

Principles of AI Planning

4. PDDL

Albert-Ludwigs-Universität Freiburg



Bernhard Nebel and Robert Mattmüller

October 29th, 2014

1 Schematic operators



Schematic operators
Schemata
PDDL

- Schematic operators

October 29th, 2014

B. Nebel, R. Mattmüller – AI Planning

3 / 26

Schematic operators



Schematic operators
Schemata
PDDL

- 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**.

October 29th, 2014

B. Nebel, R. Mattmüller – AI Planning

4 / 26

Schematic operators: example



Schematic operators
Schemata
PDDL

Schematic operator $drive_car_from_to(x, y_1, y_2)$:

$$x \in \{car1, car2\},$$

$$y_1 \in \{Freiburg, Strasbourg\},$$

$$y_2 \in \{Freiburg, Strasbourg\}$$

$$\langle in(x, y_1), in(x, y_2) \wedge \neg in(x, y_1) \rangle$$

corresponds to the operators

$$\langle in(car1, Freiburg), in(car1, Strasbourg) \wedge \neg in(car1, Freiburg) \rangle,$$

$$\langle in(car1, Strasbourg), in(car1, Freiburg) \wedge \neg in(car1, Strasbourg) \rangle,$$

$$\langle in(car2, Freiburg), in(car2, Strasbourg) \wedge \neg in(car2, Freiburg) \rangle,$$

$$\langle in(car2, Strasbourg), in(car2, Freiburg) \wedge \neg in(car2, Strasbourg) \rangle,$$

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

$$\langle in(car1, Freiburg), in(car1, Freiburg) \wedge \neg in(car1, Freiburg) \rangle.$$

October 29th, 2014

B. Nebel, R. Mattmüller – AI Planning

5 / 26

Existential quantification (for formulae only)

Finite disjunctions $\varphi(a_1) \vee \dots \vee \varphi(a_n)$ represented as $\exists x \in \{a_1, \dots, a_n\} : \varphi(x)$.

Universal quantification (for formulae and effects)

Finite conjunctions $\varphi(a_1) \wedge \dots \wedge \varphi(a_n)$ represented as $\forall x \in \{a_1, \dots, a_n\} : \varphi(x)$.

Example

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

- 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

```
(and (or (on A B) (on A C))  
      (or (on B A) (on B C))  
      (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)
               )
)
```

Schematic operators

PDDL
Overview
Domain files
Problem files
Example

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>)
```

Schematic operators

PDDL
Overview
Domain files
Problem files
Example

- effect:

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

Schematic operators

PDDL
Overview
Domain files
Problem files
Example

```
(: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)))
```

Schematic operators

PDDL
Overview
Domain files
Problem files
Example

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

The Fast Downward Planner

Fast Downward is the state-of-the-art planner, usable both for research and applications.

Main developers:

- Malte Helmert
- Gabi Röger
- Erez Karpas
- Jendrik Seipp
- Silvan Sievers
- Florian Pommerening

Example

The Fast Downward Planner

Fast Downward is available at
<http://www.fast-downward.org/>

Installation:

Follow instructions at
<http://www.fast-downward.org/ObtainingAndRunningFastDownward>

Running:

Follow instructions at
<http://www.fast-downward.org/PlannerUsage>

Example run of Fast Downward



```
# ./fast-downward.py --plan-file plan.txt \  
domain.pddl problem.pddl --search "astar(blind())"
```

```
[...]  
INFO      Running search.  
[...]  
Solution found!  
[...]  
Plan length: 6 step(s).  
[...]  
Expanded 85 state(s).  
[...]  
Search time: 0s  
[...]
```

Schematic operators

PDDL
Overview
Domain files
Problem files
Example

Example plan found by Fast Downward



```
# cat plan.txt  
(pick-up b)  
(stack b a)  
(pick-up c)  
(stack c b)  
(pick-up d)  
(stack d c)  
; cost = 6 (unit cost)
```

Schematic operators

PDDL
Overview
Domain files
Problem files
Example

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)  
               )  
)
```

Schematic operators

PDDL
Overview
Domain files
Problem files
Example

```
(: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)))
```

Schematic operators

PDDL
Overview
Domain files
Problem files
Example

```
(: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)))
)
```