



SAPIENZA
UNIVERSITÀ DI ROMA

Artificial Intelligence

Prof: Daniele Nardi

Exercises: Classical Planning - STRIPS & PDDL

Francesco Riccio
email: riccio@diag.uniroma1.it

Classical Planning

in a nutshell

(Russell & Norvig, Chapter 10)

Planning is the process of creating a **sequence of actions** to achieve an agent's goals.

- State representation - Action representation
- Fully observable, deterministic, static environments, single agent

STRIPS: **S**Tanford **R**esearch **I**nstitute **P**roblem **S**olver

Manifold algorithms for generating plans: **Forward** (progression), **Backward** (regression),

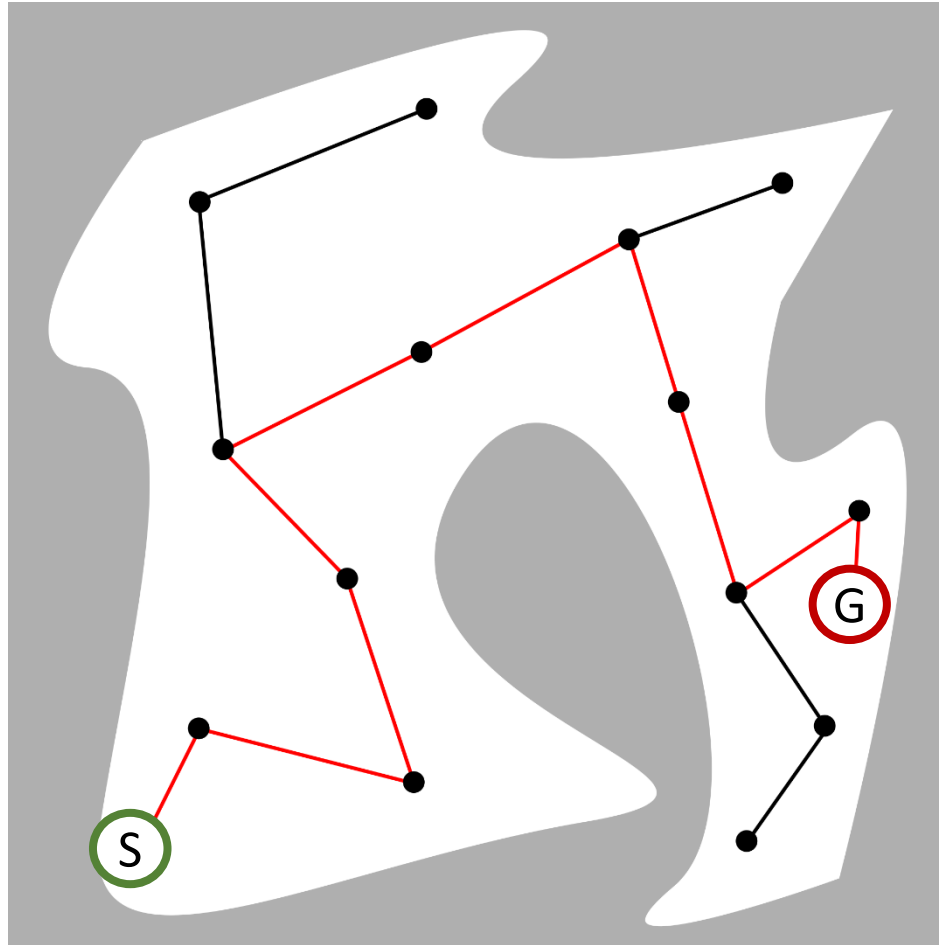
PDDL: **P**lanning **D**omain **D**efinition **L**anguage

(standard encoding language for classical planning, it does not specify an algorithm for producing plans!)

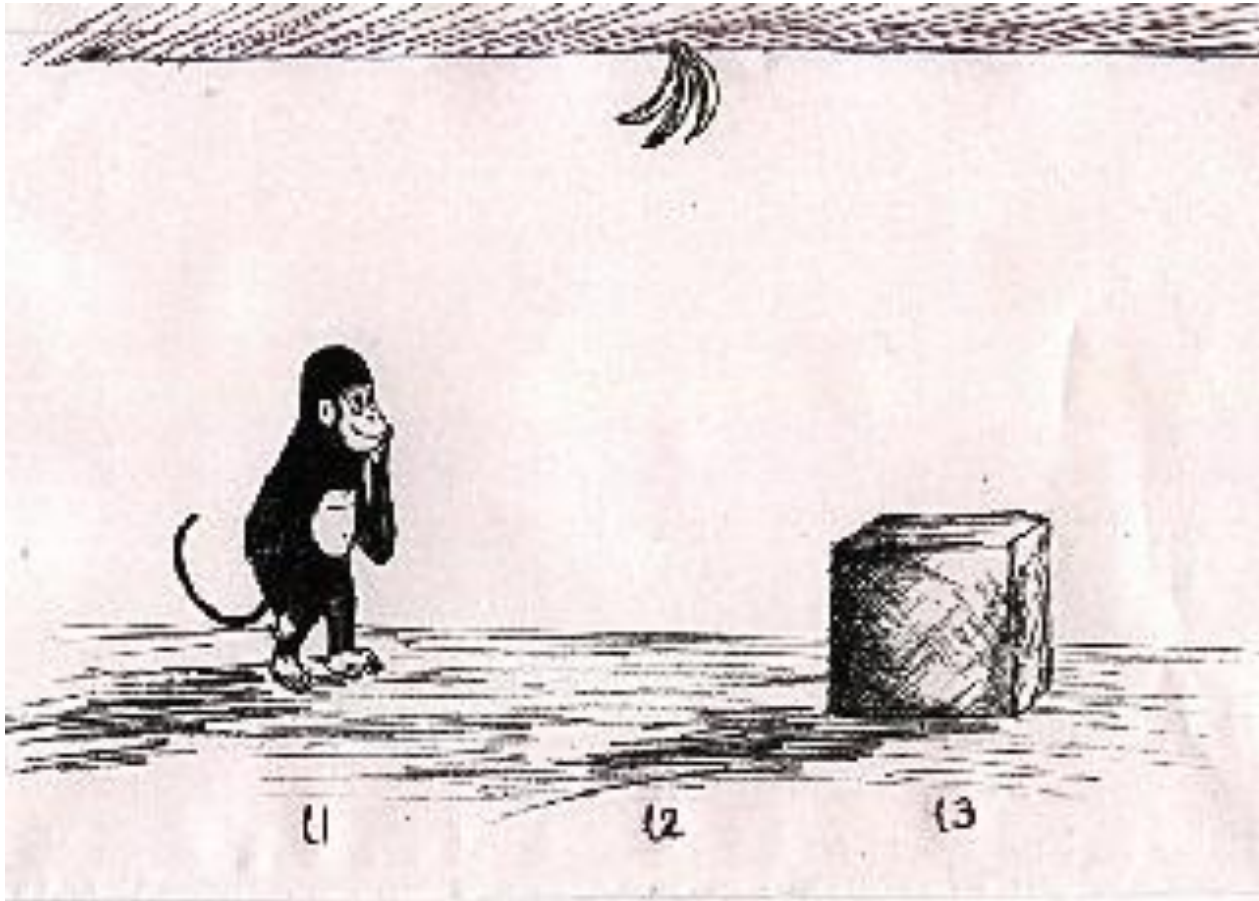
Composed by: **Objects** (landmarks of interest in the world), **Predicates** (properties of objects/landmarks), **Initial state**, **Goal specification**, **Actions/Operators** (ways of changing the state of the world).

Forward (Progression) and Backward (Regression) Search

Planning as a state-space search

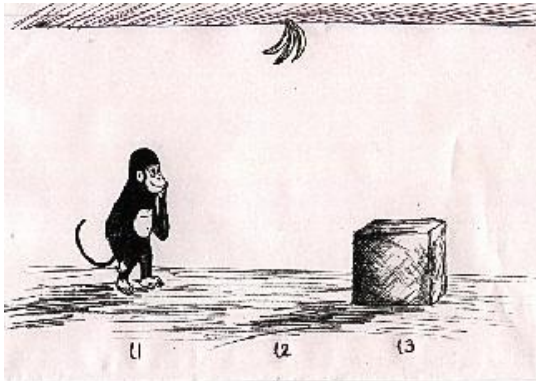


The Monkey and Bananas problem



The Monkey and Bananas problem

STRIPS



Initial state: At(A), Level(low), BoxAt(C), BananasAt(B)

Goal state: Have(bananas)

Actions:

// move to box location

GoBox(X),

PRECOND: At(X), Level(low)

EFFECT: not At(X), At(C)

// climb up on the box

ClimbBox(X),

PRECOND: At(X), BoxAt(X), Level(low)

EFFECT: Level(high), not Level(low)

// move monkey and box to bananas location

PushBoxBananas(X),

PRECOND: At(X), BoxAt(X), Level(low)

EFFECT: BoxAt(B), not BoxAt(X), At(B), not At(X)

// take the bananas

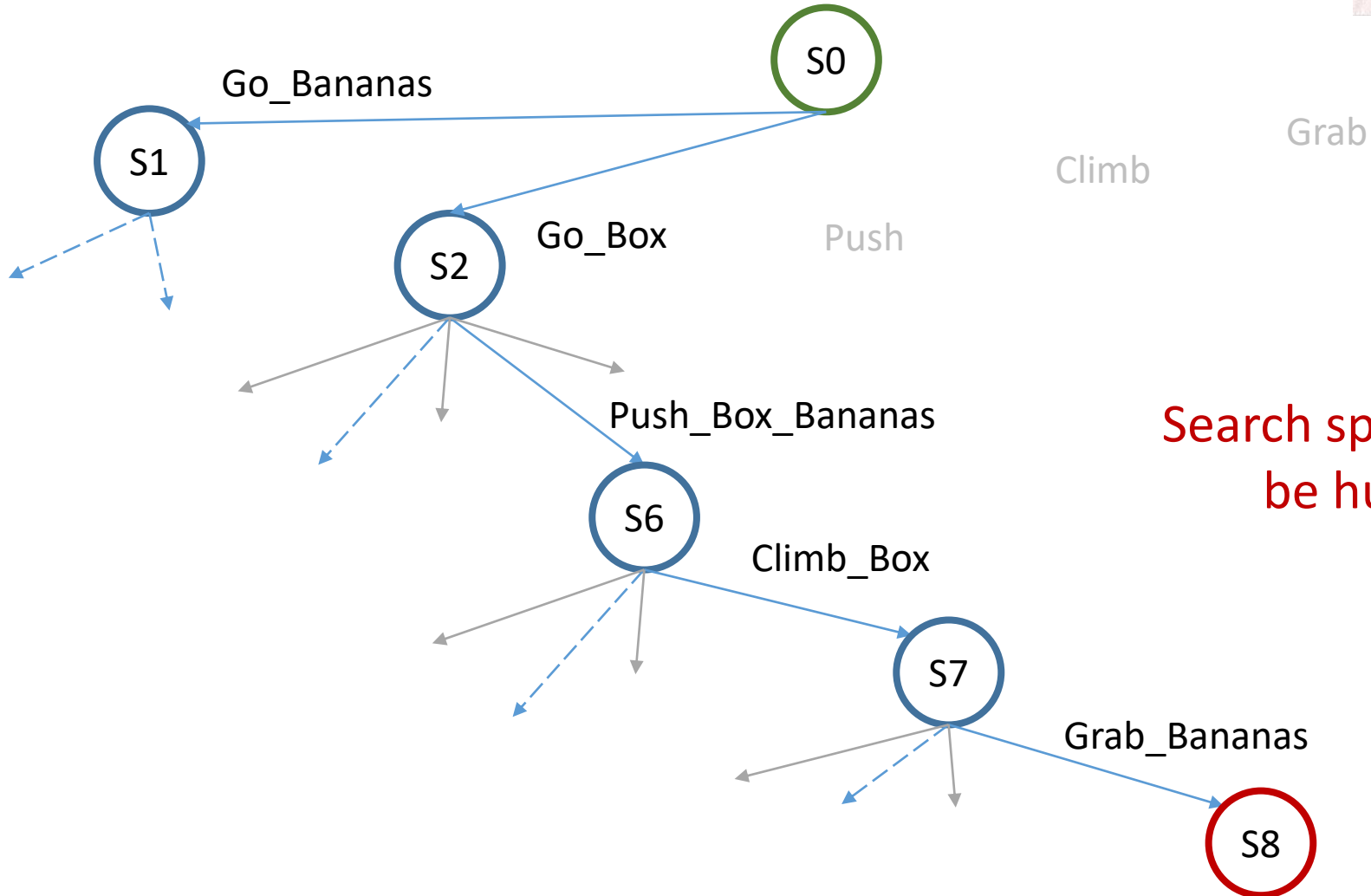
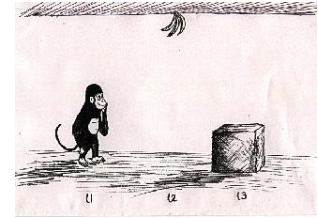
GrabBananas(X),

PRECOND: At(X), BananasAt(X), Level(high)

EFFECT: Have(bananas)

Forward Search (Progression)

The Monkey and Bananas



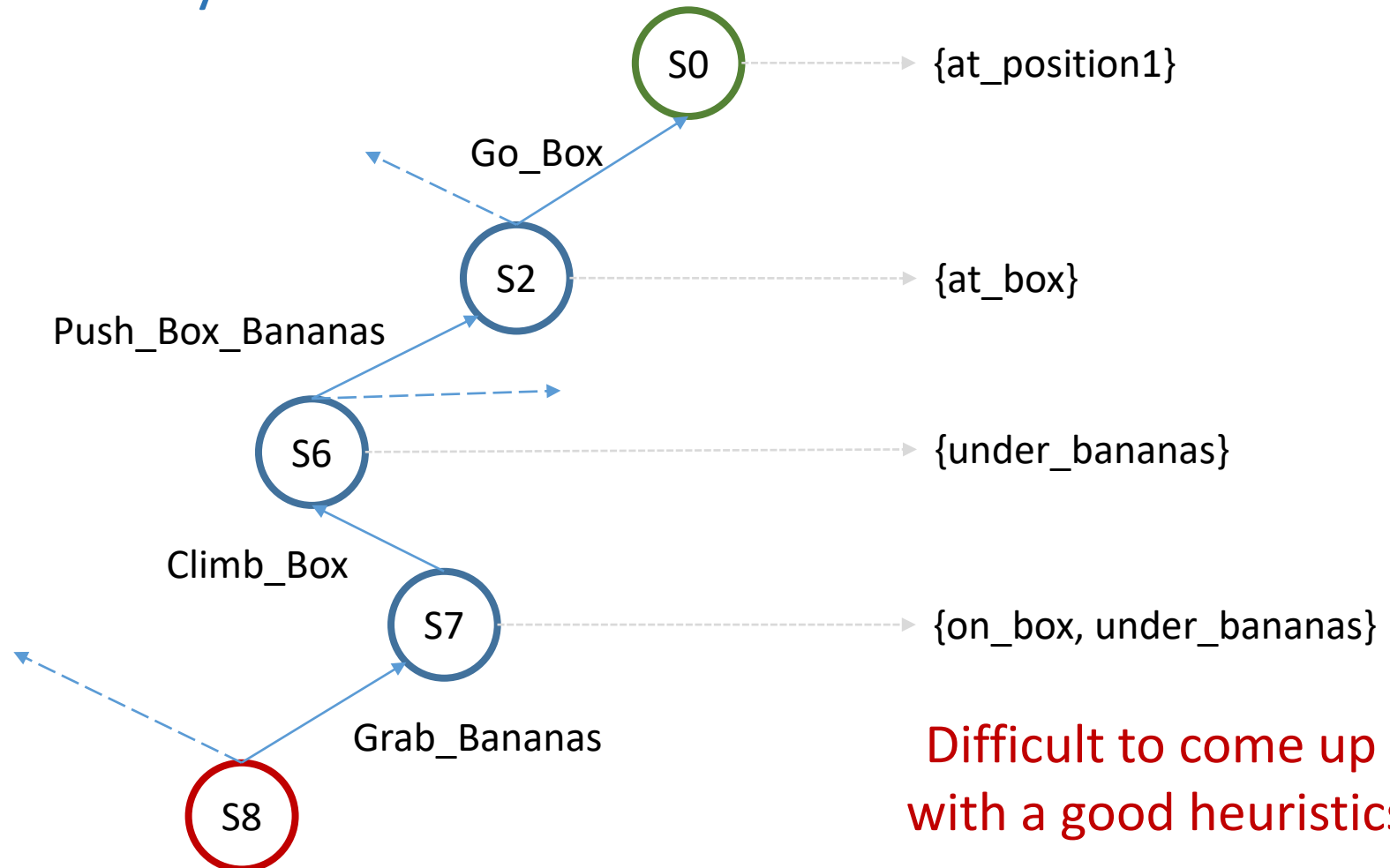
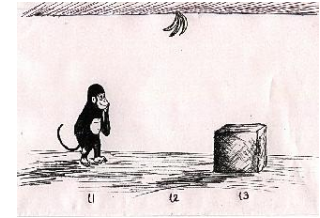
Search space can be huge

$$s' = (s - \text{DEL}(a)) \cup \text{ADD}(a)$$

Backward Search (Regression)

aka relevant-states search

The Monkey and Bananas



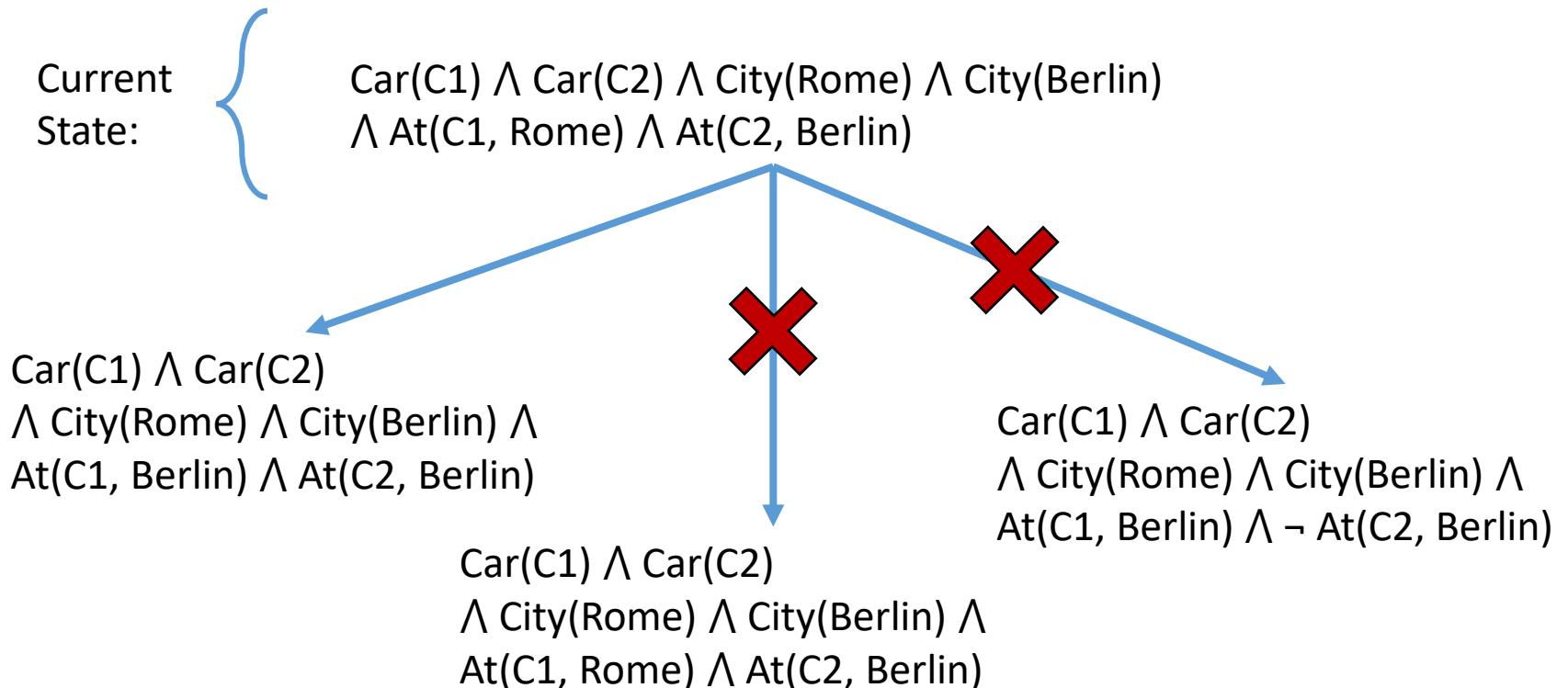
Difficult to come up
with a good heuristics

$$g' = (g - \text{ADD}(a)) \cup \{\text{PRECOND}(a)\}$$

Example

Recap

```
Drive(C1, Rome, Berlin),  
  PRECOND: At(C1, Rome)  $\wedge$  Car(C1)  
            $\wedge$  City(Rome)  $\wedge$  City(Berlin)  
  EFFECT:  $\neg$ At(C1, Rome)  $\wedge$  At(C1, Berlin)  
)
```



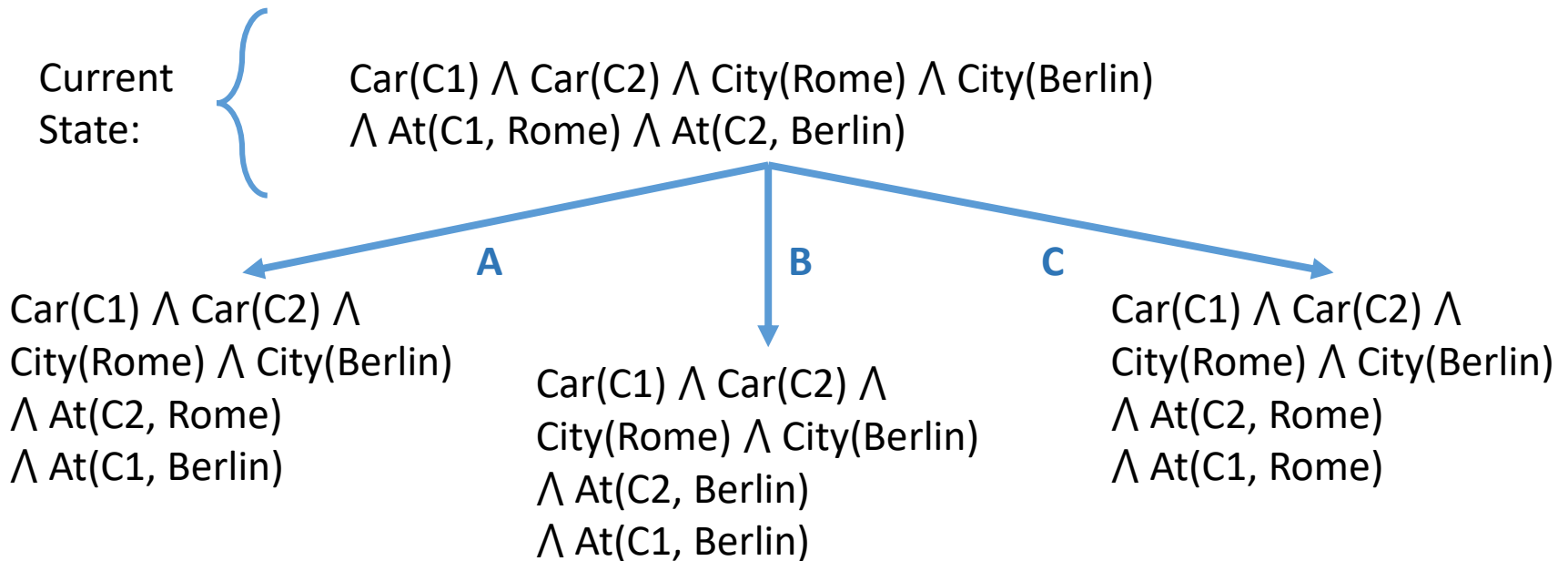
$$s' = \text{RESULT}(s, a) = (s - \text{DEL}(a)) \cup \text{ADD}(a)$$

Example

Recap

Drive(?car, ?from, ?to),
PRECOND: $\text{At}(\text{?car}, \text{?from}) \wedge \text{Car}(\text{?car})$
 $\wedge \text{City}(\text{?from}) \wedge \text{City}(\text{?to})$
EFFECT: $\neg \text{At}(\text{?car}, \text{?from}) \wedge \text{At}(\text{?car}, \text{?to})$
)

Effects of applying the **Drive** action



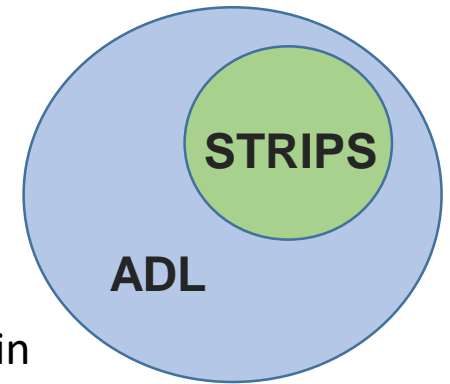
1. **A** 2. **B** 3. **C** 4. **A,B** 5. **A,C** 6. **B,C** 7. **A,B,C**

PDDL

Action description language

Action description language (ADL) is an automated planning and scheduling system in particular for robots.

In PDDL we can specify an ADL planner by adding **:adl** to the domain specification.



It means that the domain uses some or all of the ADL elements: **disjunctions** and **quantifiers** in preconditions and goals, quantified and **conditional effects**.

In an ADL domain, an **effect formula** may in addition contain:

- A Conditional effect

(when CONDITION_FORMULA EFFECT_FORMULA)

The interpretation is that the specified effect takes place only if the specified condition formula is true in the state where the action is executed.

Conditional effects are usually placed within quantifiers.

- A universally quantified formula:

(forall (?V1 ?V2) EFFECT_FORMULA)

PDDL

ADL vs. STRIPS

| STRIPS Language | ADL Language |
|--|--|
| Only positive literals in states: <i>Poor</i> \wedge <i>Unknown</i> | Positive and negative literals in states: \neg <i>Rich</i> \wedge \neg <i>Famous</i> |
| Closed World Assumption: Unmentioned literals are false. | Open World Assumption: Unmentioned literals are unknown. |
| Effect $P \wedge \neg Q$ means add P and delete Q . | Effect $P \wedge \neg Q$ means add P and $\neg Q$ and delete $\neg P$ and Q . |
| Only ground literals in goals: <i>Rich</i> \wedge <i>Famous</i> | Quantified variables in goals: $\exists x At(P_1, x) \wedge At(P_2, x)$ is the goal of having P_1 and P_2 in the same place. |
| Goals are conjunctions: <i>Rich</i> \wedge <i>Famous</i> | Goals allow conjunction and disjunction: \neg <i>Poor</i> \wedge (<i>Famous</i> \vee <i>Smart</i>) |
| Effects are conjunctions. | Conditional effects allowed: when P : E means E is an effect only if P is satisfied. |
| No support for equality. | Equality predicate ($x = y$) is built in. |
| No support for types. | Variables can have types, as in ($p : Plane$). |

Figure 11.1 Comparison of STRIPS and ADL languages for representing planning problems. In both cases, goals behave as the preconditions of an action with no parameters.

PDDL

Action description language - Example

```
(:action drive-truck

  :parameters (?truck ?loc-from ?loc-to ?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))
                   )
               ))
)
```

PDDL

ADL to PDDL – conditional effects

```
(:action goto
  :parameters (?agent ?from ?to)
  :precondition (and (at ?agent ?from))
  :effect       (and (at ?agent ?to) (not (at ?agent ?from))
                    (when (and (obj ?o) (carry ?agent ?o))
                        (and (not (at ?o ?from))
                            (at ?o ?to)))
                ))
)
```

```
(:action goto
  :parameters (?agent ?from ?to)
  :precondition (and (at ?agent ?from))
  :effect       (and (at ?agent ?to)
                    (not (at ?agent ?from)))
)
```

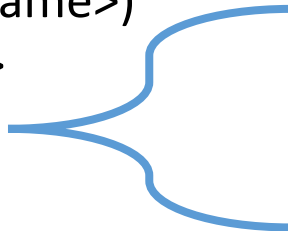
```
(:action goto
  :parameters (?agent ?from ?to ?o)
  :precondition (and (at ?agent ?from)
                    (obj ?o) (carry ?agent ?o))
  :effect       (and (at ?agent ?to)
                    (not (at ?agent ?from))
                    (at ?o ?to)
                    (not (at ?o ?from)))
)
```

PDDL

template

Domain file:

```
(define (domain <domain name>)  
  <PDDL predicates>  
  <PDDL actions>  
)
```



Action(ActionName,
 PARAM: var1 ... varN
 PRECOND: list of preconditions
 EFFECT: list of effects
)

Problem file:

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

The Shakey's world

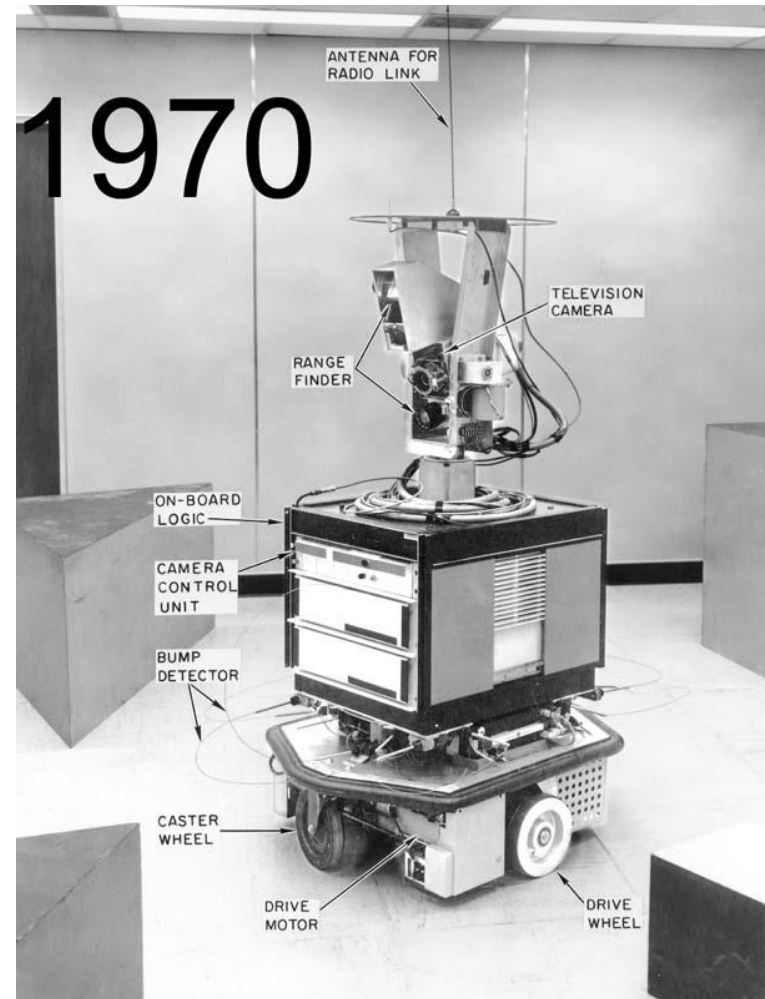
STRIPS

Stanford Robot controlled by STRIPS;

At the time planner could find plans beyond robots abilities;

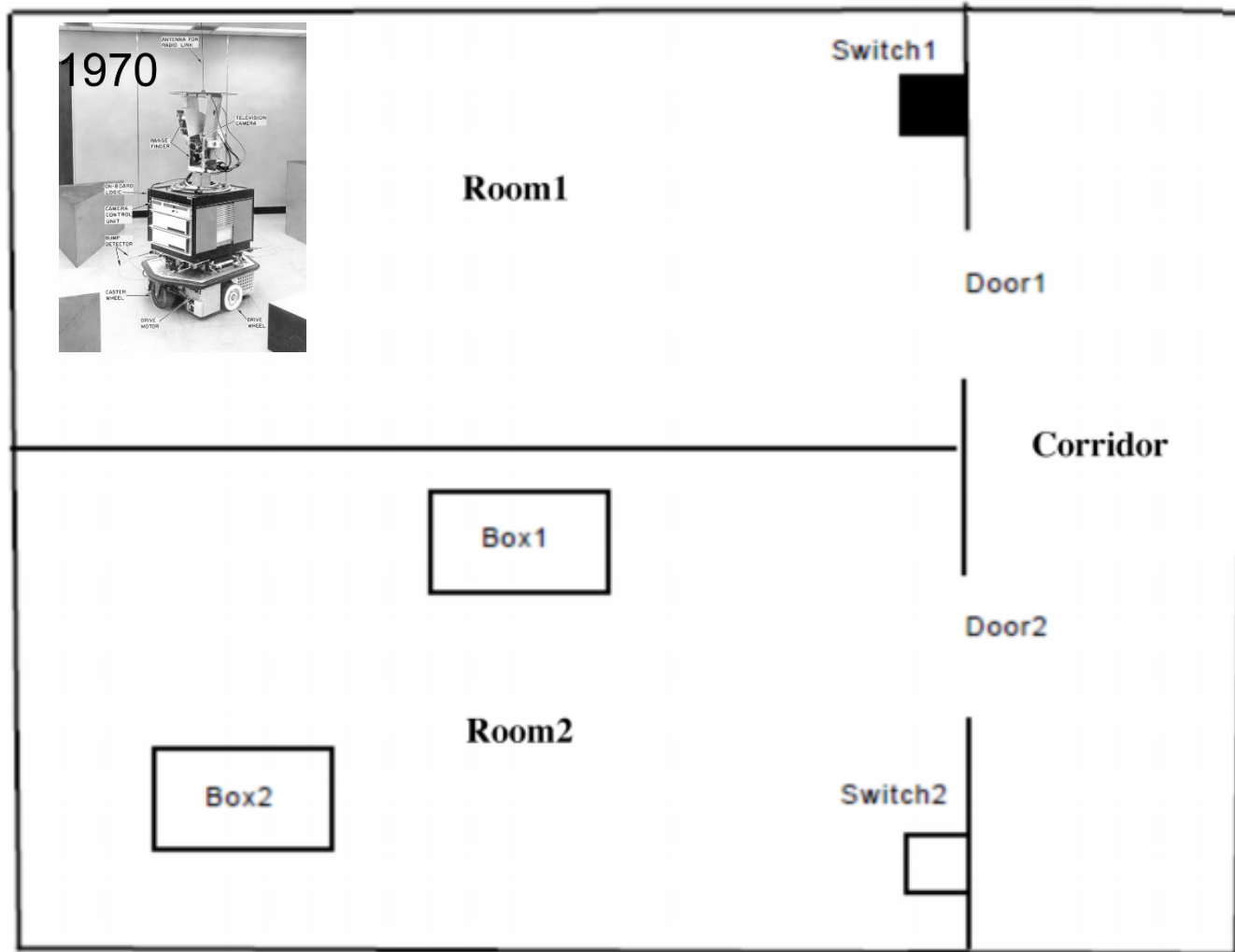
Actions:

- Move;
- Pushing movable objects;
- Climb onto and down from objects;
- Turn light switches on and off.



The Shakey's world

STRIPS



The Shakey's world

STRIPS

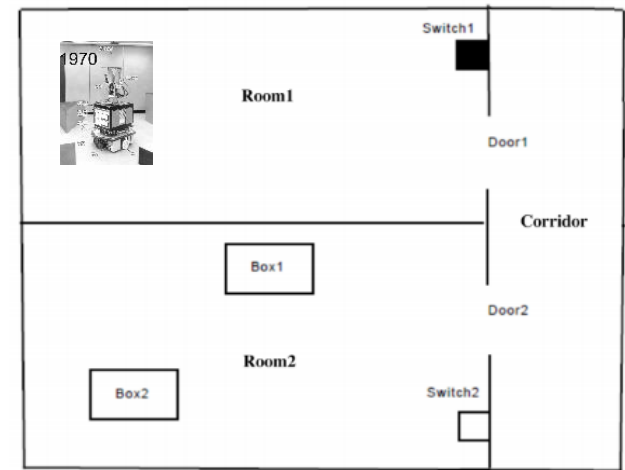
INITIAL STATE:

(init:

(on Shakey Floor) (at Shakey START) (in START Room1)
(box Box1) (at Box1 BX1) (in BX1 Room2) (box Box2)
(at Box2 BX2) (in BX2 Room2) (in Door1 Room1)
(in Door1 Corridor) (in Switch1 Room1)
(not (turnedOn Switch1)) (in Door2 Room2)
(in Door2 Corridor) (in Switch2 Room2)
(turnedOn Switch2)

)

GOAL: (:goal (at Box2 Switch1))



OBJECTS:

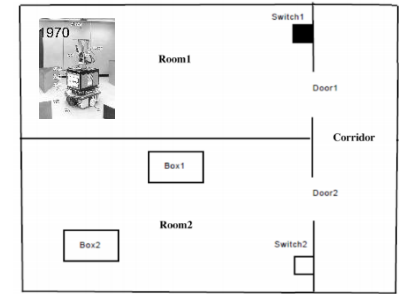
(:objects

Shakey Floor Corridor START
Room1 Room2
Door1 Door2
Switch1 Switch
Box1 Box2
BX1 BX2

)

The Shakey's world

STRIPS



PREDICATES: Is x at location y? Is x on y? Is x a box? Is x turned On?

(:predicates (at ?x ?y) (on ?x ?y) (in ?x ?y) (box ?x) (turnedOn ?x)) }

ACTIONS:

(:action **Go**

:parameters (?x ?y ?r)

:precondition (and (on Shakey Floor) (at Shakey ?x) (in ?x ?r) (in ?y ?r))

:effect (and (at Shakey ?y) (not (at Shakey ?x)))

)

(:action **Push**

:parameters (?b ?x ?y ?r)

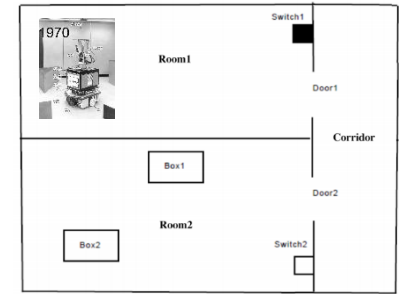
:precondition (and (on Shakey Floor) (at Shakey ?x) (box ?b) (at ?b ?x) (in ?x ?r) (in ?y ?r))

:effect (and (at Shakey ?y) (at ?b ?y) (not (at Shakey ?x)) (not (at ?b ?x)))

)

The Shakey's world

STRIPS



(:action **ClimbUp**

:parameters (?x ?b)

:precondition (and (on Shakey Floor) (at Shakey ?x) (at ?b ?x) (box ?b))

:effect (and (on Shakey ?b) (not (on Shakey Floor)))

)

(:action **ClimbDown**

:parameters (?b)

:precondition (and (on Shakey ?b) (box ?b))

:effect (and (on Shakey Floor) (not (on Shakey ?b)))

)

(:action **TurnOn**

:parameters (?s ?b)

:precondition (and (on Shakey ?b)
(box ?b) (at ?b ?s))

:effect (turnedOn ?s)

)

(:action **TurnOff**

:parameters (?s ?b)

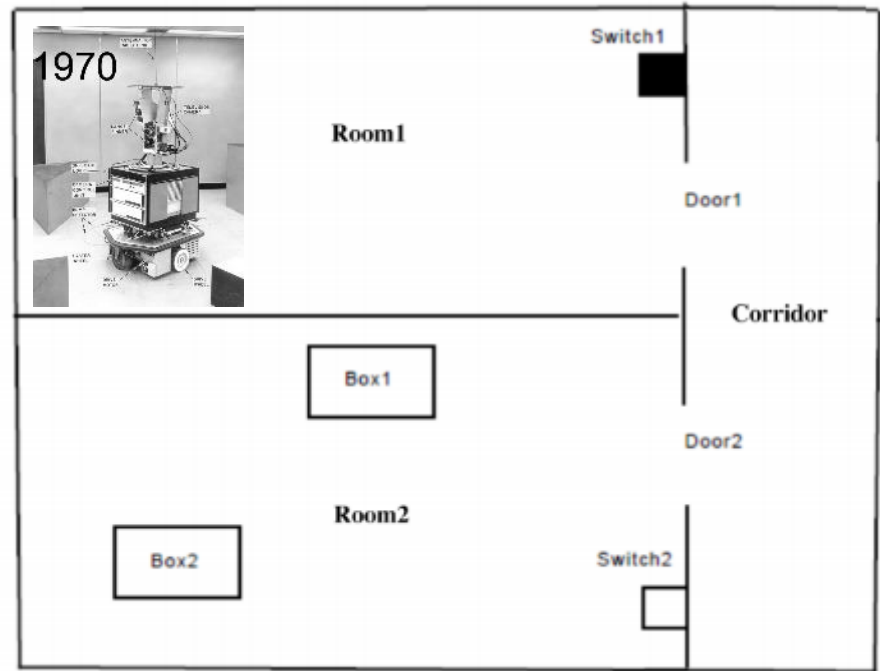
:precondition (and (on Shakey ?b)
(box ?b) (at ?b ?s))

:effect (not (turnedOn ?s))

)

The Shakey's world

STRIPS



Generated Plan:

1. (go start door1 room1)
2. (go door1 door2 corridor)
3. (go door2 bx2 room2)
4. (push box2 bx2 door2 room2)
5. (push box2 door2 door1 corridor)
6. (push box2 door1 switch1 room1)

Two Armed Robot

STRIPS

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: 2 rooms, 4 balls and 2 robot arms.

Predicates: $\text{room}(x)$, $\text{ball}(x)$, $\text{at-ball}(b, r)$, $\text{free}(x)$, etc.;

Initial state: all balls and the robot are in the first room, all robot arms are empty, etc.;

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.





Two Armed Robot

STRIPS: Domain File

```
(define (domain gripper-strips)
  (:predicates (room ?r) (ball ?b) (gripper ?g) (at-robot ?r) (at ?b ?r) (free ?g) (carry ?o ?g))
```

```
(:action move
  :parameters (?from ?to)
  :precondition (and (room ?from) (room ?to) (at-robot ?from))
  :effect (and (at-robot ?to) (not (at-robot ?from))))
```

```
(:action pick
  :parameters (?obj ?room ?gripper)
  :precondition (and (ball ?obj) (room ?room) (gripper ?gripper) (at ?obj ?room) (at-robot
    ?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-
    robot ?room))
  :effect (and (at ?obj ?room) (free ?gripper) (not (carry ?obj ?gripper)))) )
```

Two Armed Robot

STRIPS: Problem File



```
(define (problem strips-gripper2)
  (:domain gripper-strips)

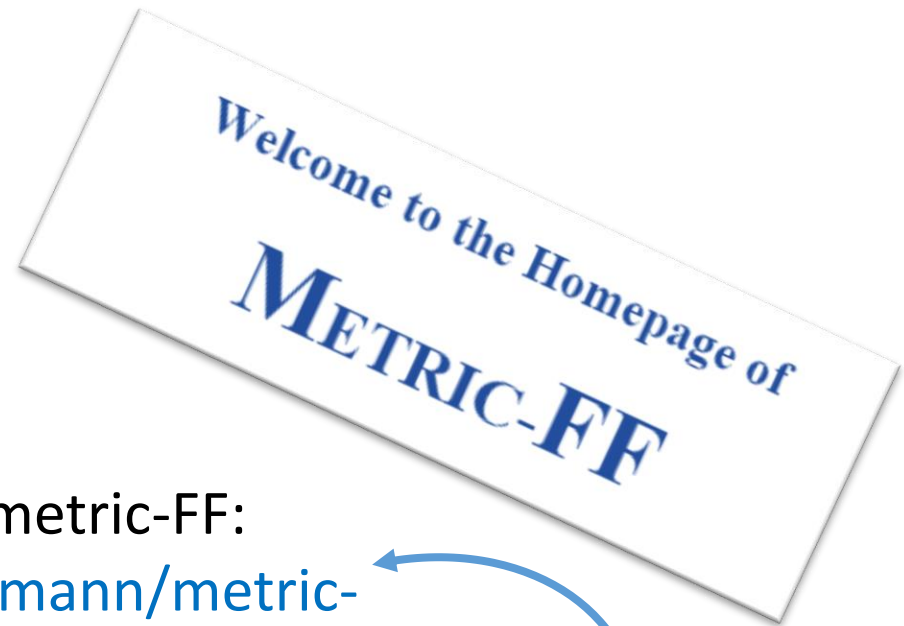
  (:objects rooma roomb ball1 ball2 ball3 ball4 left right)

  (:init (room rooma) (room roomb) (ball ball1) (ball ball2)
    (ball ball3) (ball ball4) (gripper left) (gripper right) (at-robb
    rooma) (free left) (free right) (at ball1 rooma) (at ball2
    rooma) (at ball3 rooma) (at ball4 rooma))

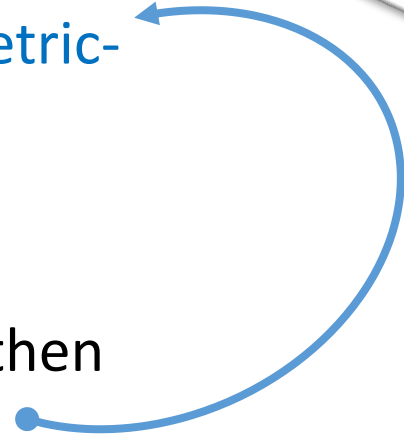
  (:goal (at ball1 roomb) (at ball2 roomb) (at ball3 roomb) (at
    ball4 roomb))

)
```

Get a plan



- For linux users go to download metric-FF:
<https://fai.cs.uni-saarland.de/hoffmann/metric-ff.html>
- For other users:
Install linux or a linux virtual machine, and then



State Search Planning

The Monkey and Bananas

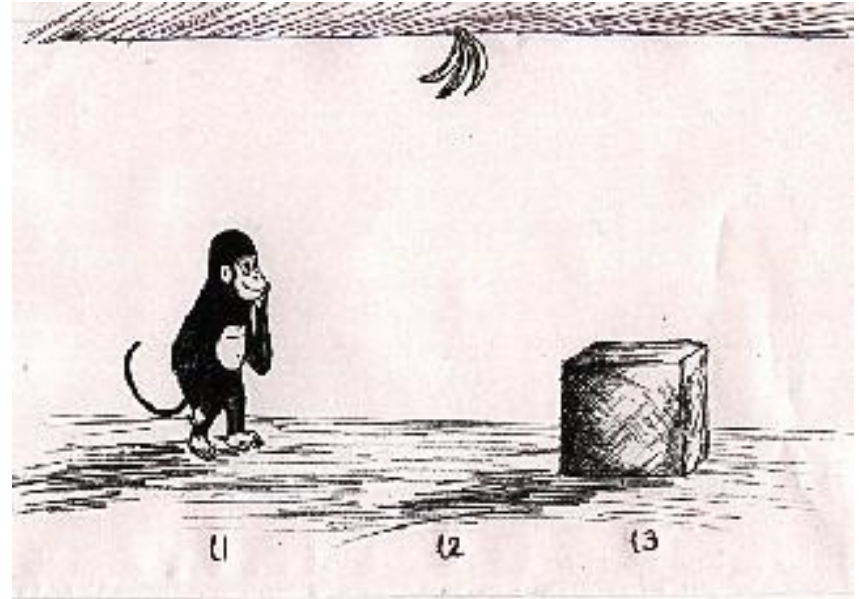
Objects: monkey, bananas, box.

Predicates: on, box, holding, on_ceiling, etc.

Initial state: as in Fig.

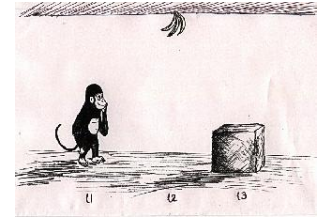
Goal specification: the monkey is holding bananas.

Actions/Operators: go, push, climb, grab.



State Search Planning

The Monkey and Bananas



(:action **go**
:parameters (?who ?from ?to)
:precondition (the monkey at ?from)
:effect (the monkey is at ?to and not at ?from))

(:action **push**
:parameters (?who ?what ?from ?to)
:precondition (the monkey at ?from, ?what is
at ?from)
:effect (the monkey is at ?to and not at ?from,
?what is at ?to and not at ?from))

(:action **climb**
:parameters (?who ?what ?under-what)
:precondition (the monkey at ? under-what,
?what is at ? under-what)
:effect (the monkey is on ?what))

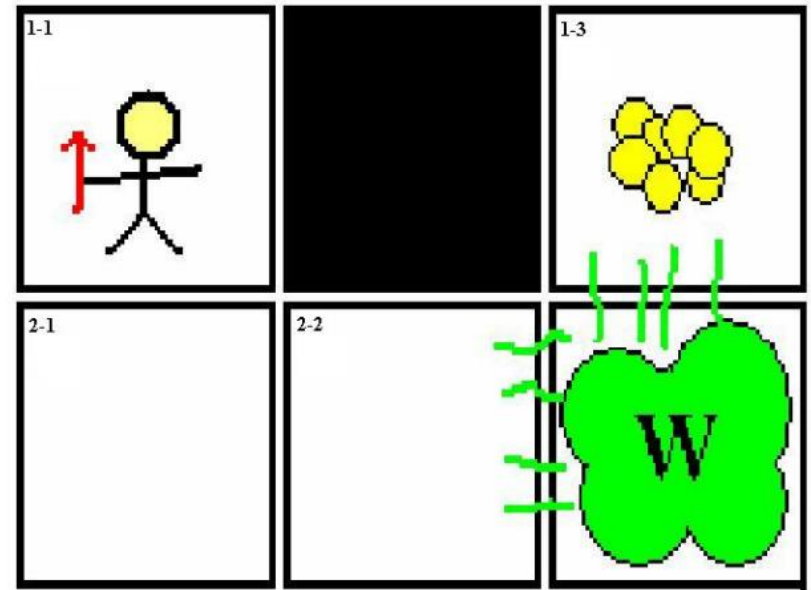
(:action **grab**
:parameters (?who ?what ?under-what)
:precondition (the monkey at ? under-what,
the monkey is on ?what)
:effect (the monkey is holding ?under-what))

The Wumpus World

STRIPS

An agent can move between squares and collect gold if any in the same square. Some of the square are not '*walkable*' as there is a pit, and a wumpus is hiding in one of the squares.

The agent can shoot at the wumpus with an arrow if they are in the adjacent squares. Of course, the agent cannot move in the same square of the wumpus (if he is alive), otherwise he will die.



Objects: 6 squares, 1 gold, 1 agent, 1 arrow, 1 wumpus

Predicates: gold(x), agent(x), wumpus(x), dead(x), at(x,y), etc.

Initial state: as in Fig

Goal specification: the agent must collect the gold and return to the initial position

Actions/Operators: the agent can move between squares, collect gold, shoot an arrow

The Wumpus World

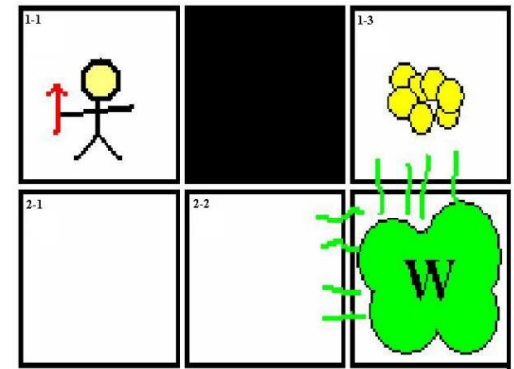
STRIPS: Domain File

```
(define (domain wumpus-c)
  (:predicates (at ?what ?square) (adj ?square-1 ?square-2)
    (pit ?square) (wumpus-in ?square) (have ?who ?what)
    (is-agent ?who) (is-wumpus ?who) (is-gold ?what) (is-arrow ?what) (dead ?who) )
```

```
(:action move
  :parameters (?who ?from ?to)
  :precondition (and (is-agent ?who) (at ?who ?from) (adj ?from ?to) (not (pit ?to)) (not (wumpus-in ?to)))
  :effect (and (not (at ?who ?from)) (at ?who ?to)))
```

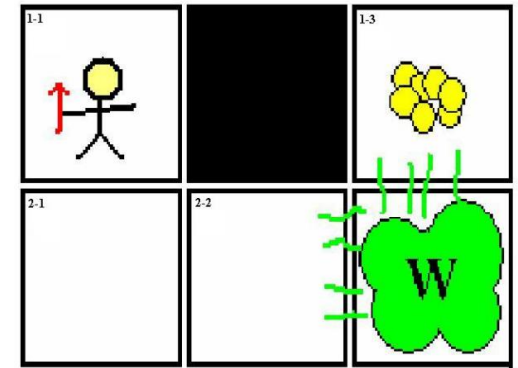
```
(:action take
  :parameters (?who ?what ?where)
  :precondition (and (is-agent ?who) (at ?who ?where) (at ?what ?where))
  :effect (and (have ?who ?what) (not (at ?what ?where))))
```

```
(:action shoot
  :parameters (?who ?where ?with-what ?victim ?where-victim)
  :precondition (and (is-agent ?who) (have ?who ?with-what) (is-arrow ?with-what) (at ?who ?where) (is-wumpus ?victim) (at ?victim ?where-victim) (adj ?where ?where-victim))
  :effect (and (dead ?victim) (not (wumpus-in ?where-victim)) (not (have ?who ?with-what)))))
```



The Wumpus World

STRIPS: Problem File



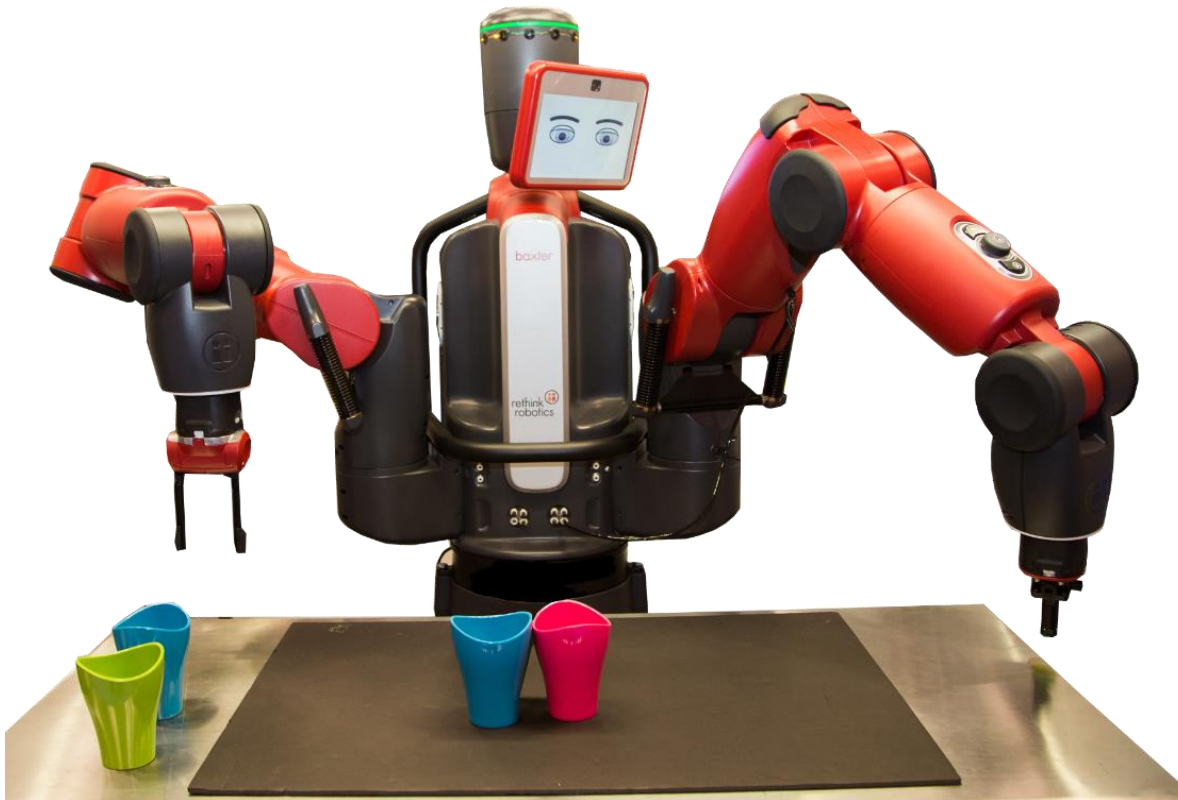
```
(define (problem wumpus-c-1)
  (:domain wumpus-c)
  (:objects s-1-1 s-1-2 s-1-3 s-2-1 s-2-2 s-2-3 gold-1 arrow-1 agent
    wumpus)

  (:init (adj s-1-1 s-1-2) (adj s-1-2 s-1-1) (adj s-1-2 s-1-3) (adj s-1-3 s-
    1-2) (adj s-2-1 s-2-2) (adj s-2-2 s-2-1) (adj s-2-2 s-2-3) (adj s-2-3 s-
    2-2) (adj s-1-1 s-2-1) (adj s-2-1 s-1-1) (adj s-1-2 s-2-2) (adj s-2-2 s-
    1-2) (adj s-1-3 s-2-3) (adj s-2-3 s-1-3) (pit s-1-2) (is-gold gold-1) (at
    gold-1 s-1-3) (is-agent agent) (at agent s-1-1) (is-arrow arrow-1)
    (have agent arrow-1) (is-wumpus wumpus) (at wumpus s-2-3)
    (wumpus-in s-2-3))

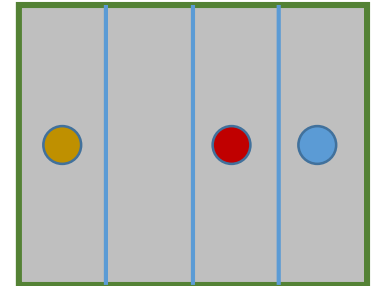
  (:goal (and (have agent gold-1) (at agent s-1-1)))
)
```

Hands-in

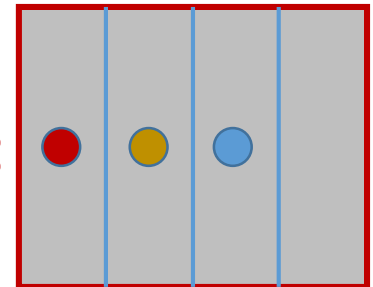
One arm robot problem



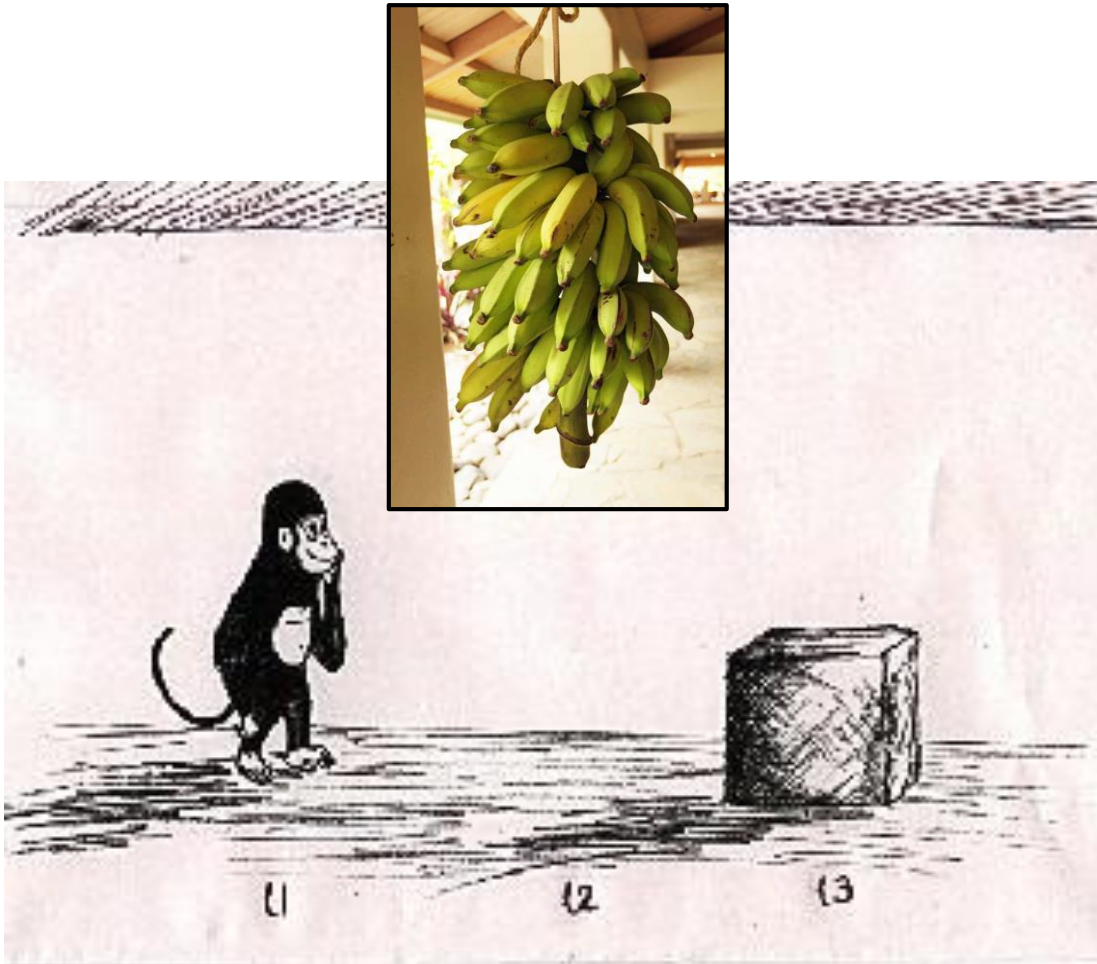
I:



G:

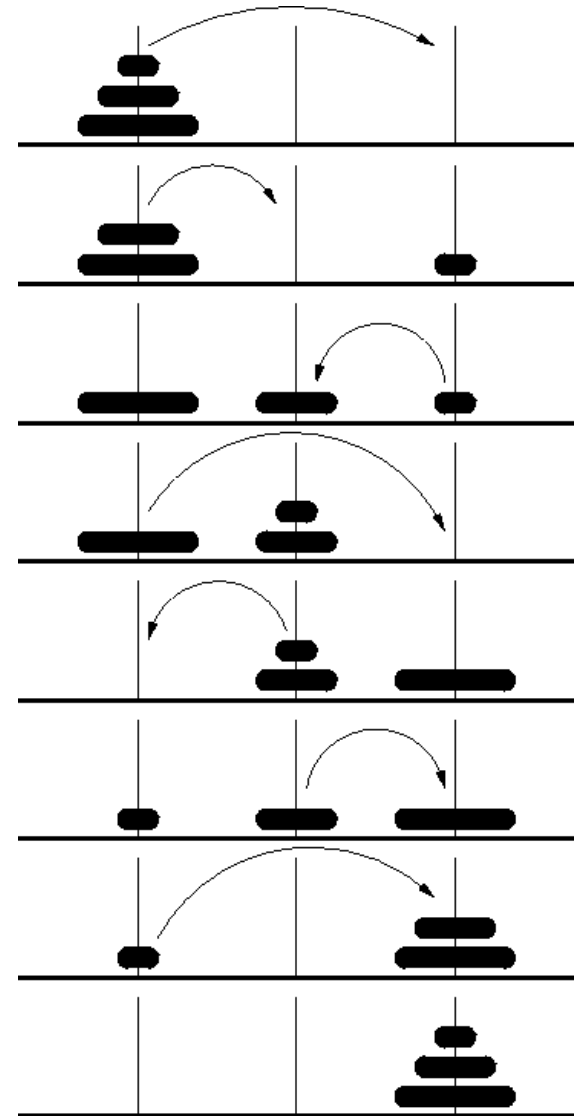


The Monkey and Bananas ++ (Hands in)



The Towers of Hanoi problem

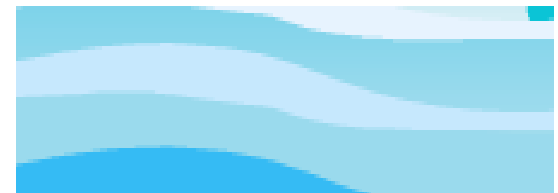
Rules for Towers of Hanoi. The goal of the puzzle is to move all the disks from the leftmost peg to the rightmost peg, adhering to the following rules: Move only **one disk at a time**. A **larger disk may not be placed on top of a smaller disk**.



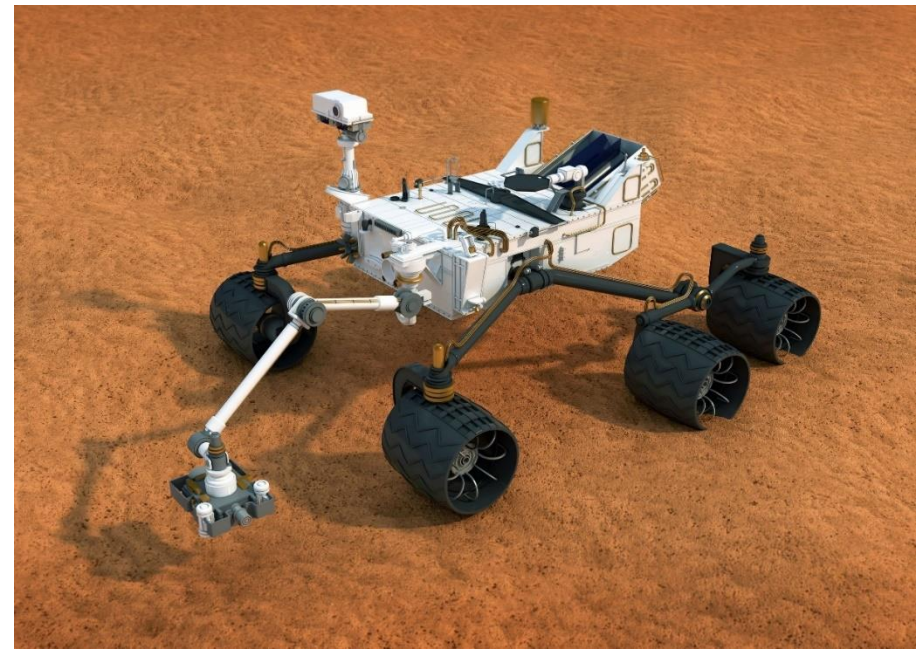
Example: Crossing the river

A man has a **wolf**, a **sheep** and a **cabbage**. He is on a river bench with a **boat**, whose maximum **load** for a single trip is the man **plus one of his 3 goods**. The man wants to cross the river with his goods, but he must avoid that - when he is far away - **the wolf eats the sheep** and that, **the sheep eats the cabbage**. How can the man plan its actions?

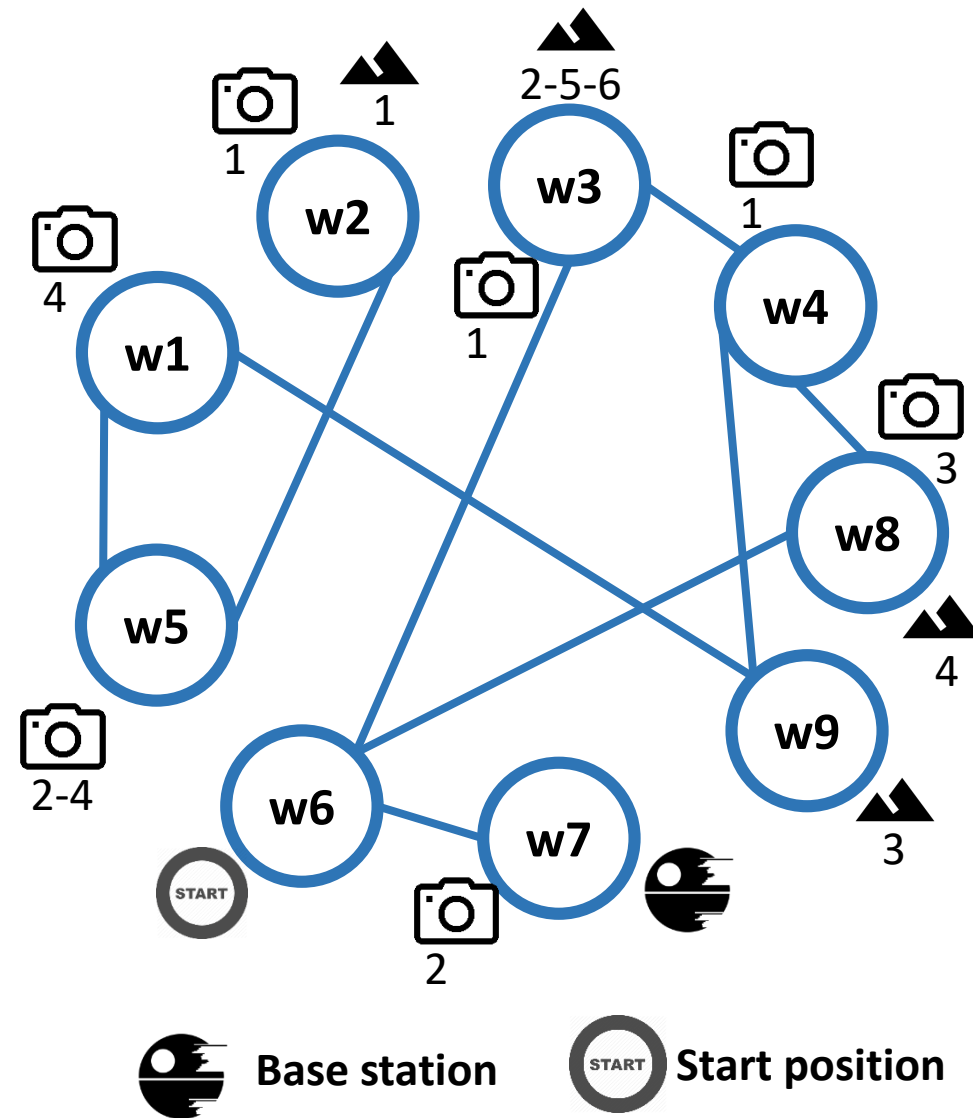
**WOLF
SHEEP
CABBAGE**




The mars rover problem

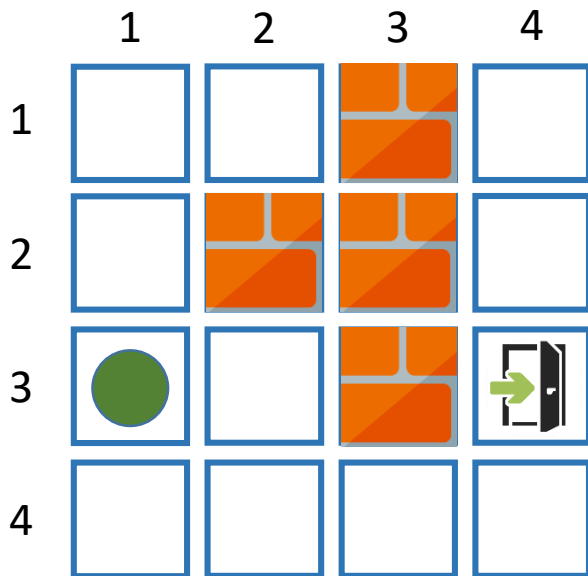



The rover has to **collect samples** from the ground and **take pictures** of different areas. **Traversability** is represented by the graph, positions of samples and desired pictures are highlighted near the nodes, as well as the **base station** and the **starting position**



Exam 17/6/2015

An agent is posed at the entrance ● of the following labyrinth and, it has to traverse it to reach the exit ➡. The symbol  represents a wall



The **state space** is the set of possible positions, that can be represented as a pair $\langle i, j \rangle$, with $0 < i < 5$ and $0 < j < 5$ and $\langle i, j \rangle \neq$ 

The **initial** state is in $\langle 3, 1 \rangle$ while the **goal** state is in $\langle 3, 4 \rangle$

At each step, the agent can move in every direction to one of the adjacent cells. It can perform an horizontal, vertical and diagonal move. Of course, the agent cannot traverse walls not move out of the labyrinth