Artificial Intelligence Foundation – JC3001

Lecture 26: Introduction to Automated Planning - II

Prof. Aladdin Ayesh (aladdin.ayesh@abdn.ac.uk)

Dr. Binod Bhattarai (binod.bhattarai@abdn.ac.uk)

Dr. Gideon Ogunniye, (g.ogunniye@abdn.ac.uk)

October 2025

Material adapted from:
Russell and Norvig (AIMA Book): Chapter 11 (11.1)
Ghallab, Nau, and Traverso (Automated Planning: Theory and Practice)

UNIVERSITY OF ABERDEEN

Course Progression

- Part 1: Introduction
  1. Introduction to AI ✓
  2. Agents ✓
- Part 2: Problem-solving
  1. Search 1: Uninformed Search ✓
  2. Search 2: Heuristic Search ✓
  3. Search 4: Local Search ✓
  4. Search 4: Adversarial Search ✓
- Part 3: Reasoning and Uncertainty
  1. Reasoning 1: Constraint Satisfaction ✓
  2. Reasoning 2: Logic and Inference ✓
  3. Probabilistic Reasoning 1: BNs ✓
  4. Probabilistic Reasoning 2: HMMs ✓
- Part 4: Planning
  1. **Planning 1: Intro and Formalism**
  2. Planning 2: Algos and Heuristics
  3. Planning 3: Hierarchical Planning
  4. Planning 4: Stochastic Planning
- Part 5: Learning
  1. Learning 1: Intro to ML
  2. Learning 2: Regression
  3. Learning 3: Neural Networks
  4. Learning 4: Reinforcement Learning
- Part 6: Conclusion
  1. Ethical Issues in AI
  2. Conclusions and Discussion

UNIVERSITY OF ABERDEEN

Objectives

- Search versus Planning ✓
- Planning Problem Representation
  - STRIPS ✓
  - PDDL
- Forward and Backward Search

Action Description Language (ADL) is a richer language than STRIPS.
It allows for

- Positive and negative literals in states
- The open world assumption
- Quantified variables in goals as well as conjunctions and disjunctions
- Conditional effects
- (in)Equality predicates
- Types associated with variables

- Planning Domain Definition Language (PDDL)
  is a 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.

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.

```
(define (domain <domain name>)
  <PDDL code for constants>
  <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`.

```
(define (domain gripper)
(:requirements :strips)
(:predicates (room ?r) (ball ?b) (gripper ?g) (at-robby ?r) (at ?b ?r)
             (free ?g) (carry ?o ?g))
 (:action move
    :parameters (?from ?to)
    :precondition (and  (room ?from) (room ?to) (at-robby ?from))
    :effect (and  (at-robby ?to) (not (at-robby ?from))))
 (:action pick
    :parameters (?obj ?room ?gripper)
    :precondition  (and  (ball ?obj) (room ?room) (gripper ?gripper)
                   (at ?obj ?room) (at-robby ?room) (free ?gripper))
    :effect (and (carry ?obj ?gripper) (not (at ?obj ?room))
              (not (free ?gripper))))
 (:action drop
     :parameters  (?obj  ?room ?gripper)
     :precondition  (and  (ball ?obj) (room ?room) (gripper ?gripper)
                          (carry ?obj ?gripper) (at-robby ?room))
    :effect (and (at ?obj ?room) (free ?gripper)
```

```
(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-one.pddl.

```
(define (problem gripper-one)
   (:domain gripper)
   (:requirements :strips)
   (:objects roomA roomB Ball1 Ball2 left right)
   (:init (room roomA) (room roomB) (ball Ball1) (ball Ball2)
          (gripper left) (gripper right) (at-robby roomA)
          (free left) (free right) (at Ball1 roomA) (at Ball2 roomA)
  )
    (:goal (and (at Ball1 roomB) (at Ball2 roomB)))
)
```

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:	at-robby(y) becomes true. at-robby(x) becomes 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 with 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:

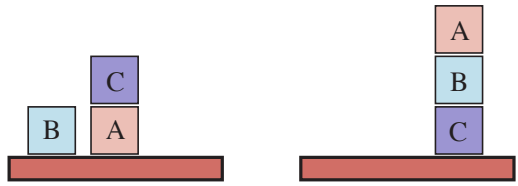
```
(forall (?v1 ...  ?vn)
        <effect>)
```

- They can be conditional:

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

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

- Formalise the actions of the Blocks World Domain:
- Predicates:
  - on(A,B) - A is **on** B
  - block(B) - B is a **block**
  - clear(B) - B is **clear**
- Objects: Blocks **a**, **b** and **c**; **table** (always clear)



Blocks World (in STRIPS)

$Init(On(A, Table) \land On(B, Table) \land On(C, A)$
$\quad \land Block(A) \land Block(B) \land Block(C) \land Clear(B) \land Clear(C))$
$Goal(On(A, B) \land On(B, C))$
$Action(Move(b, x, y),$
$\quad$ PRECOND: $On(b, x) \land Clear(b) \land Clear(y) \land Block(b) \land Block(y) \land$
$\qquad (b \neq x) \land (b \neq y) \land (x \neq y),$
$\quad$ EFFECT: $On(b, y) \land Clear(x) \land \neg On(b, x) \land \neg Clear(y))$
$Action(MoveToTable(b, x),$
$\quad$ PRECOND: $On(b, x) \land Clear(b) \land Block(b) \land (b \neq x),$
$\quad$ EFFECT: $On(b, Table) \land Clear(x) \land \neg On(b, x))$

Blocks World in PDDL Domain


```
(define (domain blocks)
   (:requirements :strips)
   (:constants-def table)
   (:predicates (on ?a ?b) (block ?b) (clear ?b) )
   (:action move
      :parameters (?b ?x ?y)
      :precondition (and (on ?b ?x) (clear ?y) (clear ?b)
                         (block ?b) (block ?y))
      :effect (and (on ?b ?y) (clear ?x)
                   (not (on ?b ?x)) (not (clear ?y)))
   )
   (:action moveToTable
      :parameters (?b ?x)
      :precondition (and (on ?b ?x) (clear ?b) (block ?b) (clear ?x))
      :effect (and (on ?b table) (clear ?x) (not (on ?b ?x)))
   )
)
```

Blocks World in PDDL Problem Definition


```
(define (problem pb1)
   (:domain blocks)
   (:objects a b c table)

   (:init (on a table) (on b table) (on c a)
          (block a) (block b) (block c) (clear b) (clear c))
   (:goal (and (on a b) (on b c)))
)
```

To continue in the next session.