SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Planning II

PROF LIM KWAN HUI

50.021 Artificial Intelligence

The following notes are compiled from various sources such as textbooks, lecture materials, Web resources and are shared for academic purposes only, intended for use by students registered for a specific course. In the interest of brevity, every source is not cited. The compiler of these notes gratefully acknowledges all such sources.

Planning Domain Definition Language (PDDL)

- A language to describe planning problems that can be used as input to planner software.
  - Used by planning researchers as a standard language for defining classical planning problems

- PDDL and STRIPS
  - PDDL includes STRIPS as special case along with more advanced features
  - PDDL includes some simple additional features, such as type specification for objects, negated preconditions, conditional add/del effects
  - Some more advanced features include allowing numeric variables and durative actions

Components of a PDDL task

- Objects: Things in the world that interest us.
- Predicates: Properties of objects that we are interested in; can be true or false
- Initial state: The state of the world that we start in.
- Goal specification: Things that we want to be true.
- Actions/Operators: Ways of changing the state of the world.

PDDL Task Specifications

- Planning tasks specified in PDDL are separated into two files:
  - A problem file for objects, initial state and goal specification.
  - A domain file for predicates (= facts) and actions.

- We will use Malte Helmert's slides for specific examples

An Introduction to PDDL

What is PDDL?

PDDL = Planning Domain Definition Language

~ standard encoding language for “classical” planning tasks

Components of a PDDL planning task:

- **Objects:** Things in the world that interest us.
- **Predicates:** Properties of objects that we are interested in; can be *true* or *false*.
- **Initial state:** The state of the world that we start in.
- **Goal specification:** Things that we want to be true.
- **Actions/Operators:** Ways of changing the state of the world.

How to Put the Pieces Together

Planning tasks specified in PDDL are separated into two files:

1. A **domain file** for predicates and actions.
2. A **problem file** for objects, initial state and goal specification.

Domain Files

Domain files look like this:

```
(define (domain <domain name>)
    <PDDL code for predicates>
    <PDDL code for first action>
    [...]
    <PDDL code for last action>
)
```

`<domain name>` is a string that identifies the planning domain, e.g. `gripper`.

Example on the web: `gripper.pddl`.

Problem Files

Problem files look like this:

```
(define (problem <problem name>)
   (:domain <domain name>)
   <PDDL code for objects>
   <PDDL code for initial state>
   <PDDL code for goal specification>
)
```

`<problem name>` is a string that identifies the planning task, e.g. `gripper-four-balls`.

`<domain name>` must match the domain name in the corresponding domain file.

Example on the web: `gripper-four.pddl`.

Running Example

Gripper task with four balls:

There is a robot that can move between two rooms and pick up or drop balls with either of his two arms. Initially, all balls and the robot are in the first room. We want the balls to be in the second room.

- **Objects:** The two rooms, four balls and two robot arms.
- **Predicates:** Is x a room? Is x a ball? Is ball x inside room y? Is robot arm x empty? [...]
- **Initial state:** All balls and the robot are in the first room. All robot arms are empty. [...]
- **Goal specification:** All balls must be in the second room.
- **Actions/Operators:** The robot can move between rooms, pick up a ball or drop a ball.

Gripper task: Objects

Objects:
Rooms: `rooma`, `roomb`
Balls: `ball1`, `ball2`, `ball3`, `ball4`
Robot arms: `left`, `right`

In PDDL:

```
(:objects rooma roomb
          ball1 ball2 ball3 ball4
          left right)
```


Gripper task: Predicates

Predicates:

`ROOM(`x`)`	– true iff x is a room
`BALL(`x`)`	– true iff x is a ball
`GRIPPER(`x`)`	– true iff x is a gripper (robot arm)
`at-robby(`x`)`	– true iff x is a room and the robot is in x
`at-ball(`x`, `y`)`	– true iff x is a ball, y is a room, and x is in y
`free(`x`)`	– true iff x is a gripper and x does not hold a ball
`carry(`x`, `y`)`	– true iff x is a gripper, y is a ball, and x holds y

In PDDL:

```
(:predicates (ROOM ?x) (BALL ?x) (GRIPPER ?x)
             (at-robby ?x) (at-ball ?x ?y)
             (free ?x) (carry ?x ?y))
```


Gripper task: Initial state

Initial state:

`ROOM(rooma)` and `ROOM(roomb)` are true.
`BALL(ball1)`, ..., `BALL(ball4)` are true.
`GRIPPER(left)`, `GRIPPER(right)`, `free(left)` and `free(right)` are true.
`at-robby(rooma)`, `at-ball(ball1, rooma)`, ..., `at-ball(ball4, rooma)` are true. Everything else is false.

In PDDL:

```
(:init (ROOM rooma) (ROOM roomb)
       (BALL ball1) (BALL ball2) (BALL ball3) (BALL ball4)
       (GRIPPER left) (GRIPPER right)   (free left) (free right)
       (at-robby rooma)
       (at-ball ball1 rooma) (at-ball   ball2 rooma)
       (at-ball ball3 rooma) (at-ball   ball4 rooma))
```


Gripper task: Goal specification

Goal specification:

`at-ball(ball1, roomb)`, ..., `at-ball(ball4, roomb)` must be true.
Everything else we don't care about.

In PDDL:

```
(:goal (and (at-ball ball1 roomb)
            (at-ball ball2 roomb)
            (at-ball ball3 roomb)
            (at-ball ball4 roomb)))
```


Gripper task: Movement operator

Action/Operator:

Description: The robot can move from x to y.
Precondition: `ROOM(`x`)`, `ROOM(`y`)` and `at-robby(`x`)` are true.
Effect: `at-robby(`y`)` becomes true. `at-robby(`x`)` becomes false. Everything else doesn't change.

In PDDL:

```
(:action move :parameters (?x ?y)
   :precondition (and (ROOM ?x) (ROOM ?y)
                      (at-robby ?x))
   :effect       (and (at-robby ?y)
                      (not (at-robby ?x))))
```


Gripper task: Pick-up operator

Action/Operator:

Description: The robot can pick up x in y with z.

Precondition: `BALL(`x`)`, `ROOM(`y`)`, `GRIPPER(`z`)`, `at-ball(`x`, `y`)`, `at-robby(`y`)` and `free(`z`)` are true.

Effect: `carry(`z`, `x`)` becomes true. `at-ball(`x`, `y`)` and `free(`z`)` become false. Everything else doesn't change.

In PDDL:

```
(:action pick-up :parameters (?x ?y ?z)
   :precondition (and (BALL ?x) (ROOM ?y) (GRIPPER ?z)
                      (at-ball ?x ?y) (at-robby ?y) (free ?z))
   :effect       (and (carry ?z ?x)
                      (not (at-ball ?x ?y)) (not (free ?z))))
```


Gripper task: Drop operator

Action/Operator:

Description: The robot can drop x in y from z.

(Preconditions and effects similar to the pick-up operator.)

In PDDL:

```
(:action drop :parameters (?x ?y ?z)
   :precondition (and (BALL ?x) (ROOM ?y) (GRIPPER ?z)
                      (carry ?z ?x) (at-robby ?y))
   :effect       (and (at-ball ?x ?y) (free ?z)
                      (not (carry ?z ?x))))
```


A Note on Action Effects

Action effects can be more complicated than seen so far.

They can be **universally quantified**:

```
(forall (?v1 ... ?vn)
        <effect>)
```

They can be **conditional**:

```
(when <condition>
      <effect>)
```

Example on the web: `crazy-switches.pddl`

Questions?