Intro to Planning

Autonomous Software Agents

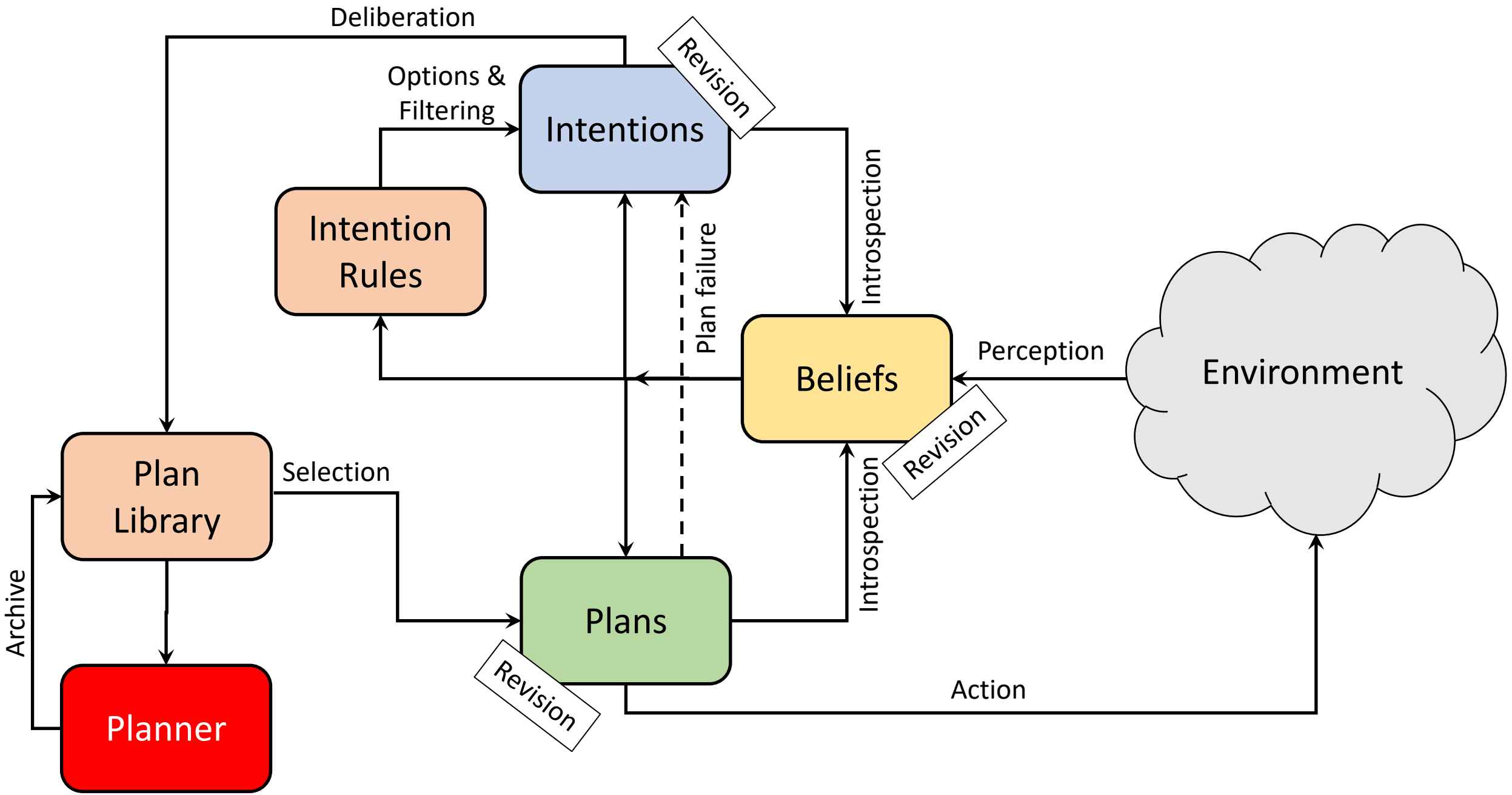
A.A. 2022-2023

Prof. Paolo Giorgini

Dr. Marco Robol

UNIVERSITY OF TRENTO - Italy

Department of Information
and Communication Technology


Deliberation
Options &
Filtering
Intentions
Revision
Intention
Rules
Plan failure
Introspection
Beliefs
Perception
Environment
Revision
Plan
Library
Selection
Introspection
Archive
Plans
Revision
Action
Planner

What is Planning ?

Planning as deliberation

An intelligent agent needs planning to decide best actions to do

- Programming-based agent - writing the agent control by hand
- Learning-based agent - infer the control by experience (Reinforcement Learning)
- Model-based approach - derive the control automatically by reasoning

These approaches are not orthogonal!

What is Model-Based Planning?

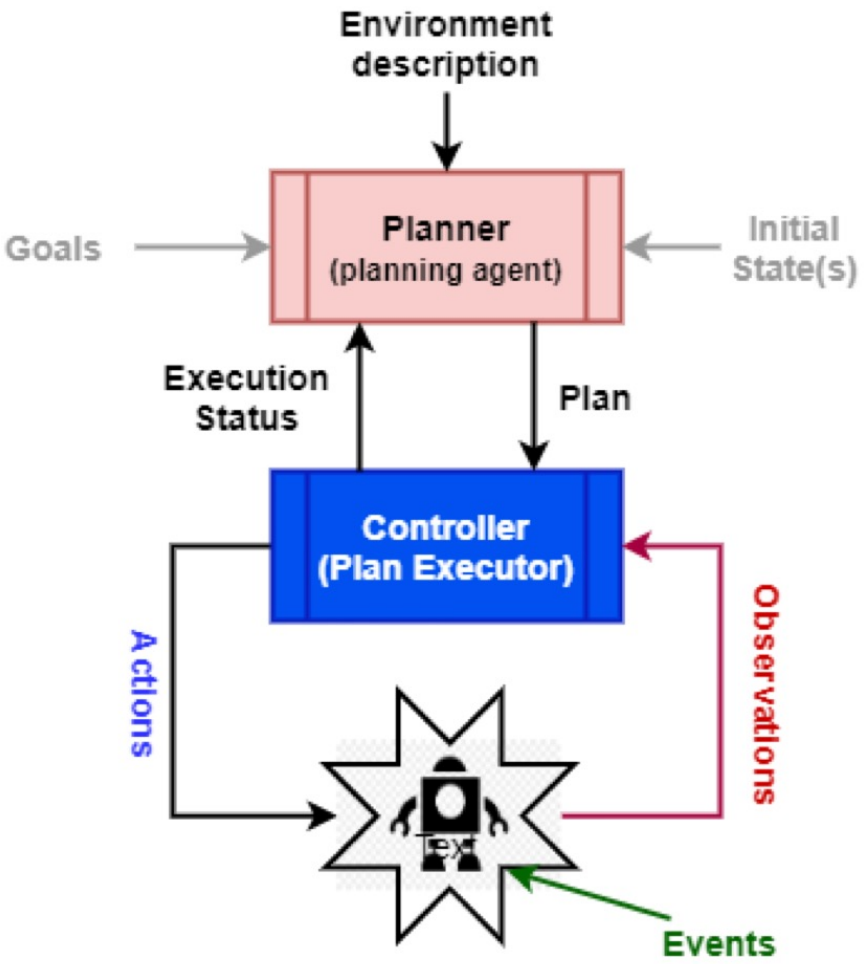
“Planning is the reasoning side of acting. It is an explicit deliberation process that chooses and organises actions, on the basis of their expected outcomes, in order to achieve some objective as best as possible.”

“...planning is the model based approach to action selection...”

“AI Planning is general problem solving”

What is Model-Based Planning?

- In AI Planning the agent has access to world dynamic
  - actions describe how the world changes
  - sensor-model describe how to update the knowledge of the world
  - goals denote what the agent wants
  - states capture evolving relevant conditions of the world the agent is operating in
- A Planner is a domain independent program that can solve (find a plan for) all the planning problems starting from
  - environment description
    - including action capabilities
  - problem description
    - initial state(s) and goals



AI Planning Problem

Given:

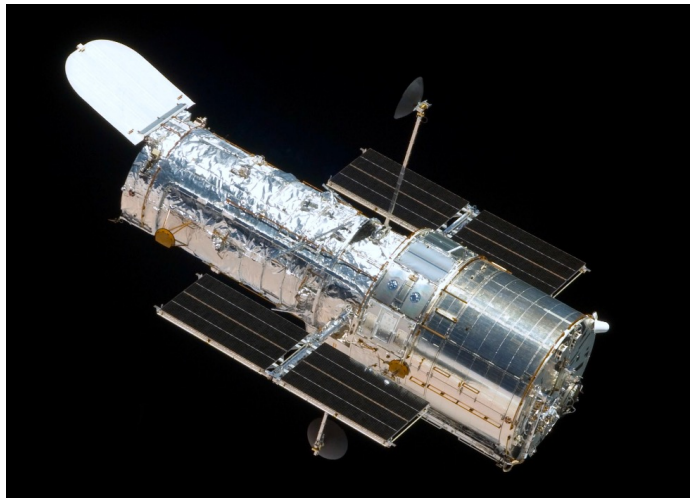
- A description of (possible) initial state(s)
- A description of desired goal states
- A description of a set of possible actions

Generate:

- A sequence of actions that leads to one of the goal states

Where do we use planning?

- Space Missions
- Hubble Telescope
- Logistics
- Games



Why is planning so difficult?

- Uncertainty about effects of actions
- Uncertainty about environment
- Uncertain sensing and perception
- Agents own actions may have bad effects
- Time and resources are limited!
- Preferences
- Complex goals
- Optimality

Let’s simplify things a bit...

- Known initial state
- Deterministic actions
- Simple action representation
- Instantaneous actions
- No deadlines and sufficient resources
- Fully observable
- Single agent /No external events
- No concurrent actions

We introduce constraints and simplification to focus on the reasoning behind planning problems.

Planning Problem

Given

- A description of (possible) initial state(s)
- A description of desired goals
- A description of a set of possible actions

Generate:

- A sequence of actions that leads to one of the goal states.

We can go back to the definition of planning problems and start reasoning about concepts.

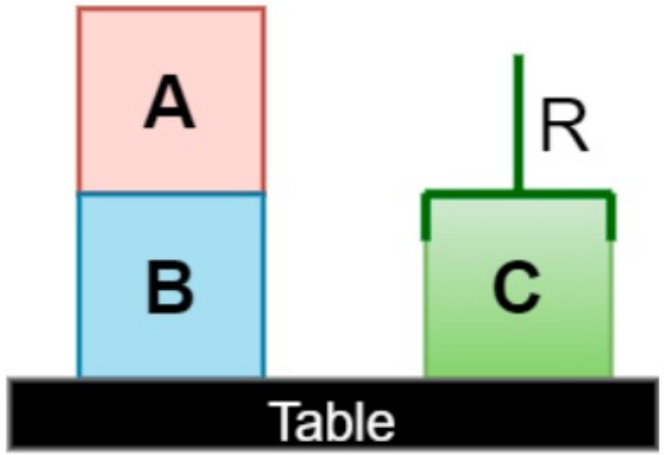
Planning concepts

Speaking about planning problems we deal with:

- State
- Action
- Goal
- Plan

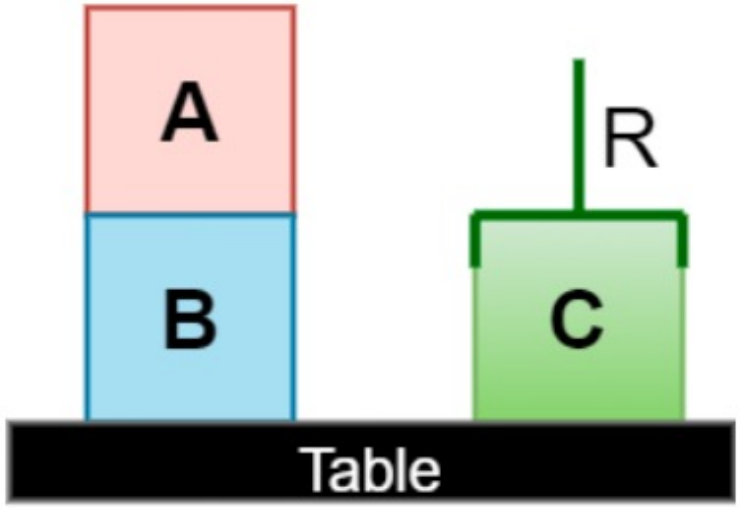
Example: The Blocks World

- Objects
  - Blocks: `A`, `B`, `C`
  - Table: `Table`
  - Robot: `R`
- States: Conjunctions of ground literals built from *predicates*
  - `(On A B)`, `(On B Table)`, `(On C Table)`, `(Clear A)`, `(Holding R C)`, `(Free R)`
- Actions: Operator schemas with variables
  - `(Pickup ?R ?A)`, `(Putdown ?R ?B)`
- Domain Axioms:
  - 'At most one block on top of another'
  - 'Hand must be empty and block must be clear to pick up'



What is a state?

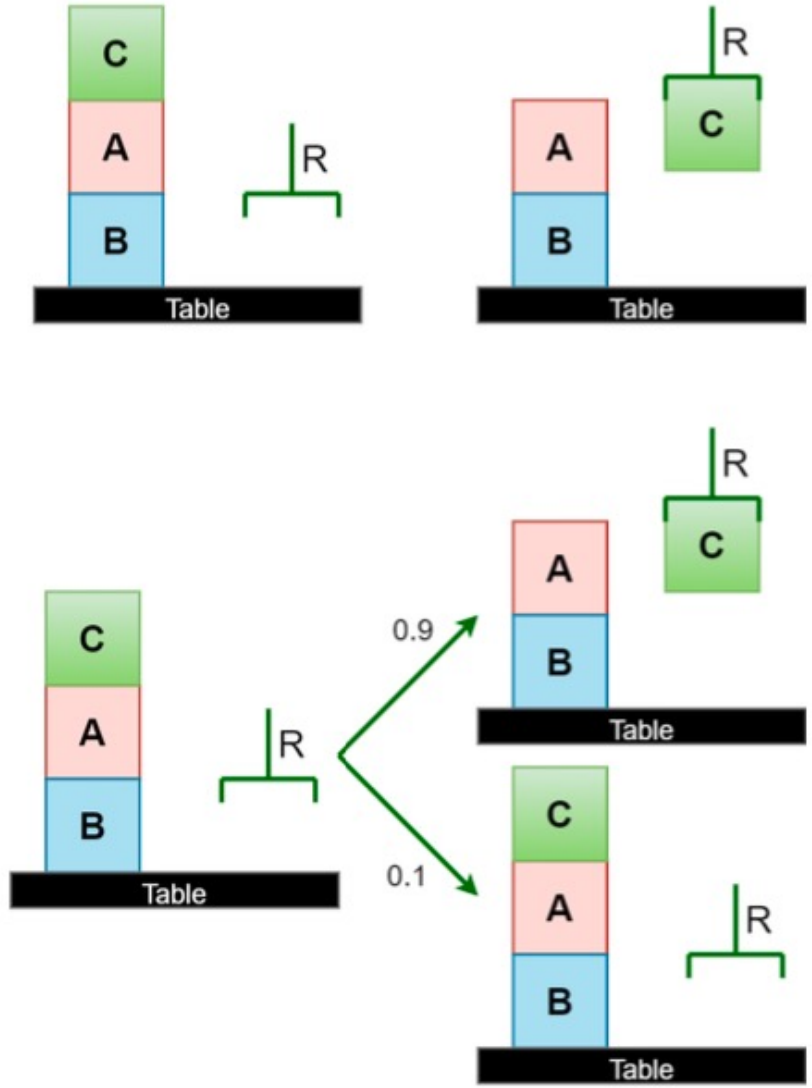
- A complete description of the world.
- Instantiation of all state variables
- Truth assignment of all variables



What is an action?

Transition from one state to another

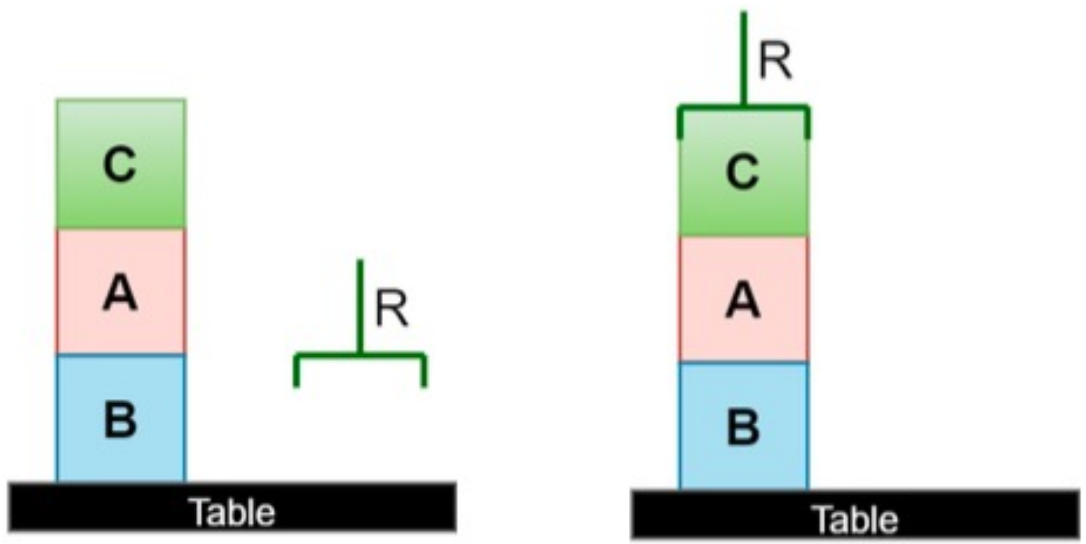
- Some actions may be applicable to some of the states.
  - `(Pickup ?R ?X)` applicable if `(Block ?X)`, `(Clear ?X)`, `(Free ?R)`
- Have given effects on the state.
  - Deterministic
  - Non-deterministic
  - Probabilistic



What is a goal?

A desired set of states

- (On C A), (On A B), (On B Table)
  - (On C A), (On A B), (On B Table), (Clear C), (Free R)
  - (On C A), (On A B), (On B Table), (Holding R C)



What is a plan?

- A sequence of instantiated actions
- A set of instantiated actions
- A tree of instantiated actions
- A policy of instantiated actions

```
; example of a sequence of
;   instantiated actions
(pickup a)
(stack a b)
```

How to find a plan?

Solution Techniques

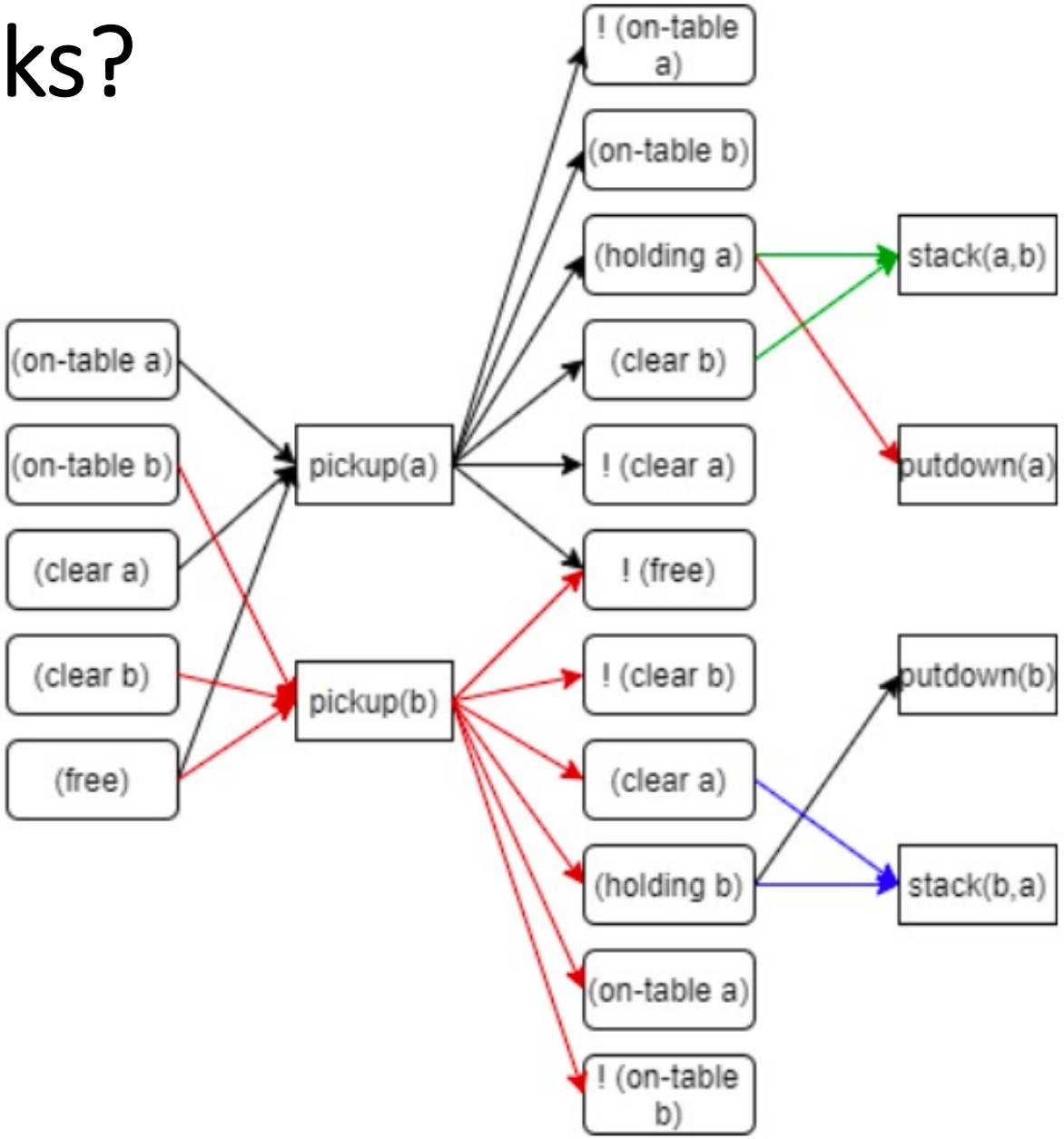
- Graph search
- Heuristic search
- Solve the transitioning matrix

Complexity

- Exponential in the number of actions
- Polynomial in the number of states

How does a planner works?

- The planning algorithm creates the plan graph exploiting e.g. Graphplan agorithm Blum and Furst [1]
  - Start with initial condition
  - Add actions with satisfied conditions
  - Add all effects of actions at previous levels
  - Add frame conditions
  - Repeat!



How does it works?

- The planner creates the plan graph exploiting e.g. Graphplan algorithm.
- At each step it checks whether all of its goals exist.
  - if true $\rightarrow$ there is a chance that the plan graph contains a solution, and the planner has to search for it.
  - if false $\rightarrow$ the planner has to grow the graph out another level and try again.

  The algorithm also performs a pruning phase, in which it finds and marks pairs of actions/propositions that are mutually exclusive.
  - A more in-depth look of the Graphplan algorithm (with a visual example) is given at this link `http://bit.ly/MIT-graphplan`.

STRIPS

In 1971 STRIPS (Stanford Research Institute Problem Solver) was developed as an automated planner. Later, the name STRIPS has been used to refer only to the formal language of the inputs

To actually deal with planning, we use specific language to express the domain and the problem. STRIP is a language to express planning problems in which:

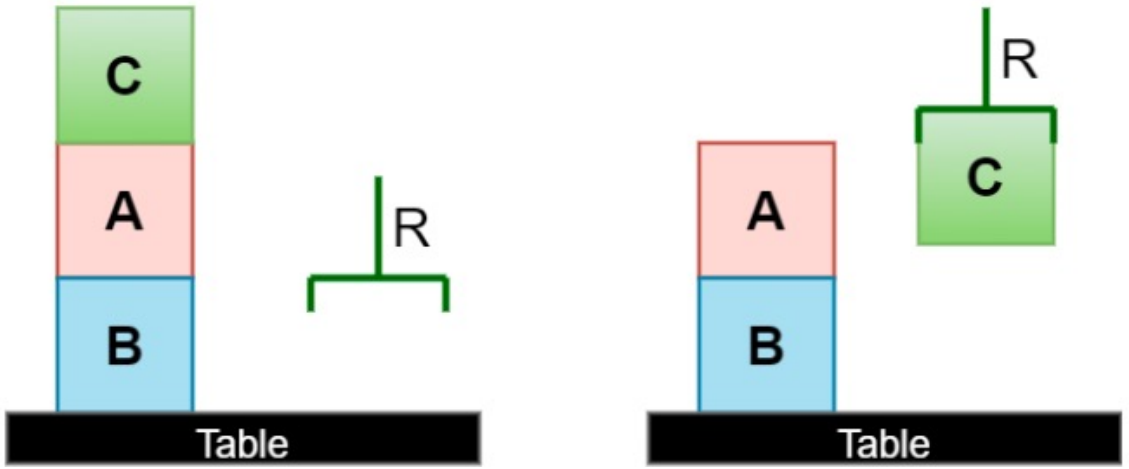
- State is expressed by a set of ground literals, assumed false if not in the set
- Effects of actions described with two sets of ground literals, one to be added and one to be removed from the set describing the world state. (add and delete lists)
- No explicit representation of time
- No logical inference rules

Something more expressive 1

- Conditional Effects

```
(:action pickup
  :parameters (?X)
  :preconditions (and (BLOCK ?X) (free) (clear ?X) (On ?X ?Y)
  :effects (and (holding ?X) (when (and (BLOCK ?Y)) (and (Clear ?Y))) ; add
                (not (free)) (not (On ?X ?Y)))                        ; delete
```

- Quantified effects `(forall (?x) (when (and (in ?x ?y))...))`
- Disjunctive and negated preconditions `(or (conn ?x ?y) (not (in ?y ?x)))`



Something more expressive 2

- Functional effects `(increment ?x 10)`
- Disjunctive effects
- Probabilistic effects
- Duration (actions no more instantaneous)
- External events, agents, concurrent events, etc
- Inference operators

PDDL

Planning Domain Definition Language : standard encoding language for classical planning

Objects	Things in the world
Predicates	Properties of the objects
Initial state	The state of the world we start in
Goal specification	Things we want to be true
Actions	Ways of changing the state of the world

How to put pieces together?

It works with two files, splitting content of planning problem in a domain file and a problem file

- A domain file for predicates and actions
- A problem file for objects, initial states, and goal descriptions

Which one can we re-use?

PDDL: Domain file

```
(define (domain <domain name>)
   <PDDL directives>              ; specify PDDL language features
   <PDDL code for predicates>
   <PDDL code for actions>
)
```

The blocks world domain file

```
;; domain file: block-domain-complete.pddl
(define (domain blocksworld)
  (:requirements :strips)

  (:predicates (clear ?x) (on-table ?x) (holding ?x) (on ?x ?y) (free))

  (:action pickup
           :parameters (?ob)
           :precondition (and (clear ?ob) (on-table ?ob) (free))
           :effect (and (holding ?ob)
                        (not (clear ?ob)) (not (on-table ?ob)) (not (free))))

  (:action putdown
      ...
)
```

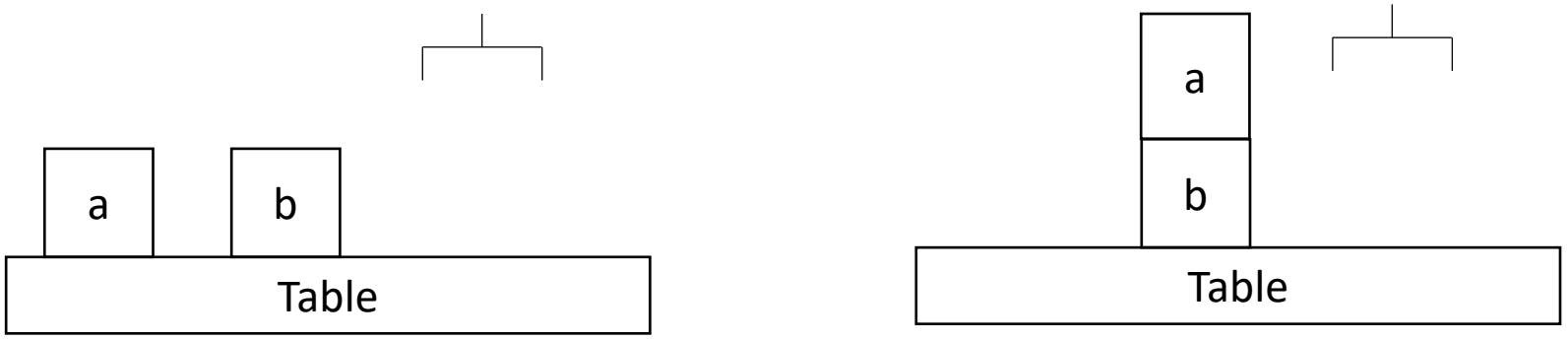
Problem file

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

- *<problem name>* is the string for this specific planning problem
- *<domain name>* must match your domain file!

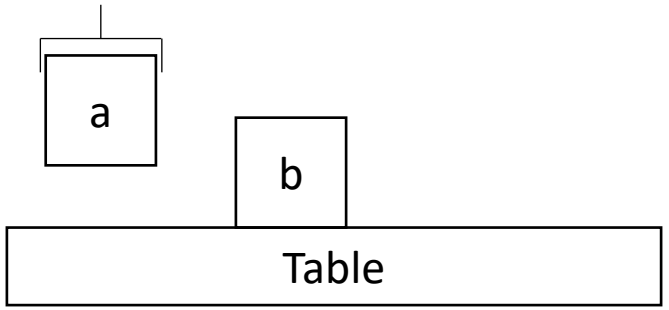
The blocks world problem file

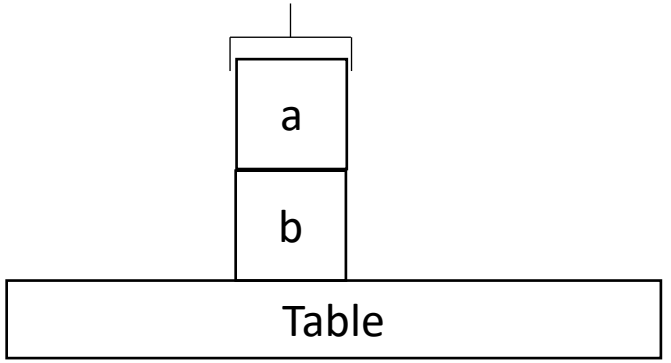
```
;; problem file: blocks-problem1.pddl

(define (problem blocksworld-prob1)
  (:domain blocksworld)
  (:objects a b)
  (:init (on-table a) (on-table b) (clear a) (clear b) (free))
  (:goal (and (on a b))))
```



Sample output

```
command_prompt > planner blocks-domain-complete.pddl blocks-problem1.pddl
...
Parsed Domain file blocks-domain-complete.pddl successfully
Parsed Problem file blocks-problem1.pddl successfully
Performing search ...
...
Found a solution
(pickup a)
(stack a b)
Instantiation Time = 0.07sec
Planning Time = 0.03sec
```





Closed-World Assumption

!!! IMPORTANT !!!

- PDDL adopts the Closed-World Assumption (CWA):
  - Predicates not mentioned in
    - actions' precondition and
    - problems' init

    are assumed to be false.

  - Examples:
    - In pickup's precondition is assumed that `(not (holding ?ob))` holds.
    - In problem's init is assumed that `(not (on a b))`

!!! CWA does not hold for the goal !!!

GRIPPER 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.

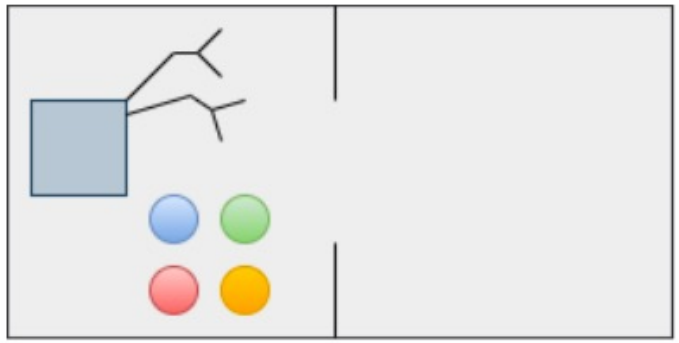
OBJECTS: The two rooms, four balls and two robot arms.

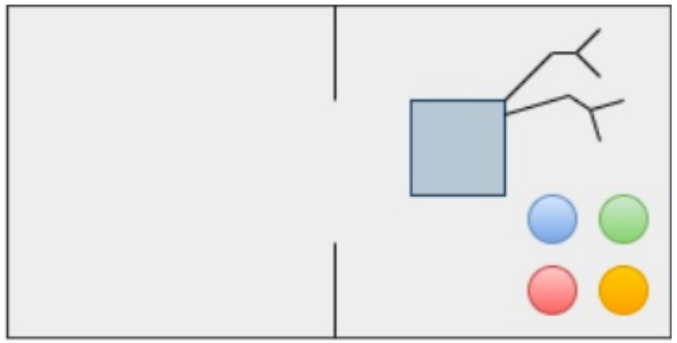
PREDICATES: Is x a room? Is x a ball? Is ball x inside room y? Is robot arm x empty? [...]

INITIAL STATE: All balls and the robot are in the first room. All robot arms are empty. [...]

GOAL STATES: All balls must be in the second room.

ACTIONS: The robot can move between rooms, pick up a ball or drop a ball.





Objects

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

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

Predicates

- ROOM(x)
- BALL(x)
- GRIPPER(x)
- at-robby(x)
- at-ball(x,y)
- free(x), carry(x,y)

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

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.

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

Goal Specification

at-ball(ball1, roomb), ..., at-ball(ball4, roomb) must be true.

PDDL

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

Movement Action

Action

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 does not change.

PDDL

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

Pick up action

ACTION

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

PRECONDITION: ROOM(x), ROOM(y) and at-robby(x) are true. 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 does not change.

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))))
```

Drop action

ACTION

DESCRIPTION: The robot can drop x in y from z

PRECONDITION: BALL(x), ROOM(y), GRIPPER(z), carry(z,x), at-robby(y)

EFFECT: at-ball(x,y) and free(z) become true, carry(z,x) becomes false, everything else does not change

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))))
```

Example complete

```
(define (domain gripper)
  (:requirements :strips)
  (:predicates (ROOM ?x) (BALL ?x) (GRIPPER ?x) (at-robby ?x) (at-ball ?x ?y)
               (free ?x) (carry ?x ?y) )
  (:action move
           :parameters (?x ?y)
           :precondition (and (ROOM ?x) (ROOM ?y) (at-robby ?x))
           :effect (and (at-robby ?y) (not (at-robby ?x))) )
  (: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 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))) )
  )
```

Example complete

```
(define (problem gripper-prob)
  (:domain gripper)
  (:objects rooma roomb
            ball1 ball2 ball3 ball4
            left right)
  (: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 (and (at-ball ball1 roomb)
              (at-ball ball2 roomb)
              (at-ball ball3 roomb)
              (at-ball ball4 roomb)))
  )
```

Extras ...1: Typing

```
(define (domain gripper-typed)
   (:requirements :typing)
   (:types room ball gripper)
   (:constants left right - gripper)
   (:predicates (at-robby ?r - room)
                (at ?b - ball ?r - room)
                (free ?g - gripper)
                (carry ?o - ball ?g - gripper))
   (:action move :parameters (?from ?to - room)
       :precondition (at-robby ?from)
       :effect (and (at-robby ?to)
                    (not (at-robby ?from))))
```

Extras … 2: Type Hierarchy

```
(define (domain logistics-adl)
   (:requirements :adl :domain-axioms)
   (:types physobj - object
           obj vehicle - physobj
           truck airplane - vehicle
           location city - object
           airport - location)
  (:predicates (at ?x - physobj ?l - location)
               (in ?x - obj ?t - vehicle)
               (in-city ?l - location ?c - city))
```

Extras...3: Conditional Effects

```
(:action drive-truck
 :parameters (?truck - truck ?loc-from ?loc-to - location ?city - city)
 :precondition (and (at ?truck ?loc-from)
                    (in-city ?loc-from ?city)
                    (in-city ?loc-to ?city))
 :effect (and (at ?truck ?loc-to)
              (not (at ?truck ?loc-from))
              (forall (?x - obj)
                 (when (and (in ?x ?truck))
                       (and (not (at ?x ?loc-from))
                            (at ?x ?loc-to))))))
```

Next

- Hands-on Session!
- We will create and solve planning problems
- Download Blackbox planner
  http://www.cs.rochester.edu/u/kautz/satplan/blackbox/
- Alternatively use an online planner:
  https://web-planner.herokuapp.com/
  http://planning.domains/