Course "Automated Planning: Theory and Practice"
Chapter Lab 01: PDDL Planning Exercices

Teacher: Marco Roveri - marco.roveri@unitn.it

M.S. Course: Artificial Intelligence Systems (LM)
A.A.: 2025-2026
Where: DISI, University of Trento
URL: https://shorturl.at/A81hf

Last updated: Wednesday 1st October, 2025

Terms of Use and Copyright

The material (text, figures) in these slides is authored by Jonas Kvarnström and Marco Roveri.

Hands-on Session

- We will create and solve planning problems
- On your laptops!
  - Experiment with command line planners
    - Planutils (several pre-compiled planners on Linux)
      `https://pypi.org/project/planutils/`
      `https://github.com/AI-Planning/planutils`
    - Fast Downward[1] `https://www.fast-downward.org/`
  - Experiment with online planners
    - `http://editor.planning.domains`

[1] Instructions to compile it for Linux/Window/MacOS X `https://github.com/aibasel/downward/blob/main/BUILD.md`

Useful Tips for PLANUTILS

- Be sure to use Python version $\geq$ 3.6.*
- To install `pip` or `pip3` (in Debian like Linux distributions e.g. Ubuntu):
  - Run `apt-get install python-pip` or `apt-get install python3-pip`
- Run `planutils list` for a list of available planners
- To install a planner run `<name>` using names from previous command
- Under Linux singularity do not mount all the host files by default.
  - Edit file `/usr/local/etc/singularity/singularity.conf` (or `/etc/singularity/singularity.conf`) changing `mount hostfs = no` to `mount hostfs = yes`.
- If for some reason the singularity framework is not installed by the `pip3 install planutils`, then install it manually as discussed in `https://docs.sylabs.io/guides/latest/user-guide/quick_start.html#quick-installation-steps`
  - Before running singularity, a reboot may be needed!
- To update planutils run `pip3 install --upgrade planutils` and then `planutils upgrade` to download latest versions of the singularity images!

Using Fast Downward

- `downward --help` for the full list of options
- `downward --show-alias` list of pre-defined configurations
- `downward --alias lama-first domain.pddl problem.pddl` (use pre-defined `lama-first` pre-defined configuration)
- `downward domain.pddl problem.pddl --search "astar(lmcut())"` to specify search/heuristic to be used (e.g. A* with landmarks) for fine tuning the planner!
  - See `https://www.fast-downward.org/Doc/SearchEngine`
  - See `https://www.fast-downward.org/Doc/Evaluator`

Specify the DWR domain

- Types: `location`, `pile`, `robot`, `crane`, `container`
- Constants: `pallet` of type `container`
- Predicates:

`adjacent`	location l1 is adjacent to l2
`attached`	pile p attached to location l
`belong`	crane k belongs to location l
`atl`	robot r is at location l
`free`	there is no robot at location l
`loaded`	robot r is loaded with container c
`unloaded`	robot r is empty
`holding`	crane k is holding a container c
`empty`	crane k is empty
`in`	container c is within pile p
`top`	container c is on top of pile p
`on`	container k1 is on container k2

- Actions:

`move`	moves a robot r between two adjacent locations from and to
`load`	loads an empty robot r with a container c held by a nearby crane k at location l
`unload`	unloads a robot r holding container c to a nearby crane k at location l
`take`	takes a container c on c1 from a pile p with a crane k at location l
`put`	puts a container c on c1 held by a crane k on a nearby pile p at location l

Specify the DWR domain (cont.)

- Actions:

`move`	moves a robot r between two adjacent locations from and to pre: adjacent from and to, robot r at from, free to
`load`	loads an empty robot r with a container c held by a nearby crane k at location l pre: robot r at l, belong k to l, holding k c, and unloaded r
`unload`	unloads a robot r holding container c to a nearby crane k at location l pre: robot r at l, belong k to l, loaded r with c, empty k
`take`	takes a container c on c1 from a pile p with a crane k at location l pre: belong k to l, attached p to l, empty k, c in p, top c of p, on c c1
`put`	puts a container c on c1 held by a crane k on a nearby pile p at location l pre: belong k to l, attached p to l, holding k c, top c1 p

Specify a DWR problem

- Objects:

`r1`	robot
`l1 l2`	location
`k1 k2`	crane
`p1 q1 p2 q2`	pile
`ca cb cc cd ce cf`	container

- Initial state:

`adjacent`	(l1,l2), (l2,l1)
`attached`	(p1. l1), (q1, l1), (p2,l2), (q2, l2)
`belong`	(k1, l1), (k2, l2)
`atl`	(r1, l1)
`free`	(l2)
`loaded`	
`unloaded`	(r1)
`holding`	
`empty`	(k1), (k2)
`in`	(ca, p1), (cb, p1), (cc, p1), (cd, q1), (ce, q1), (cf, q1)
`top`	(cc, p1), (cf, q1), (pallet p2), (pallet q2)
`on`	(ca, pallet), (cb, ca), (cc, cb), (cd, pallet), (ce, cd), (cf, ce)

- Goal: `in` (ca,p2), (cb,q2), (cc,p2), (cd,q2), (ce,q2), (cf,q2)

DWR Domain and Problem

- PDDL Domain file:
  { `Examples/dwr/dwr-domain.pddl` }
- PDDL Problem file:
  { `Examples/dwr/dwr-pb1.pddl` }

Specify the MICONIC Domain

- Types: `passenger`, `floor`
- Predicates:

`notboarded`	not boarded passenger p
`down`	down floor f
`boarded`	boarded passenger p
`depart`	depart passenger p from floor f
`notserved`	not server passenger p
`origin`	origin passenger p from floor f
`board`	board passenger p at floor f
`lift-at`	lift at floor f
`served`	served passenger p
`destin`	destination of passenger p is floor f
`up`	up floor f
`above`	floor f1 above floor f2

- Actions:

`down`	moves lift down from floor f1 to floor f2
`load`	board passenger p at floor l
`up`	move lift up from floor f1 to floor f2
`depart`	passenger p depart from floor f

Specify the MICONIC Domain (cont.)

- Actions:

`down`	moves lift down from floor f1 to floor f2 pre: above f2 of f1, down f2, lift at f1
`load`	board passenger p at floor l pre: board passenger p at floor f, lift at f, origin passenger floor f
`up`	move lift up from floor f1 to floor f2 pre: above f1 of f2, lift at f, up f2
`depart`	passenger p depart from floor f pre: boarded passenger p, depart passenger p at floor f, lift at f, destination of p is f

Specify a MICONIC problem

- Objects: `f0 f1 f2 f3 f4 f5` floor
  `p0 p1 p2` passenger
- Initial state:

`notboarded`	
`down`	f0, f1, f2, f3, f4, f5
`boarded`	
`depart`	(f0,p0), (f0,p1), (f0,p2), (f1,p0), (f1,p1), (f1,p2), (f2,p0), (f2,p1), (f2,p2), (f3,p0), (f3,p1), (f3,p2), (f4,p0), (f4,p1)
`notserved`	
`origin`	(p0,f1), (p1,f3), (p2,f5)
`board`	(f0,p0), (f0,p1), (f0,p2), (f1,p0), (f1,p1), (f1,p2), (f2,p0), (f2,p1), (f2,p2), (f3,p0), (f3,p1), (f3,p2), (f4,p0), (f4,p1)
`lift-at`	f0
`served`	
`destin`	(p0, f4), (p1,f1), (p2,f1)
`up`	f0, f1, f2, f3, f4, f5
`above`	(f0,f1), (f0,f2), (f0,f3), (f0,f4), (f0,f5), (f1,f2), (f1,f3), (f1,f4), (f1,f5), (f2,f3), (f2,f4), (f2,f5), (f3,f4), (f3,f5), (f4,f

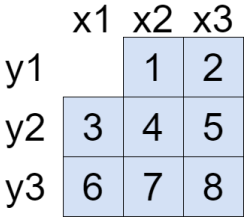
- Goal: `served` p0, p1, p2

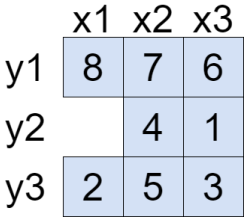
MICONIC Domain and Problem

- PDDL Domain file:
  { `Examples/elevator/elevator.pddl` }
- PDDL Problem file:
  { `Examples/elevator/problem.pddl` }

SlideTile Domain

- The sliding-tile puzzle (i.e. the eight/fifteen puzzle).
- Tile positions are encoded by a predicate `(at <tile> <x> <y>)`, i.e, using one object for horizontal position and one for vertical (there's a separate predicate for the position of the blank.
- The predicates `inc` and `dec` encode addition/subtraction of positions.
- Four actions: `move-up`, `move-down`, `move-left`, `move-right` with obvious preconditions (empty destination and there is a successor/predecessor)





SlideTile Domain and problems

- PDDL Domain file:
  { `Examples/slidetile/slidetile.pddl` }
- PDDL problem file:
  { `Examples/slidetile/eight01.pddl` }
- PDDL problem file:
  { `Examples/slidetile/eight01x.pddl` }

Blocks World with Derived Predicates

- $above(x, y) \leftrightarrow (on(x, y) \vee \exists z.(on(x, z) \wedge above(z, y)))$
- $busy(x, y) \leftrightarrow (handfull(x) \wedge holding(y))$
- PDDL Domain file:
  { `Examples/derived/derivedblocks.pddl` }
- PDDL Problem file:
  { `Examples/derived/problem.pddl` }

Fridge Domain with Exists/Forall

- PDDL Domain file:
  { `Examples/fridge/fridge.pddl` }
- PDDL problem file:
  { `Examples/fridge/problem.pddl` }

Citycar Domain with conditional effects

- PDDL Domain file:
  { `Examples/citycar/domain.pddl` }
- PDDL problem file:
  { `Examples/citycar/problem.pddl` }

Assembly Domain with conditional effects

- PDDL Domain file:
  { `Examples/assembly/domain.pddl` }
- PDDL problem file:
  { `Examples/assembly/problem.pddl` }

References I

[1] Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2013. ISBN 9781608459698. doi: 10.2200/S00513ED1V01Y201306AIM022. URL https://doi.org/10.2200/S00513ED1V01Y201306AIM022.

[2] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning - theory and practice*. Elsevier, 2004. ISBN 978-1-55860-856-6.

[3] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated Planning and Acting*. Cambridge University Press, 2016. ISBN 978-1-107-03727-4. URL http://www.cambridge.org/de/academic/subjects/computer-science/artificial-intelligence-and-natural-language-processing/automated-planning-and-acting?format=HB.

[4] Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, and Christian Muise. *An Introduction to the Planning Domain Definition Language*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2019. doi: 10.2200/S00900ED2V01Y201902AIM042. URL https://doi.org/10.2200/S00900ED2V01Y201902AIM042.