Fundamentals of Artificial Intelligence
Graph-based Planning
Lab Session

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Luca Giuliani, Matteo Francobaldi
Dipartimento di Informatica – Scienza e Ingegneria
Università di Bologna

luca.giuliani13@unibo.it matteo.francobaldi2@unibo.it

Outline

Topic: Planning based on graphs

- **Graphplan**
  - Recall on graphic planning
  - Graphplan in action
- **PDDL Modeling**
  - PDDL
- **SATPlan and Blackbox**
  - Planning as a satisfiability problem
  - Blackbox in action
- **Fast Forward**
  - Recalls + FF in action
- **Exercises with PDDL**

Graphplan

Graphplan

What is Graphplan?

- Planner of the STRIPS type introduced by Blum and Furst (CMU) in 1995
- Based on a data structure called planning graph
- Planning as a search on the planning graph
- It is an off-line planner
- It is complete and always produces the shortest possible plan
- Produces partially ordered plans

Graphplan

- The actions are represented as those of STRIPS with

  ADD LIST

  DELETE LIST

  PRECONDITIONS
- A no-op action is implicitly defined that does not change the status
- State = objects + set of predicates
- Objects have a type (typed)!

Graphplan

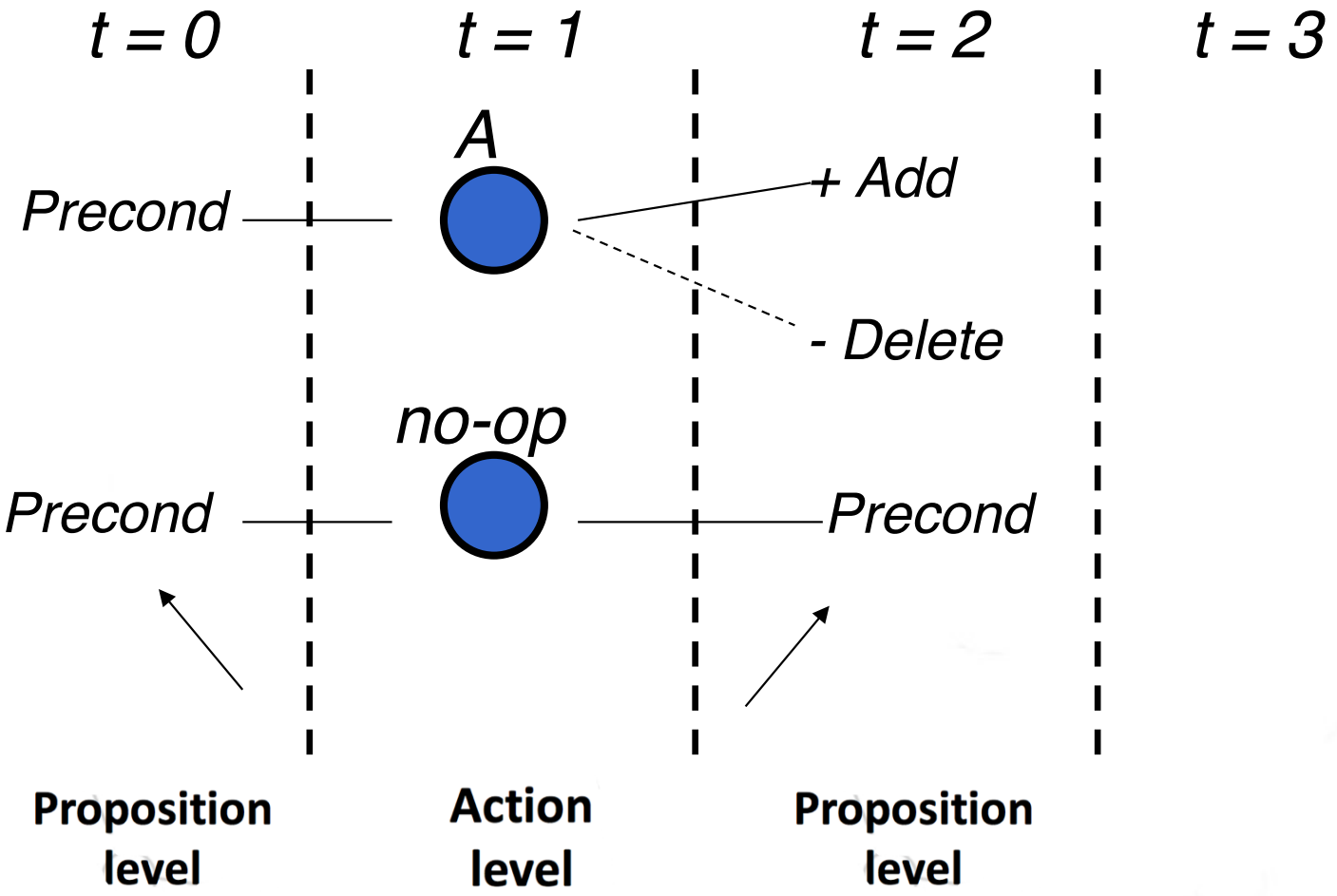
- A planning graph is a direct level graph:
  - ✓ the nodes are grouped into levels
  - ✓ the arcs connect nodes at adjacent levels

- A planning graph alternates:
  - ✓ Proposition levels, containing proposition nodes
  - ✓ Action levels, containing action nodes

- Level 0 corresponds to the initial state (it is a proposition level)

- Arcs are divided into:
  - ✓ Precondition arcs (proposition → action)
  - ✓ Add arcs (action → proposition)
  - ✓ Delete arcs (action → proposition)

Graphplan


t = 0
t = 1
t = 2
t = 3
A
Precond
+ Add
- Delete
no-op
Precond
Precond
Proposition level
Action level
Proposition level

Graphplan

During the construction of the planning graph any inconsistencies are identified, in particular

- Two actions can be inconsistent in the same time step
- Two propositions can be inconsistent at the same time step

In this case the actions / propositions are mutually exclusive

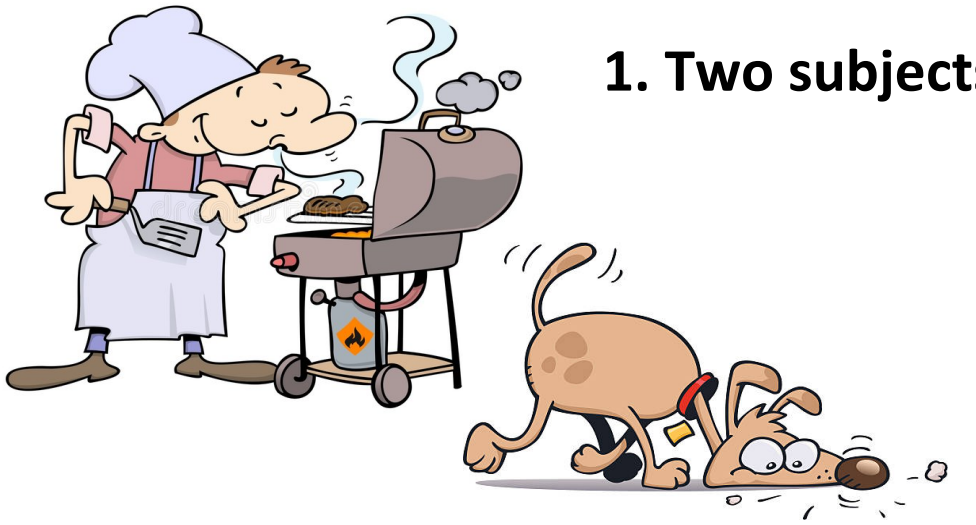
Graphplan

- Once the planning graph is built, we have to extract a **valid-plan, i.e.,** a connected and consistent subgraph of the planning graph

Features valid plan

- Actions in the same time step can be performed in any order (do not interfere)
- Propositions at the same time step are not mutually exclusive
- The last time step contains all the literals of the goal and these are not marked as mutually exclusive

A classic example



1. Two subjects

2. A starting place

3. A means of transportation

4. A destination place

A classic example

Actions:

- **MOVE (R, PosA, PosB)**

  PRECONDITIONS: at (R, PosA), hasFuel (R)

  ADD LIST: at (R, PosB)

  DELETE LIST: at (R, PosA), hasFuel (R)

- **LOAD (Object, R, Pos)**

  PRECONDITIONS: at (R, Pos), at (Object, Pos)

  ADD LIST: in (R, Object)

  DELETE LIST: at (Object, Pos)

- **UNLOAD (Object, Pos)**

  PRECONDITIONS: in (R, Object), at (R, Pos)

  ADD LIST: at (Object, Pos)

  DELETE LIST: in (R, Object)

A classic example

Types, state, goals:

- **OBJECTS:**

  cart: car (c)

  objects: jack (j), bobby (t)

  locations: home (h), mushrooms (m)

- **INITIAL STATE:**

  at (j, h)

  at (b, h)

  at (c, h)

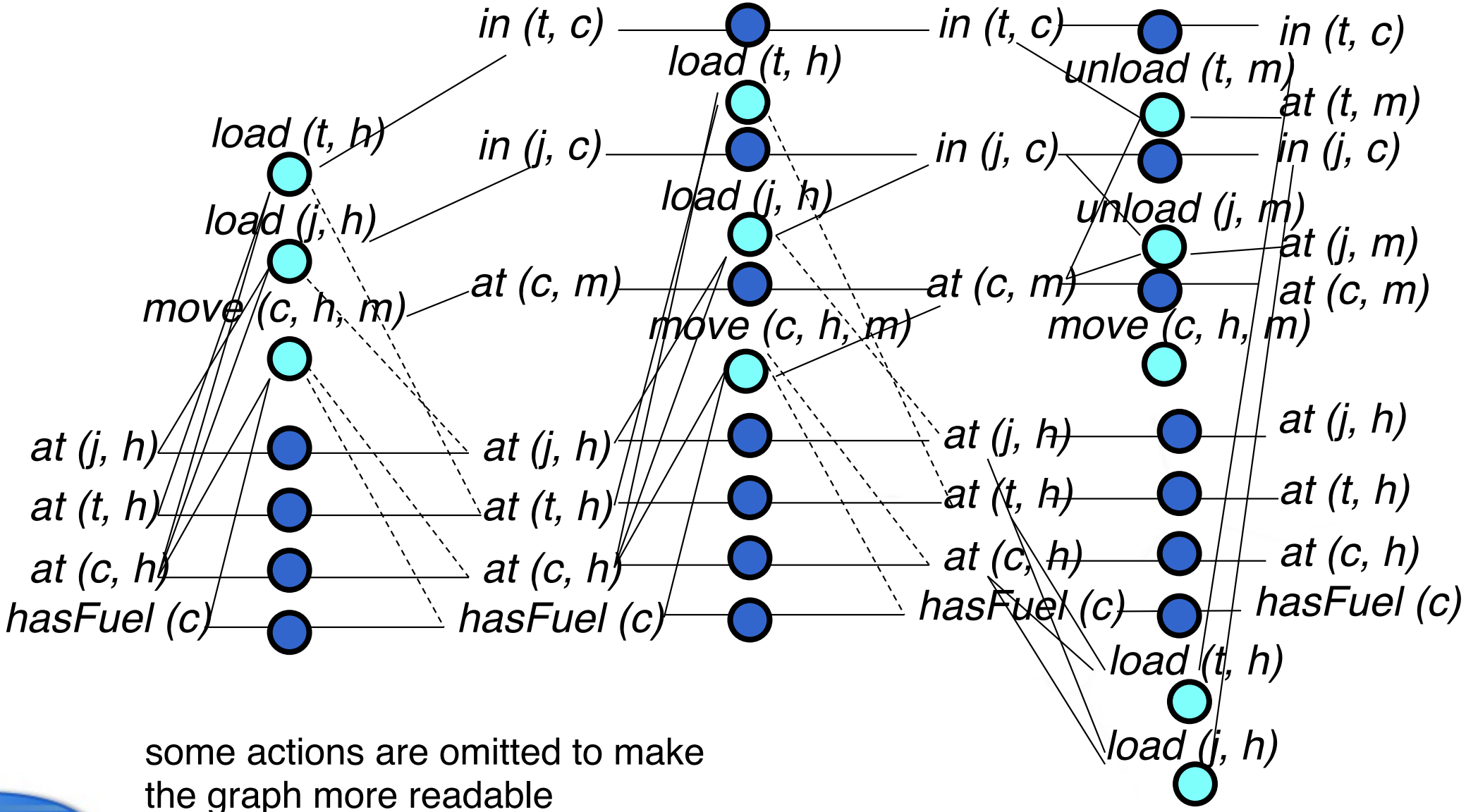
  hasFuel (c)

- **GOAL:**

  at (j, m)

  at (b, m)

A classic example



some actions are omitted to make the graph more readable

Virtual Machine

https://virtuale.unibo.it/pluginfile.php/751246/mod_resource/content/1/Lab-VirtualMachineInstallation.pdf

Start the Virtual Machine

- Open the command line
- Graphplan can run from the command line:

  info@info-VirtualBox:~$ `graphplan -h`

- On your desktop, you can find the “cart_example_gp” folder:

```
operator files: cart.gp.ops
Fact file: cart.gp.facts
```

cart.gp.facts

- **OBJECTS**

```
(home PLACE)
(mushrooms PLACE)
(car CART)
(jack CARGO)
(bobby CARGO)
```

- **INITIAL STATE:**

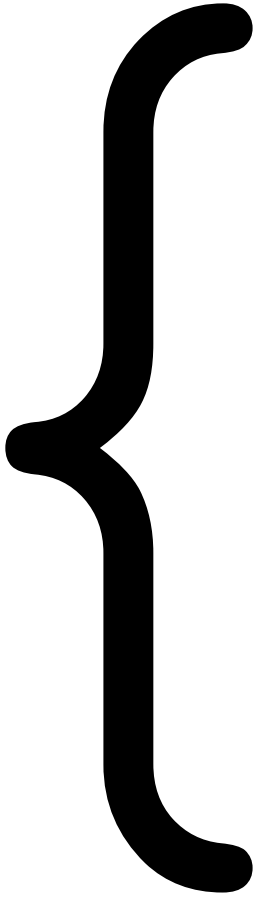
```
(preconds
(at car home)
(at jack home)
(at bobby home)
(has-fuel car))
```

- **GOAL:**

```
(effects
(at jack mushrooms)
(at bobby mushrooms))
```

cart.gp.ops

ACTIONS:



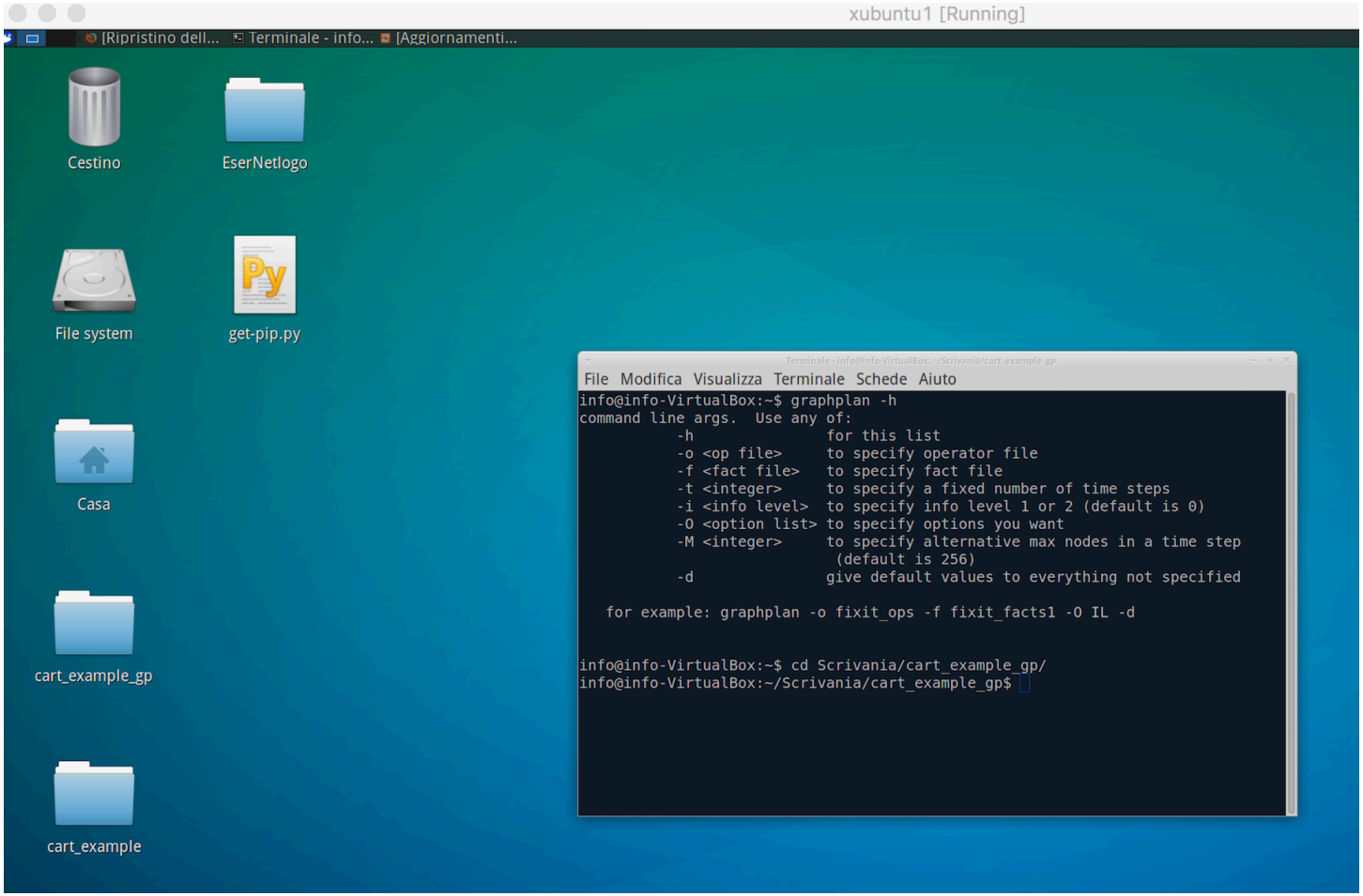
```
(operator
 LOAD
 (params
  (<object> CARGO) (<cart> CART) (<place> PLACE))
 (preconds
  (at <cart> <place>)  (at <object> <place>))
 (effects
  (on <object> <cart>) (del at <object> <place>)))

(operator
 UNLOAD
 (params
  (<object> CARGO) (<cart> CART) (<place>  PLACE))
 (preconds
  (at <cart> <place>) (on <object> <cart>))
 (effects
  (at <object> <place>) (del on <object> <cart>)))

(operator
 MOVE
 (params
  (<cart> CART) (<from> PLACE) (<to> PLACE))
 (preconds
  (has-fuel <cart>) (at <cart> <from>))
 (effects
  (at <cart> <to>) (del has-fuel <cart>) (del at <cart> <from>)))
```

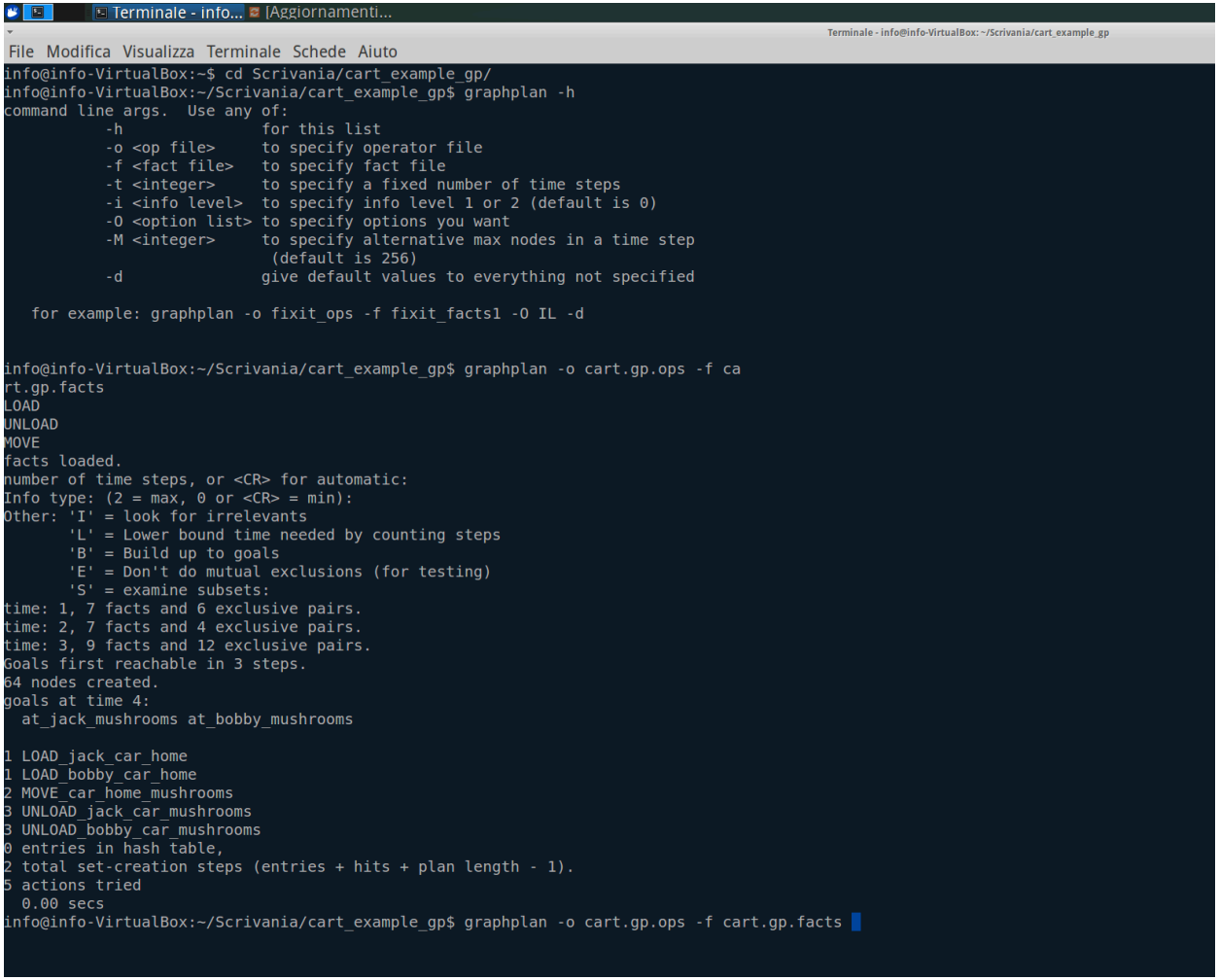
Exercise

Construction of the graph...



Exercise

Construction of the graph...



Planning Domain Definition Language

Planning Domain Definition Language

Other planners (e.g. Blackbox and FF) require that the files of the facts and actions are in PDDL.

PDDL = Planning Domain Definition Language
Attempt to standardizes a language for modeling planning problems

- Developed to support the International Planning Competition

A PDDL definition consists of two parts

- A domain definition for predicates and actions
- A problem file for the initial state and the goal specification

Typically in separate files

PDDL - Domain

Approximate syntax of a domain definition

```
(define (domain DOMAIN_NAME)
(:requirements [:strips] [:equality] [:typing] [:adl])
[(:types NAME_1 ... NAME_N)]
(:predicates
      (PREDICATE_1_NAME [?A1 ?A2 ... ?AN])
      ...
)

(:action ACTION_1_NAME
[:parameters (?P1 ?P2 ... ?PN)]
[:precondition PRECOND_FORMULA]
[:effect EFFECT_FORMULA] )
...
)
```

PDDL - Domain

Approximate syntax of a domain definition

```
(define (domain DOMAIN_NAME)
(:requirements [:strips] [:equality] [:typing] [:adl])
[(:types NAME_1 ... NAME_N)]
(:predicates
      (PREDICATE_1_NAME [?A1 ?A2 ... ?AN])
      ...
)

(:action ACTION_1_NAME
[:parameters (?P1 ?P2 ... ?PN)]
[:precondition PRECOND_FORMULA]
[:effect EFFECT_FORMULA] )
...
)
```

Object types that are part of the domain

PDDL - Domain

Approximate syntax of a domain definition

```
(define (domain DOMAIN_NAME)
(:requirements [:strips] [:equality] [:typing] [:adl])
[(:types NAME_1 ... NAME_N)]
(:predicates
        (PREDICATE_1_NAME [?A1 ?A2 ... ?AN])
        ...
)

(:action ACTION_1_NAME
[:parameters (?P1 ?P2 ... ?PN)]
[:precondition PRECOND_FORMULA]
[:effect EFFECT_FORMULA] )
...
)
```

Types of predicate; eg at (home, car)

PDDL - Domain

Approximate syntax of a domain definition

```
(define (domain DOMAIN_NAME)
(:requirements [:strips] [:equality] [:typing] [:adl])
[(:types NAME_1 ... NAME_N)]
(:predicates
      (PREDICATE_1_NAME [?A1 ?A2 ... ?AN])
      ...
)

(:action ACTION_1_NAME
[:parameters (?P1 ?P2 ... ?PN)]
[:precondition PRECOND_FORMULA]
[:effect EFFECT_FORMULA] )
...
)
```

operators Description

PDDL - predicates and actions

Syntax for variables with type:

```
?<Var name> - <type>
```

Syntax of a formula

Aggregation of predicates using logical operators:

```
(and PREDICATE_1 ... PREDICATE_N)
(or PREDICATE_1 ... PREDICATE_N)
(not PREDICATE)
```

Effects

Add and delete lists are aggregated (the delete effect are preceded by a "not" :

```
:effect FORMULA
```

PDDL - .facts File

```
(define (<problem name>)
        (:domain <reference domain>)
        (:objects
                {<object name> - <type>}
        )
        (:init
                {<predicate in the initial state>}
        )
        (:goal
                (and {<predicate in the final state>})
        )
)
```

Input of a planner = two files:

- Description of actions and types of objects (domain, .pddl file)
- Description of the initial state and the goal (.facts file)

SATPLAN & Blackbox

SATPLAN & Blackbox

SATsfability problem:
Decide if a logical formula is satisfiable

Declarative approach:

1. We model a problem by:
   - logical variables (0-1 domain)
   - operators and, or, not
2. Using a solver generic to find an assignment of the variables that satisfies all the constraints

Why SAT?

Because there are very efficient SAT solvers that would become usable as planners

Exercise

Start blackbox

- Blackbox is pre-installed on the VM
- Try to run it with:

  info@info-VirtualBox:~$ `blackbox`

- Run the program using as input the two .pddl and .facts files that you have found in the "Cart Example" folder:

  `blackbox -o <.pddl file> -f <.facts file>`

Exercise

```
info@info-VirtualBox:~/Scrivania/cart_example$ blackbox -o cart.pddl -f cart.facts
blackbox version 43
command line:  blackbox -o cart.pddl -f cart.facts

Begin solver specification
    -maxint        0   -maxsec 10.000000  graphplan
    -maxint        0   -maxsec 0.000000  chaff
End solver specification
Loading domain file: cart.pddl
Loading fact file: cart.facts
Problem name: cart-example
Facts loaded.
time: 1, 7 facts and 6 exclusive pairs.
time: 2, 7 facts and 4 exclusive pairs.
time: 3, 9 facts and 12 exclusive pairs.
Goals first reachable in 3 steps.
64 nodes created.

################################################
goals at time 4:
  at-crg_jack_mushrooms at-crg_bobby_mushrooms


------------------------------------------------
Invoking solver graphplan
Result is Sat
Iteration was 12
Performing plan justification:
   0 actions were pruned in 0.00 seconds

------------------------------------------------
Begin plan
1 (load jack car home)
1 (load bobby car home)
2 (move car home mushrooms)
3 (unload bobby car mushrooms)
3 (unload jack car mushrooms)
End plan
------------------------------------------------

5 total actions in plan
0 entries in hash table,
2 total set-creation steps (entries + hits + plan length - 1)
5 actions tried

################################################
Total elapsed time:   0.00 seconds
Time in milliseconds: 0

################################################
info@info-VirtualBox:~/Scrivania/cart_example$
```

FF: Fast Forward

FF: Fast Forward - Exercises

Fast Forward

FF is an extremely efficient heuristic planner introduced by Hoffmann in 2000

- Heuristic = at each state S is an evaluation of the distance from the goal using a heuristic function

Basic operation:

1. Starting from a state S, all successors S' are examined
2. If you find a successor state S* better than S, move to it and go back to step 1
3. If no state is found with better evaluation, a complete search is performed A*, using the same heuristic

FF: Fast Forward - Exercises

Start FF

- On the VM Fast Forward is also pre-installed
- Try to run it on the example of the trolley with:

```
info@info-VirtualBox:~$  ff -f <dataFile>
                         -o <operatorFile>
```

- Try running the program using as input the two .pddl and and .facts files from the same classic example used with graphplan

FF: Fast Forward - Exercises

```
info@info-VirtualBox:~/Scrivania/cart_example$ ff -f cart.facts -o cart.pddl

ff: parsing domain file
domain 'CART' defined
 ... done.
ff: parsing problem file
problem 'CART_EXAMPLE' defined
 ... done.



Cueing down from goal distance:    5 into depth [1]
                                   4            [1]
                                   3            [1]
                                   2            [1]
                                   1            [1]
                                   0

ff: found legal plan as follows

step    0: LOAD JACK CAR HOME
        1: LOAD BOBBY CAR HOME
        2: MOVE CAR HOME MUSHROOMS
        3: UNLOAD JACK CAR MUSHROOMS
        4: UNLOAD BOBBY CAR MUSHROOMS


time spent:    0.00 seconds instantiating 12 easy, 0 hard action templates
               0.00 seconds reachability analysis, yielding 9 facts and 12 actions
               0.00 seconds creating final representation with 9 relevant facts
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 6 states, to a max depth of 1
               0.00 seconds total time

info@info-VirtualBox:~/Scrivania/cart_example$
```

References

- **GRAPHPLAN**
  - http://www.cs.cmu.edu/~avrim/graphplan.html
  - A. M. Blum and Furst, "Fast Planning Planning Through Graph Analysis", Artificial Intelligence, 90: 281-300 (1997).
- **Blackbox**
  - Henry Kautz and Bart Selman, "Unifying SAT-based and Graph-based Planning", Proc. IJCAI-99, Stockholm, 1999.
- **Fast Forward**
  - J. Hoffmann, "FF: The Fast-Forward Planning System", in: AI Magazine, Volume 22, Number 3, 2001, Pages 57-62

Exercise 1

Modeling in the PDDL the **nine puzzle problem**

initial state

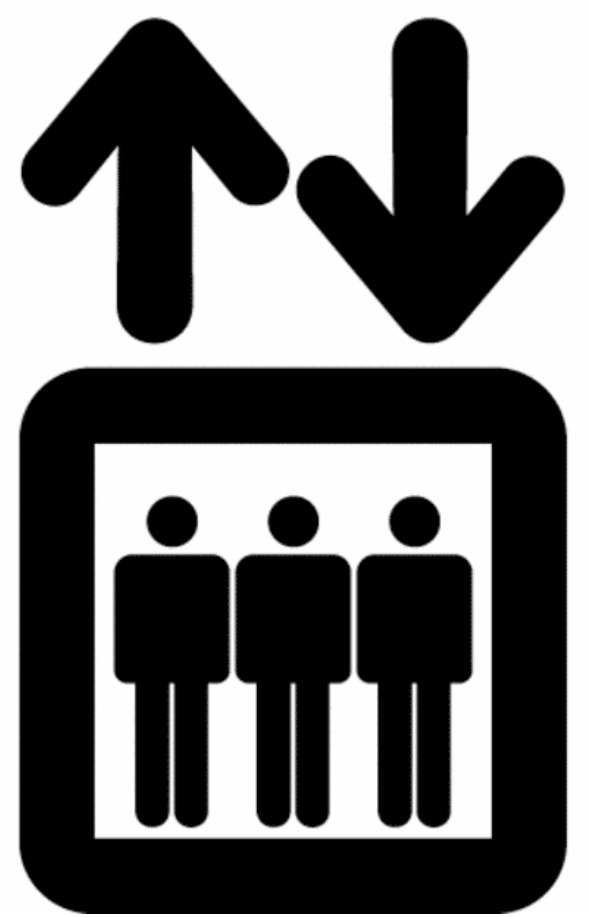
7	3	4
2		1
8	6	5

Goal

1	2	3
4	5	6
7	8	

- Try to solve it with the various planners
- How to vary performances by "mixing" a little '?

Exercise 2



- A building has 5 floors, 5 tenants and two lifts (currently at the ground floor and at the 4th floor)
- Each tenant is at a starting floor and must go to another floor
- We need to find a plan that moves each tenant to its destination and that requires the minimum possible time (Load / unload the lift takes the same time to move up one level).

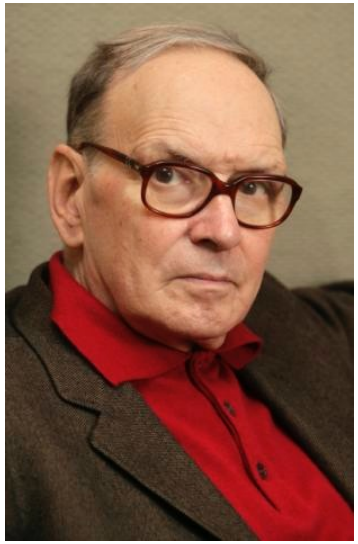
Model the problem in PDDL and solve it!!

Exercise 2

the young **Enrico** is at the ground floor after a hard morning at work and is finally back home on the 3rd floor

Elena is at the ground floor should bring her shopping bags on the 2nd floor

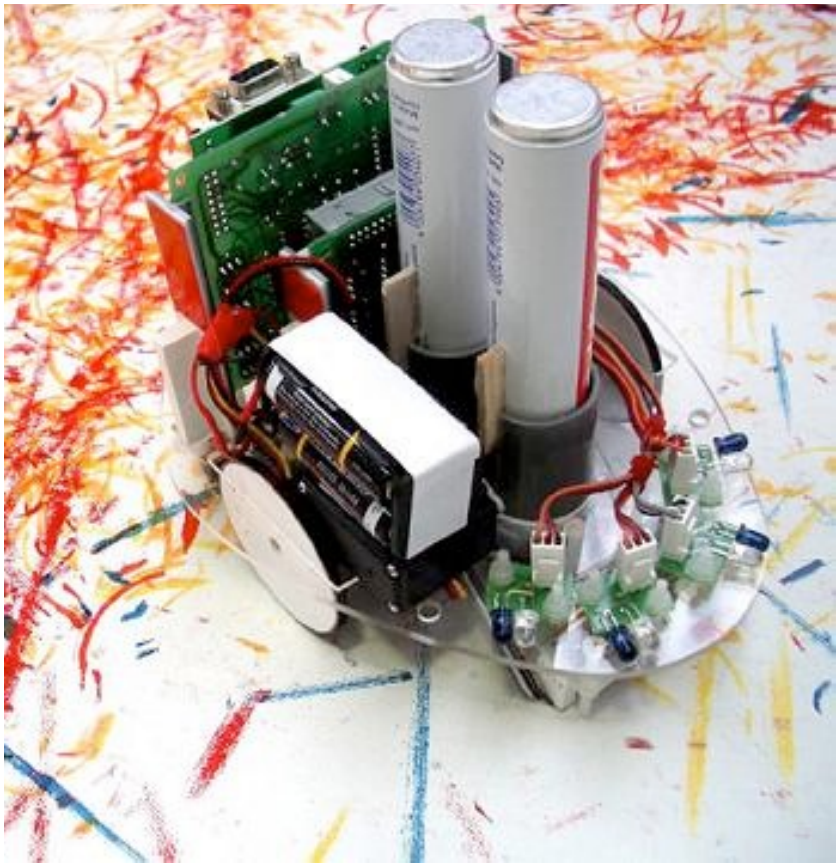
After a family lunch on the 3rd floor, **Ettore** should go back to his workshop on the ground floor



Elvira and **Ennio** are on the 1st (to play cards) and on the 2nd (composing music) and should go back home on the 4th floor

Exercise 3

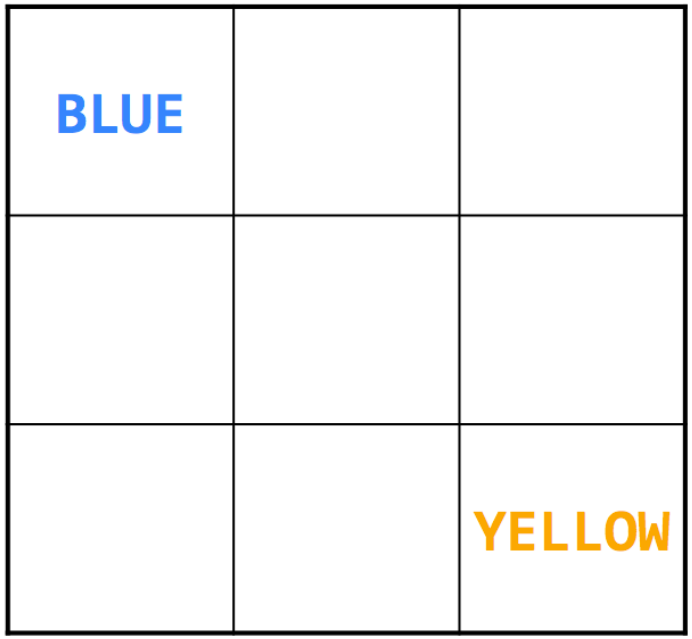
Two robots have to paint the tiles on a floor:



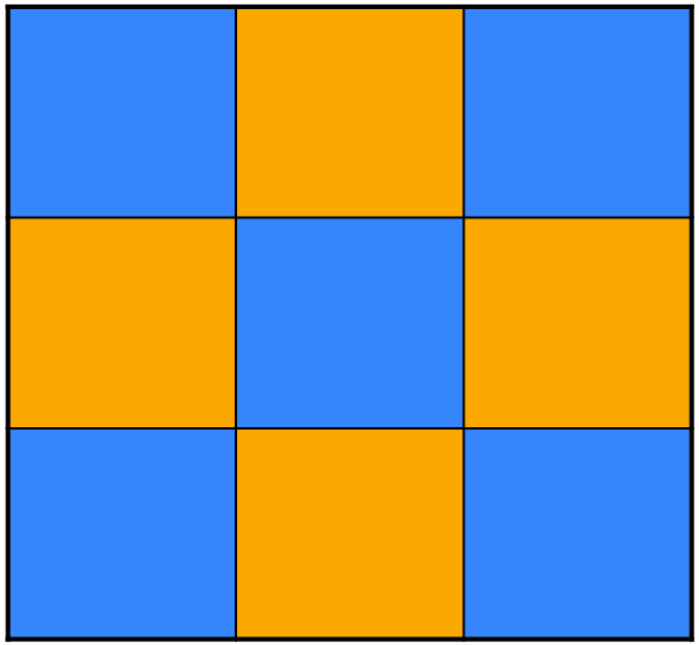
- Each robot can move between adjacent tiles
- Each robot is loaded with a specific color and can use the color on the tile it is located to.
- Robots cannot pass on a painted tile
- The two robots can be on the same tile

Exercise 3

initial state



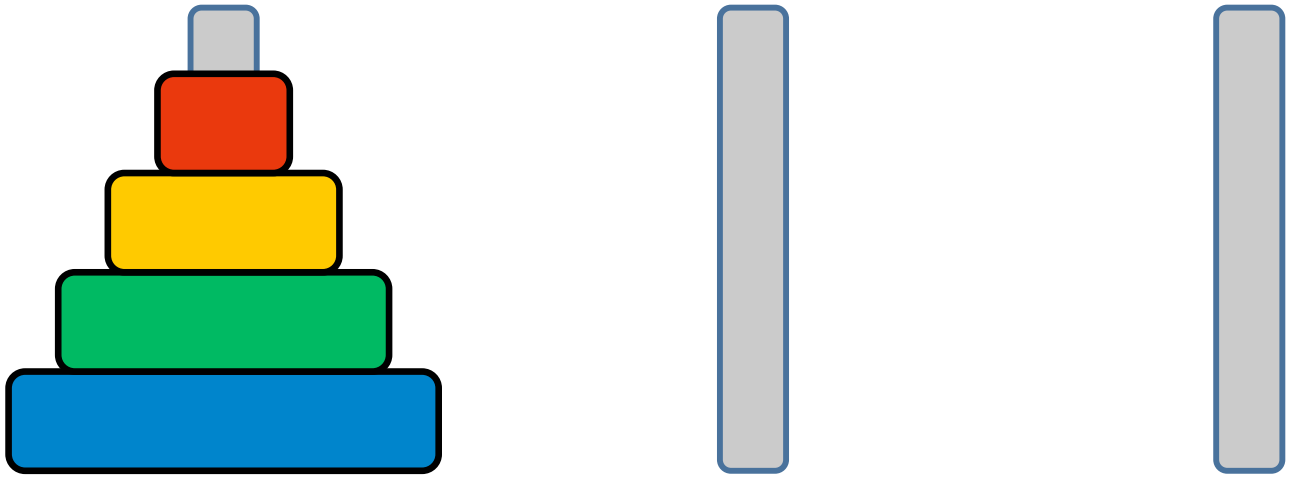
Goal



Note that a robot can color a tile, even if the other robot there is located above at the time

Exercise 4

Shape in the puzzle of PDDL **Hanoi Tower**



- Move a circle at a time, from one busbar
- A smaller circle can never go above a larger one

Exercise 4

Goal: