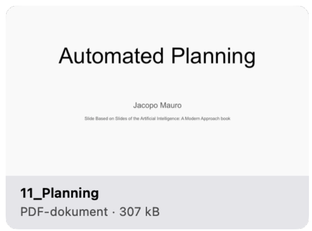
AUTOMATED PLANNING

Read: 11.1, 11.2, 11.4



11_Planning
PDF-dokument · 307 kB

Classical Planning Definition

Classical planning is defined as the task of finding a sequence of actions to accomplish a goal in a discrete, deterministic, static, fully observable environment.

Planning Domain Definition Language (PDDL) is the de-facto language

- Basic PDDL can handle classical planning domains
- Extensions can handle non-classical domains that are continuous, partially observable, concurrent, and multi-agent.

State: represented as a conjunction of ground atomic fluents

Closed-world assumption: any fluent that are not mentioned is false

Algorithms for Classical Planning

Forward state-space search for planning

- Start at initial state
- Compute the set of actions in which preconditions are satisfied
- For all actions compute the next state applying effects
- Repeat until goal is found

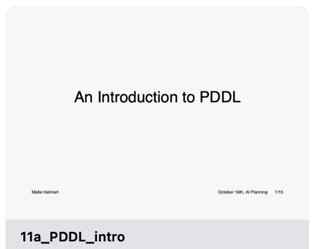
Backward search also possible

Encoding into SAT

Graph plan uses a specialized data structure, a planning graph

- Backward search with smaller search space

PDDL



11a_PDDL_intro
PDF-dokument · 29 kB

Syntax: Gripper example

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

In PDDL:

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

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))
```

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))
```

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)))
```

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))))
```

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))))
```

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))))
```

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 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`.