg \text{in}(\text{car1}, \text{Strasbourg}) \rangle, \]
\[ \langle \text{in}(\text{car2}, \text{Freiburg}), \text{in}(\text{car2}, \text{Strasbourg}) \land \neg \text{in}(\text{car2}, \text{Freiburg}) \rangle, \]
\[ \langle \text{in}(\text{car2}, \text{Strasbourg}), \text{in}(\text{car2}, \text{Freiburg}) \land \neg \text{in}(\text{car2}, \text{Strasbourg}) \rangle \]
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**

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

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

PDDL
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

\[
\text{(and } \text{(or } \text{(on A B) (on A C)}) \\
\text{(or } \text{(on B A) (on B C)}) \\
\text{(or } \text{(on C A) (on A B)})\text{)}
\]
A domain file consists of

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

(define (domain BLOCKS)
  (:requirements :adl :typing)
  (:types block - object
    blueblock smallblock - block)
  (:predicates (on ?x - smallblock ?y - block)
    (ontable ?x - block)
    (clear ?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>)
effect:

<schematic-state-var>
(not <schematic-state-var>)
(and <effect> ... <effect>)
(when <formula> <effect>)
(forall (?x1 - type1 ... ?xn - typen) <effect>)
(:action fromtable
  :parameters (?x - smallblock ?y - block)
  :precondition (and (not (= ?x ?y))
                   (clear ?x)
                   (onetable ?x)
                   (clear ?y))
  :effect
   (and (not (onetable ?x))
        (not (clear ?y))
        (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 - smallblock)
    d e - block
    f - blueblock)
  (:init (clear a) (clear b) (clear c)
    (clear d) (clear e) (clear f)
    (ontable a) (ontable b) (ontable c)
    (ontable d) (ontable e) (ontable f))
  (:goal (and (on a d) (on b e) (on c f)))
)
Example run on the FF planner

```bash
# ./ff -o blocks-dom.pddl -f blocks-ex.pddl
ff: parsing domain file, domain 'BLOCKS' defined
ff: parsing problem file, problem 'EXAMPLE' defined
ff: found legal plan as follows
step 0: FROMTABLE A D
  1: FROMTABLE B E
  2: FROMTABLE C F
0.01 seconds total time
```
Example: blocks world in PDDL

(define (domain BLOCKS)
  (:requirements :adl :typing)
  (:types block)
  (:predicates (on ?x - block ?y - block)
    (on table ?x - block)
    (clear ?x - block)
  )
(:action fromtable
   :parameters (?x - block ?y - block)
   :precondition (and (not (= ?x ?y))
                   (clear ?x)
                   (ontable ?x)
                   (clear ?y))
   :effect (and (not (ontable ?x))
             (not (clear ?y))
             (on ?x ?y)))
(:action totable
  :parameters (?x - block ?y - block)
  :precondition (and (clear ?x) (on ?x ?y))
  :effect
    (and (not (on ?x ?y))
      (clear ?y)
      (ontable ?x)))
(:action move
  :parameters (?x - block
  ?y - block
  ?z - block)
  :precondition (and (clear ?x) (clear ?z)
  (on ?x ?y) (not (= ?x ?z)))
  :effect
  (and (not (clear ?z))
  (clear ?y)
  (not (on ?x ?y))
  (on ?x ?z)))
(define (problem blocks-10-0)
 (:domain BLOCKS)
 (:objects d a h g b j e i f c - block)
 (:init (clear c) (clear f)
  (ontable i) (ontable f)
  (on c e) (on e j) (on j b) (on b g)
  (on g h) (on h a) (on a d) (on d i))
 (:goal (and (on d c) (on c f) (on f j)
  (on j e) (on e h) (on h b)
  (on b a) (on a g) (on g i)))
)