



UNIVERSIDAD  
DE GRANADA



# Técnicas de los Sistemas Inteligentes

Grado en Ingeniería Informática

Curso 2021-22. Seminario 3

Planificación Clásica (PDDL) y Práctica 3

Jesús Giráldez Crú y Pablo Mesejo Santiago

**Departamento de Ciencias de la  
Computación e Inteligencia Artificial**

<http://decsai.ugr.es>

# Índice

1. ¿Qué es la Planificación?
  - i. Problema de ejemplo
  - ii. Acciones
  - iii. Representación del mundo
2. Planning Domain Definition Language (PDDL)
  - i. Ontología PDDL
  - ii. Acciones PDDL
  - iii. PDDL avanzado
3. Planificador Fast-Forward
4. Otras consideraciones
5. Práctica 3



# Índice

1. **¿Qué es la Planificación?**
  - i. Problema de ejemplo
  - ii. Acciones
  - iii. Representación del mundo
2. Planning Domain Definition Language (PDDL)
  - i. Ontología PDDL
  - ii. Acciones PDDL
  - iii. PDDL avanzado
3. Planificador Fast-Forward
4. Otras consideraciones
5. Práctica 3

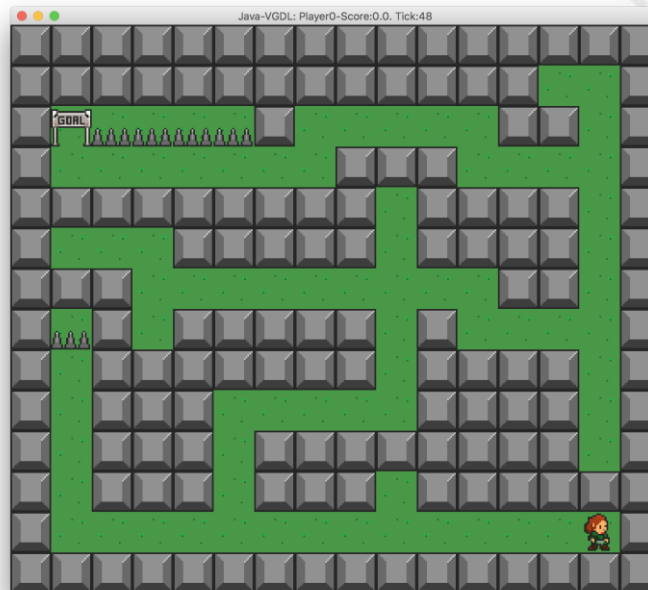


# ¿Qué es la Planificación?

En las prácticas anteriores hemos visto técnicas para que un agente resuelva un problema simple, ajustando su ejecución a ciertas restricciones de bajo nivel.

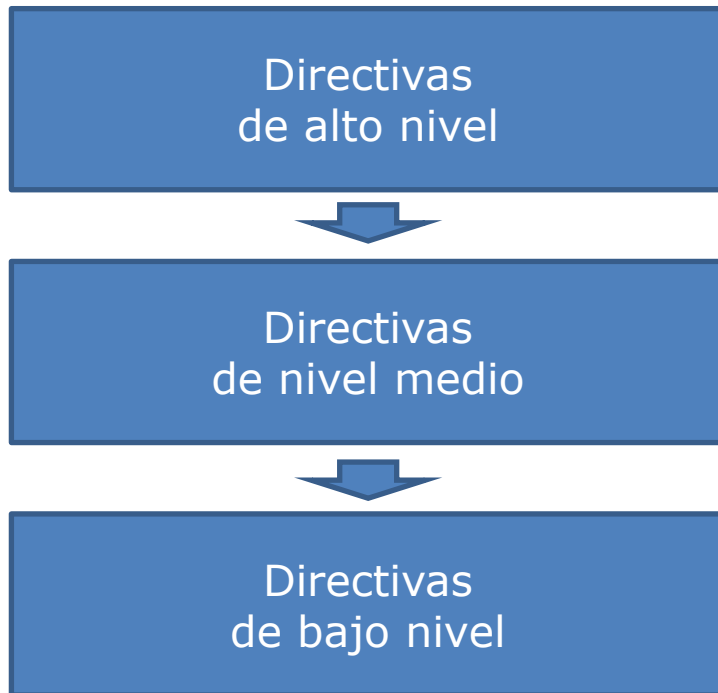
Pero...

Estas técnicas no son suficientes para resolver problemas complejos que requieran sincronizar distintos tipos de subtareas o agentes, o que requieran **razonar sobre acciones que tienen un efecto sobre el mundo.**



# ¿Qué es la Planificación?

## Planificación automática y agentes:



Determinar a alto nivel qué acciones deben ejecutarse, y en qué orden, para alcanzar un objetivo propuesto. **(Planificación automática)**

Desarrollar un plan de acción para ejecutar una tarea concreta. **(Técnicas Deliberativas de la Práctica 1)**

Ajustar el plan de acción a los requisitos del mundo/problema para ejecutarlo correctamente **(Técnicas de la Práctica 2 + Técnicas Reactivas de la Práctica 1)**

# ¿Qué es la Planificación?

Las técnicas de planificación requieren dos elementos:

1. Dominio: Un conjunto de acciones (y sus efectos esperados en el mundo).
2. Problema: Una definición del estado inicial y objetivo.

Partiendo de una representación del mundo (estado inicial), un planificador proporciona una lista de acciones que, al ser aplicadas en orden, generan una representación del mundo deseada (estado objetivo).



# Índice

1. ¿Qué es la Planificación?
  - i. **Problema de ejemplo**
  - ii. Acciones
  - iii. Representación del mundo
2. Planning Domain Definition Language (PDDL)
  - i. Ontología PDDL
  - ii. Acciones PDDL
  - iii. PDDL avanzado
3. Planificador Fast-Forward
4. Otras consideraciones
5. Práctica 3





## Problema de ejemplo: El mono y los plátanos

- Un mono en un laboratorio tiene lejos de su alcance un racimo de plátanos.
- Una caja permite alcanzar los plátanos si éste se sube en ella.
- El mono está en una posición desde la que no alcanza las bananas. Las bananas y la caja están en posiciones distintas.
- El mono puede **desplazarse** de una ubicación a otra, **empujar** la caja, **subir** a la caja, **coger** los plátanos y **bajar** de la caja.





# Índice

1. ¿Qué es la Planificación?
  - i. Problema de ejemplo
  - ii. Acciones**
  - iii. Representación del mundo
2. Planning Domain Definition Language (PDDL)
  - i. Ontología PDDL
  - ii. Acciones PDDL
  - iii. PDDL avanzado
3. Planificador Fast-Forward
4. Otras consideraciones
5. Práctica 3

# Acciones tipo STRIPS

**Acción** cogerPlátanos

Parámetros(.....)

Precondiciones(.....)

Postcondiciones(.....)

- **Cabecera**
  - Nombre de la acción y parámetros.
- **Precondiciones**
  - Condiciones que deben cumplirse para poder ejecutar la acción.
- **Postcondiciones (Efectos)**
  - Cómo modifica el mundo la acción. Definidos como:
    - Una lista de cosas que añadir al mundo.
    - Una lista de cosas que eliminar del mundo.

Stanford Research Institute Problem Solver (**STRIPS**) es el lenguaje formal usado para codificar las acciones.

- Fikes, Richard E., and Nils J. Nilsson. "STRIPS: A new approach to the application of theorem proving to problem solving." *Artificial Intelligence* 2.3-4 (1971): 189-208.

## El problema del marco

- Cada acción se aplica sobre un estado previo del mundo (pre-estado) y genera un estado sucesor (post-estado)
- ¿Qué aspectos del mundo cambian cuando la ejecutamos?
  - Esto es conocido como el Problema del Marco:
    - **¿Qué es lo que se mantiene (marco) y qué es lo que cambia en el mundo tras llevar a cabo una acción?**
- Se trata de un problema “irresoluble”, en el sentido de que en el mundo real no podemos saber con absoluta seguridad cuáles son todos los efectos de una acción.
- STRIPS intenta resolver el problema del marco a través de la hipótesis del mundo cerrado:
  - si no se sabe si una acción ha cambiado una cierta característica del dominio, se asume que no la ha cambiado.

McCarthy, John, and Patrick J. Hayes. "Some philosophical problems from the standpoint of artificial intelligence." *Machine Intelligence* 4 (1969): 463-502.

# Índice

1. ¿Qué es la Planificación?
  - i. Problema de ejemplo
  - ii. Acciones
  - iii. **Representación del mundo**
2. Planning Domain Definition Language (PDDL)
  - i. Ontología PDDL
  - ii. Acciones PDDL
  - iii. PDDL avanzado
3. Planificador Fast-Forward
4. Otras consideraciones
5. Práctica 3

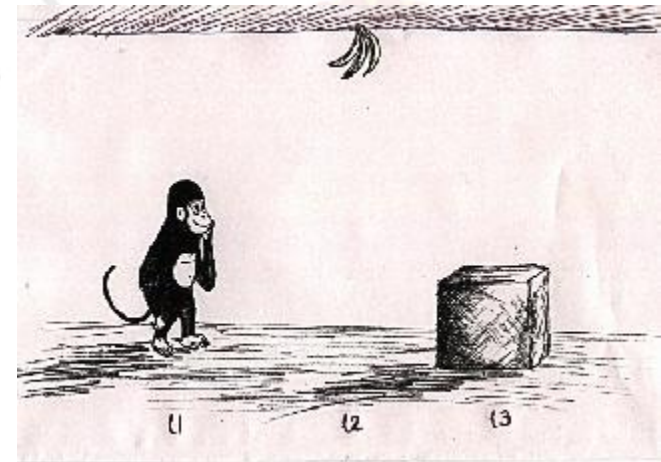


# Representación del mundo

- La representación del mundo (**Ontología**) es una descripción formal de los objetos y relaciones entre objetos del mundo.
  1. **Objetos:** los elementos del mundo.
    - Representan a los elementos del mundo.
    - Pueden tener un tipo (mono, caja)
  2. **Relaciones:** indican propiedades de los objetos.
    - Estas propiedades pueden ser intra-objeto (el mono tiene plátanos) o inter-objeto (el mono está encima de la caja)
    - Se representan usando lógica de primer orden (predicados)
      - tienePlátanos(mono)
      - sobre(mono, caja)

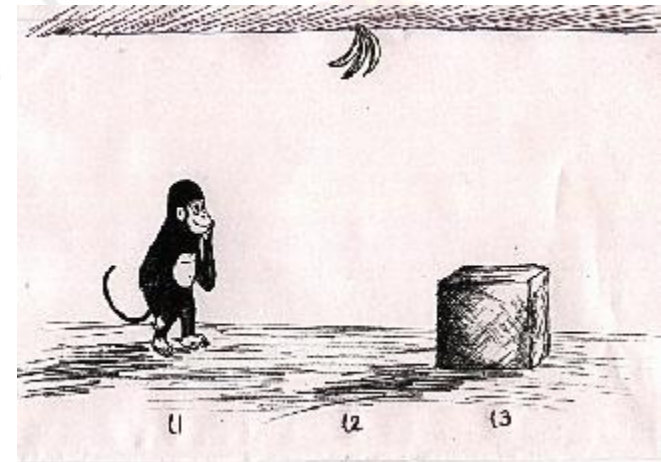
# Representación del mundo

- **Estado:** Representación del mundo en un momento concreto. Se define como una conjunción de predicados lógicos junto con una colección de objetos.
- **Objetos:**
  - Mono1 – Mono
  - Caja1 – Caja
  - Localización1 – Localización
  - Localización2 – Localización
  - Localización3 – Localización



# Representación del mundo

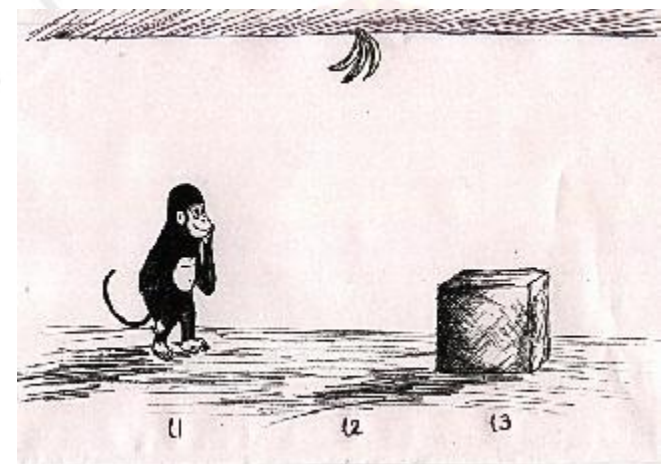
- **Estado:** Representación del mundo en un momento concreto. Se define como una conjunción de predicados lógicos junto con una colección de objetos.
- **Relaciones:**
  - En(Mono1, Localización1)
  - En(Caja1, Localización3)
  - PlatanoEn(Localización2)





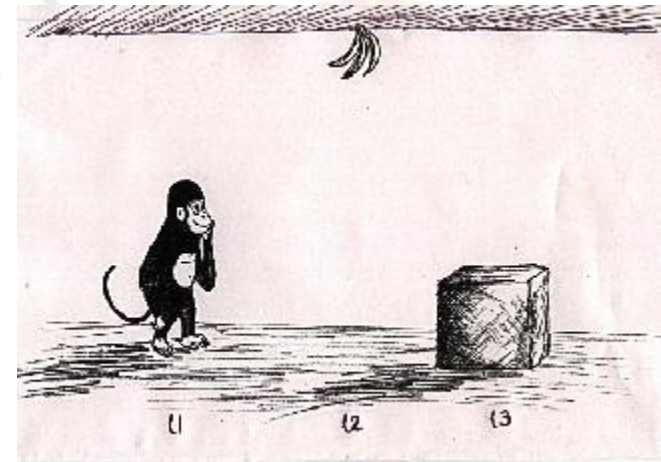
# Representación del mundo

- Los planificadores actuales siguen la **Hipótesis del Mundo Cerrado (CWA)**
  - “Si no conozco algo lo considero falso.”
- Esto evita escribir todas las relaciones falsas
  - $\neg \text{En}(\text{Mono1}, \text{Localización2})$
  - $\neg \text{En}(\text{Mono1}, \text{Localización3})$
  - $\neg \text{tienePlátano}(\text{Mono1})$
  - ...



# Representación del mundo

- **Estado Objetivo:** Representación del mundo al que queremos llegar.
  - No se requiere definir el estado completo. Solo el pequeño subconjunto de predicados que queremos que sean ciertos al finalizar el plan.
- tienePlátano(Mono1)



# Índice

1. ¿Qué es la Planificación?
  - i. Problema de ejemplo
  - ii. Acciones
  - iii. Representación del mundo
2. **Planning Domain Definition Language (PDDL)**
  - i. **Ontología PDDL**
  - ii. **Acciones PDDL**
  - iii. **PDDL avanzado**
3. Planificador Fast-Forward
4. Otras consideraciones
5. Práctica 3



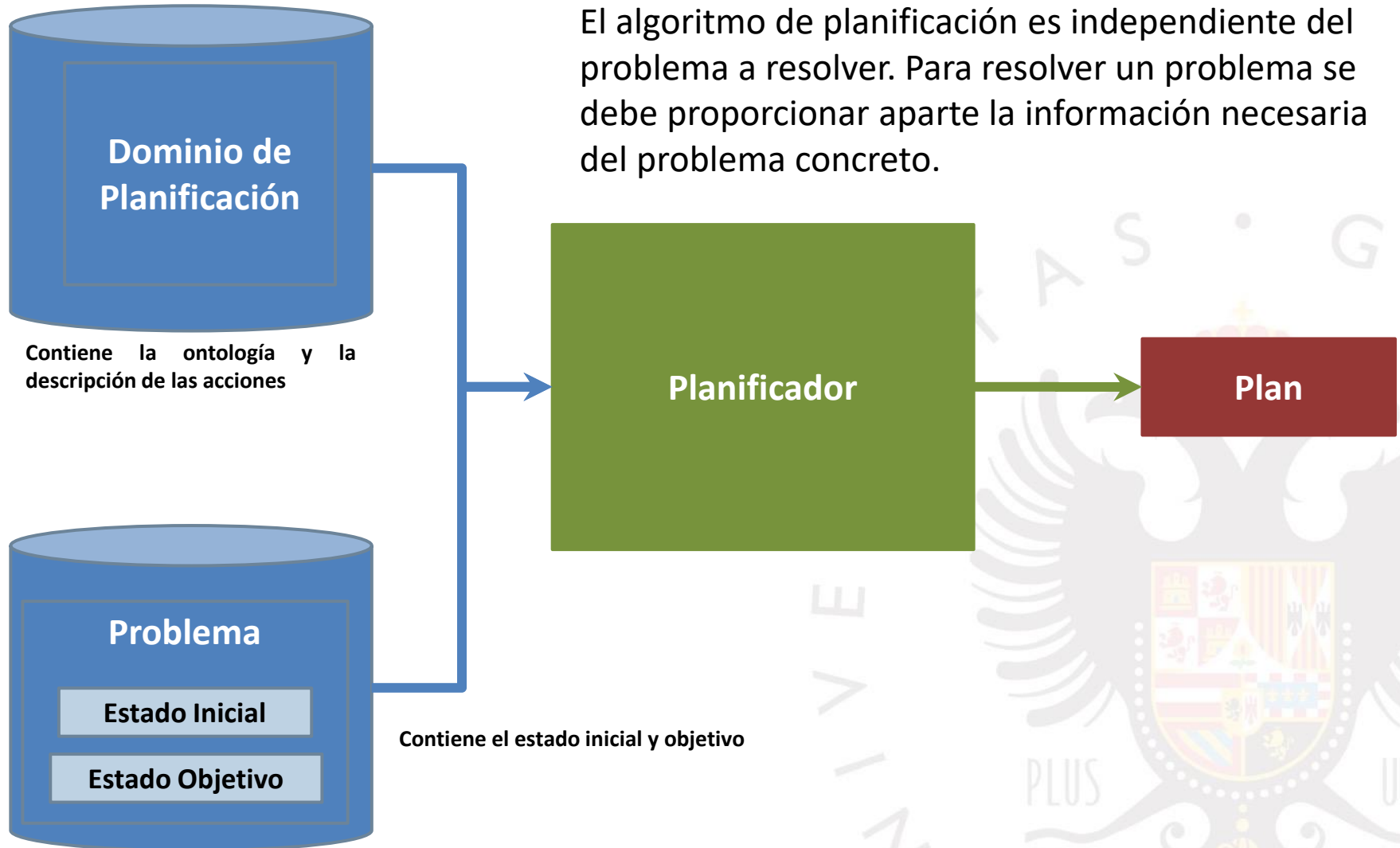
# PDDL

- PDDL
  - McDermott, Drew, et al. "PDDL-the planning domain definition language." (1998).
- Lenguaje estándar para la representación de dominios de planificación clásicos. Inspirado en STRIPS y basado en LISP.
- Todas las propiedades y relaciones entre los objetos o bien se conocen inicialmente o bien pueden conocerse durante el proceso de planificación (Conocimiento Completo)
  - Sistema completamente determinista, observable y estático.
  - Número finito de estados.
  - Los efectos de las acciones son conocidos a priori.
  - Cambios en el mundo producidos por la ejecución de las acciones.
  - Los objetivos no cambian.

# PDDL

- Material online adicional sobre PDDL
  - Writing Planning Domains and Problems (breve introducción)  
<http://users.cecs.anu.edu.au/~patrik/pddlman/writing.html>
  - Online PDDL Reference (Muy recomendado!)  
<https://github.com/jan-dolejsi/pddl-reference#contents>
  - Otros recursos didácticos sobre PDDL y planificación  
<http://education.planning.domains/>  
<https://planning.wiki/>

# Planificación Independiente de Dominio



# Dominio PDDL

```
(define (domain <domain name>)
  (:requirements <requirements spec>)
  (:types <types spec>)
  (:predicates <predicates spec>)
  (:action <action spec>)
  (:action <action spec>)
  ...
)
```

Un dominio tiene las siguientes partes:

- cabecera
- requisitos
- tipos
- predicados
- acciones





# Dominio PDDL

```
(define (domain <domain name>)
  (:requirements <requirements spec>)
  (:types <types spec>)
  (:predicates <predicates spec>)
  (:action <action spec>)
  (:action <action spec>)
  ...
)
```

Cabecera del dominio y nombre

# Dominio PDDL

```
(define (domain <domain name>)
  (:requirements <requirements spec>)
  (:types <types spec>)
  (:predicates <predicates spec>)
  (:action <action spec>)
  (:action <action spec>)
  ...
)
```

Requisitos que debe cumplir el planificador para resolver problemas con el dominio.

```
(:requirements
  :req1
  :req2
  :req3
)
```

<https://planning.wiki/ref/pddl/requirements>

# Dominio PDDL

```

(define (domain <domain name>)
  (:requirements <requirements spec>)
  (:types <types spec>)
  (:predicates <predicates spec>)
  (:action <action spec>)
  (:action <action spec>)
  ...
)
```

Permite el uso de tipado.

Similar a clases y subclases  
en Programación Orientada  
a Objetos.

```

(:requirements
 :typing
)
```

```

(:types
  movable loc - object
  mono caja plátano - movable
)
```

Tipos de los objetos del dominio

- Los tipos pueden depender de un supertipo

**Nota:** los comentarios, en PDDL, se introducen con ;

# Dominio PDDL

```

(define (domain <domain name>)
  (:requirements <requirements spec>)
  (:types <types spec>)
  (:predicates <predicates spec>)
  (:action <action spec>)
  (:action <action spec>)
  ...
)

```

Predicados de la ontología

“El objeto *x* de tipo *movible* está en la localización *y*”

“El mono *x* tiene el plátano”

“El mono *x* está sobre la caja *y*”

“El plátano está en la localización *x*”

```

(:predicates
  (en ?x - movible ?y - loc)
  (tienePlátano ?x - mono)
  (sobre ?x - mono ?y - caja)
  (plátanoEn ?x - loc)
)

```

# Dominio PDDL

```
(define (domain <domain name>)
  (:requirements <requirements spec>)
  (:types <types spec>)
  (:predicates <predicates spec>)
  (:action <action spec>)
  (:action <action spec>)
  ...
)
```

Lista de acciones del dominio

# Dominio PDDL: Acciones

- Acciones:
  - Requirement **:strips**

```
(:action irA
  :parameters (?m - mono ?x ?y - localizacion)
  :precondition
    (and
      (en ?m ?x)
    )
  :effect
    (and
      (en ?m ?y)
      (not (en ?m ?x))
    )
)
```

# Dominio PDDL: Acciones

- Cabecera:

```
(:action irA
  :parameters (?m - mono ?x ?y - localizacion)
  :precondition
    (and
      (en ?m ?x)
    )
  :effect
    (and
      (en ?m ?y)
      (not (en ?m ?x))
    )
)
```

- Tiene que contener un nombre de la acción (irA) y una lista de parámetros (?m, ?x, ?y) junto con sus tipos.

**Importante:** En las precondiciones y efectos solo se puede acceder a aquellos predicados de los que conocemos **TODOS** los argumentos.

Por ejemplo, no se podría acceder a un predicado “en” usando una localización “z” si esta no ha sido incluida en los argumentos de la acción.



# Dominio PDDL: Acciones

- Precondiciones:

```

(:action irA
  :parameters (?m - mono ?x ?y - localización)
  :precondition
    (and
      (en ?m ?x)
    )
  :effect
    (and
      (en ?m ?y)
      (not (en ?m ?x))
    )
)

```

- Las precondiciones se representan como una conjunción de predicados del mundo.

$$P1 \text{ Y } P2 \text{ Y } P3 \text{ Y } \dots \text{ Y } Pn$$

# Dominio PDDL: Acciones

- Efectos:

```
(:action irA
  :parameters (?m - mono ?x ?y - localización)
  :precondition
    (and
      (en ?m ?x)
    )
  :effect
    (and
      (en ?m ?y)
      (not (en ?m ?x))
    )
)
```

- Recordemos. Los efectos se separan en dos tipos:

- Lista de predicados a añadir al estado previo.
- Lista de predicados a borrar al estado previo (se borran con **not**)

**NOTA IMPORTANTE:** una acción NO se puede ejecutar si no se pueden insertar en el mundo los efectos de la post-condición

# Dominio PDDL

```

(define (domain mono)
  (:requirements :strips :typing)
  (:types
    movable localizacion - object
    mono caja - movable
  )
  (:predicates
    (en ?obj - movable ?x - localizacion)
    (tienePlatano ?m - mono)
    (sobre ?m - mono ?c - caja)
    (platanoEn ?x - localizacion)
  )

  (:action cogerPlatanos
    :parameters (?m - mono ?c - caja)
    :precondition
      (and
        (sobre ?m ?c)
      )
    :effect
      (and
        (tienePlatano ?m)
      )
  )
)

```

**Nota:** Los primeros pasos para trabajar con PDDL que os recomendamos más adelante, incluirán trabajar este dominio para completarlo y hacerlo más funcional.

# Fichero de problema

```
(define (problem <problem id>)
  (:domain <domain name>)
  (:objects <object spec>)
  (:init <initial facts spec>)
  (:goal <goal spec>)
)
```

Un fichero de problema tiene las siguientes partes:

- cabecera
- nombre del dominio al que pertenece
- los objetos
- el estado inicial
- el objetivo

# Fichero de problema

```
(define (problem <problem id>)
  (:domain <domain name>)
  (:objects <object spec>)
  (:init <initial facts spec>)
  (:goal <goal spec>)
)
```

Cabecera del problema y nombre

# Fichero de problema

```
(define (problem <problem id>)
```

```
  (:domain <domain name>)
```

```
  (:objects <object spec>)
```

```
  (:init <initial facts spec>)
```

```
  (:goal <goal spec>)
```

```
)
```

Nombre del dominio asociado al problema.

# Fichero de problema

```
(define (problem <problem id>)
  (:domain <domain name>)
  (:objects <object spec>)
  (:init <initial facts spec>)
  (:goal <goal spec>)
)
```

Nombre y tipo de los objetos del problema

```
(:objects
  mono1 - mono
  caja1 - caja
  platano1 - platano
  loc1 loc2 loc3 - loc
)
```



# Fichero de problema

```
(define (problem <problem id>)
  (:domain <domain name>)
  (:objects <object spec>)
  (:init <initial facts spec>)
  (:goal <goal spec>)
)
```

Estado Inicial

```
(:init
  (en mono1 loc1)
  (platanosEn loc2)
  (en caja1 loc3)
)
```

Recordad que PDDL sigue el CWA (Closed-World Assumption)

# Fichero de problema

```
(define (problem <problem id>)
  (:domain <domain name>)
  (:objects <object spec>)
  (:init <initial facts spec>)
  (:goal <goal spec>)
)
```

Objetivo a resolver

```
(:goal
  (and
    (tienePlátano mono1)
  )
)
```

# Problema PDDL

```
(define (problem monosp1)
  (:domain mono)
  (:objects
    mono1 - mono
    caja1 - caja
    localizacion1 localizacion2 localizacion3 - localizacion
  )
  (:init
    (en caja1 localizacion2)
    (platanoEn localizacion2)

    (sobre mono1 caja1)
  )
  (:goal
    (and
      (tienePlatano mono1)
    )
  )
)
```

## Consejos sobre PDDL

A la hora de escribir una acción hay que pensar que la acción es parte de un todo y se va a ejecutar junto a otras:

- Ya sea antes o después de otras acciones.

Cada acción debe cumplir un único objetivo concreto.

- Las Acciones NO deben “pisarse” entre ellas
  - Las precondiciones solo deben restringir los predicados necesarios.
  - Los efectos solo deben modificar los predicados imprescindibles.

# PDDL Avanzado (I)

## Precondiciones con disyunciones

ADL es un super-requerimiento que añade varios, entre otros :*disjunctive-preconditions*

- Requirement :**adl**
- La expresión lógica de la precondición de una acción no se limita a una conjunción (and), además puede incluir el operador de disyunción (or) y el anidado de expresiones

$P1 \text{ Y } (P2 \text{ O } (P3 \text{ Y } P4)) \text{ Y } \dots \text{ Y } Pn$

## Precondiciones con cuantificadores lógicos

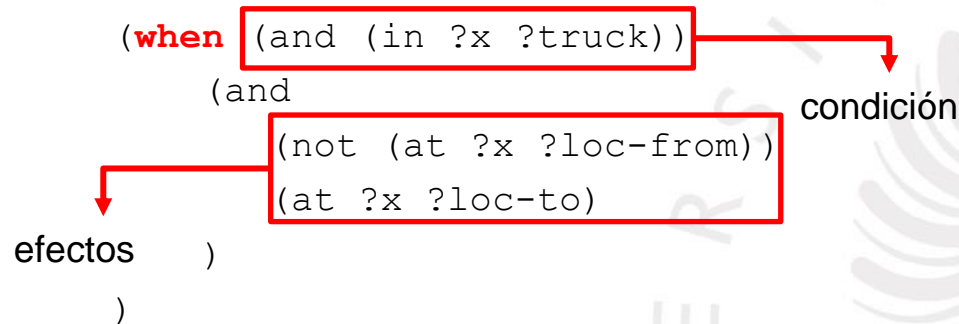
- Permiten extender precondiciones y comprobar que una expresión lógica se cumple para un conjunto de objetos.
  - Forall (todos deben cumplirla)
  - Exists (al menos uno debe cumplirla)

(**forall/exists** (?x - obj)  
(expresión lógica)  
)

# PDDL Avanzado (II)

## Efectos condicionales

- Efectos que se activan si es cierta alguna condición en el estado.



Si la condición es cierta, se ejecuta un bloque de efectos.

En el ejemplo ([https://www.cs.toronto.edu/~sheila/384/w11/Assignments/A3/veloso-PDDL\\_by\\_Example.pdf](https://www.cs.toronto.edu/~sheila/384/w11/Assignments/A3/veloso-PDDL_by_Example.pdf)): “Si un objeto *x* está dentro del camión [y el camión ha llegado a su destino], entonces dicho objeto *x* no está en la localización de origen sino en la de destino”

# PDDL Avanzado (III)

## Precondiciones condicionales

- Similar a “when”, pero para precondiciones.

```
(imply (walls-built ?s)
  (
    (foundations-set ?s)
  )
)
```

La precondición es verdadera cuando el antecedente es falso, o si antecedente y consecuente son ciertos (recordad que  $A \rightarrow B \equiv \neg A \vee B$ ).

En el ejemplo (<https://planning.wiki/ref/pddl/domain>): “Si los muros han sido construidos, entonces ya se han puesto los cimientos.”

# PDDL Avanzado (IV)

## Constantes

- Son un tipo especial de objeto.
  - Se declaran en el dominio (antes de los predicados).
  - Pueden tener tipo propio, como los objetos.
  - Una acción SIEMPRE puede acceder a una constante sin necesidad de que sea pasada como argumento. Es como una especie de “variable global”.

```
(define (domain <domain name>)
  (:requirements <requirements spec>)
  (:types <types spec>)
  (:constants
    norte sur este oeste - orientación
  )
  (:predicates <predicates spec>)
```



# PDDL Numérico (I)

## Operaciones aritméticas

- A partir de la versión 2.1 de PDDL, éste es capaz de trabajar con predicados numéricos. Al ser invocados, estos predicados devuelven como valor un número en vez de Verdadero o Falso.
  - Permite definir efectos para gestionar variables numéricas
  - Permite definir precondiciones para comprobar relaciones lógicas ( $>$ ,  $<$ ,  $!=$ )
- Requirement :**fluents**

# PDDL Numérico (II)

```
(define (domain <domain name>)
  (:requirements <requirements spec>)
  (:types <types spec>)
  (:predicates <predicates spec>)
  (:functions <functions spec>)
  (:action <action spec>)
  (:action <action spec>)
  ...
)
```

Funciones

```
(:functions
  (distancia ?x ?y - loc)
)
```

## PDDL Numérico (III)

PDDL 2.1 permite la definición de expresiones aritméticas o relacionales para las precondiciones y efectos de las acciones.

- Usando el formato LISP:

(operador (operando1) (operando2))

(>   ←   Operador  
      5   ←   Operando1  
      10   ←   Operando2  
      )

Los operandos pueden ser números

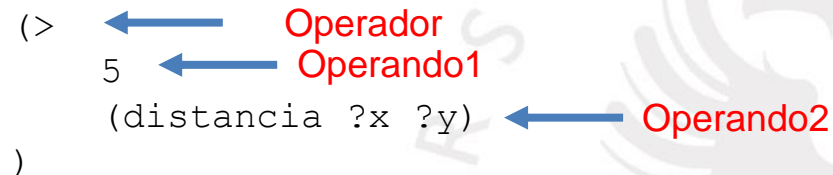
## PDDL Numérico (IV)

PDDL 2.1 permite la definición de expresiones aritméticas o relacionales para las precondiciones y efectos de las acciones.

- Usando el formato LISP:

(operador (operando1) (operando2))

(> 5 (distancia ?x ?y))



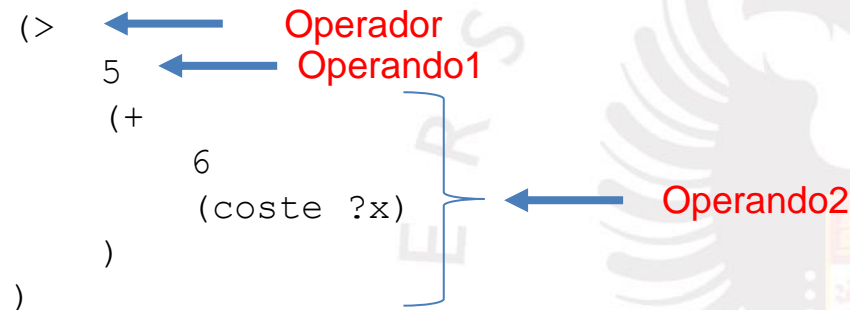
Los operandos pueden ser funciones

## PDDL Numérico (V)

PDDL 2.1 permite la definición de expresiones aritméticas o relacionales para las precondiciones y efectos de las acciones.

- Usando el formato LISP:

(operador (operando1) (operando2))



```
(> 5 (+ 6 (coste ?x)))
```

Operador

Operando1

Operando2

Los operandos pueden ser otras expresiones

## PDDL Numérico (VI)

PDDL 2.1 permite la definición de expresiones aritméticas o relacionales para las precondiciones y efectos de las acciones.

- Usando el formato LISP:

(operador (operando1) (operando2))

(>  
5  
10  
)

Operador Relacional	Operador Aritmético
!=	+
>	-
<	*
>=	/
<=	

## PDDL Numérico (VII)

Se implementan operadores para modificar las funciones:

- (assign (funcion) (valor))
- (increase (funcion) (valor))
- (decrease (funcion) (valor))

```
(decrease
  (energíaTotal ?x)
  (coste ?x ?y)
)
```

El valor puede ser un número  
constante, otra función o una  
operación aritmética



Estos operadores son la **ÚNICA** forma que tienen las acciones de modificar las funciones.

## PDDL Numérico (VIII)

En el estado inicial las funciones se instancian a un valor concreto con las sentencias

```
(= (tienePlatanos mono1) 0)
```

PDDL permite definir métricas numéricas que optimizar

- Por defecto, un planificador intenta minimizar el numero de acciones de un plan.
- Definir una métrica fuerza al planificador a explorar nuevas soluciones para cumplir con el requisito.

```
(:init <initial facts spec>)  
(:goal <goal spec>)  
(:metric minimize (operación aritmética))
```



# Índice

1. ¿Qué es la Planificación?
  - i. Problema de ejemplo
  - ii. Acciones
  - iii. Representación del mundo
2. Planning Domain Definition Language (PDDL)
  - i. Ontología PDDL
  - ii. Acciones PDDL
  - iii. PDDL avanzado
3. **Planificador Fast-Forward**
4. Otras consideraciones
5. Práctica 3



# Planificador FF (Fast-Forward)

- FF:
  - Hoffmann, Jörg, and Bernhard Nebel. "The FF planning system: Fast plan generation through heuristic search." *Journal of Artificial Intelligence Research* 14 (2001): 253-302.
  - Es uno de los más usados y referenciados en la actualidad.
  - Es de los pocos planificadores que admiten información numérica y permiten "optimizar".
- Implementa un algoritmo de búsqueda heurística para estudiar el espacio de posibles estados del mundo
  - Genera sucesores a partir del estado inicial aplicando operadores (acciones) permitidos en cada estado hasta llegar al objetivo.
    - Una acción se permite si todas sus precondiciones son satisfechas por el estado.
    - Sucesores: Estados posteriores de cada acción.
  - Condición de parada
    - Todos los predicados del objetivo están incluidos en el estado actual.

# Planificador FF (Fast-Forward)

Para el desarrollo de esta práctica usaremos el planificador MetricFF en su primera versión, que se puede descargar en: <https://fai.cs.uni-saarland.de/hoffmann/ff/Metric-FF.tgz>

Para ejecutar:

**`./ff -o <dominio.pddl> -f <problema.pddl> -O -g 1 -h 1`**

La opción “-O” le indica al planificador que se desea optimizar el plan encontrado (si bien no existen garantías de que el plan encontrado sea óptimo), mientras que las opciones “-g 1” y “-h 1” nos sirven para establecer la función de coste y la función heurística usada por A\* para encontrar el plan. Usando estos valores se facilita que MetricFF se acerque a la solución óptima.

Hoffmann, Jörg. "The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables." *Journal of artificial intelligence research* 20 (2003): 291-341.

# Planes óptimos en MetricFF

- MetricFF incluye una opción de optimización:
  - En general, intenta optimizar la longitud del plan (*plan length*)
  - Si se define una métrica concreta, intenta optimizar ésta
- No obstante, esta optimización se debe entender como una **aproximación**:
  - Demostrar que un plan es óptimo es una tarea computacionalmente muy costosa.
  - Por esta razón, la mayoría de los planificadores "se conforman" con aproximar (y no demostrar) un plan suficientemente "bueno" (es decir, "cercano" al óptimo).

**Nota:** independientemente de estas consideraciones, en esta práctica veremos una forma de obtener un plan óptimo de forma manual.

# Planificador FF (Fast-Forward)

Breves notas con respecto a la instalación de FF:

- Os recomendamos emplear Linux.
  - Suele haber problemas para encontrar *planners* que funcionen en Windows, pero si es para Linux hay más donde elegir
- 1. Descomprimir el fichero (`tar zxvf Metric-FF.tgz`)
- 2. Instalar flex (`sudo apt-get install flex`) y bison (`sudo apt-get install bison`)
- 3. Compilar el planificador usando `make`
- 4. Si usáis gcc 11.2.0 podéis tener problemas. Nuestro consejo es que instaléis una versión anterior de gcc, por ejemplo, gcc 9.3.0.
  - `sudo apt install gcc-9`
  - `sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-9 100`

# Planificador FF (Fast-Forward)

```
pablo@pablo-VirtualBox:/media/sf_VirtualBox_Shared_Folders/PDDL/Metric-FF$ ./ff -o ../MundoBloquesDom.pddl -f ../MundoBloquesProb.pddl -O -g 1 -h 1

ff: parsing domain file
domain 'BLOCKSWORLD' defined
... done.
ff: parsing problem file
problem 'BWP1' defined
... done.

no metric specified. plan length assumed.

checking for cyclic := effects --- OK.

ff: search configuration is best-first on  $1*g(s) + 1*h(s)$  where
    metric is plan length

advancing to distance:  5
                      4
                      3
                      2
                      1
                      0

ff: found legal plan as follows

step   0: PUT_DOWN B5
       1: UNSTACK B2 B4
       2: PUT_DOWN B2
       3: PICK_UP B5
       4: STACK B5 B2
       5: UNSTACK B3 B1
       6: STACK B3 B5

time spent:  0.00 seconds instantiating 60 easy, 0 hard action templates
            0.00 seconds reachability analysis, yielding 41 facts and 60 actions
            0.00 seconds creating final representation with 41 relevant facts, 0 relevant fluents
            0.00 seconds computing LNF
            0.00 seconds building connectivity graph
            0.00 seconds searching, evaluating 28 states, to a max depth of 0
            0.00 seconds total time
```

# Índice

1. ¿Qué es la Planificación?
  - i. Problema de ejemplo
  - ii. Acciones
  - iii. Representación del mundo
2. Planning Domain Definition Language (PDDL)
  - i. Ontología PDDL
  - ii. Acciones PDDL
  - iii. PDDL avanzado
3. Planificador Fast-Forward
4. **Otras consideraciones**
5. Práctica 3





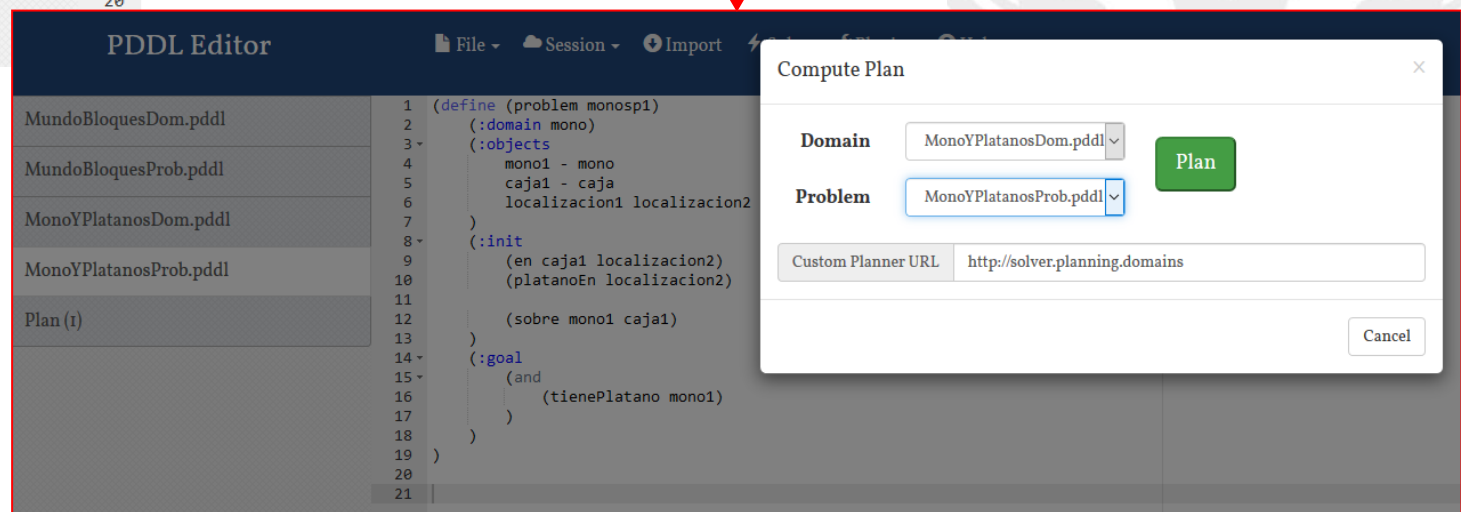
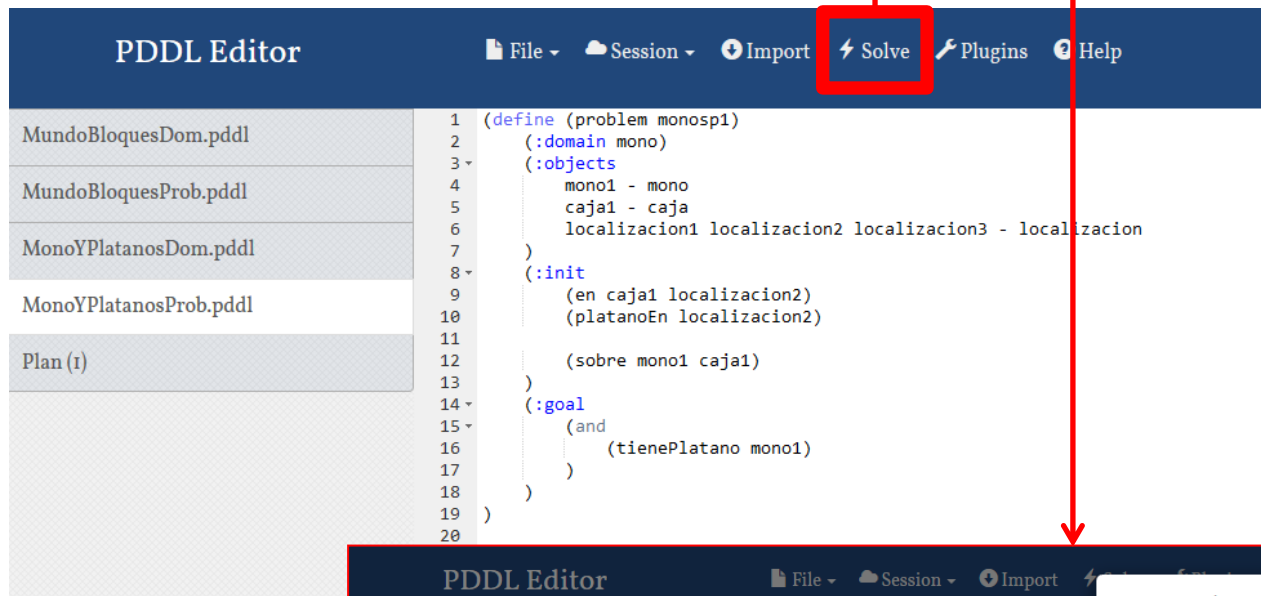
# Editores

1. Un editor de texto (Gedit, Atom o Sublime Text)
2. PDDL Online Editor (<http://editor.planning.domains/>)
  - Permite ejecutar los dominios según se construyen.
  - Adecuado para ejercitarse online con problemas de prueba y, luego, usar FF para resolver los problemas de la práctica.
  - El editor online tiene autoguardado. Pero es recomendable no abandonar la página antes de guardar localmente todo el trabajo hecho.
3. Visual Studio tiene una extensión para PDDL
  - <https://marketplace.visualstudio.com/items?itemName=jan-dolejsi.pddl>



# PDDL Online Editor

(<http://editor.planning.domains/>)



# Cómo depurar en PDDL

Cómo verificar **sintaxis** y **semántica** de la acción.

- Análisis sintáctico

- Solo se puede hacer lanzando un dominio + problema en un planificador y viendo dónde falla.

- Análisis semántico

- Diseñar y verificar un dominio de planificación es un proceso manual:
  1. Definir un dominio con una sola acción.
  2. Definir un problema cuyo plan sea ejecutar esa acción.
  3. Lanzar el planificador y comprobar que el dominio sea correcto.
  4. Incluir una nueva acción al dominio.
  5. Modificar el problema para que el plan contenga la nueva acción.
  6. Lanzar el planificador y comprobar que el dominio sea correcto.
  7. Repetir 4 – 6 hasta terminar.

## Primeros pasos

1. Revisar con calma los ejemplos (mundo de bloques y mono/plátanos) proporcionados en [http://editor.planning.domains/#read\\_session=wQdLTQZvUN](http://editor.planning.domains/#read_session=wQdLTQZvUN)
2. En relación al problema del mono y los plátanos:
  - I. Modificar la localización de los plátanos.
    - ¿Es correcto el resultado del planificador? Arreglar si no lo es.
  - II. Incluir una nueva acción para que el mono empuje la caja.
    - El mono debe estar en la misma posición que la caja.
    - Cuando la empuja el mono debe moverse con ella.
  - III. Incluir una nueva acción para que el mono suba a la caja.
    - El mono debe estar en la misma posición que la caja.
    - Y no debe estar ya encima de ella.
  - IV. Definir una acción para que el mono sea capaz de moverse por el mundo sin necesidad de empujar la caja.

## Consideraciones generales y preguntas frecuentes

### 1. Si tenemos un objeto, ¿este puede tener atributos?

- Supongamos que tenemos un objeto de tipo persona que se llama Pablo. Si queremos que ese objeto tenga información extra (por ejemplo, el nombre de su padre) deberíamos usar un predicado lógico (padreDe Pablo Jesus). Si queremos, podríamos determinar su peso con  $(= (\text{peso Pablo}) 65)$  usando funciones. Así es como se puede añadir información extra a un objeto.

### 2. ¿Cómo es posible definir un *grid* en PDDL?

- De nuevo, empleando lógica de predicados. Si tenemos un grid de 2x2, y las localizaciones (A, B, C, D) están conectadas entre ellas si son adyacentes (horizontal y verticalmente, pero no diagonalmente), se podría representar así con lógica de predicados en el fichero del problema:

A	B
C	D

(conectado A B)  
 (conectado A C)  
 (conectado B A)  
 (conectado B D)  
 (conectado C A)  
 (conectado C D)  
 (conectado D B)  
 (conectado D C)

## Consideraciones generales y preguntas frecuentes

3. Ejecuto mi dominio y mi problema, no obtengo ningún error, pero tampoco se genera ningún plan. ¿Qué puede estar pasando?
  - Este comportamiento suele ir asociado al mensaje *predicate xxxxxx is declared to use unknown or empty type yyyyyy*. Generalmente, se corresponde a no incluir un espacio entre el objeto definido y su tipo, por ejemplo: *A1 A2 A3- Unidad* en lugar de *A1 A2 A3 - Unidad*
4. Mi código tarda mucho en ejecutarse. ¿A qué puede deberse?
  - Evidentemente, este comportamiento puede estar asociado con muchas causas. Pero una de ellas es poner al revés un *forall* y un *exist*: dependiendo de su colocación, podríamos anular la capacidad de *pruning* del algoritmo de búsqueda, llegando hasta las hojas y creando un árbol inmenso.

# Índice

1. ¿Qué es la Planificación?
  - i. Problema de ejemplo
  - ii. Acciones
  - iii. Representación del mundo
2. Planning Domain Definition Language (PDDL)
  - i. Ontología PDDL
  - ii. Acciones PDDL
  - iii. PDDL avanzado
3. Planificador Fast-Forward
4. Otras consideraciones
5. **Práctica 3**



- La práctica consiste en resolver 8 ejercicios de planificación inspirados en el videojuego StarCraft (ver enunciado de la práctica):
  - **Cada ejercicio, a nivel de código, tiene una puntuación de 1 punto.** Los ejercicios que tengan **errores sintácticos califican con cero puntos**, independientemente de la documentación entregada para ese ejercicio. **El código debe estar bien comentado.**
  - Se entrega **memoria (3 págs. como máximo)**, donde se responda a algunas preguntas planteadas. La calidad de la memoria (contenido+forma) puntúa **hasta un máximo de 2 puntos.**
- La entrega se realizará a través de PRADO, y consistirá en un **fichero ZIP** que contenga **2 ficheros escritos en PDDL para cada ejercicio** (uno de dominio y otro de problema), y el **fichero PDF** con la memoria anteriormente mencionada.

**Fecha de entrega: 08 de Junio de 2022 23:59**





UNIVERSIDAD  
DE GRANADA



# Técnicas de los Sistemas Inteligentes

Grado en Ingeniería Informática

Curso 2021-22. Seminario 3

Planificación Clásica (PDDL) y Práctica 3

Jesús Giráldez Crú y Pablo Mesejo Santiago

**Departamento de Ciencias de la  
Computación e Inteligencia Artificial**

<http://decsai.ugr.es>