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 \texttt{drive\_car\_from\_to}(x,y_1,y_2):

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

corresponds to the operators

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

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

\[
\langle \text{in}(\text{car1}, \text{Freiburg}), \text{in}(\text{car1}, \text{Freiburg}) \land \neg \text{in}(\text{car1}, \text{Freiburg}) \rangle.
\]
Schematic operators: quantification

Existential quantification (for formulae only)
Finite disjunctions \( \phi(a_1) \lor \cdots \lor \phi(a_n) \) represented as
\[ \exists x \in \{a_1, \ldots, a_n\} : \phi(x). \]

Universal quantification (for formulae and effects)
Finite conjunctions \( \phi(a_1) \land \cdots \land \phi(a_n) \) represented as
\[ \forall x \in \{a_1, \ldots, a_n\} : \phi(x). \]

Example
\[ \exists x \in \{A, B, C\} : \text{in}(x, \text{Freiburg}) \] is a short-hand for
\[ \text{in}(A, \text{Freiburg}) \lor \text{in}(B, \text{Freiburg}) \lor \text{in}(C, \text{Freiburg}). \]

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
  (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)
  - :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
  :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>)

  :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)))
PDDL: problem files

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)

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**

```bash
# ./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
 [...] 
```

**Example plan found by Fast Downward**

```bash
# 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)
```

**PDDL Editor**

...in the cloud

In case you are looking for a decent PDDL editor:

- Check out the PDDL editor in the cloud:
  [http://editor.planning.domains/](http://editor.planning.domains/)
- The website also includes a built-in solver:
  [http://solver.planning.domains/](http://solver.planning.domains/)
- ...and an API + domain repository:
  [http://api.planning.domains/](http://api.planning.domains/)

**Example: blocks world in PDDL**

```pddl
(define (domain BLOCKS)
  (:requirements :strips :typing)
  (:types 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))))