Artificial Intelligence

Planning

E. Giunchiglia, F. Leofante, A. Tacchella

Computer Engineering
Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi

Last update: April 4, 2024

Agenda - Planning

1. Why and What is planning?
2. Representing classical domain independent planning problems
   - STRIPS
   - First order extension of STRIPS
   - PDDL
3. Beyond Classical domain independent planning

Agenda - Planning

1. Why and What is planning?
2. Representing classical domain independent planning problems
   - STRIPS
   - First order extension of STRIPS
   - PDDL
3. Beyond Classical domain independent planning

Why?

Alice and The Cheshire Cat debating on the relevance of planning

non ha sensa senso parlarer di pianificazione senza un obbiettivo

A:	Would you tell me, please, which way I ought to go from here?
TCC:	That depends a good deal on where you want to get to.
A:	I don’t much care where.
TCC:	Then it doesn’t much matter which way you go.

Why?

“A goal without a plan is just a wish”

Antoine de Saint-Exupéry

... also in “50 Ways to Lose Ten Pounds" (1995) by Joan Horbiak, p. 95

What is planning anyway?

Planning is . . .

the art and practice of thinking before acting

What is planning anyway?

Planning is . . .

the art and practice of thinking before acting

That is, given a model that represents

- the set of possible **initial** states I,
- the set of **actions** $\mathcal{A}$ that can be performed and their effects,
- the **goal** G.

what are the actions in A that I should perform to reach G from I?

Planning problem

Abstractly, once defined what is a state, a *planning problem* is a triple $\langle I, \mathcal{A}, G \rangle$, and to define it I have to specify

1. what is a state in general, and the possible initial states in particular,
2. when an action is executable in a state (the preconditions), and what are the effects of executing the action, i.e., the set of possible resulting states, and
3. the set of goal states,
4. what is a plan and what are the valid plans.

che tipo di piano voglio ci sono diversi tipi di soluzioni e di azioni

Solving a planning problem amounts to finding a valid plan.

con un numero finito di variabili e di if hai un numero finito di stati, invece se ci sono loop ci son un numero infinito di possibili stati

dato uno stato e una azione il risultato sono possibili stati futuri SxA -> 2^S

pero non ogni azione è possibile eseguirla in ogni stato

ci sono anche azioni condizionate con if

Planning problem

proprieta delle azioni, quale è l'effetto di eseguire un azione

Depending on the properties of

1. the set of admissible states,
2. the initial state I, set of actions $\mathcal{A}$ and goal states G,
3. the set of admissible solutions/plans

I have different types of planning, like:

- task, path/motion, . . . planning
- conformant, conditional, universal, . . . planning
- deterministic vs non deterministic planning
- single vs multi-agent planning
- . . .

...we will focus on **classical domain-independent** planning!

Agenda - Planning

1 Why and What is planning?

2 Representing classical domain independent planning problems
- STRIPS
- First order extension of STRIPS
- PDDL

3 Beyond Classical domain independent planning

Assumptions

il fatto che siano booleane non è una grossa restrizione in quanto qualsiasi var è rappresentabile con var boleane

In classical planning, the following assumptions are made:

- **Finite domain:** a state of the world is completely described by finitely many variables assumed to be Boolean
- **Atomic time:** actions have no durations, their execution is instantaneous
- **Single agent:** there is a single agent who can execute a single action at each time

Assumptions (cont.d)

- **Determinism:** *single* initial state and the effect of executing any action is a *single* state, which is a function of the action and the state of the world when the action is executed
- **Omniscience:** complete knowledge of the initial states as well as of the consequences of executing the actions
- **Sole cause of change:** the actions are the sole cause of change

Agenda - Planning

1 Why and What is planning?

2 Representing classical domain independent planning problems
- STRIPS
- First order extension of STRIPS
- PDDL

3 Beyond Classical domain independent planning

STRIPS

language to model this scenario

STanford Research Institute Problem Solver is the first system for automated planning (1971)

A *state* is a finite set of propositional variables (the variables that are true in the state, the others assumed to be false). S:{P}

The *initial* state I is a single state, represented as a standard state. I:{Q}

The *goal* G is again represented as a set of variables, this time denoting the set of states S such that $G \subseteq S$. G:{Q,R}

STRIPS

An *action* A is a triple $\langle P, E, D \rangle$ such that $E \cap D = \emptyset$, where

1. P is the set of preconditions (variables that must be true before the execution of A),
2. E is the set of positive effects (variables that are true after the execution of A),
3. D is the set of negative effects (variables that are false after the execution of A).

STRIPS

Given an action $A = \langle P, E, D \rangle$ and a single state S,

1. A is *executable* in S iff $P \subseteq S$, and
2. if A is executable in S then the state $Res(A, S)$ *resulting from the execution of A in S* is

$$Res(A, S) = S \setminus D \cup E = S \cup E \setminus D$$

Example

Given the action

$$Go\_Home\_Office = \langle \{At\_Home\}, \{At\_Office\}, \{At\_Home\} \rangle,$$

and the state $S = \{At\_Home, Have\_Banana\}$,

$$Res(Go\_Home\_Office, S) = \{At\_Office, Have\_Banana\}.$$

STRIPS

l'obbiettivo è trovare una sequenza dia azioni che porta dallo stato iniziale fino allo stato desiderato

A *plan* is a sequence of actions $A_1; \ldots; A_n$ ($n \geq 0$).

Given a plan $\Pi = A_1; \ldots; A_n$ and a state S, the state resulting from execution Π in S is

$$Res(A_1; \ldots; A_n, S) = Res(A_n, Res(A_1; \ldots; A_{n-1}, S)).$$

Given an initial state I and a goal state G, a plan $A_1; \ldots; A_n$ is *valid* iff

$$G \subseteq Res(A_1; \ldots; A_n, I),$$

goal condition satisfy in the p

i.e., if, starting in the initial state I and then executing the action A_1, then A_2, ..., then A_n, we end up in a state denoted by G.

STRIPS

in sostanza rappresenti tutti i vari stati con tutti i le varie azioni: un grafo ,pero il numero di nodi aumenta esponenzialmente con il numero di variabili

in ogni nodo (possibile stadio raggiungibile) è collegato in uscita con n edges dove n è il numero di azioni che soddisfano i prerequisiti in quello stato

Note: The previous definition assumes that each A_i is executable in $Res(A_1; \ldots; A_{i-1}, S)$. If the assumption is not satisfied, $Res(A_1; \ldots; A_n, S)$ is undefined.

Note: Determining the existence of a valid plan is a PSPACE-complete problem.

STRIPS

Example

Given the planning problem with

1. actions

P E D

$$Go\_Home\_Office = \langle \{At\_Home\}, \{At\_Office\}, \{At\_Home\} \rangle,$$

$$Buy\_Banana = \langle \{\}, \{Have\_Banana\}, \{\} \rangle,$$

2. initial state $I = \{At\_Home\}$, and
3. goal state $G = \{At\_Office, Have\_Banana\}$,

$$Res(Go\_Home\_Office, Res(Buy\_Banana, I)) = Res(Buy\_Banana, Res(Go\_Home\_Office, I)) = \{At\_Office, Have\_Banana\}.$$

Go_Home_Office; *Buy_Banana* and *Buy_Banana*; *Go_Home_Office* are valid plans.

Agenda - Planning

1. Why and What is planning?
2. Representing classical domain independent planning problems
   - STRIPS
   - First order extension of STRIPS
   - PDDL
3. Beyond Classical domain independent planning

A first-order language for planning

To concisely describe planning problems, it is common to use a first-order language with:

- finitely many predicate symbols;
- finitely many constant symbols;
- no function symbols.

and with the intended interpretation $\langle D, g\rangle$ in which D is the set of constants and for each constant c, $g(c) = c$.

Each ground atom corresponds to a propositional variable and is called *fluent*.

States and initial state

State

A *state* is an interpretation of the set of fluents. States can be represented in many ways:

1. as the set of fluents that are true, all the other assumed to be false,
2. as the set of fluents and negated fluents that are true,
3. as the conjunction of the set of fluents and negated fluents that are true,
4. ...

Initial state

Because of the assumption of determinism, there is a single initial state.

States (cont.d)

Example

Assume we have the unary predicate *At* and the constants *Home* and *Office*.

1. $\{At(Home)\}$,
2. $\{At(Home), \neg At(Office)\}$,
3. $(At(Home) \wedge \neg At(Office))$,

are the three representations corresponding to the state in which $At(Home)$ is true and $At(Office)$ is false.

Goal states

In general, in a planning problem there can be more than one goal states. In the literature, various representations are used:

1. as the set of fluents that must be true, all the others we do not care,
2. as the set of fluents and negated fluents that must be true, all the others we do not care,
3. as a closed formula.
4. …

Note: Representing goal states with a formula gives more expressivity since it allows, e.g., to write $(At(Home) \vee At(Office))$.

Note: Given the intended interpretation, quantifiers can be eliminated.

Operators and actions

Operators are action descriptions consisting of

- name: a unique expression that defines all variables appearing in the operator

  e.g. $Buy(p, x)$

- preconditions: a condition, in general expressed as a formula, that must be true for the operator to be executable in the state

  e.g. $(At(p) \wedge Sells(p, x) \wedge Location(p) \wedge Object(x))$

- effects: in general, a conjunction of function-free literals describing how the state changes when the operator is applied,

  e.g. $(Have(x) \wedge \neg Sells(p, x))$

Execution of an action

The set of actions is obtained by grounding the operators in all possible ways.

Given an action A and a state S

1. A is *executable* in S iff its preconditions are satified by S;
2. if A is executable in S then the state $Res(A, S)$ *resulting from the execution of A in S* is the interpretation $Res(A, S)$ satisfying the effects of A and equal to S when the fluent is not mentioned in the effects of A.

Plan

A plan is a sequence of actions.

Given a plan $\Pi = A_1; \ldots; A_n$ and a state S, the state resulting from execution Π in S is

$$Res(A_1; \ldots; A_n, S) = Res(A_n, Res(A_1; \ldots; A_{n-1}, S)).$$

Given an initial state I and a goal state G, a plan $A_1; \ldots; A_n$ is *valid* iff G is satisfied by $Res(A_1; \ldots; A_n, I)$.

Agenda - Planning

1 Why and What is planning?

2 Representing classical domain independent planning problems
- STRIPS
- First order extension of STRIPS
- PDDL

3 Beyond Classical domain independent planning

An introduction to PDDL

PDDL is a commonly used language for specifying planning problems.

In this part we will switch to slides by Malte Helmert.

What is PDDL?

PDDL = Planning Domain Definition Language

$\rightsquigarrow$ 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?

Agenda - Planning

1 Why and What is planning?

2 Representing classical domain independent planning problems
- STRIPS
- First order extension of STRIPS
- PDDL

3 Beyond Classical domain independent planning

Beyond classical planning

We made several assumptions: finite domain, atomic time, single agent, determinism, omniscience, sole cause of change.

If we relax some assumptions, we get:

- planning in infinite domains,
- temporal planning,
- multi-agent planning,
- planning in nondeterministic domains (conformant, contingent, universal planning),
- planning in unknown environment
- planning with exogenous events

The complexity of planning in such assumptions can become higher than PSPACE-complete, up to become undecidable.