Welcome!

AI for Robotics II

(if you are in the wrong call, you still have time to leave..)

University of HUDDERSFIELD
Inspiring tomorrow's professionals

Where is Huddersfield?

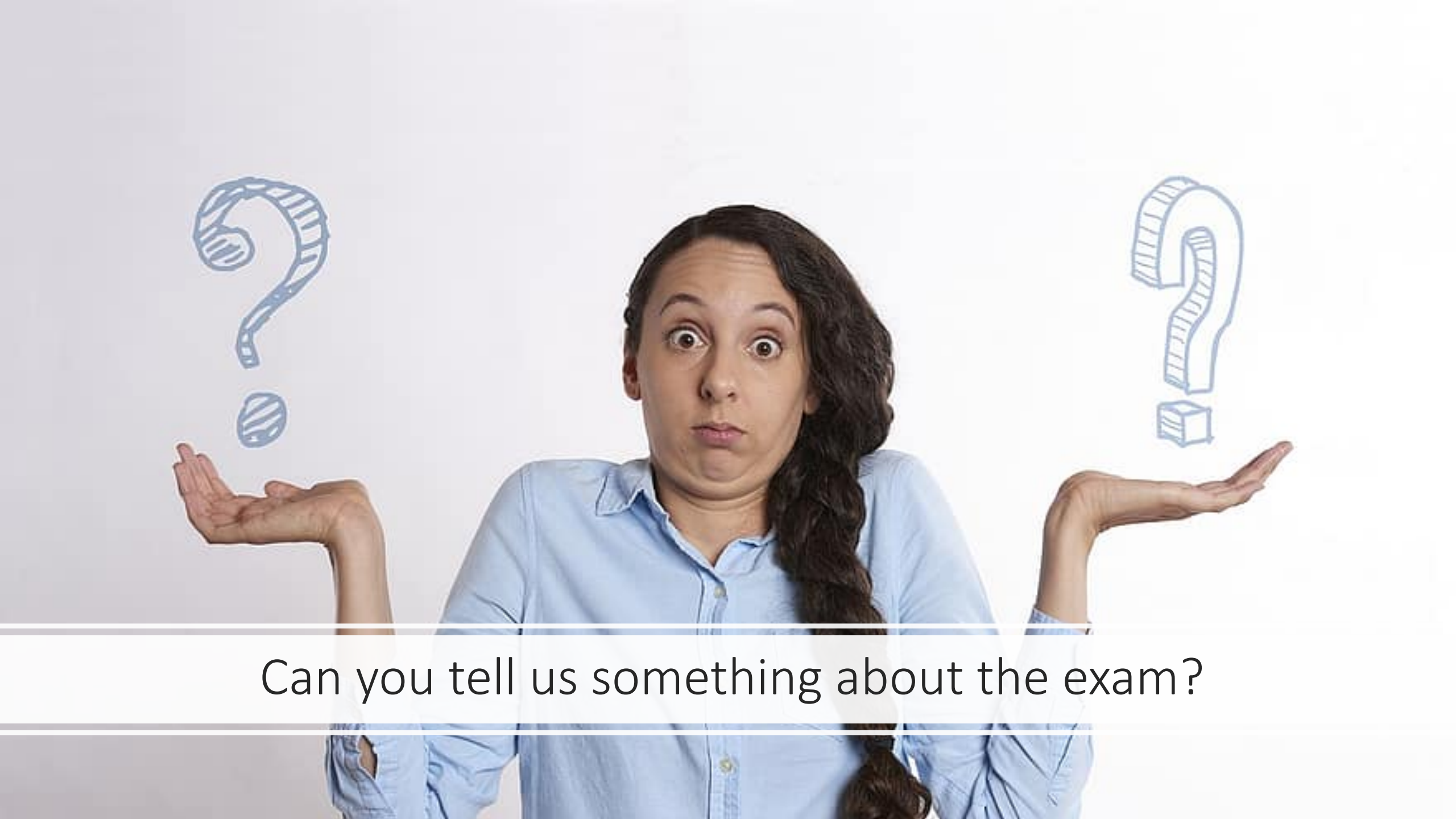
Who am I?

Reader in AI

http://maurovallati.blogspot.com/

Structure of this module

- AI Planning
  - Overview of Planning
  - Approaches for Solving Classical Planning
  - Beyond Classical Planning
  - Hybrid Planning
  - Knowledge Engineering for Planning
- Fulvio!
  - …..



Can you tell us something about the exam?

What is Automated Planning?

(and who cares?) -- a quick recap of planning, transition systems, planning task, and PDDL

Automated Planning

planning is reasoning about actions

Planning is the art and practice of thinking before acting.
-- Patrick Haslum

Selecting a goal-leading course of action based on a high-level description of the world.
— Joerg Hoffmann

Who cares?

- Planning is a pivotal aspect of what is termed “Intelligence”
  - We don’t have a definition of intelligence, but we know some of the components.
- 2 main reasons for studying planning
  - **Scientific goal of AI**: understand Intelligence
  - **Engineering goal of AI**: build autonomous agents (ring a bell?)
    - Planning helps agents in choosing and organising actions to achieve goals.
- Planning is one of the major subfields of AI.
  - Represented at major conferences (IJCAI, AAAI, ECAI)
  - Specialised conference (ICAPS)
  - Represented in major academic journals (AIJ, JAIR, …)

Do we (humans) plan?

- Humans rarely plan before acting in routine situations:
  - Immediate goals to achieve (turn on light)
  - Well-trained behaviour (driving a car)

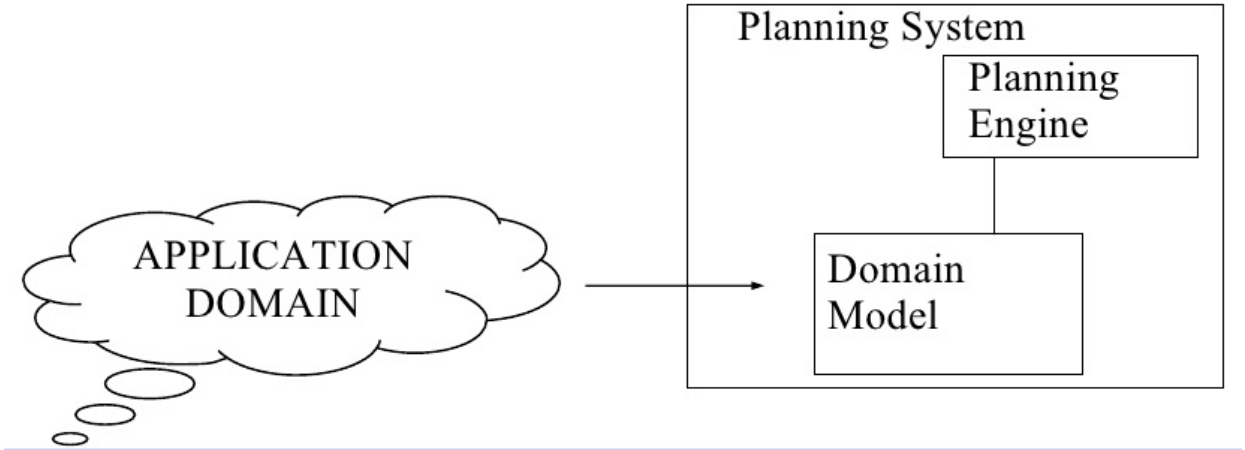
- But they do in other situations:
  - Complex tasks (decide what to put in this slide)
  - New situation (new work, move house)
  - High risk/cost environment (nuclear power station, oil drilling)

Two main classes of approaches

- Domain-specific
  - Use of specific techniques, approaches and models that are adapted for the application domain.
    - Examples: manipulation planning, perception planning, etc.
- Domain-independent
  - Very generic representation and techniques that can be used in “any” domain
    - Leads to general understanding of planning

We will focus on domain-independent planning!

Domain-independent Planning



Overall aim:
Create one planning approach that performs sufficiently well on any application domain (including future ones)

General Perspective

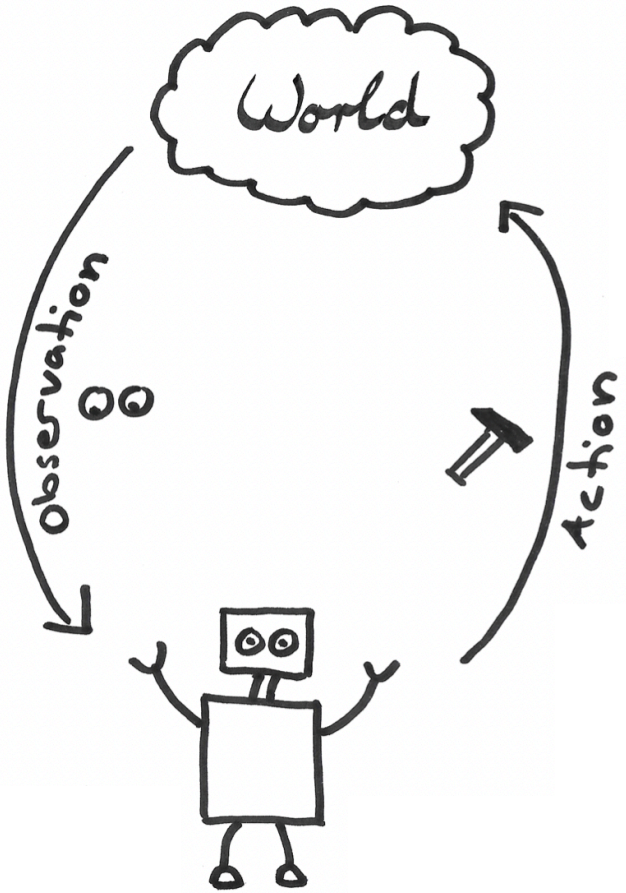


Image from: https://ai.dmi.unibas.ch/_files/teaching/hs19/po/slides/po-a02.pdf

General Perspective

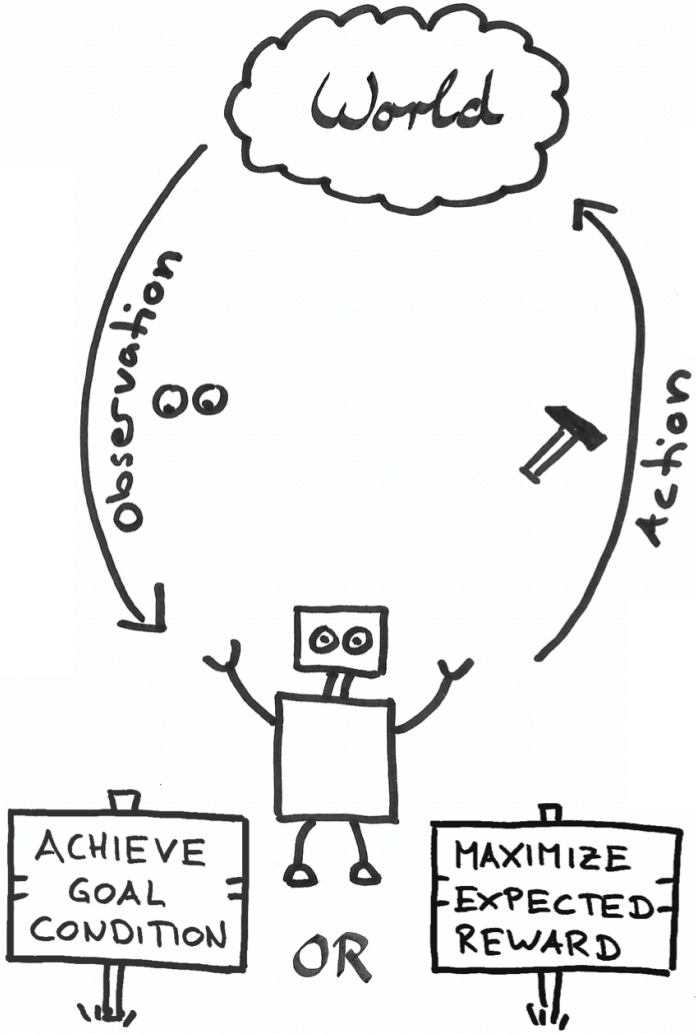


Image from: https://ai.dmi.unibas.ch/_files/teaching/hs19/po/slides/po-a02.pdf

General Perspective

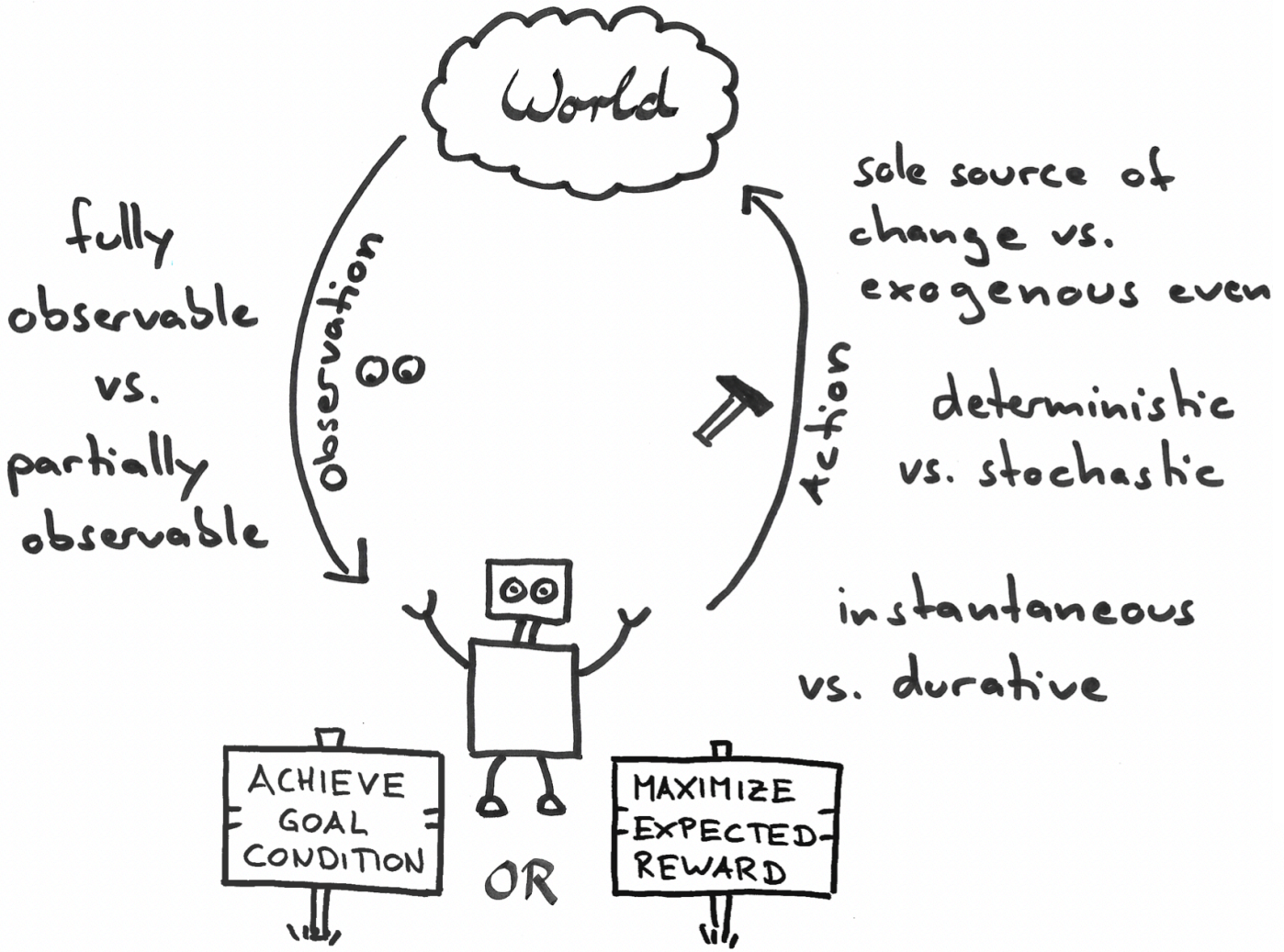
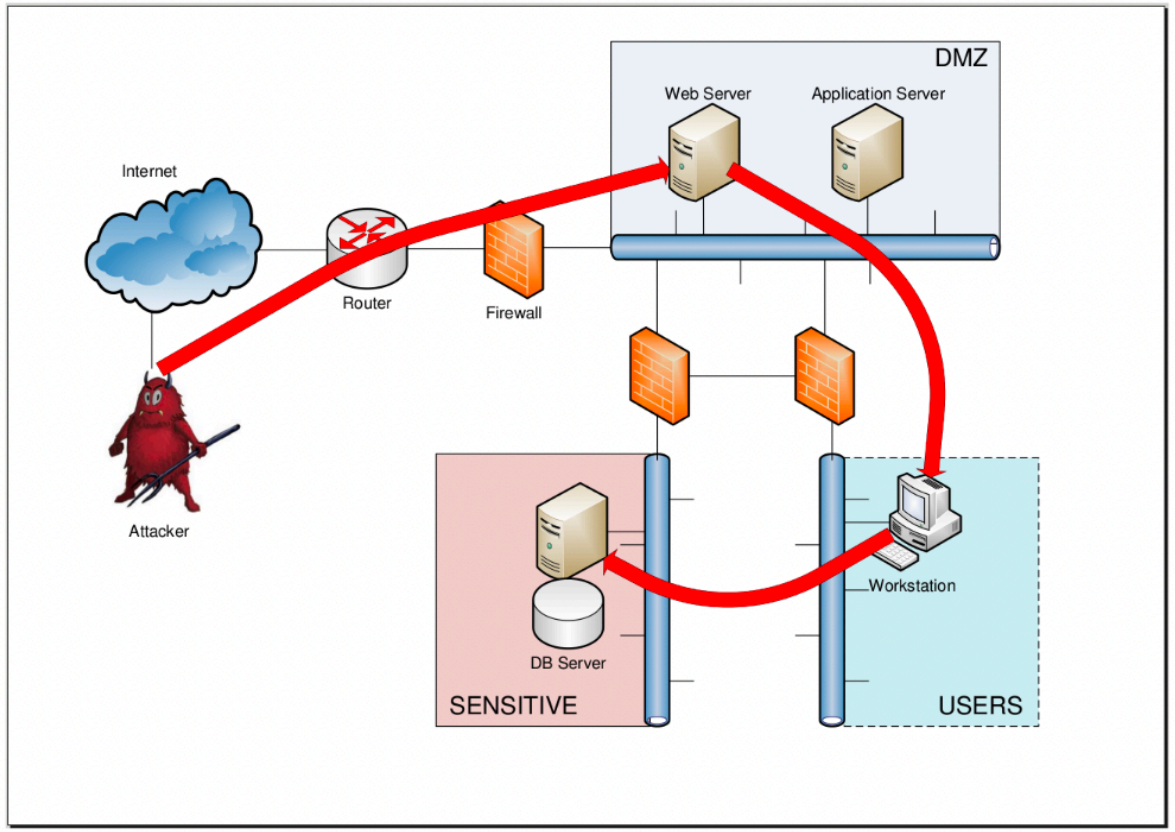


Image from: https://ai.dmi.unibas.ch/_files/teaching/hs19/po/slides/po-a02.pdf

Example: Cybersecurity

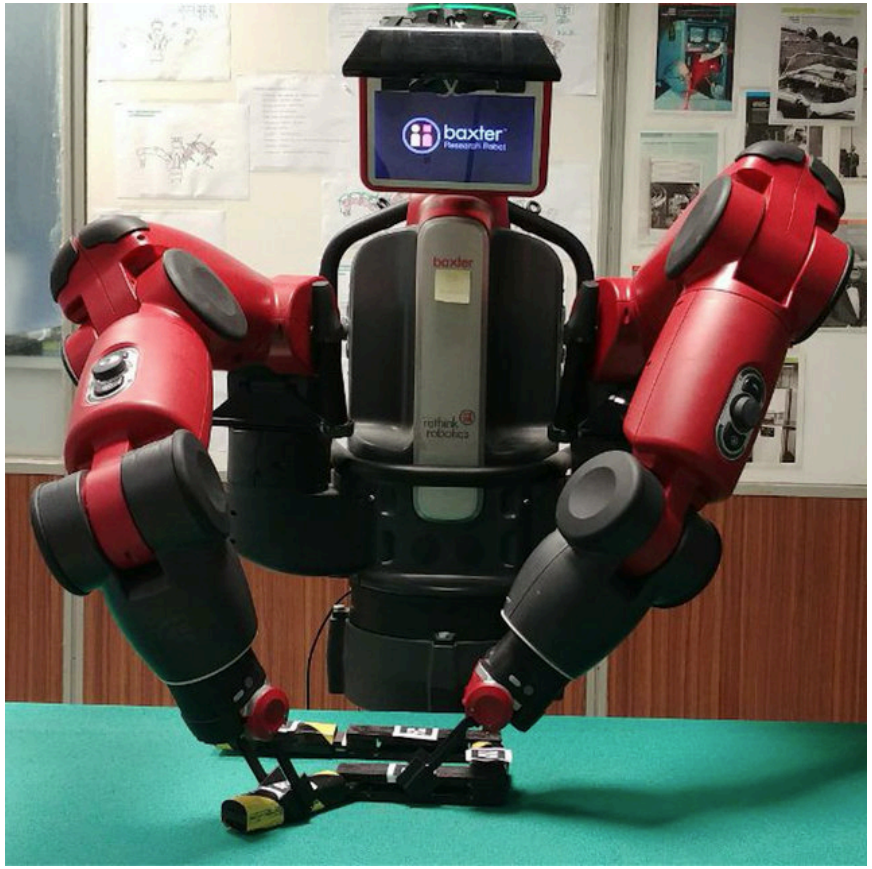


- CALDERA automated adversary emulation system
- Log analysis to identify potential breaches

Example: Urban Traffic Control

Automated generation of strategies to resolve issues such as congestion or poor air quality

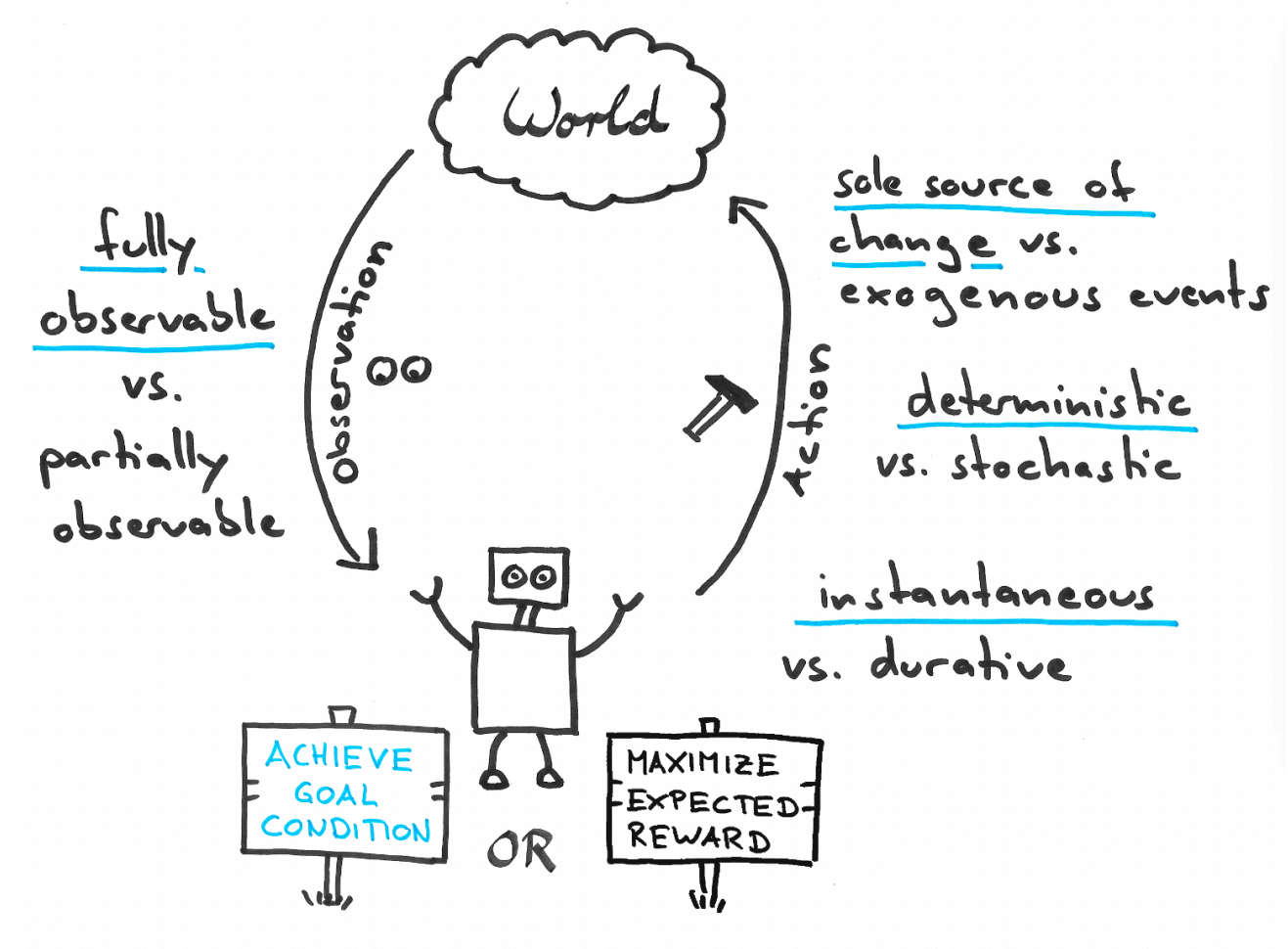
Example: Robotic Manipulation



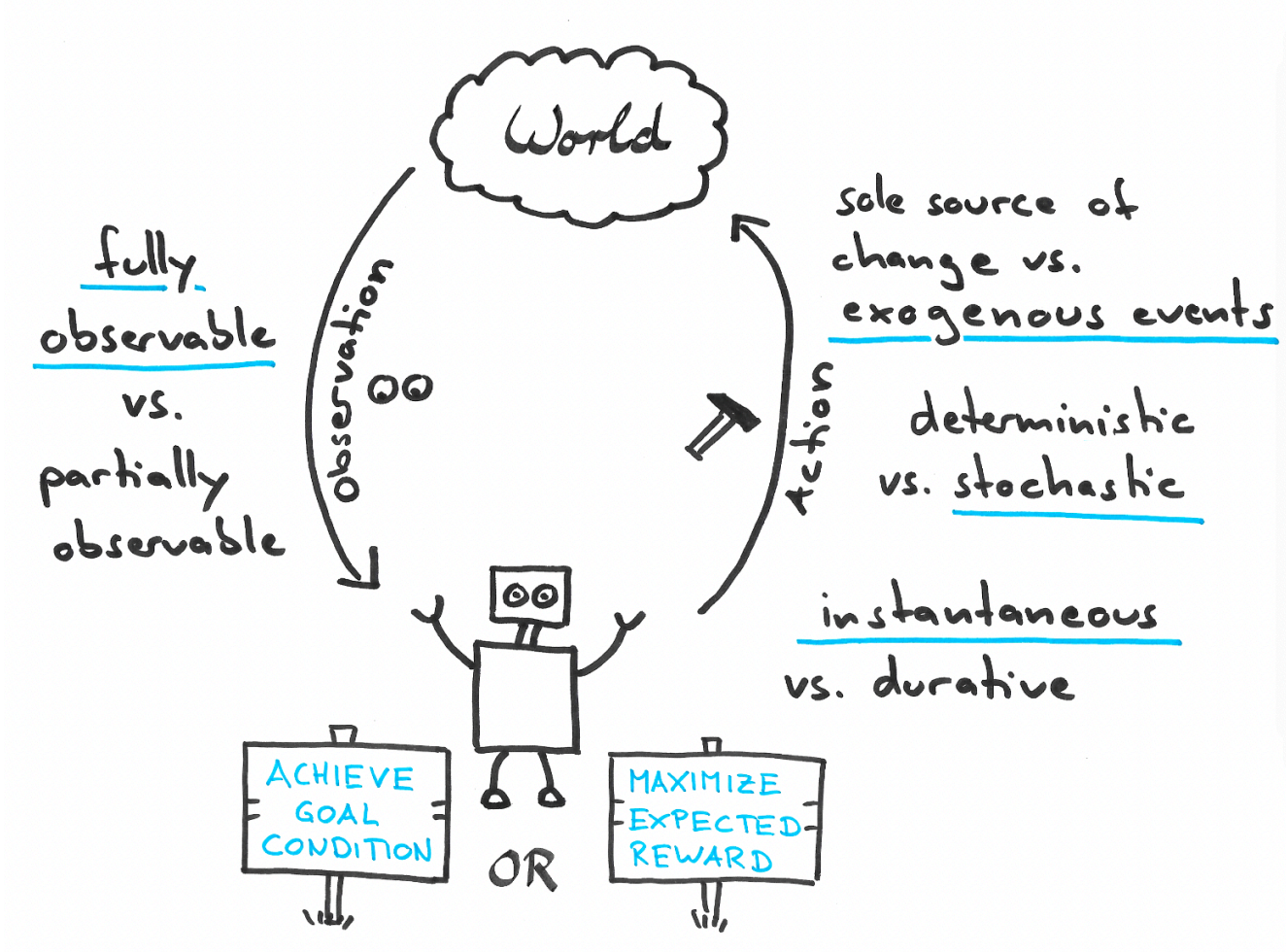
Generation of sequences of movements to allow the robot to manipulate an articulated object

https://www.youtube.com/watch?v=dMdzCB5FBMI

Classical Planning



Probabilistic Planning



Planning Task

- Input to provide for a planning engine: planning task
  - Initial state of the world
  - actions that can change the state
  - Goal to be achieved

- Output:
  - A solution plan (hopefully)
    - A sequence of actions that allows to achieve the goal starting from the initial state
  - A policy (probabilistic settings)
    - function that returns for each state the action to take

How Complex is Planning?

Is Planning Difficult?

- Generating a solution (plan existence) for classical planning is PSPACE-complete
  - Satisficing planning – generate any solution
- Bounded-cost plan existence is still PSPACE-complete
  - Optimal planning – generate an optimal solution according to some given criteria
- If we leave classical planning, we quickly move into EXPSPACE-completeness

Transition System

Simplified transition system

- A state-transition system is a tuple $\Sigma = (S, A, \gamma)$, where:
  - $S = \{s_1, s_2, \ldots\}$ is a finite set of states;
  - $A = \{a_1, a_2, \ldots\}$ is a finite set of actions;
  - γ: $S \times (A) \rightarrow 2^S$ is a state transition function.
- if $a \in A$ and $\gamma(s, a) \neq \varnothing$ then a is applicable in s
- applying a in s will take the system to $s' \in \gamma(s, a)$

Properties and terminology

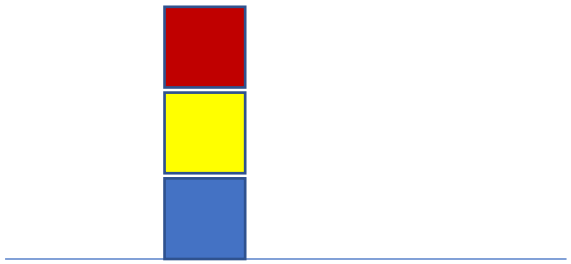
- A transition system is deterministic if for all states and all actions, there is at most one state s' $\gamma(s,a) = s'$

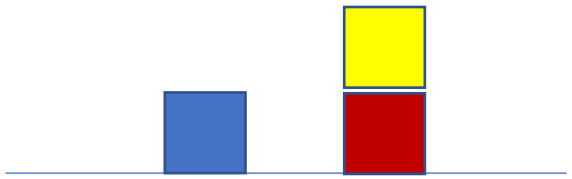
- s' is a **successor** of s
- s is a **precedessor** (parent) of s'

- A state s' is reachable if there exists a sequence of actions that, starting from a given initial state, allows to reach s'
- A solution, is a path connecting a given initial state to one of the goal states

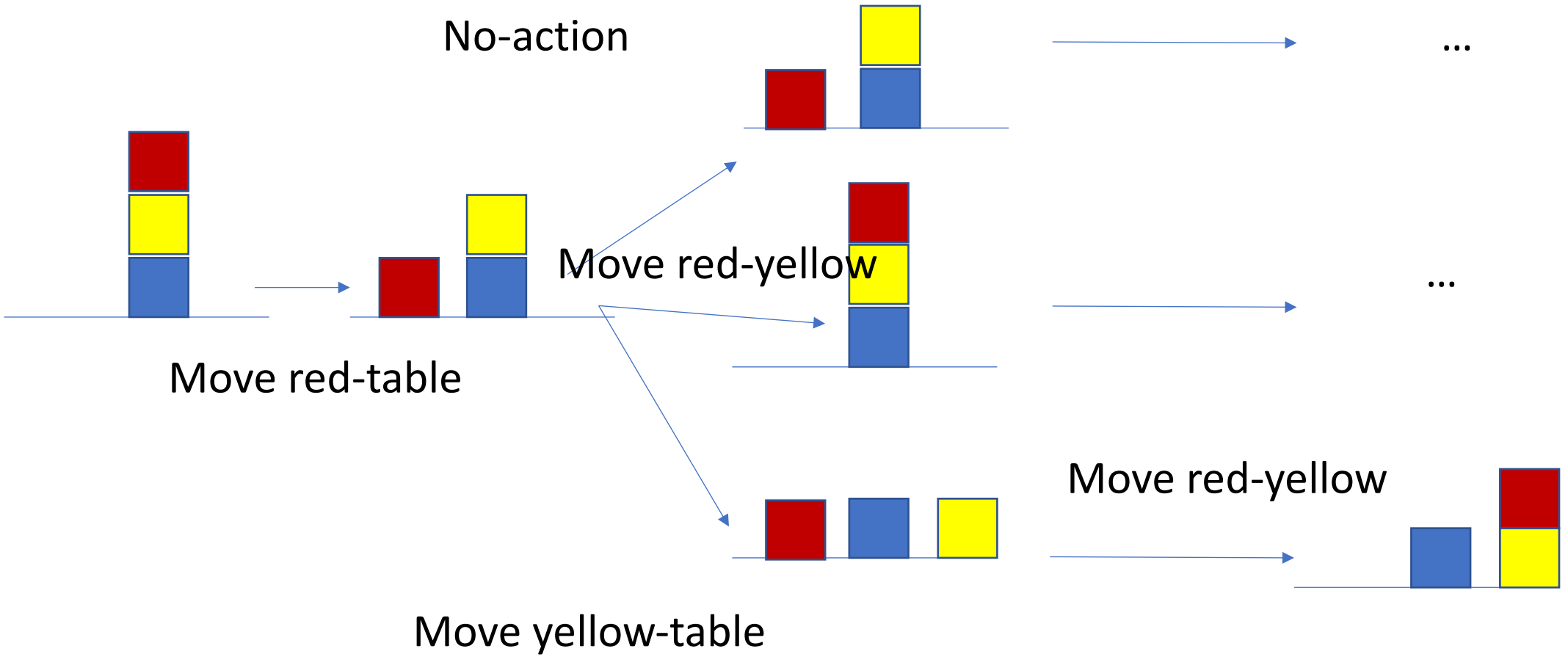
Example

- Blocksworld:
  - Some blocks in an initial condition
  - Actions: move blocks
  - Goal: a given configuration





Example (1)

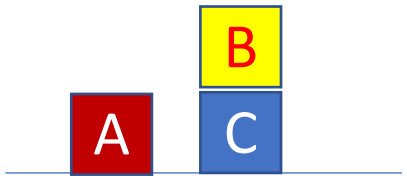

No-action
...
Move red-yellow
...
Move red-table
Move red-yellow
Move yellow-table

Planning Task

State variables

- How to specify huge transition systems without enumerating the states?
  - represent different aspects of the world in terms of different state variables
  - individual state variables induce atomic propositions
    - a state of the transition system is a valuation of state variables
  - n Boolean state variables induce 2^n states
    - exponentially more compact than “flat” representations, where you have one variable per state

Blocksworld with propositional state variables



- s(A-on-B) =F
- s(A-on-C) =F
- s(A-on-table) =T
- s(B-on-A) =F
- s(B-on-C) =T
- s(B-on-table) =F
- s(C-on-A) =F
- s(C-on-B) =F
- …..

Propositional state variable

A propositional state variable is a symbol X.

Let V be a finite set of propositional state variables.

- A state s over V is a valuation for V, i.e.,
  a truth assignments: V→{**T**,**F**}.
- A formula over V is a propositional logic formula using V as the set of atomic propositions.

From state variables to transition systems

Use formulas to describe sets of states

- states: all assignments to the state variables
- goal states: defined by a formula
- transitions: defined by operators

- An operator o over state variables V is an object with three properties:
  - a precondition pre(o), a formula over V
  - an effect eff(o) over V
  - a cost cost(o)$\in$R+

Intuition on Operators

- The operator precondition describes the set of states where the operator can be applied.
- The operator effect describes how taking such a transition changes the state s into s’.
- The operator cost describes the cost of applying the operator

PDDL

What is PDDL?

- To solve a problem automatically, you need to design an input in a language that your computer can read.
- That language is PDDL: The Planning Domain Definition Language.
- **Why PDDL**?
  - it is the de-facto standard language in the planning area.
  - It is used in International Planning Competitions!

We do need a standard language so that we can share info and talk to each other…

PDDL Basics

- Syntax is (strongly) inspired by Lisp

```
(and (or (on A B) (on A C))
     (or (on B A) (on B C))
      (or (on C A) (on A B)))
```

The input is divided into 2 files:

- A domain file (information about the domain and the application; predicates, types, operators)
  - Lifted representation, using variables
- A problem file (the actual problem to solve; initial state, goal, objects)
  - The actual objects that are used in the planning task, and their initial states

PDDL ideas

- The language is focused around the notion of predicates, which are properties of the world that we do care about for the sake of the problems at hand.
  - State variables!
- Going back to the state notion, a state is completely defined by the value of the predicates
- The value of predicates can be changed by applying actions
- Goals are described in terms of predicates that we do care about

PDDL domain files

- A PDDL domain file consists of:
  1. `(define (domain name)`
  2. A requirements definition (use “:adl :typing” by default).
  3. Definitions of types (each object variable can have a type).
  4. Definitions of predicates.
  5. Definitions of operators.

Types and Predicates: Gripper

```
(define (domain gripper)
   (:requirements :adl :typing)
   (:types room ball gripper)
   (:predicates (at-robby ?r1 - room)
                (at ?b - ball ?r - room)
                (free ?g - gripper)
                   (carry ?o - ball ?g - gripper))
```

Syntax: Operator

- (:action <name>
- List of parameters:
  - (?x - type1 ?y - type2 ?z - type3)
- The precondition is a formula:
  - <predicate>
  - (and <formula> ... <formula>)
  - (or <formula> ... <formula>)
  - (not <formula>)
  - (forall (?x1 - type1 ... ?xn - typen) <formula>)
  - (exists (?x1 - type1 ... ?xn - typen) <formula>)

Syntax: Operator (2)

- The effect is a combination of literals, conjunction, conditional effects, and quantification over effects:
  - <predicate>
  - (not <predicate>)
  - (and <effect> ... <effect>)
  - (when <formula> <effect>)
  - (forall (?x1 - type1 ... ?xn - typen) <effect>)

Operators (Example)

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

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

Syntax: Problem file

- **A PDDL problem file consists of:**
  1. (define (problem <name>)
  2. (:domain <name>)
     - to which domain does this problem belong?
  3. Definitions of objects belonging to each type.
  4. Definition of the initial state (list of ground predicates initially true).
  5. Definition of the goal (a formula like action preconditions).
     - This does not need to be fully describing the state of the world.

Problem file: Example Gripper

```
(define (problem gripper-x-1)
(:domain gripper)
(:objects rooma roomb - room
                ball2 ball1 - ball
                left - gripper )
(:init (at-robby rooma)
         (free left)
         (at ball2 rooma)
         (at ball1 rooma))
 (:goal (and (at ball2 roomb)
                (at ball1 roomb)))
)
```

How does a solution look like?

Example of execution output of a planning engine

```
Match tree built with 10 nodes.
PDDL problem description loaded:
             Domain: GRIPPER
             Problem: GRIPPER-X-1
#Actions: 10
#Fluents: 9
Landmarks found: 2
Starting search with IW (time budget is 60 secs)...
rel_plan size: 5
#RP_fluents 6
Caption {#goals, #UNnachieved, #Achieved} -> IW(max_w)

{2/2/0}:IW(1) -> [2][3][4]rel_plan size: 3
#RP_fluents 4
{2/1/1}:IW(1) -> [2][3][4][5]rel_plan size: 0
#RP_fluents 0
Plan found with cost: 7
Total time: -8.64267e-10
Nodes generated during search: 18
Nodes expanded during search: 12
IW search completed
```

```
(pick ball1 rooma left)
(move rooma roomb)
(drop ball1 roomb left)
(move roomb rooma)
(pick ball2 rooma left)
(move rooma roomb)
(drop ball2 roomb left)
```

Show

- Example from planning.domains

Additional info on PDDL

Closed-world assumption

predicates that are not explicitly defined as **T**, are considered to be **F**.

Add-after-Delete Semantics

if a predicate is simultaneously “added” (set to **T**) and “deleted” (set to **F**), the value **T** takes precedence.

PDDL: a bit of history

- PDDL has been introduced for the IPC 1998
- **2000, PDDL1.2** derives from STRIPS: predicate-centric representation (classical representation), and supports ADL
- **2002, PDDL2.1**: numeric fluents and durative actions
- **2006, PDDL3.1**: trajectory constraints, preferences and object fluents
- **2016, PDDL+**: continuous processes and events
- **PPDDL**: probabilistic action effects
- **MA-PDDL**: planning by and for multiple agents

What are IPCs?

- International planning competitions have been organised since 1998
- Aiming at evaluating and improving the state of the art of planning engines
  - Providing benchmarks
- Different tracks according to what is assessed
  - E.g., satisficing planning, optimal, multi-core, etc.
- Huddersfield organised the 2014 edition.

Is PDDL the only one?

- NDDL
  - Focus on timeline and activities
- ANML
  - Mix between NDDL and PDDL (focus on timeline, but notions of actions and predicates)
- RDDL
  - Probabilistic, allows to model partial observability
- PICAT
  - Structured representation. Supports the use of “control knowledge”

Some references

- Drew McDermott et al. The PDDL Planning Domain Definition Language. The AIPS-98 Planning Competition Comittee, 1998.
- Fahiem Bacchus. Subset of PDDL for the AIPS2000 Planning Competition. The AIPS-00 Planning Competition Comittee, 2000.
- Blai Bonet and Robert Givan. 5th international planning competition: Non-deterministic track – call for participation. In Proceedings of the 5th International Planning Competition (IPC'06), 2006.
- Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. Journal of Artificial Intelligence Research, 20:61–124, 2003.
- Alfonso Gerevini, Patrik Haslum, Derek Long, Alessandro Saetti, and Yannis Dimopoulos. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. Artificial Intelligence, 173(5-6):619–668, 2009.
- Joerg Hoffmann and Stefan Edelkamp. The deterministic part of ipc-4: An overview. Journal of Artificial Intelligence Research, 24:519–579, 2005.