Intelligent Robotics Lab

Planificación y Sistemas Cognitivos

Bloque 1

Planificación Clásica

Universidad Rey Juan Carlos

Objetivo del Tema

1. Aprenderemos qué es Classic Planning y las aproximaciones típicas
2. Introduciremos PDDL (Panning Definition Domain language)
3. Aprenderemos a plantear problemas que resolveremos aplicando planning
4. Usaremos POPF, una planner de PDDL

Introducción

¿Qué es Planning?

1. Planning es el **proceso de cálculo de un plan para conseguir uno o varios objetivos**
2. Un plan es una secuencia de acciones que conduce de un estado inicial a un estado que contiene los objetivos a conseguir

¿Cuáles son los elementos que encontramos en el Planning?

1. El estado de un problema lo conforma un conjunto de predicados

$$Perdido \wedge huerfano$$

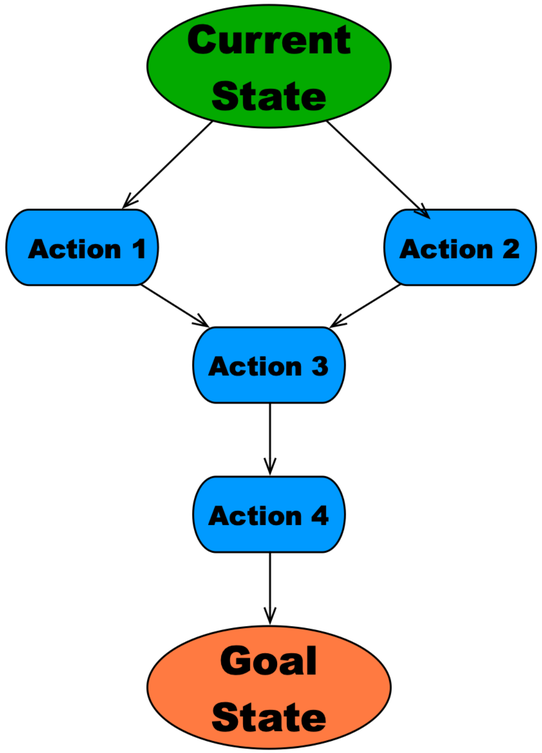
$$robotAt(r2d2, Tatooine) \wedge robotAt(c3po, Devaron)$$

2. No son válidos

$\neg rico$	es negativo
$\neg robotAt(bb8, Dagobah)$	es negativo
$areFighting(owner(bb8), DarthVader)$	No funciones
$robotAt(x, y)$	No grounded

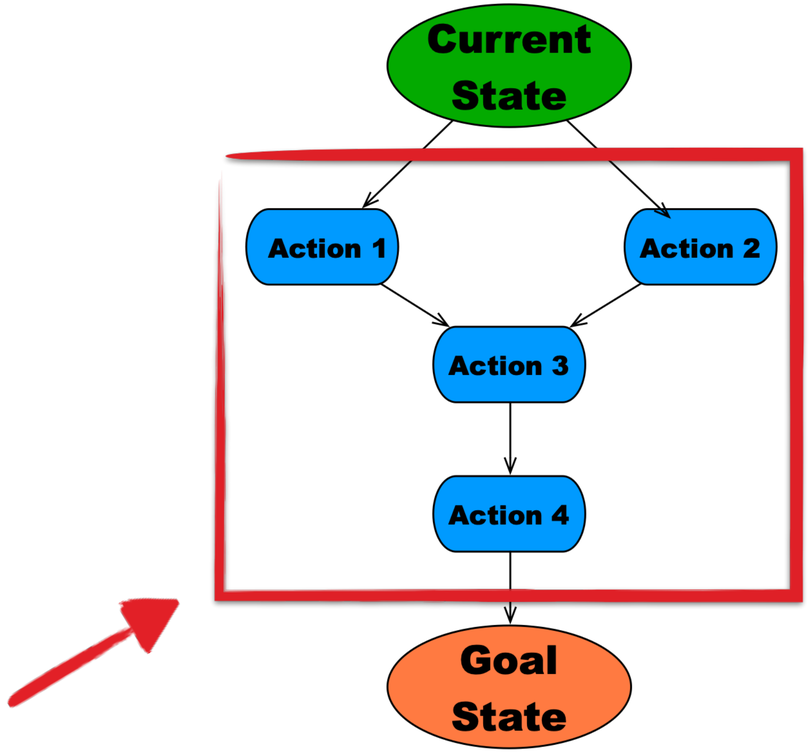
¿Qué es Planning?

1. Planning es el **proceso de cálculo de un plan para conseguir uno o varios objetivos**
2. Un plan es una secuencia de acciones que conduce de un estado inicial a un estado que contiene los objetivos a conseguir



¿Qué es Planning?

1. Planning es el **proceso de cálculo de un plan para conseguir uno o varios objetivos**
2. Un plan es una secuencia de acciones que conduce de un estado inicial a un estado que contiene los objetivos a conseguir



Esto se calcula automáticamente!!!

Más elementos en Planning

1. Una acción permite cambiar el estado del problema
2. Tiene una precondición
3. Tiene un efecto que cambia el estado del problema
4. Un acción es aplicable si su precondición se cumple

$$
\begin{aligned}
&Action(Travel(r, s, from, to),\\
&PRECOND: robotAt(r, from) \wedge spaceshipAt(s, from) \wedge robot(r) \wedge spaceship(s) \wedge\\
&planet(from) \wedge planet(to)\\
&EFFECT: \neg robotAt(r, from) \wedge robotAt(r, to))
\end{aligned}
$$

5. Add List (ADD) es la lista de predicados que añade una acción
6. Delete List (DEL) es la lista de predicados que añade una acción

$$(state, action) = (state - DEL(action)) \cup ADD(action)$$

$$Travel(r2d2, millenniumFalcon, Tatooine, Naboo)$$

$$
\begin{aligned}
&robotAt(r2d2, Tatooine) \wedge\\
&robot(r2d2) \wedge\\
&spaceship(millenniumFalcon) \wedge\\
&planet(Tatooine) \wedge\\
&planet(Naboo)
\end{aligned}
\longrightarrow
\begin{aligned}
&\cancel{robotAt(r2d2, Tatooine) \wedge}\\
&robot(r2d2) \wedge\\
&spaceship(millenniumFalcon) \wedge\\
&planet(Tatooine) \wedge\\
&planet(Naboo) \wedge\\
&robotAt(r2d2, Naboo)
\end{aligned}
$$

Intelligent Robotics Lab

Ejemplo: Viajes estelares

$$
\begin{aligned}
&Init(\\
&robotAt(r2d2, Naboo) \wedge robotAt(c3p0, Tatooine) \wedge spaceshipAt(milleniunFalcon, Alderaan) \wedge\\
&spaceshipAt(Xwing, Kessel) \wedge robot(r2d2) \wedge robot(c3p0) \wedge spaceship(milleniunFalcon) \wedge\\
&spaceship(Xwing) \wedge planet(Alderaan) \wedge planet(Kessel))
\end{aligned}
$$

$$
\begin{aligned}
&Goal(\\
&robotAt(r2d2, Tatooine) \wedge robotAt(c3p0, Naboo))
\end{aligned}
$$

$$
\begin{aligned}
&Action(Load(r, s, p),\\
&PRECOND\ :\ robotAt(r, p) \wedge spaceshipAt(s, p) \wedge robot(r) \wedge spaceship(s) \wedge\\
&planet(p)\\
&EFFECT : \neg robotAt(r, p) \wedge robotIn(r, s)
\end{aligned}
$$

$$
\begin{aligned}
&Action(Unload(r, s, p),\\
&PRECOND\ :\ robotIn(r, s) \wedge spaceshipAt(s, p) \wedge robot(r) \wedge spaceship(s) \wedge\\
&planet(p)\\
&EFFECT : \neg robotIn(r, s) \wedge robotAt(r, p)
\end{aligned}
$$

$$
\begin{aligned}
&Action(Fly(s, from, to),\\
&PRECOND : spaceshipAt(s, from) \wedge spaceship(s) \wedge planet(from) \wedge planet(to)\\
&EFFECT : \neg spaceshipAt(s, from) \wedge spaceshipAt(s, to)
\end{aligned}
$$

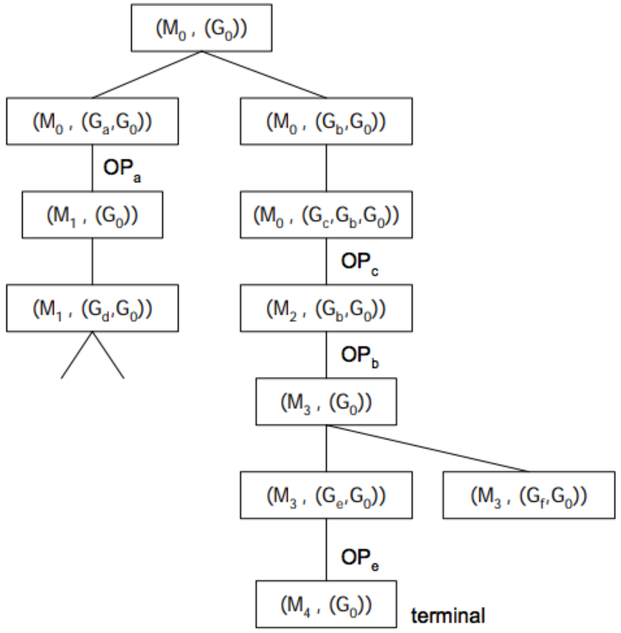
Ejercicio: Viajes estelares

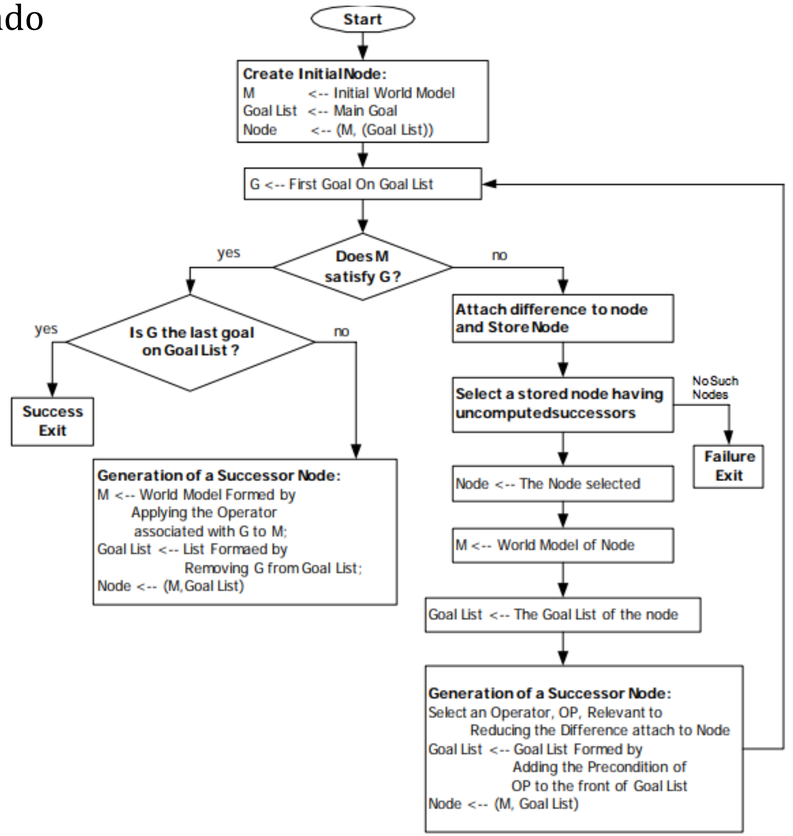
1. En una hoja de papel pon el estado inicial
2. Calcula la secuencia de acciones para obtener el goal
3. Por cada acción, tacha los predicados que se eliminan, y escribe los que se añaden
4. Encuentra el error, y qué efecto produce

10 minutos!!!

STanford Research Institute Problem Solver
STRIPS

- 1971. Imperial College London
- Primer lenguaje de planning ampliamente usado
- Lo usó Shakey en 1984 (SRI)





http://ai.stanford.edu/~nilsson/OnlinePubs-Nils/shakey-the-robot.pdf

http://www.cs.cmu.edu/~mmv/planning/readings/strips.pdf

Intelligent Robotics Lab

PDDL

Planning Domain Definition Language
PDDL

- PDDL1.2 fue el lenguaje oficial del primer IPC (International Planning Competition) en 1998
- Inspirado en STRIPS
- Un modelo puede aplicarse a muchos problemas distintos
- La versión 2.1 es la más popular, ya que incluye durative actions.
- POPF

http://www.cs.cmu.edu/~mmv/planning/readings/98aips-PDDL.pdf

Planning Domain Definition Language
PDDL

- Establecer las reglas (Domain)
- Presentar una situación y un goal (Problem)
- Usar un plan solver
- Conseguir un plan

```
(define (domain simple)
(:types robot room)
(:predicates
  (robot_at ?r - robot ?ro - room)
  (connected ?ro1 ?ro2 - room))
(:durative-action move
  :parameters (?r - robot ?r1 ?r2 - room)
  :duration ( = ?duration 5)
  :condition (and
    (at start(connected ?r1 ?r2))
    (at start(robot_at ?r ?r1)))
  :effect (and
    (at start(not(robot_at ?r ?r1)))
    (at end(robot_at ?r ?r2))))
)
```

Domain.pddl

Planning Domain Definition Language PDDL

- Establecer las reglas (Domain)
- Presentar una situación y un goal (Problem)
- Usar un plan solver
- Conseguir un plan

```
(define (domain simple)
(:types robot room)
(:predicates
  (robot_at ?r - robot ?ro - room)
  (connected ?ro1 ?ro2 - room))
(:durative-action move
  :parameters (?r - robot ?r1 ?r2 - room)
  :duration ( = ?duration 5)
  :condition (and
    (at start(connected ?r1 ?r2))
    (at start(robot_at ?r ?r1)))
  :effect (and
    (at start(not(robot_at ?r ?r1)))
    (at end(robot_at ?r ?r2))))
)
```

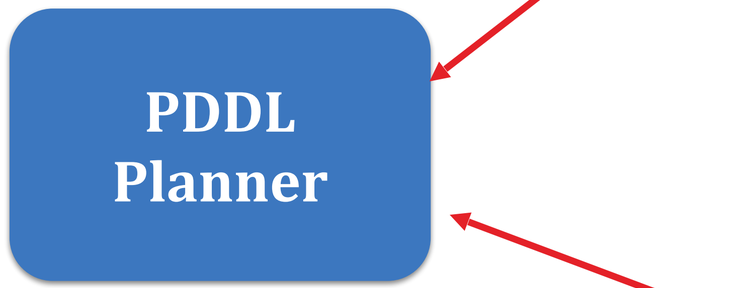
Domain.pddl

```
(define (problem problem_1)
(:domain simple)
(:objects
  r2d2 - robot
  bedroom living kitchen - room
)
(:init
  (robot_at r2d2 bedroom)
  (connected living bedroom)
  (connected bedroom living)
  (connected living kitchen)
  (connected kitchen living))
(:goal (and(robot_at r2d2 kitchen)))
)
```

Problem1.pddl

Planning Domain Definition Language PDDL

- Establecer las reglas (Domain)
- Presentar una situación y un goal (Problem)
- Usar un plan solver
- Conseguir un plan




```
(define (domain simple)
(:types robot room)
(:predicates
  (robot_at ?r - robot ?ro - room)
  (connected ?ro1 ?ro2 - room))
(:durative-action move
  :parameters (?r - robot ?r1 ?r2 - room)
  :duration ( = ?duration 5)
  :condition (and
    (at start(connected ?r1 ?r2))
    (at start(robot_at ?r ?r1)))
  :effect (and
    (at start(not(robot_at ?r ?r1)))
    (at end(robot_at ?r ?r2))))
)
```

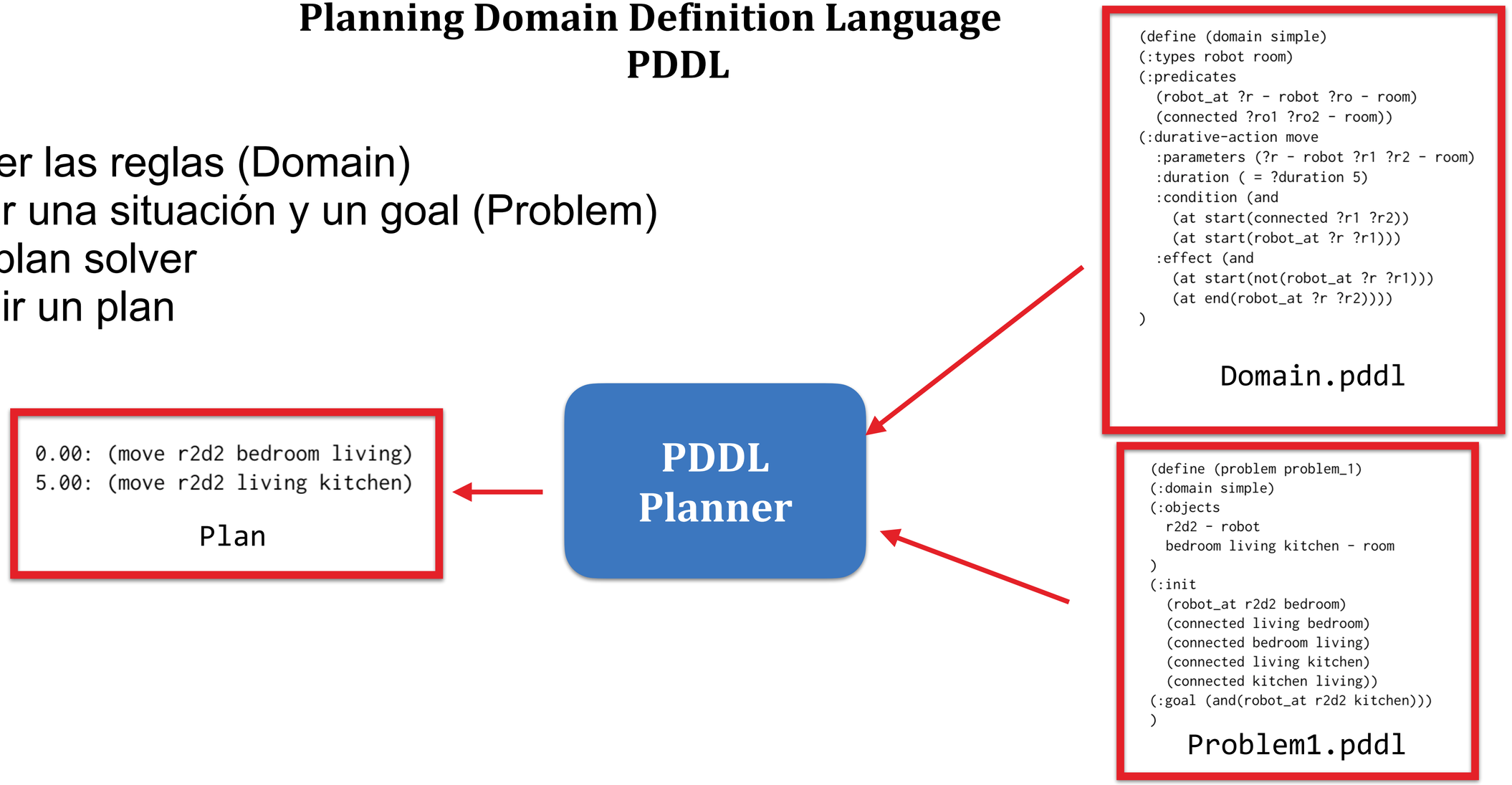
Domain.pddl

```
(define (problem problem_1)
(:domain simple)
(:objects
  r2d2 - robot
  bedroom living kitchen - room
)
(:init
  (robot_at r2d2 bedroom)
  (connected living bedroom)
  (connected bedroom living)
  (connected living kitchen)
  (connected kitchen living))
(:goal (and(robot_at r2d2 kitchen)))
)
```

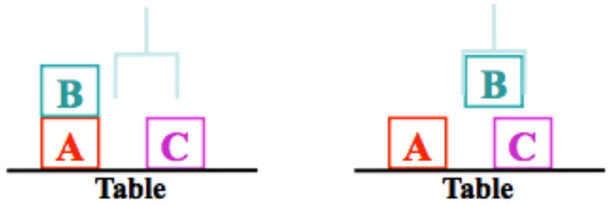
Problem1.pddl

Planning Domain Definition Language PDDL

- Establecer las reglas (Domain)
- Presentar una situación y un goal (Problem)
- Usar un plan solver
- Conseguir un plan



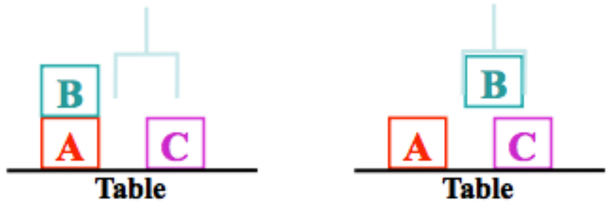
El mundo de los bloques




```
(define (domain blocks)
(:requirements :strips :equality)
(:predicates
  (on ?x ?y)
  (clear ?x)
  (table ?t)
  (block ?b)
)

(:constants Table)
```

https://github.com/fmrico/planning_cognitve_systems_course/tree/main/pddl

El mundo de los bloques

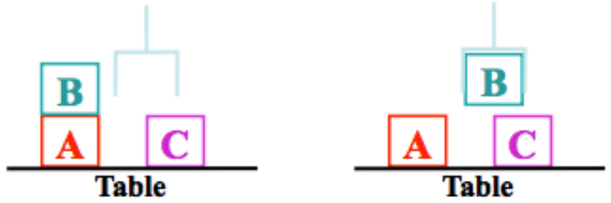



```
(:action move_to_table
  :parameters (?b ?x)
  :precondition
    (and
      (block ?b)
      (block ?x)
      (on ?b ?x)
      (clear ?b)
      (not (= ?b ?x))
    )
  :effect
    (and
      (on ?b Table)
      (clear ?x)
      (not (on ?b ?x))
    )
)
)
```

```
(:action move
  :parameters (?b ?x ?y)
  :precondition
    (and
      (block ?b)
      (block ?y)
      (clear ?b)
      (clear ?y)
      (on ?b ?x)
      (not (= ?b ?x))
      (not (= ?b ?y))
      (not (= ?x ?y))
    )
  :effect
    (and
      (on ?b ?y)
      (clear ?x)
      (not (on ?b ?x))
      (not (clear ?y))
    )
)
```

https://github.com/fmrico/planning_cognitve_systems_course/tree/main/pddl

El mundo de los bloques



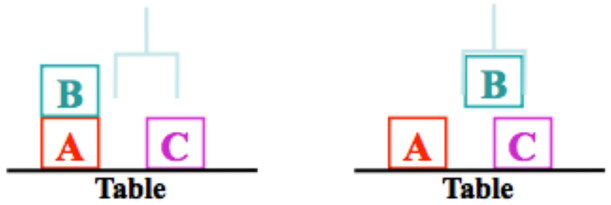

```
(define (problem blocks1)
(:domain blocks)
(:objects
  a b c
)
(:init
  (block a)
  (block b)
  (block c)
  (clear b)
  (clear c)
  (on b Table)
  (on a Table)
  (on c a)
)
```

```
(:goal (and
  (on b c)(on a b))
)

)
```

- Ejercicio 1: Prueba a resolverlo con popf
- Ejercicio 2: Añade otro bloque y prueba una configuración inicial más compleja

https://github.com/fmrico/planning_cognitve_systems_course/tree/main/pddl

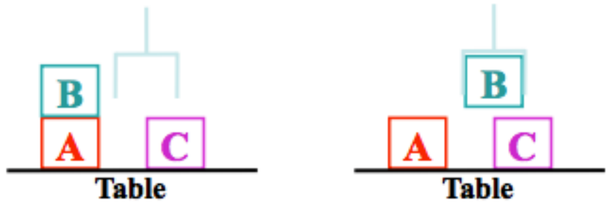
El mundo de los bloques con tipos




```
(define (domain blocks)
(:requirements :strips :equality :typing)

(:types
  table block - object
)

(:predicates
  (on ?x ?y - object)
  (clear ?x - object)
)

(:constants Table - table)
```

https://github.com/fmrico/planning_cognitve_systems_course/tree/main/pddl

El mundo de los bloques con tipos

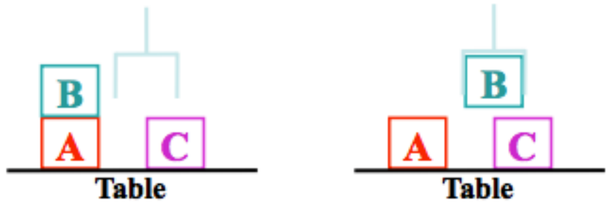



```
(:action move_to_table
  :parameters (?b ?x - block)
  :precondition
    (and
     (on ?b ?x)
      (clear ?b)
      (not (= ?b ?x))
    )
  :effect
    (and
      (on ?b Table)
      (clear ?x)
      (not (on ?b ?x))
    )
)
)
```

```
(:action move
  :parameters (?b - block ?x - object ?y - block)
  :precondition
    (and
      (clear ?b)
      (clear ?y)
      (not (= ?b ?y))
   )
  :effect
    (and
      (on ?b ?y)
      (clear ?x)
      (not (on ?b ?x))
      (not (clear ?y))
    )
)
```

https://github.com/fmrico/planning_cognitve_systems_course/tree/main/pddl

El mundo de los bloques con tipos




```
(define (problem blocks1)
(:domain blocks)
(:objects
  a b c d - block
)
(:init
  (clear b)
  (clear d)
  (on c Table)
  (on a Table)
  (on b a)
  (on d c)
)
```

```
(:goal (and
  (on b a)
  (on c b)
  (on d c)
  (on a Table)
)
)
```

https://github.com/fmrico/planning_cognitve_systems_course/tree/main/pddl

Numeric Fluents

- Introducido en PDDL 2.1
- Similar a un predicado, es una variable que mantiene un valor numérico a lo largo del plan

```
(:functions
    (<variable_name> <parameter_name> - <object_type>)
    ...
    (<variable_name> <parameter_name> - <object_type>)
)
```

```
(:functions
    (battery-level ?r - rover)
)
```

```
(:functions
    (distance ?wp1 - waypoint ?wp2 - waypoint)
)
```

Numeric Fluents

Numeric Expressions

```
(+ (sample-capacity) (battery-capacity))
```

```
(/ (sample-capacity) (battery-capacity))
```

```
(- (sample-capacity) (battery-capacity))
```

```
(* (sample-capacity) (battery-capacity))
```

Numeric Fluents

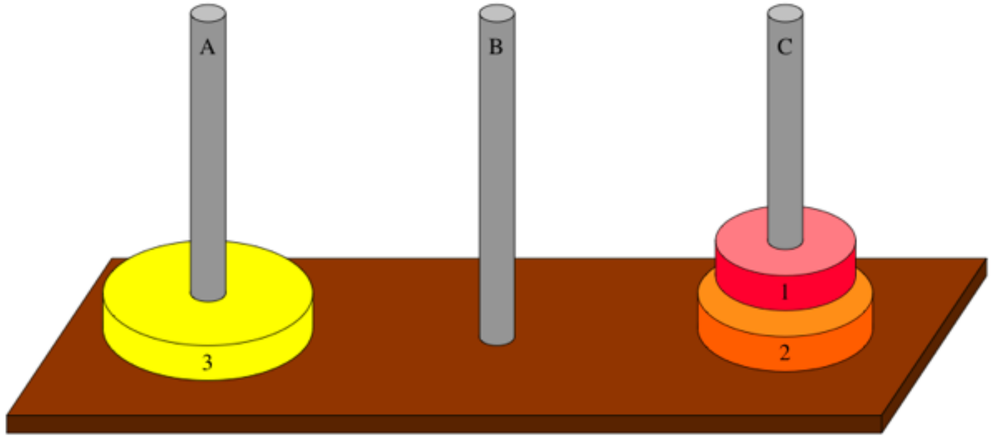
Numeric Expressions

```
(increase (battery-level ?r) (charge-available - ?solarpanel))
```

```
(decrease (battery-level ?r) (power-needed-for-work - ?task))
```

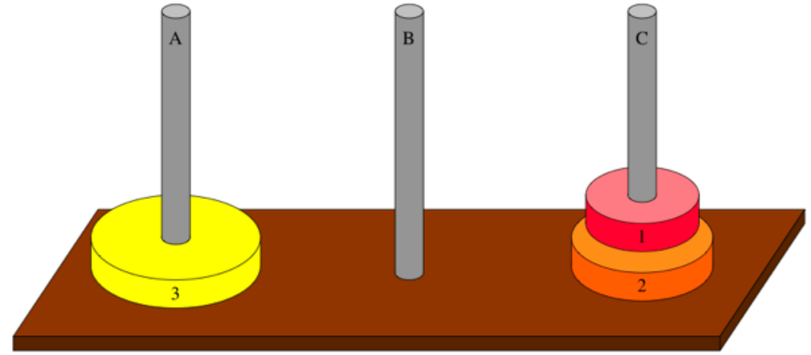
```
(assign (battery-level ?r) 10)
```

Las torres de Hanoi



- Mover todos los discos de un pilar a otro pilar
- Un disco puede apilarse encima de otro más pequeño

Las torres de Hanoi

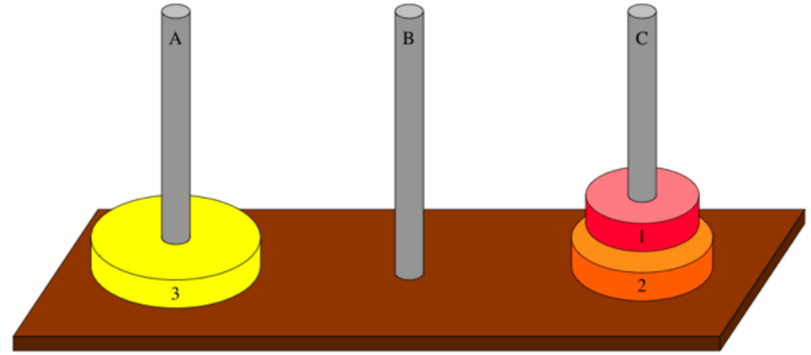



```
(define (domain hanoi)
  (:requirements :strips)

(:predicates
  (clear ?x)
  (on ?x ?y)
  (smaller ?x ?y)
)

(:action move
  :parameters (?disc ?from ?to)
  :precondition (and
    (smaller ?to ?disc)
    (on ?disc ?from)
       (clear ?disc)
    (clear ?to)
  )
  :effect  (and
    (clear ?from)
    (on ?disc ?to)
    (not (on ?disc ?from))
       (not (clear ?to))
  )
)
)
```

```
(define (problem hanoi3)
  (:domain hanoi)
  (:objects peg1 peg2 peg3 d1 d2 d3)
  (:init
   (smaller peg1 d1) (smaller peg1 d2) (smaller peg1 d3)
   (smaller peg2 d1) (smaller peg2 d2) (smaller peg2 d3)
   (smaller peg3 d1) (smaller peg3 d2) (smaller peg3 d3)
   (smaller d2 d1) (smaller d3 d1) (smaller d3 d2)
   (clear peg2) (clear peg3) (clear d1)
   (on d3 peg1) (on d2 d3) (on d1 d2))
  (:goal (and (on d3 peg3) (on d2 d3) (on d1 d2)))
  )
```

Las torres de Hanoi

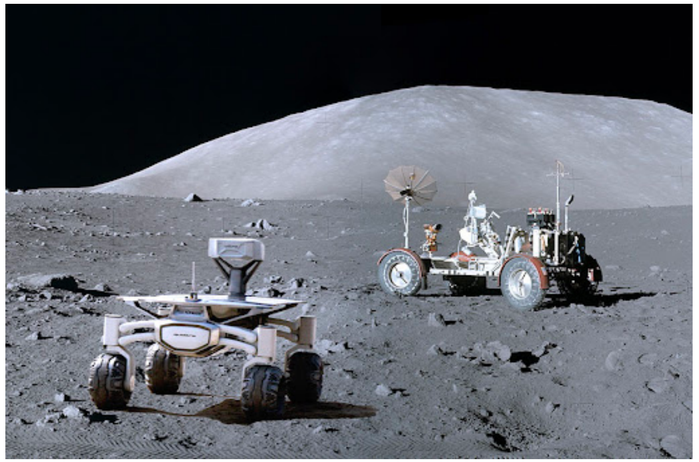
con fluents !!!




```
(define (domain hanoi)
  (:requirements :strips :fluents)

(:functions
    (size ?x)
)

(:predicates
  (clear ?x)
  (on ?x ?y)
)

(:action move
  :parameters (?disc ?from ?to)
  :precondition (and
    (< (size ?to) (size ?disc))
    (on ?disc ?from)
       (clear ?disc)
    (clear ?to)
  )
  :effect  (and
    (clear ?from)
    (on ?disc ?to)
    (not (on ?disc ?from))
       (not (clear ?to))
  )
)
)
```

```
(define (problem hanoi3)
  (:domain hanoi)
  (:objects peg1 peg2 peg3 d1 d2 d3)
  (:init

   (= (size peg1) 0)
   (= (size peg2) 0)
   (= (size peg3) 0)
   (= (size d1) 3)
   (= (size d2) 2)
   (= (size d3) 1)

   (clear peg2) (clear peg3) (clear d1)
   (on d3 peg1) (on d2 d3) (on d1 d2))
  (:goal (and (on d3 peg3) (on d2 d3) (on d1 d2)))
  )
```

Rover problem



- Un robot debe moverse entre waypoints para explorar un entorno
- Tiene una batería con cierta capacidad, y debe recargarse para poder continuar

Rover problem

```
(define (domain rover)

  (:requirements :typing :fluents)

  (:types robot waypoint - object)

  (:predicates
    (robot_at ?r - robot ?wp - waypoint)
    (connected ?c1 ?c2 - waypoint)
    (visit ?wp - waypoint)
  )


  (:action move
    :parameters (?r - robot ?from ?to - waypoint)
    :precondition (and
      (connected ?from ?to)
      (robot_at ?r ?from)
    )
    :effect (and
      (not (robot_at ?r ?from))
      (robot_at ?r ?to)
      (visit ?from)
      (visit ?to)
    )
  )
)
```

```
(define (problem rover-problem)
  (:domain rover)

  ;; Instantiate the objects.
  (:objects
    curiosity - robot
    wp1 wp2 wp3 wp4 wp_recharge - waypoint
  )

  (:init
    ; Define the initial state predicates.
    (robot_at curiosity wp1)

    (connected wp1 wp2)
    (connected wp2 wp1)
    (connected wp1 wp3)
    (connected wp3 wp1)
    (connected wp1 wp4)
    (connected wp4 wp1)
    (connected wp2 wp3)
    (connected wp3 wp2)
    (connected wp2 wp4)
    (connected wp4 wp2)
    (connected wp3 wp4)
    (connected wp4 wp3)
    (connected wp2 wp_recharge)
    (connected wp_recharge wp2)
  )

  (:goal (and
    (visit wp1)
    (visit wp2)
    (visit wp3)
    (visit wp4)
  ))
)
```

- Añade un nivel de batería
- Cada movimiento cuesta 1
- Hay que ir a recargarse al punto de recarga

Travel problem

```
(define (domain travel)

  (:requirements :typing :fluents)

  (:types vehicle city - object)

  (:predicates
    (vehicle_at ?v - vehicle ?c - city)
    (connected ?c1 ?c2 - city)
  )

  (:functions
    (deposit_level)
    (fuel ?c1 ?c2 - city)
  )

  (:action drive
    :parameters (?v - vehicle ?c1 ?c2 - city)
    :precondition (and
      (connected ?c1 ?c2)
      (vehicle_at ?v ?c1)
    )
    :effect (and
      (not (vehicle_at ?v ?c1))
      (vehicle_at ?v ?c2)
      (decrease (deposit_level) (fuel ?c1 ?c2))
    )
  )
)
```

```
(define (problem travel-problem)
  (:domain travel)

  ;; Instantiate the objects.
  (:objects
    minicooper - vehicle
    alcorcon leganes fuenlabrada mostoles madrid - city
  )

  (:init
    ; Define the initial state predicates.
    (vehicle_at minicooper fuenlabrada)

    (connected alcorcon leganes)
    (connected leganes alcorcon)
    (connected mostoles alcorcon)
    (connected alcorcon mostoles)
    (connected fuenlabrada leganes)
    (connected leganes fuenlabrada)
    (connected mostoles fuenlabrada)
    (connected fuenlabrada mostoles)
    (connected madrid leganes)
    (connected leganes madrid)
    (connected madrid alcorcon)
    (connected alcorcon madrid)

    (= (deposit_level) 100)

    (= (fuel alcorcon leganes) 2)
    (= (fuel leganes alcorcon) 2)
    (= (fuel mostoles alcorcon) 1)
    (= (fuel alcorcon mostoles) 1)
    (= (fuel fuenlabrada leganes) 5)
    (= (fuel leganes fuenlabrada) 5)
    (= (fuel mostoles fuenlabrada)  2)
    (= (fuel fuenlabrada mostoles) 2)
    (= (fuel madrid leganes) 3)
    (= (fuel leganes madrid) 3)
    (= (fuel madrid alcorcon) 3)
    (= (fuel alcorcon madrid) 3)
  )

  (:goal (and
    (vehicle_at minicooper madrid)
  ))

  (:metric maximize (deposit_level))
)
```

Intelligent Robotics Lab

Durative actions

- Introducido en PDDL 2.1
- Las acciones tardan un tiempo, sobre todo en robots

```
(:durative-action <action_name>
    :parameters (<arguments>)
    :duration (= ?duration <duration_number>)
    :condition (logical_expression)
    :effect (logical_expression)
)
```

Durative actions

Duration

```
:duration (= ?duration <duration_number>)
```

```
:duration (> ?duration <duration_number>)
```

```
:duration (< ?duration <duration_number>)
```

```
:duration (and
    (> ?duration <duration_number>)
    (< ?duration <duration_number>))
```

Pueden usarse fluents

Durative actions

Condition

```
:condition (<logical_temporal_expression>)
```

```
(at start (at ?rover ?from-waypoint))
```

```
(at end (>= (battery-amount ?rover) 0))
```

```
(over all (can-move ?from-waypoint ?to-waypoint))
```

Durative actions

Effect

```
:effect (<logical_temporal_condition>)
```

```
:effect
    (and
        (at start (not (at ?rover ?from-waypoint)))
        (at start (decrease (battery-amount ?rover) 8))
        (at end (at ?rover ?to-waypoint))
        (at end (been-at ?rover ?to-waypoint)))
```

Travel problem with Durative actions

```
(define (domain travel)

  (:requirements :typing :fluents :durative-actions)

  (:types vehicle city - object)

  (:predicates
    (vehicle_at ?v - vehicle ?c - city)
    (connected ?c1 ?c2 - city)
  )

  (:functions
    (time ?c1 ?c2 - city)
  )

  (:durative-action drive
    :parameters (?v - vehicle ?c1 ?c2 - city)
    :duration (= ?duration (time ?c1 ?c2))
    :condition (and (at start (connected ?c1 ?c2))
                    (at start (vehicle_at ?v ?c1))
                    )
    :effect (and (at start (not (vehicle_at ?v ?c1)))
                 (at end (vehicle_at ?v ?c2))

                 )
  )
)
```

```
(define (problem travel-problem)
  (:domain travel)

  ;; Instantiate the objects.
  (:objects
    minicooper - vehicle
    alcorcon leganes fuenlabrada mostoles madrid - city
  )

  (:init
    ; Define the initial state predicates.
    (vehicle_at minicooper fuenlabrada)

    (connected alcorcon leganes)
    (connected leganes alcorcon)
    (connected mostoles alcorcon)
    (connected alcorcon mostoles)
    (connected fuenlabrada leganes)
    (connected leganes fuenlabrada)
    (connected mostoles fuenlabrada)
    (connected fuenlabrada mostoles)
    (connected madrid leganes)
    (connected leganes madrid)
    (connected madrid alcorcon)
    (connected alcorcon madrid)

    (= (time alcorcon leganes) 10)
    (= (time leganes alcorcon) 10)
    (= (time mostoles alcorcon) 5)
    (= (time alcorcon mostoles) 5)
    (= (time fuenlabrada leganes) 10)
    (= (time leganes fuenlabrada) 10)
    (= (time mostoles fuenlabrada)  10)
    (= (time fuenlabrada mostoles) 10)
    (= (time madrid leganes) 15)
    (= (time leganes madrid) 15)
    (= (time madrid alcorcon) 15)
    (= (time alcorcon madrid) 15)
  )

  (:goal (and
    (vehicle_at minicooper madrid)
  ))

)
```

Durative actions

Continuos Effect

```
(increase (fuel ?tank) #t)
```

```
(decrease (battery ?battery) (* 5 #t))
```

```
(define (domain travel)

  (:requirements :typing :fluents :durative-actions
   :continuous-effects)

  (:types vehicle city)

  (:predicates
    (vehicle_at ?v - vehicle ?c - city)
    (visited ?c - city)
  )

  (:functions
    (fuel ?v -vehicle)
    (distance ?from ?to - city)
  )

  (:durative-action drive
    :parameters (?v - vehicle ?c1 ?c2 - city)
    :duration (= ?duration (distance ?c1 ?c2))
    :condition (and
      (at start (vehicle_at ?v ?c1))
      (at start (> (fuel ?v) (distance ?c1 ?c2)))
    )
    :effect (and
      (at start (not (vehicle_at ?v ?c1)))
      (at start (visited ?c1))
      (at end (vehicle_at ?v ?c2))
      (at end (visited ?c2))
      (decrease (fuel ?v) (* #t 1.0))
    )
  )

  (:durative-action recharge
   :parameters (?v - vehicle)
   :duration (= ?duration 10)
   :condition (and
   )
   :effect (and
     (increase (fuel ?v) (* #t 10.0))
   )
  )
)
```

```
(define (problem travel-problem)
  (:domain travel)

  ;; Instantiate the objects.
  (:objects
    minicooper - vehicle
    alcorcon leganes fuenlabrada mostoles madrid - city
  )

  (:init
    ; Define the initial state predicates.
    (vehicle_at minicooper fuenlabrada)
    (= (fuel minicooper) 100)

    (= (distance alcorcon leganes) 10)
    (= (distance leganes alcorcon) 10)
    (= (distance mostoles alcorcon) 5)
    (= (distance alcorcon mostoles) 5)
    (= (distance fuenlabrada leganes) 10)
    (= (distance leganes fuenlabrada) 10)
    (= (distance mostoles fuenlabrada)  10)
    (= (distance fuenlabrada mostoles) 10)
    (= (distance madrid leganes) 15)
    (= (distance leganes madrid) 15)
    (= (distance madrid alcorcon) 15)
    (= (distance alcorcon madrid) 15)
  )

  (:goal (and
    (forall (?c - city)
      (visited ?c)
    )
  ))
)
```

Existential preconditions

```
(exists (?c - crane)
    (crane-is-free ?c)
)
```

```
(define (domain robot-mover)
  (:requirements :strips :typing :existential-preconditions)
  (:types
    robot box location
  )
  (:predicates
    (at ?r - robot ?l - location)
    (robot_has_box ?r - robot ?b - box)
    (box-at ?b - box ?l - location)
    (adjacent ?l1 ?l2 - location)
  )

  (:action move
    :parameters (?r - robot ?from ?to - location)
    :precondition (and (at ?r ?from) (adjacent ?from ?to))
    :effect (and
              (not (at ?r ?from))
              (at ?r ?to))
  )

  (:action pick_up
    :parameters (?r - robot ?b - box ?l - location)
    :precondition (and (at ?r ?l) (box-at ?b ?l))
    :effect (and
      (not (box-at ?b ?l))
      (robot_has_box ?r ?b)
      )
  )

  (:action put_down
    :parameters (?r - robot ?b - box ?l - location)
    :precondition (and
      (at ?r ?l)
      (robot_has_box ?r ?b))
    :effect (and
      (box-at ?b ?l)
      (not (robot_has_box ?r ?b)))
  )
)
```

```
(define (problem move-box-to-target)
  (:domain robot-mover)
  (:objects
    robot1 - robot
    box1 box2 - box
    location1 location2 location3 - location
  )
  (:init
    (at robot1 location1)
    (box-at box1 location1)
    (box-at box2 location1)
    (adjacent location1 location2)
    (adjacent location2 location3)
  )
  (:goal
    (exists (?b - box)
      (box-at ?b location3)
    )
  )
)
```

Existential preconditions

```
(exists (?c - crane)
    (crane-is-free ?c)
)
```

```
(define (domain delivery-service)
  (:requirements :strips :typing :equality :existential-preconditions)
  (:types
    robot package location
  )
  (:predicates
    (at-robot ?r - robot ?l - location)
    (at-package ?p - package ?l - location)
    (connected ?l1 ?l2 - location)
    (package-destined-for ?p - package ?l - location)
  )

  (:action move-robot
    :parameters (?r - robot ?from ?to - location)
    :precondition (and (at-robot ?r ?from) (connected ?from ?to))
    :effect (and
              (not (at-robot ?r ?from))
              (at-robot ?r ?to))
  )

  (:action deliver-package
    :parameters (?r - robot ?p - package ?from ?to - location)
    :precondition (and
                    (at-robot ?r ?from)
                    (at-package ?p ?from)
                    (package-destined-for ?p ?to)
                    (exists (?l - location) (connected ?from ?l))
                    (exists (?l - location) (connected ?l ?to)))
    :effect (and
              (not (at-package ?p ?from))
              (at-package ?p ?to))
  )
)
```

```
(define (problem simple-delivery)
  (:domain delivery-service)
  (:objects
    robot1 - robot
    package1 - package
    location1 location2 location3 location4 - location
  )
  (:init
    (at-robot robot1 location1)
    (at-package package1 location1)
    (package-destined-for package1 location4)
    (connected location1 location2)
    (connected location2 location3)
    (connected location3 location4)
  )
  (:goal
    (at-package package1 location4)
  )
)
```

Existential preconditions

```
(exists (?c - crane)
    (crane-is-free ?c)
)
```

```
(define (domain library-system)
(:requirements :strips :typing :equality :existential-preconditions)
(:types
  book copy - object
  shelf - object
)
(:predicates
  (is-copy ?c - copy ?b - book)
  (available ?c - copy)
  (read ?b - book)
)

(:action borrow
  :parameters (?b - book ?s - shelf ?c - copy)
  :precondition (and
                  (exists (?c - copy) (and
                    (is-copy ?c ?b)
                    (available ?c)))
                )
  :effect (and
            (not (available ?c))
            (read ?b)
           )
)
)
```

```
(define (problem borrow-hamlet)
  (:domain library-system)
  (:objects
    hamlet - book
    copy1 copy2 - copy
    shelf1 - shelf
  )
  (:init
    (available copy1)
    (is-copy copy1 hamlet)
    (is-copy copy2 hamlet)

  )
  (:goal
    (read hamlet)
  )
)
```

Universal preconditions

```
(forall (?c - crane)
    (crane-is-free ?c)
)
```

```
(define (domain painting-service)
  (:requirements :strips :typing :universal-preconditions)
  (:types
    room house
  )
  (:predicates
    (room_clean ?r - room)
    (paint_room ?r - room)
    (paint_house ?h - house)
  )

  (:action paint_house
    :parameters (?house - house)
    :precondition (and
      (forall (?r - room)
        (paint_room ?r))
      )
    :effect (and
      (paint_house ?house)
    )
  )

  (:action paint_room
    :parameters (?r - room)
    :precondition (and
      (room_clean ?r)
    )
    :effect (paint_room ?r)
  )
)
```

```
(define (problem painting-house)
  (:domain painting-service)
  (:objects
    room1 room2 room3 - room
    houseA - house
  )
  (:init
    (room_clean room1)
    (room_clean room2)
    (room_clean room3)
  )
  (:goal
    ;(forall (?r - room)
    ;    (paint_room ?r))
    (paint_house houseA)
  )
)
```

Conditional effects

```
(when
    ;Antecedent
    (and (has-hot-chocolate ?p ?c) (has-marshmallows ?c))
    ;Consequence
    (and (person-is-happy ?p))
)
```

```
(define (domain moving)

  (:requirements :strips :typing
   :universal-preconditions :conditional-effects)

  (:types location robot thing)

  (:predicates
    (robot_at ?r - robot ?l - location)
    (thing_at ?t - thing ?l - location)
    (visible ?t - thing)
    (hidden ?t - thing)
    (checked ?t - thing)
  )

  (:action move
    :parameters (?r - robot ?from ?to - location)
    :precondition (and
      (robot_at ?r ?from)
    )
    :effect (and
          (not (robot_at ?r ?from))
          (robot_at ?r ?to)
    )
  )

  (:action check
    :parameters (?r - robot ?l - location ?t - thing)
    :precondition (and
      (robot_at ?r ?l)
      (thing_at ?t ?l)
    )
    :effect (and
      (when (visible ?t) (checked ?t))
    )
  )

  (:action look_for
    :parameters (?r - robot ?l - location ?t - thing)
    :precondition (and
      (robot_at ?r ?l)
      (thing_at ?t ?l)
      (hidden ?t)
    )
    :effect (and
      (visible ?t)
      (not (hidden ?t))
    )
  )
)
```

```
(define (problem moving-problem)
  (:domain moving)

  ;; Instantiate the objects.
  (:objects
    walle - robot
    kitchen bedroom bathroom entrance - location
    spoon knife umbrella soap book glasses - thing
  )

  (:init
    (robot_at walle entrance)
    (thing_at spoon kitchen)
    (thing_at knife kitchen)
    (thing_at umbrella entrance)
    (thing_at soap bathroom)
    (thing_at book bedroom)
    (thing_at glasses bedroom)
    (visible spoon)
    (hidden knife)
    (visible umbrella)
    (visible soap)
    (visible book)
    (visible glasses)
  )

  (:goal (and
    (forall (?t - thing)
      (checked ?t)
    )
  ))

)
```

Referencia PDDL

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

`(domain <name>)`

- Un dominio siempre empieza por `(define` y lo siguiente es la especificación del nombre del dominio del dominio.
- La mayor parte de los planificadores usan el nombre del fichero en lugar de este nombre

https://planning.wiki/

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

- No está muy soportado
- “Hereda” de otro dominio “padre” la mayor parte de sus componentes

```
(:extends <domain_name>)
```

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

- Simular a `#include` o `import`
- “Hereda” de otro dominio “padre” la mayor parte de sus componentes

```
(:requirements <requirement_name>)
```

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

- Simular a `#include` o `import`
- “Hereda” de otro dominio “padre” la mayor parte de sus componentes

```
(:requirements <requirement_name>)
```

- `:strips`
- `:typing`
- `:disjunctive-preconditions`
- `:equality`
- `:existential-preconditions`
- `:universal-preconditions`
- `:quantified-preconditions`
- `:conditional-effects`
- `:action-expansions`
- `:foreach-expansions`
- `:dag-expansions`
- `:domain-axioms`
- `:subgoal-through-axioms`
- `:safety-constraints`
- `:expression-evaluation`
- `:fluents`
- `:open-world`
- `:true-negation`
- `:adl`
- `:ucpop`
- `:numeric-fluents`
- `:durative-actions`
- `:continuous-effects`
- `:negative-preconditions`

https://planning.wiki/

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

- Permite añadir o borrar efectos

```
:effect (walls-built ?s)
:effect (not (walls-built ?s))
```

- **:strips**
- :typing
- :disjunctive-preconditions
- :equality
- :existential-preconditions
- :universal-preconditions
- :quantified-preconditions
- :conditional-effects
- :action-expansions
- :foreach-expansions
- :dag-expansions
- :domain-axioms
- :subgoal-through-axioms
- :safety-constraints
- :expression-evaluation
- :fluents
- :open-world
- :true-negation
- :adl
- :ucpop
- :numeric-fluents
- :durative-actions
- :continuous-effects
- :negative-preconditions

https://planning.wiki/

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

- Permite usar tipos

```
(:types
    site material - object
    bricks cables windows - material
)
```

- :strips
- **:typing**
- :disjunctive-preconditions
- :equality
- :existential-preconditions
- :universal-preconditions
- :quantified-preconditions
- :conditional-effects
- :action-expansions
- :foreach-expansions
- :dag-expansions
- :domain-axioms
- :subgoal-through-axioms
- :safety-constraints
- :expression-evaluation
- :fluents
- :open-world
- :true-negation
- :adl
- :ucpop
- :numeric-fluents
- :durative-actions
- :continuous-effects
- :negative-preconditions

https://planning.wiki/

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

- Permite usar or en las precondiciones

```
(or
    (walls-built ?s)
    (windows-fitted ?s)
)
```

- :strips
- :typing
- **:disjunctive-preconditions**
- :equality
- :existential-preconditions
- :universal-preconditions
- :quantified-preconditions
- :conditional-effects
- :action-expansions
- :foreach-expansions
- :dag-expansions
- :domain-axioms
- :subgoal-through-axioms
- :safety-constraints
- :expression-evaluation
- :fluents
- :open-world
- :true-negation
- :adl
- :ucpop
- :numeric-fluents
- :durative-actions
- :continuous-effects
- :negative-preconditions

https://planning.wiki/

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

- :strips
- :typing
- :disjunctive-preconditions
- **:equality**
- :existential-preconditions
- :universal-preconditions
- :quantified-preconditions
- :conditional-effects
- :action-expansions
- :foreach-expansions
- :dag-expansions
- :domain-axioms
- :subgoal-through-axioms
- :safety-constraints
- :expression-evaluation
- :fluents
- :open-world
- :true-negation
- :adl
- :ucpop
- :numeric-fluents
- :durative-actions
- :continuous-effects
- :negative-preconditions

- Permite comparar si dos objetivos son el mismo

```
(not (= ?s1 ?s2))
```

https://planning.wiki/

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

- Permite usar `forall` es goals y precondiciones

```
(forall (?c - crane)
    (crane-is-free ?c)
)
```

- :strips
- :typing
- :disjunctive-preconditions
- :equality
- :existential-preconditions
- **:universal-preconditions**
- :quantified-preconditions
- :conditional-effects
- :action-expansions
- :foreach-expansions
- :dag-expansions
- :domain-axioms
- :subgoal-through-axioms
- :safety-constraints
- :expression-evaluation
- :fluents
- :open-world
- :true-negation
- :adl
- :ucpop
- :numeric-fluents
- :durative-actions
- :continuous-effects
- :negative-preconditions

https://planning.wiki/

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

- Es equivalente a

```
(:requirements :existential-preconditions :universal-preconditions)
```

- :strips
- :typing
- :disjunctive-preconditions
- :equality
- :existential-preconditions
- :universal-preconditions
- **:quantified-preconditions**
- :conditional-effects
- :action-expansions
- :foreach-expansions
- :dag-expansions
- :domain-axioms
- :subgoal-through-axioms
- :safety-constraints
- :expression-evaluation
- :fluents
- :open-world
- :true-negation
- :adl
- :ucpop
- :numeric-fluents
- :durative-actions
- :continuous-effects
- :negative-preconditions

https://planning.wiki/

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

- Permite definir la misma acción con diferentes tipos

- :strips
- :typing
- :disjunctive-preconditions
- :equality
- :existential-preconditions
- :universal-preconditions
- :quantified-preconditions
- :conditional-effects
- **:action-expansions**
- :foreach-expansions
- :dag-expansions
- :domain-axioms
- :subgoal-through-axioms
- :safety-constraints
- :expression-evaluation
- :fluents
- :open-world
- :true-negation
- :adl
- :ucpop
- :numeric-fluents
- :durative-actions
- :continuous-effects
- :negative-preconditions

https://planning.wiki/

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

- :strips
- :typing
- :disjunctive-preconditions
- :equality
- :existential-preconditions
- :universal-preconditions
- :quantified-preconditions
- :conditional-effects
- :action-expansions
- **:foreach-expansions**
- :dag-expansions
- :domain-axioms
- :subgoal-through-axioms
- :safety-constraints
- :expression-evaluation
- :fluents
- :open-world
- :true-negation
- :adl
- :ucpop
- :numeric-fluents
- :durative-actions
- :continuous-effects
- :negative-preconditions

- Permite usar `foreach` en expansiones de acciones
- Equivalente a

```
(:requirements :action-expansions :foreach-expansions)
```

https://planning.wiki/

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

- :strips
- :typing
- :disjunctive-preconditions
- :equality
- :existential-preconditions
- :universal-preconditions
- :quantified-preconditions
- :conditional-effects
- :action-expansions
- :foreach-expansions
- :dag-expansions
- **:domain-axioms**
- :subgoal-through-axioms
- :safety-constraints
- :expression-evaluation
- :fluents
- :open-world
- :true-negation
- :adl
- :ucpop
- :numeric-fluents
- :durative-actions
- :continuous-effects
- :negative-preconditions

- Permite usar axiomas

```
(:derived (clear ?x)
   (and (not (holding ?x))
   (forall (?y) (not (on ?y ?x))))))
```

https://planning.wiki/

Intelligent Robotics Lab

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

- Permite usar axiomas como subgoals

- :strips
- :typing
- :disjunctive-preconditions
- :equality
- :existential-preconditions
- :universal-preconditions
- :quantified-preconditions
- :conditional-effects
- :action-expansions
- :foreach-expansions
- :dag-expansions
- :domain-axioms
- **:subgoal-through-axioms**
- :safety-constraints
- :expression-evaluation
- :fluents
- :open-world
- :true-negation
- :adl
- :ucpop
- :numeric-fluents
- :durative-actions
- :continuous-effects
- :negative-preconditions

https://planning.wiki/

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

- Permite definir predicados que deben ser válidos al final de la ejecución de un plan

- :strips
- :typing
- :disjunctive-preconditions
- :equality
- :existential-preconditions
- :universal-preconditions
- :quantified-preconditions
- :conditional-effects
- :action-expansions
- :foreach-expansions
- :dag-expansions
- :domain-axioms
- :subgoal-through-axioms
- **:safety-constraints**
- :expression-evaluation
- :fluents
- :open-world
- :true-negation
- :adl
- :ucpop
- :numeric-fluents
- :durative-actions
- :continuous-effects
- :negative-preconditions

https://planning.wiki/

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

- :strips
- :typing
- :disjunctive-preconditions
- :equality
- :existential-preconditions
- :universal-preconditions
- :quantified-preconditions
- :conditional-effects
- :action-expansions
- :foreach-expansions
- :dag-expansions
- :domain-axioms
- :subgoal-through-axioms
- :safety-constraints
- **:expression-evaluation**
- :fluents
- :open-world
- :true-negation
- :adl
- :ucpop
- :numeric-fluents
- :durative-actions
- :continuous-effects
- :negative-preconditions

- Permite usar `eval` en axiomas
- Si dos predicados son equivalentes, devuelve true

```
(eval (im-not-true ?a) (im-true ?b))
```

https://planning.wiki/

Intelligent Robotics Lab

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

- Permite usar `(fluents t)` en axiomas
- Cambia en PDDL 2.1, y no está claro el uso

- `:strips`
- `:typing`
- `:disjunctive-preconditions`
- `:equality`
- `:existential-preconditions`
- `:universal-preconditions`
- `:quantified-preconditions`
- `:conditional-effects`
- `:action-expansions`
- `:foreach-expansions`
- `:dag-expansions`
- `:domain-axioms`
- `:subgoal-through-axioms`
- `:safety-constraints`
- `:expression-evaluation`
- **`:fluents`**
- `:open-world`
- `:true-negation`
- `:adl`
- `:ucpop`
- `:numeric-fluents`
- `:durative-actions`
- `:continuous-effects`
- `:negative-preconditions`

https://planning.wiki/

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

- En planning, el predicado que no existe se considera falso (*closed-world assumption*)
- Esto permite que no sea así

- :strips
- :typing
- :disjunctive-preconditions
- :equality
- :existential-preconditions
- :universal-preconditions
- :quantified-preconditions
- :conditional-effects
- :action-expansions
- :foreach-expansions
- :dag-expansions
- :domain-axioms
- :subgoal-through-axioms
- :safety-constraints
- :expression-evaluation
- :fluents
- **:open-world**
- :true-negation
- :adl
- :ucpop
- :numeric-fluents
- :durative-actions
- :continuous-effects
- :negative-preconditions

https://planning.wiki/

Intelligent Robotics Lab

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

- No considera negación como fallo

- :strips
- :typing
- :disjunctive-preconditions
- :equality
- :existential-preconditions
- :universal-preconditions
- :quantified-preconditions
- :conditional-effects
- :action-expansions
- :foreach-expansions
- :dag-expansions
- :domain-axioms
- :subgoal-through-axioms
- :safety-constraints
- :expression-evaluation
- :fluents
- :open-world
- **:true-negation**
- :adl
- :ucpop
- :numeric-fluents
- :durative-actions
- :continuous-effects
- :negative-preconditions

https://planning.wiki/

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

- Implica otros requirements

- :strips
- :typing
- :disjunctive-preconditions
- :equality
- :existential-preconditions
- :universal-preconditions
- :quantified-preconditions
- :conditional-effects
- :action-expansions
- :foreach-expansions
- :dag-expansions
- :domain-axioms
- :subgoal-through-axioms
- :safety-constraints
- :expression-evaluation
- :fluents
- :open-world
- :true-negation
- **:adl**
- :ucpop
- :numeric-fluents
- :durative-actions
- :continuous-effects
- :negative-preconditions

https://planning.wiki/

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

- Implica otros requirements

- :strips
- :typing
- :disjunctive-preconditions
- :equality
- :existential-preconditions
- :universal-preconditions
- :quantified-preconditions
- :conditional-effects
- :action-expansions
- :foreach-expansions
- :dag-expansions
- :domain-axioms
- :subgoal-through-axioms
- :safety-constraints
- :expression-evaluation
- :fluents
- :open-world
- :true-negation
- :adl
- **:ucpop**
- :numeric-fluents
- :durative-actions
- :continuous-effects
- :negative-preconditions

https://planning.wiki/

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

- Permiten usar funciones, que representan valores numéricos

```
(:functions
    (battery-amount ?r - rover)
)
```

- :strips
- :typing
- :disjunctive-preconditions
- :equality
- :existential-preconditions
- :universal-preconditions
- :quantified-preconditions
- :conditional-effects
- :action-expansions
- :foreach-expansions
- :dag-expansions
- :domain-axioms
- :subgoal-through-axioms
- :safety-constraints
- :expression-evaluation
- :fluents
- :open-world
- :true-negation
- :adl
- :ucpop
- **:numeric-fluents**
- :durative-actions
- :continuous-effects
- :negative-preconditions

https://planning.wiki/

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

- Permiten usar `durative-action`

```
(:durative-action move
    :parameters (<arguments>)
    :duration (= ?duration 5)
    :condition (logical_expression)
    :effect (logical_expression)
)
```

- `:strips`
- `:typing`
- `:disjunctive-preconditions`
- `:equality`
- `:existential-preconditions`
- `:universal-preconditions`
- `:quantified-preconditions`
- `:conditional-effects`
- `:action-expansions`
- `:foreach-expansions`
- `:dag-expansions`
- `:domain-axioms`
- `:subgoal-through-axioms`
- `:safety-constraints`
- `:expression-evaluation`
- `:fluents`
- `:open-world`
- `:true-negation`
- `:adl`
- `:ucpop`
- `:numeric-fluents`
- **`:durative-actions`**
- `:continuous-effects`
- `:negative-preconditions`

https://planning.wiki/

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

- :strips
- :typing
- :disjunctive-preconditions
- :equality
- :existential-preconditions
- :universal-preconditions
- :quantified-preconditions
- :conditional-effects
- :action-expansions
- :foreach-expansions
- :dag-expansions
- :domain-axioms
- :subgoal-through-axioms
- :safety-constraints
- :expression-evaluation
- :fluents
- :open-world
- :true-negation
- :adl
- :ucpop
- :numeric-fluents
- :durative-actions
- **:durative-inequalities**
- :continuous-effects
- :negative-preconditions

- Permiten usar desigualdades para expresar duración

```
(= ?duration
 (/ (- 80 (battery-amount ?rover))
    (recharge-rate ?rover)))
```

https://planning.wiki/

Intelligent Robotics Lab

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

- :strips
- :typing
- :disjunctive-preconditions
- :equality
- :existential-preconditions
- :universal-preconditions
- :quantified-preconditions
- :conditional-effects
- :action-expansions
- :foreach-expansions
- :dag-expansions
- :domain-axioms
- :subgoal-through-axioms
- :safety-constraints
- :expression-evaluation
- :fluents
- :open-world
- :true-negation
- :adl
- :ucpop
- :numeric-fluents
- :durative-actions
- :durative-inequalities
- **:continuous-effects**
- :negative-preconditions

- Permite el uso de efectos continuos sobre números dentro de acciones durativas.

```
:effect
            (at end
                (increase (battery-amount ?rover)
                    (* ?duration (recharge-rate ?rover))))
        )
```

https://planning.wiki/

Intelligent Robotics Lab

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

- Permite usar `not` en precondiciones

- `:strips`
- `:typing`
- `:disjunctive-preconditions`
- `:equality`
- `:existential-preconditions`
- `:universal-preconditions`
- `:quantified-preconditions`
- `:conditional-effects`
- `:action-expansions`
- `:foreach-expansions`
- `:dag-expansions`
- `:domain-axioms`
- `:subgoal-through-axioms`
- `:safety-constraints`
- `:expression-evaluation`
- `:fluents`
- `:open-world`
- `:true-negation`
- `:adl`
- `:ucpop`
- `:numeric-fluents`
- `:durative-actions`
- `:durative-inequalities`
- `:continuous-effects`
- **`:negative-preconditions`**

https://planning.wiki/

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

- Definición de tipos

```
(:types
    <type_name_1> ... <type_name_n> - object
    <sub_name_1> ... <sub_name_n> - <type_name_1>
)
```

Planning Domain Definition Language
PDDL

```
(define
    (domain construction)
    (:extends building)
    (:requirements :strips :typing)
    (:types
        site material - object
        bricks cables windows - material
    )
    (:constants mainsite - site)
```

- Definición constantes que se pueden usar en el dominio

Planning Domain Definition Language
PDDL

```
(:predicates
        (walls-built ?s - site)
        (windows-fitted ?s - site)
        (foundations-set ?s - site)
        (cables-installed ?s - site)
        (site-built ?s - site)
        (on-site ?m - material ?s - site)
        (material-used ?m - material)
    )

    (:timeless (foundations-set mainsite))
```

- Definición de predicados

```
(:predicates
    (<predicate_name> <argument_1> ... <argument_n>)
)
```

Planning Domain Definition Language
PDDL

```
(:predicates
        (walls-built ?s - site)
        (windows-fitted ?s - site)
        (foundations-set ?s - site)
        (cables-installed ?s - site)
        (site-built ?s - site)
        (on-site ?m - material ?s - site)
        (material-used ?m - material)
    )

    (:timeless (foundations-set mainsite))
```

- Un predicado que siempre es verdadero

Planning Domain Definition Language
PDDL

```
(:action BUILD-WALL
        :parameters (?s - site ?b - bricks)
        :precondition (and
            (on-site ?b ?s)
            (foundations-set ?s)
            (not (walls-built ?s))
            (not (material-used ?b))
        )
        :effect (and
            (walls-built ?s)
            (material-used ?b)
        )
        ; :expansion ;deprecated
    )

    (:axiom
        :vars (?s - site)
        :context (and
            (walls-built ?s)
            (windows-fitted ?s)
            (cables-installed ?s)
        )
        :implies (site-built ?s)
    )

    ;Actions omitted for brevity
)
```

- Una acción cons sus componentes

https://planning.wiki/

Planning Domain Definition Language
PDDL

```
(:action BUILD-WALL
        :parameters (?s - site ?b - bricks)
        :precondition (and
            (on-site ?b ?s)
            (foundations-set ?s)
            (not (walls-built ?s))
            (not (material-used ?b))
        )
        :effect (and
            (walls-built ?s)
            (material-used ?b)
        )
        ; :expansion ;deprecated
    )

    (:axiom
        :vars (?s - site)
        :context (and
            (walls-built ?s)
            (windows-fitted ?s)
            (cables-installed ?s)
        )
        :implies (site-built ?s)
    )

    ;Actions omitted for brevity
)
```

- Un axioma es un predicado que se deriva de una condición

https://planning.wiki/

Planning Domain Definition Language
PDDL

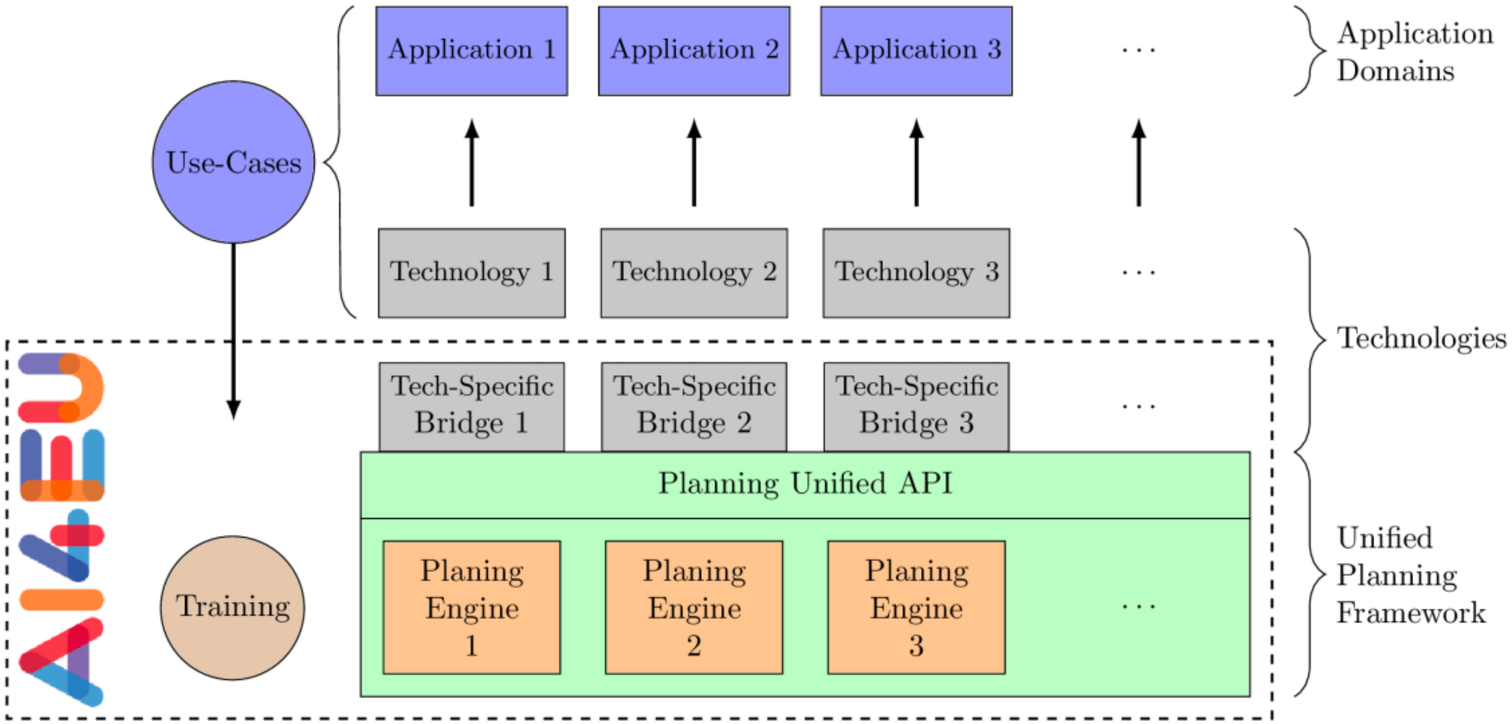
```
(:action BUILD-WALL
        :parameters (?s - site ?b - bricks)
        :precondition (and
            (on-site ?b ?s)
            (foundations-set ?s)
            (not (walls-built ?s))
            (not (material-used ?b))
        )
        :effect (and
            (walls-built ?s)
            (material-used ?b)
        )
        ; :expansion ;deprecated
    )

    (:axiom
        :vars (?s - site)
        :context (and
            (walls-built ?s)
            (windows-fitted ?s)
            (cables-installed ?s)
        )
        :implies (site-built ?s)
    )

    ;Actions omitted for brevity
)
```

- Un axioma es un predicado que se deriva de una condición

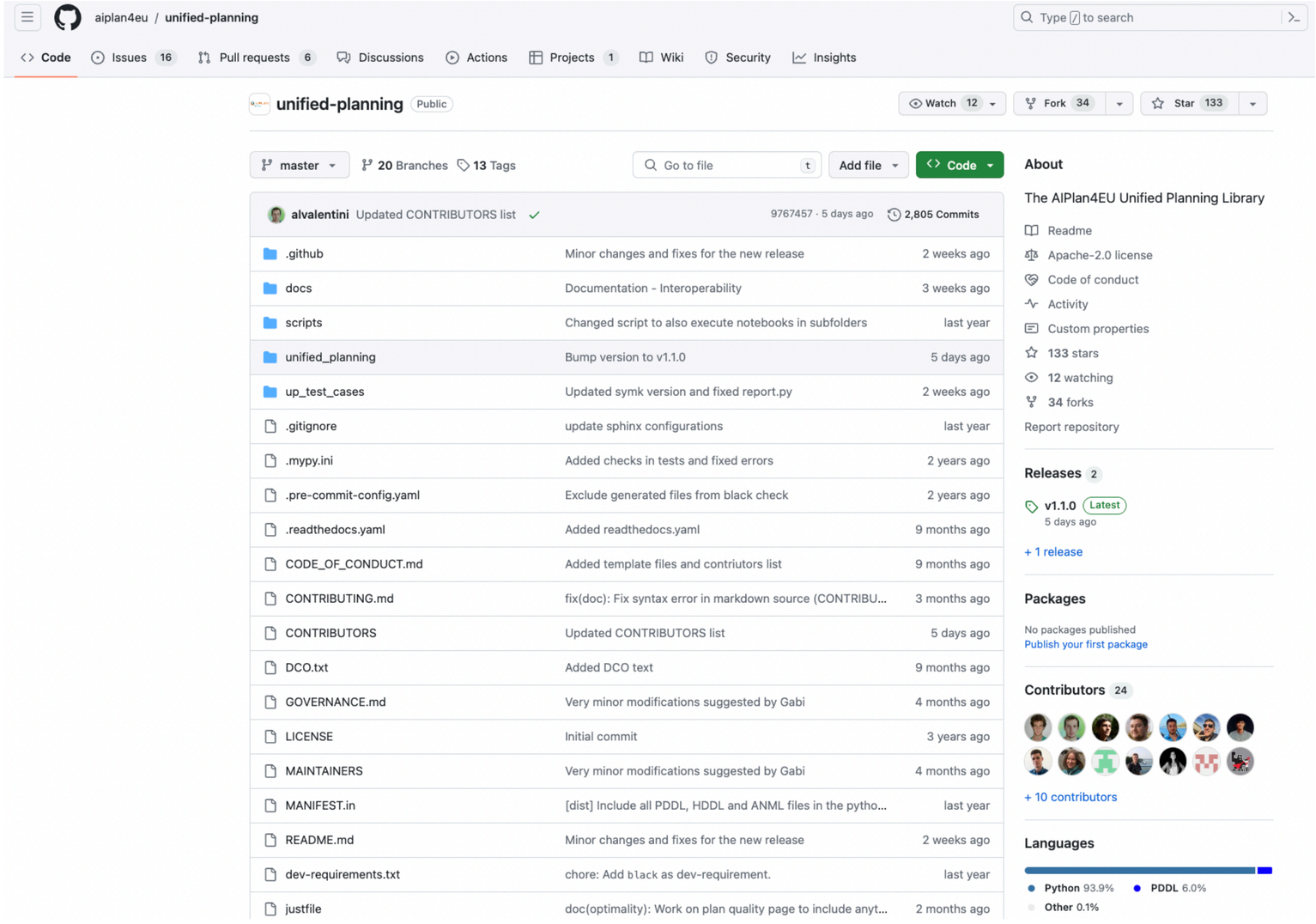
https://planning.wiki/

Universal Planning Framework (UPF)

- Proyecto en curso de Horizon Europe
- Librería escrita en Python que proporciona una capa de abstracción para usar múltiples planners



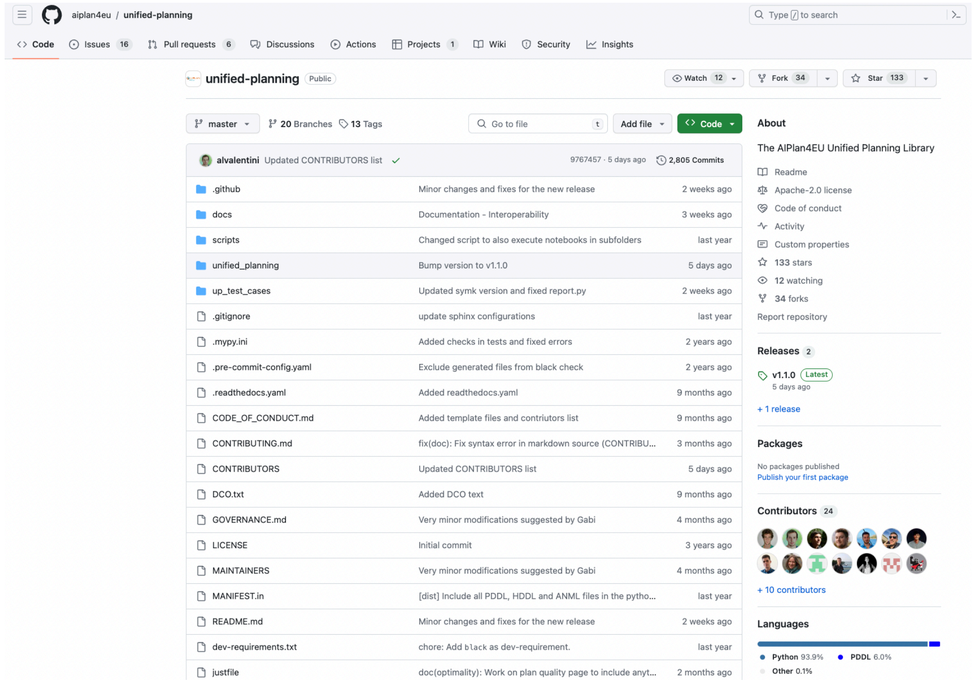
https://github.com/aiplan4eu/unified-planning

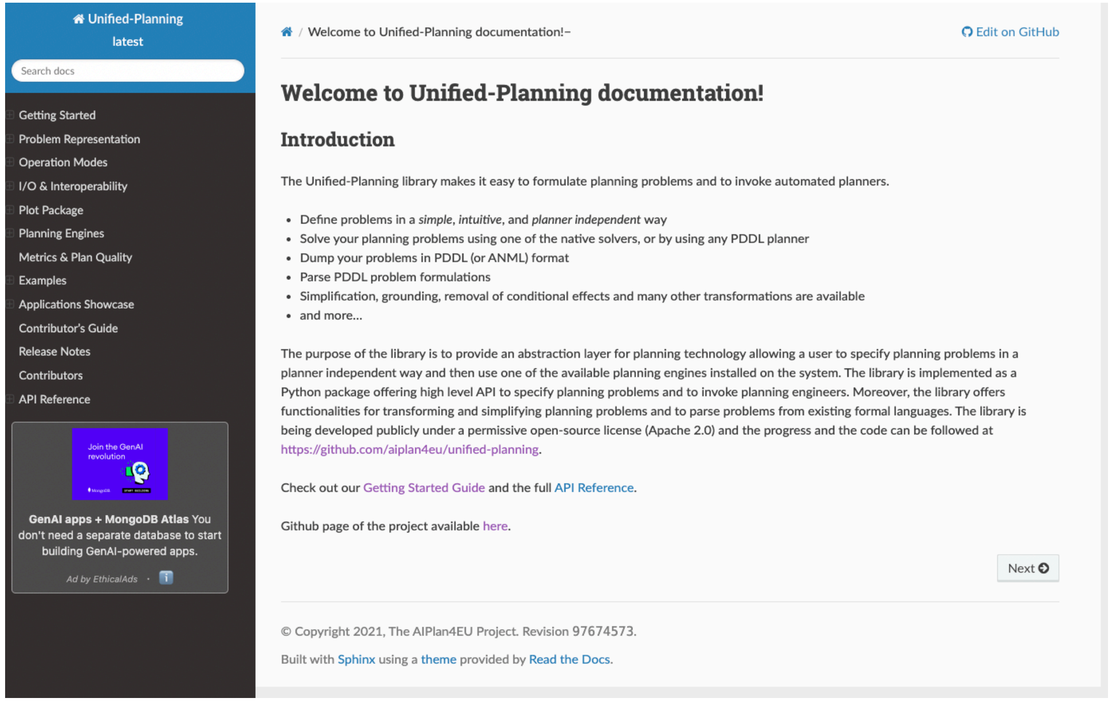


Intelligent Robotics Lab

https://github.com/aiplan4eu/unified-planning

https://upf.readthedocs.io/en/latest/





```
from unified_planning.shortcuts import *

x = Fluent("x")

a = InstantaneousAction("a")
a.add_precondition(Not(x))
a.add_effect(x, True)

problem = Problem("basic")
problem.add_fluent(x)
problem.add_action(a)
problem.set_initial_value(x, False)
problem.add_goal(x)

with OneshotPlanner(problem_kind=problem.kind) as planner:
    result = planner.solve(problem)
    if result.status in unified_planning.engines.results.POSITIVE_OUTCOMES:
        print(f"{planner.name} found this plan: {result.plan}")
    else:
        print("No plan found.")
```

```
# Import all the shortcuts, an handy way of using the unified_planning framework
from unified_planning.shortcuts import *

# Declaring types
Location = UserType("Location")

# Creating problem ‘variables’
robot_at = Fluent("robot_at", BoolType(), location=Location)
battery_charge = Fluent("battery_charge", RealType(0, 100))

# Creating actions
move = InstantaneousAction("move", l_from=Location, l_to=Location)
l_from = move.parameter("l_from")
l_to = move.parameter("l_to")
move.add_precondition(GE(battery_charge, 10))
move.add_precondition(robot_at(l_from))
move.add_precondition(Not(robot_at(l_to)))
move.add_effect(robot_at(l_from), False)
move.add_effect(robot_at(l_to), True)
move.add_effect(battery_charge, Minus(battery_charge, 10))

# Declaring objects
l1 = Object("l1", Location)
l2 = Object("l2", Location)
```

```
# Populating the problem with initial state and goals
problem = Problem("robot")
problem.add_fluent(robot_at)
problem.add_fluent(battery_charge)
problem.add_action(move)
problem.add_object(l1)
problem.add_object(l2)
problem.set_initial_value(robot_at(l1), True)
problem.set_initial_value(robot_at(l2), False)
problem.set_initial_value(battery_charge, 100)
problem.add_goal(robot_at(l2))

with OneshotPlanner(problem_kind=problem.kind) as planner:
    result = planner.solve(problem)
    if result.status in unified_planning.engines.results.POSITIVE_OUTCOMES:
        print(f"{planner.name} found this plan: {result.plan}")
    else:
        print("No plan found.")
```

Ejercicio Clase Propuesto

- Grupos de 4. Exposición Lunes 26 - 10 minutos por grupo.
- Hacer una tabla indicando qué características de PDDL soportan tres planners: POPF, OPTICS y otro a tu elección
- La fila de cada caraterística tendrá asociado un dominio y problema. Puedes usar el mismo para varias filas.
- El planner a tu elección puede ser cualquiera que encuentres y hagas funcionar: https://planning.wiki/ref/planners/atoz