Planning Representation: PDDL

Dra. Mª Dolores Rodríguez Moreno

Objectives

Specific Objectives

- Model in PDDL 1.2, PDDL 2.1 & PDDL 3.1
- Run SoA planners

Source

- Stuart Russell & Peter Norvig (2009). Chapter 10. Artificial Intelligence: A Modern Approach. (3rd Edition). Ed. Pearsons
- Ghallab, Nau &Traverso (2004). Automated Planning: Theory & Practice. The Morgan Kaufmann Series in Artificial Intelligence

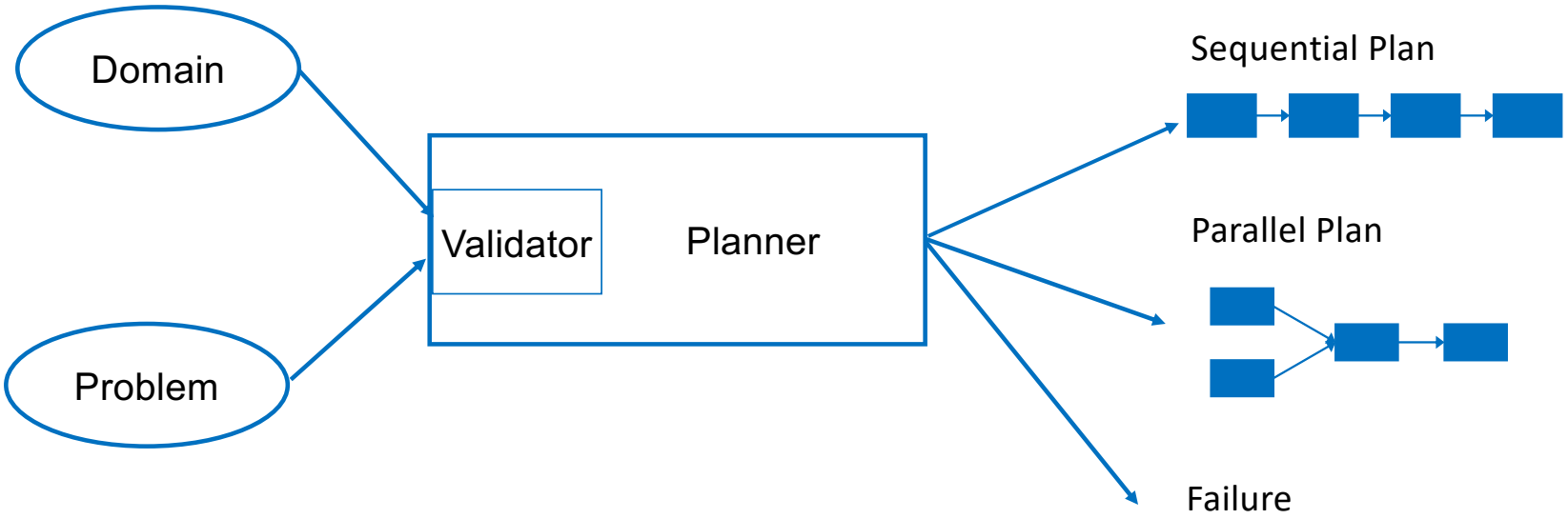
Outline

- **PDDL syntax**
- PDDL editors
- PDDL planners
- Blocks-world domain
- Conclusions

PDDL syntax

- Standard encoding language for “classical” planning tasks
- Components of PDDL :
  - Objects: things in the world that interest us
  - Predicates: properties of objects that we are interested in (can be true or false)
  - Initial state: the state of the world that we start in
  - Goal specification: things that we want to be true
  - Actions/Operators: ways of changing the state of the world (parametrized)

PDDL input/output



PDDL: Formally

- A *planning problem* is a tuple P= <A,I,G>, where A is a finite set of actions, I is the initial state, and G is a conjunctive goal
- *An action a is a tuple, a=pre(a), add(a) deal(a)*
- Given a planning problem Q=<P,A,I,G>, a sequence of actions $[a_1, a_2 \ldots, a_n]$ is a *solution* (*plan*) to Q if $Res([a_1, a_2 \ldots, a_n],I)$ is defined and G holds in $Res([a_1, a_2 \ldots, a_n],I)$

PDDL syntax: Domain

(define (domain <name>)
  (:requirements <require-key>)
  (:types <typed_list (name)>)
  (:constants <typed_list (name)>)
<PDDL list of predicates in the domain>
<PDDL code for first action>
…
<PDDL code for last action>
)

PDDL syntax: Actions (domain)

(:action <action name>
 :parameters (<list>)
 :precondition (<predicate list>)
 :effect (<predicate list>)
)

PDDL syntax: Problem

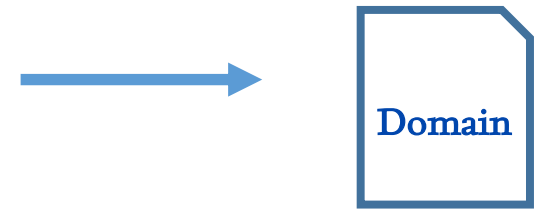
```
(define (problem <problem name>)
  (:domain <domain name>)
  <PDDL code for objects>
  <PDDL code for initial state>
  <PDDL code for goal specification>
)
```

PDDL syntax: Tip

- Use the extensión .pddl for your files
- Name the files as the domain and problem files

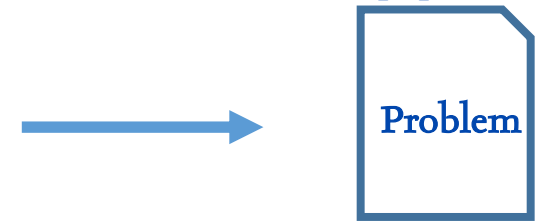
d-prueba.pddl

(define (domain d-prueba)



p1.pddl

(define (problem p1)



Outline

- PDDL1.2 syntax
- **PDDL editors**
- PDDL planners
- Blocks-world domain
- Conclusions

PDDL: editors

- Any editor that supports “( )” checking
  - Emacs
  - myPDDL
  - Gedit
  - Notepad++
  - Sublime

PDDL Planners: Online

- Use the FF planner (PDDL1.2 & partially PDDL 2.1)
  http://editor.planning.domains/

- Extended the previous editor for PDDL 2.1 – Level 3 (done at ISG lab, uses SIW planner
  https://still-fortress-36748.herokuapp.com/

Outline

- PDDL syntax
- PDDL editors
- **PDDL planners**
- Blocks-world domain
- Conclusions

PDDL Planners

- **SGPlan**
  https://wah.cse.cuhk.edu.hk/wah/programs/SGPlan/
- **LAMA**
  https://github.com/rock-planning/planning-lama
- **LPG-TD**
  http://zeus.ing.unibs.it/lpg/
- **OPTIC**
  https://nms.kcl.ac.uk/planning/software/optic.html
  https://planning.wiki/ref/planners/optic
- **International Plannning Competition (IPS)**
  http://icaps-conference.org/index.php/main/competitions

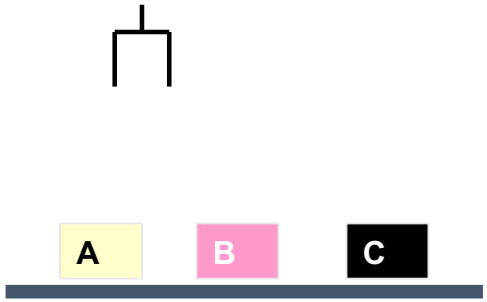
Execute SGPlan (Linux)

Outline

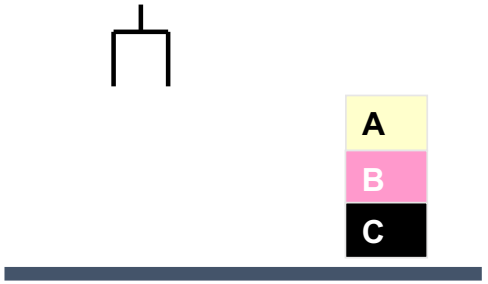
- PDDL syntax
- Gripper domain
- PDDL editors
- PDDL planners
- **Blocks-world domain**
- Conclusions

Blocks-world description (I)

- There is a hand robot that can pick up or drop blocks for recycling purpose. Initially, all blocks are on the conveyor belt (infinite). We want the blocks (finite) to be stacked as in the figure
- The hand robot distinguished between blocks on the belt or on top or other blocks. Then 4 different actions are needed



Initial State



Goal

Blocks-world description (II)

- Planning model
  - **Objects:** the 3 blocks (problem)
  - **Predicates:** Is x on the belt/table? Is nothing (clear) on top x? Is the hand empty? Is the hand holding x? Is x on top of y? (domain)
  - **Initial state:** all blocks on the conveyor belt. Hand robot is empty (problem)
  - **Goal specification:** each block is on top on each other as on the figure (problem)
  - **Actions/Operators:** the hand can pick up/drop a block or stack/unstack 2 blocks (domain)

Blocks-world description (III)

- UNSTACK(x; y)
  - preconditions: encima(x; y), libre(x), brazo-libre
  - add: sujeto(x), libre(y)
  - del: encima(x; y), brazo-libre, libre(x)
- STACK(x; y)
  - preconditions: sujeto(x), libre(y)
  - add: encima(x; y), libre(x), brazo-libre
  - del: sujeto(x), libre(y)
- PUT-DOWN(x)
  - preconditions: sujeto(x)
  - add: en-mesa(x), libre(x), brazo-libre
  - del: sujeto(x)
- PICK-UP(x)
  - preconditions: en-mesa(x), libre(x), brazo-libre
  - add: sujeto(x)
  - del: en-mesa(x), brazo-libre, libre(x)

Blocks-world: domain

```
(define (domain BLOCKS)
  (:requirements :strips)
  (:predicates (on ?x ?y)
               (ontable ?x)
               (clear ?x)
               (handempty)
               (holding ?x)
               )

  (:action pick-up
           :parameters (?x)
           :precondition (and (clear ?x) (ontable ?x) (handempty))
           :effect
           (and (not (ontable ?x))
             (not (clear ?x))
             (not (handempty))
             (holding ?x)))


  (:action unstack
           :parameters (?x ?y)
           :precondition (and (on ?x ?y) (clear ?x) (handempty))
           :effect
           (and (holding ?x)
             (clear ?y)
             (not (clear ?x))
             (not (handempty))
             (not (on ?x ?y)))))
```

Blocks-world: problem

```
(define (problem BLOCKS-1)
    (:domain BLOCKS)
    (:objects a b c)
    (:init (clear c)
           (clear b)
           (clear a)
           (ontable a)
           (ontable b)
           (ontable c)
           (handempty))
    (:goal (and (on b c)
                (on a b)
           )
    )
)
```

Conclusions (I)

- We have modelled our first PDDL domains
- Use the same name problem and domain for naming the files
- Use the extension .pddl for your files
- Use the *requirements* list to specify the type of problem

  (:requirements :strips :typing :equality :fluents :conditional-effects)

Conclusions (II)

- Use *type* for creating a hierarchy of objects (recommended)

(:types place vehicle - store
resource)

- Action effects can be more complicated than seen so far
  - They can be **universally quantified**

(forall (?v1 ... ?vn)
<effect>)

  - They can be **conditional**

(when <condition>
<effect>)