Planning Representation: PDDL

Dra. Mª Dolores Rodríguez Moreno

Objectives

Specific Objectives

- Model in PDDL 1.2
- 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

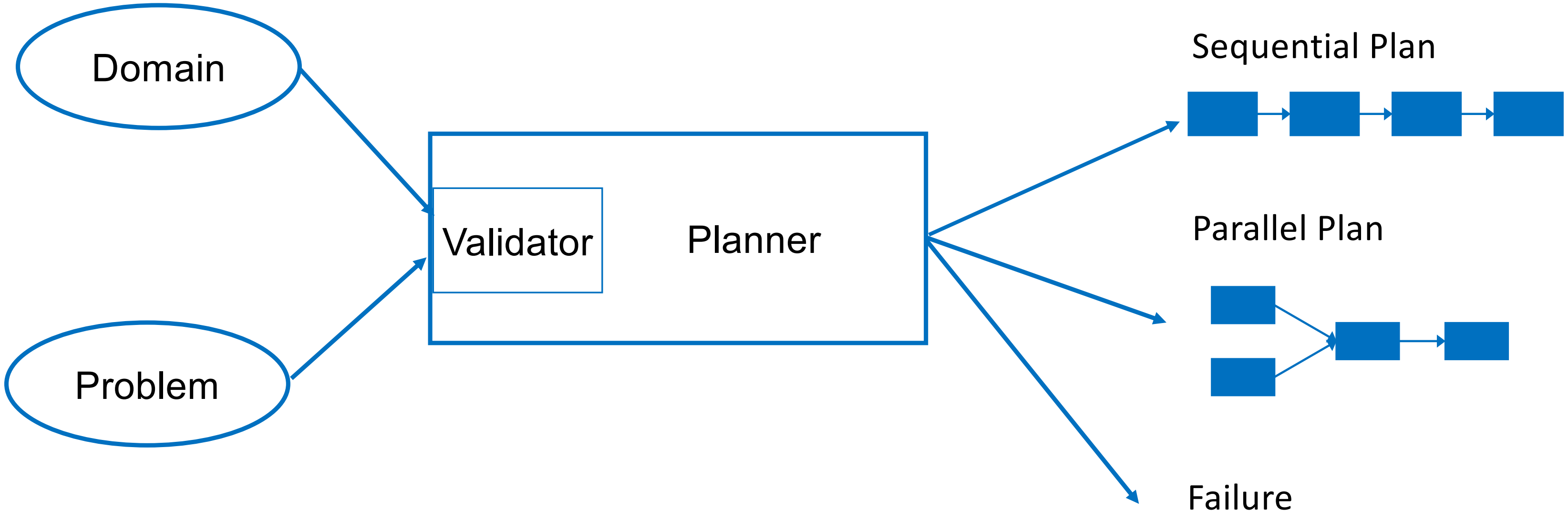
Outline

- **PDDL syntax**
- Gripper domain
- 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

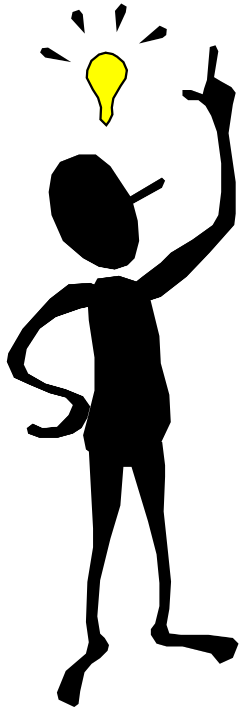
```
(define (problem <problem name>)
  (:domain <domain name>)
  <PDDL code for objects>
  <PDDL code for initial state>
  <PDDL code for goal specification>
)
```

PDDL syntax: Tip

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

(define (domain d-prueba) → d-prueba.pddl **Domain**

(define (problem p1) → p1.pddl **Problem**

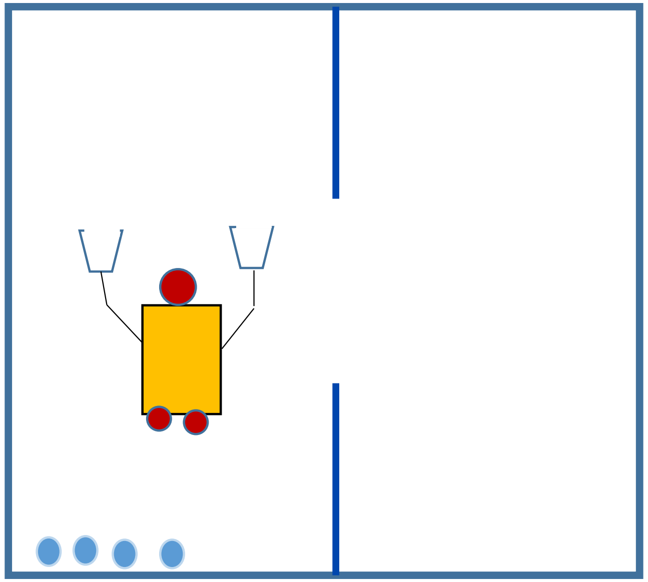


Outline

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

Gripper domain (I)

- There is a robot that can move between two rooms and pick up or drop balls (4) 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.



Gripper domain (II)

- Planning model
  - **Objects:** the two rooms, four balls and two robot arms (problem)
  - **Predicates:** Is x a room? Is x a ball? Is ball x inside room y? Is robot arm x empty? (domain)
  - **Initial state:** all balls and the robot are in the first room. All robot arms are empty (problem)
  - **Goal specification:** all balls must be in the second room (problem)
  - **Actions/Operators:** the robot can move between rooms, pick up a ball or drop a ball (domain)

- Let's get started:
  - http://editor.planning.domains/

Gripper domain (III)

- **Objects**
  - Rooms: rooma, roomb
  - Balls: ball1, ball2, ball3, ball4
  - Robot arms: left, right
- PDDL problem file:

```
(define (problem gripper-four-balls)
   (:domain gripper)
   (:objects rooma roomb
             ball1 ball2 ball3 ball4
             left right)
```

Gripper domain (IV)

- **Predicates**

ROOM(r)	– true iff r is a room
BALL(b)	– true iff b is a ball
GRIPPER (g)	– true iff g is a gripper (robot arm)
at-robby (r)	– true iff r is a room and the robot is in r
at-ball (b,r)	– true iff b is a ball, r is a room and b is in r
free(g)	– true iff g is a gripper and g does not hold a ball
carry (g,b)	– true iff g is a gripper, b is a ball, and g holds b

- Pddl domain file

```
(define (domain gripper)
(:requirements :strips)
   (:predicates (ROOM ?r)
                (BALL ?b)
                (GRIPPER ?g)
                (at-robby ?r)
                (at-ball ?b ?r)
                (free ?g)
                (carry ?g ?b))

```

Gripper domain (V)

- **Initial state**
  - ROOM(rooma) and ROOM(roomb) are true
  - BALL(ball1), ..., BALL(ball4) are true
  - GRIPPER(left), GRIPPER(right), free(left) and free(right) are true
  - at-robby(rooma), at-ball(ball1, rooma), ..., at-ball(ball4, rooma) are true
  - Everything else is false

Gripper domain: actions (I)

- **Operator *Move***
  - Description: The robot can move from x to y
  - Precondition: ROOM(x), ROOM(y) and at-robby(x) are true
  - Effect: at-robby(y) becomes true
    at-robby(x) becomes false
    Everything else doesn't change

Gripper domain: actions (II)

- Operator *Move* in PDDL (domain file)

```
(:action move
    :parameters (?x ?y)
    :precondition (and (ROOM ?x)
                       (ROOM ?y)
                       (at-robby ?x)
                  )
    :effect (and (at-robby ?y)
                 (not (at-robby ?x))
            )
)
```

Gripper domain: actions (III)

- **Operator *Pick-up***
  - Description: The robot can pick up b in r with g
  - Precondition: BALL(b), ROOM(r), GRIPPER(g), at-ball(b, r), at-robby(r) and free(b) are true
  - Effect: carry(g, b) becomes true

    at-ball(b, r) and free(g) become false

    Everything else doesn't change

Gripper domain: actions (IV)

- **Operator *Pick-up* in PDDL (domain file)**

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

Gripper domain: actions (V)

- **Operator *Drop***
  - Description: The robot can drop b in r from g
  - Precondition: BALL(b), ROOM(r), GRIPPER(g), at-ball(b, r), at-robby(r) and free(g) are true
  - Effect: carry(g, b) becomes true

    at-ball(b, r) and free(g) become false

    Everything else doesn't change

Gripper domain: actions (VI)

- **Operator *Drop* in PDDL (domain file)**

```
(:action drop
    :parameters (?ball ?room ?gripper)
    :precondition (and (BALL ?ball)
                       (ROOM ?room)
                       (GRIPPER ?gripper)
                       (carry ?gripper ?ball)
                       (at-robby ?room)
                  )
    :effect (and (at-ball ?ball ?room)
                 (free ?gripper)
                 (not (carry ?gripper ?ball))
            )
)
```

Gripper domain: Initial state

- **Initial state in PDDL (problem file)**

```
(:init (ROOM rooma)
       (ROOM roomb)
       (BALL ball1)
       (BALL ball2)
       (BALL ball3)
       (BALL ball4)
       (GRIPPER left)
       (GRIPPER right)
       (at-robby rooma)
       (free left)
       (free right)
       (at-ball ball1 rooma)
       (at-ball ball2 rooma)
       (at-ball ball3 rooma)
       (at-ball ball4 rooma)
)
```

Gripper domain: goals

- **Goal state:**
  - at-ball(ball1, roomb), ..., at-ball(ball4, roomb) must be true
  - Everything else we don't care about
- **Goal state in PDDL (problem file)**

```
    (:goal (and (at-ball ball1 roomb)
                (at-ball ball2 roomb)
                (at-ball ball3 roomb)
                (at-ball ball4 roomb)
           )
    )
)
```

Outline

- PDDL syntax
- Gripper domain
- **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
- Gripper domain
- 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)

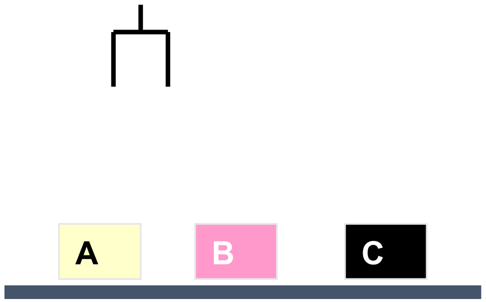
```
viki@c3po: ~/Documents
viki@c3po:~/Documents$ ./sgplan522
#
# Copyright (C) 2006, Board of Trustees of the University of Illinois.
#
# The program is copyrighted by the University of Illinois, and should
# not be distributed without prior approval.  Commercialization of this
# product requires prior licensing from the University of Illinois.
# Commercialization includes the integration of this code in part or
# whole into a product for resale.
#
#-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
# Author: C. W. Hsu, B. W. Wah, R. Y. Huang, Y. X. Chen
#-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-


SGPlan-5 settings:
-o <string>             specifies the file of the operators

-f <string>             specifies the file of (init/goal) facts

-out <string>           specifies the file name for computed plans, standard output if not specified

-cputime <number>       specifies the maximum CPU-time (in seconds)


 viki@c3po:~/Documents$ ./sgplan522 -o blocksWorld.pddl -f blocks1.pddl
```

Outline

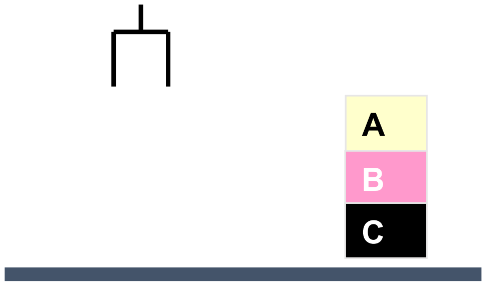
- PDDL syntax
- Gripper domain
- PDDL editors
- PDDL planners
- **Blocks-world domain**
- Conclusions

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)
           )
    )
)
```

Outline

- Introduction
- Logic
- Type of problems
- Modelling in planning
- STRIPS
- PDDL
- IPC
- **Conclusions**

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>)