PDDL
(Planning Domain Definition Language)

Jan Hrnčíř

A4M33PAH Tutorial

Introduction

- Standard encoding language for classical planning tasks
  - International Planning Competition (IPC)
  - Based on STRIPS notation
- Several versions with increasing set of features
  - **1.2** … basic version
  - **2.1** … functions, metric
  - 3.0 … preferences, soft constraints
- Resources
  - Writing Planning Domains and Problems in PDDL

Components of planning task

- **Objects:** Things in the world that interest us.
- **Predicates:** Properties of objects that we are interested in; can be true or false.
- **Functions:** Fluents that return a number.
- **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.

Gripper task

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. The rooms are connected with a corridor of a specified length.

Domain & problem file

- Planning tasks specified in PDDL are separated into two files:

  Domain file: for types, predicates, functions and actions

  Problem file: for objects, initial state and goal specification.

Domain file

- Domain files look like this:

```
(define (domain <domain name>)
     (:requirements :typing :action-costs)
     (:types <list of types>)
     <PDDL code for predicates>
     <PDDL code for functions>
     <PDDL code for first action>
     [...]
     <PDDL code for last action>
)
```

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

Types

- **Types:** room, ball, arm

In PDDL domain file:

```
(:types room ball arm)
```

Constants

- Constants are objects that can be used in the domain file: `(:constants left right – arm)`

Predicates

- **Predicates:**
  - at-robby(x) – true iff the robot is in room x
  - at (x, y) – true iff ball x is in room y
  - free(x) – true iff arm x does not hold a ball
  - carry(x, y) – true iff arm y holds ball x

In PDDL domain file:

```
(:predicates
     (at-robby ?x - room)
     (at ?x - ball ?y - room)
     (free ?x - arm)
     (carry ?x – ball ?y - arm)
)
```

Functions

- Functions
  - length(x, y) … length of the corridor between the two rooms
  - total-cost() … total cost of a plan

In PDDL domain file:

```
(:functions
    (length ?rooma - room ?roomb - room)
    (total-cost)
)
```

Actions

- **Description:** The robot can move from x to y.
- **Precondition:** at-robby(x) is true.
- **Effect:** at-robby(y) becomes true. at-robby(x) becomes false. Everything else doesn't change.
- **Cost:** The cost of the action is the length of the corridor between the two rooms

In PDDL domain file:

```
(:action move
    :parameters (?x ?y - room)
    :precondition (at-robby ?x)
    :effect (and (at-robby ?y)
            (not (at-robby ?x))
            (increase (total-cost) (length ?x ?y))
))
```

Problem file

- 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>
      <PDDL code for metric>
)
```

- `<problem name>` is a string that identifies the planning task, e.g., `gripper4`.
- `<domain name>` must match the domain name in the corresponding domain file.

Objects

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

In PDDL problem file:

```
(:objects
    rooma roomb - room
    ball1 ball2 ball3 ball4 - ball
    left right - arm
)
```

Initial state

Initial state: Initially, all balls and the robot are in the first room.

In PDDL problem file:

```
(:init
    (at-robby rooma)
    (at ball1 rooma) (at ball2 rooma)
    (at ball3 rooma) (at ball4 rooma)

    (= (length rooma roomb) 30 )
    (= (length roomb rooma) 30 )
    (= (total-cost) 0 )
)
```

Goal state

Goal state: We want the balls to be in the second room.

In PDDL problem file:

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

Metric

Metric: We want the planner to minimize the cost of the plan.

In PDDL problem file:

```
(:metric minimize (total-cost))
```

How to run a planner

- **Practical issues**
  - PDDL expressivity vs. planner capabilities → keep PDDL simple as possible
    - Use only positive preconditions
    - Various settings via `(requirements: … )`
  - Compilation of the planner (transfer of binary code)
  - Planner is a blackbox

- **Validation of PDDL files**

```
./validate -v test-domain.pddl test-problem.pddl
```

- **Planner execution**

```
./plan test-domain.pddl test-problem.pddl test-solution.txt
```