TÉCNICAS DE LOS SISTEMAS INTELIGENTES

Práctica3: Planificación.
Sesion1. Planificación clásica

Objetivos

- Sesión 1
  - Conocer el lenguaje PDDL como el estándar para la descripción de dominios y problemas de planificación.
  - Definir dominios y problemas de planificación.
  - Usar el planificador FF (Fast Forward) para resolver problemas de planificación descritos en PDDL.
- Sesión 2
  - Conocer el lenguaje HPDL como un lenguaje para describir problemas de planificación jerárquica de tareas (HTN: Hierarchical Task Networks)
  - Definir dominios y problemas de planificación jerárquica
  - Usar el planificador lactive Planner (desarrollado por nuestro grupo de investigación y comercializado por lactive) para resolver problemas de planificación jerárquica.

PDDL

- Planning Domain Definition Language
- https://en.wikipedia.org/wiki/Planning_Domain_Definition_Language
- Lenguaje estándar para la representación de dominios de planificación clásicos
  - Conocimiento completo : todas las propiedades y relaciones entre los objetos o se conocen inicialmente o pueden conocerse durante la planificación
    - Hipótesis del mundo cerrado: los hechos no especificados son falsos.
  - Acciones deterministas
    - Los efectos de las acciones son conocidos a priori
  - Cambios en el mundo sólo producidos por la ejecución de acciones.
    - No se consideran eventos exógenos.

Componentes de PDDL

- Objetos: cosas del mundo que nos interesa representar
- Predicados: propiedades o relaciones de objetos de interés (pueden ser ciertos o falsos)
- Estado inicial: El estado del mundo a partir del que empezamos a planificar.
- Objetivo: Cosas que queremos que sean ciertas cuando se ejecute el plan.
- Acciones/Operadores: Maneras de cambiar el estado del mundo.

Dominio y problema

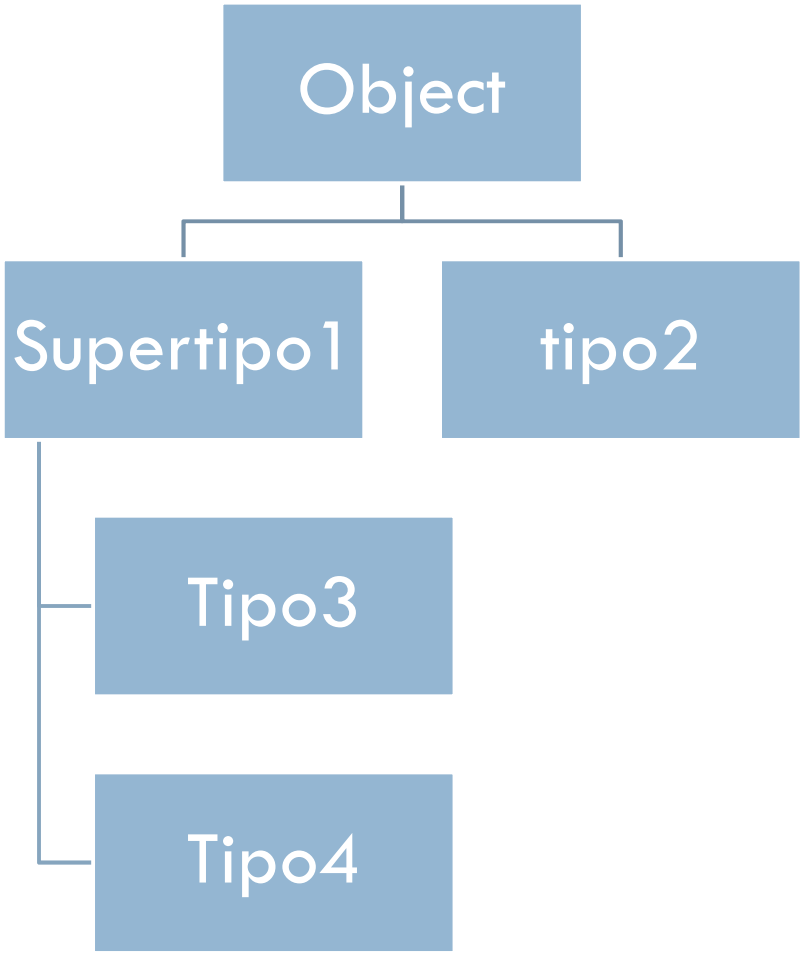
- Una tarea de planificación en PDDL se especifica en dos ficheros:
- Un fichero de dominio:
  - especificación de objetos, predicados y acciones
- Un fichero de problema
  - Objetos (constantes), estado inicial y especificación del objetivo.
- Writing Planning Domains and Problems
  - http://users.cecs.anu.edu.au/~patrik/pddlman/writing.html

Ficheros de dominio

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

- Requirements:
  - :strips
    - El subconjunto más básico de PDDL, solo la representación introducida en STRIPS
  - :adl
    - Disyunciones y cuantificadores en precondiciones y objetivos
    - Efectos condicionales y universalmente cuantificados.
  - :typing
    - El dominio usa tipos.

```
(:types supertipo1 tipo2 - object
        tipo3 tipo4 - supertipo1
```



Ejemplo

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

Ejemplo - Acciones

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

(:action put-down
        :parameters (?x - block)
        :precondition (holding ?x)
        :effect
        (and (not (holding ?x))
          (clear ?x)
          (handempty)
          (ontable ?x)))
```

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

Fichero de problema

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

Ejemplo - Problema

```
(define (problem BLOCKS-4-0)

(:domain BLOCKS)

(:objects D B A C - block)

(:INIT  (CLEAR C)
   (CLEAR A)
   (CLEAR B)
   (CLEAR D)
   (ONTABLE C)
   (ONTABLE A)
   (ONTABLE B)
   (ONTABLE D)
   (HANDEMPTY))

(:goal (AND (ON D C) (ON C B) (ON B A)))
)
```

1. Copiar blocks_domain.pddl y blocks_problem.pddl en directorio <dir>
2. Cd <metric-ff>
3. ff –p <dir> -o blocks_domain.pddl –f blocks_problem.pddl

Efectos condicionales

```
(:action drive-truck
   :parameters (?truck - truck ?loc-from ?loc-to - location
                           ?city - city)
   :precondition (and  (at ?truck ?loc-from)
                       (in-city ?loc-from ?city)
                       (in-city ?loc-to ?city))
   :effect (and  (at ?truck ?loc-to)
                 (not (at ?truck ?loc-from))
                 (forall (?x - obj)
                   (when (and (in ?x ?truck))
                         (and (not (at ?x ?loc-from))
                              (at ?x ?loc-to))
                   )
                 )
            )
)
```

- PDDL ha evolucionado en distintas versiones:
  - «Planning Domain Definition Language», *Wikipedia, the free encyclopedia*. 13-feb-2015.
  - PDDL 2.1: uso de recursos y tiempo (y otras extensiones).
  - PDDL 3: uso de preferencias (y otras extensiones).
- PDDL 2.1 permite representar el uso de recursos
  - Ejemplo el consumo de fuel de un camión: cada vez que se mueve el camión una cantidad de fuel usado por el camión se actualiza.
  - Ejemplo: verter el contenido de una jarra en otra.

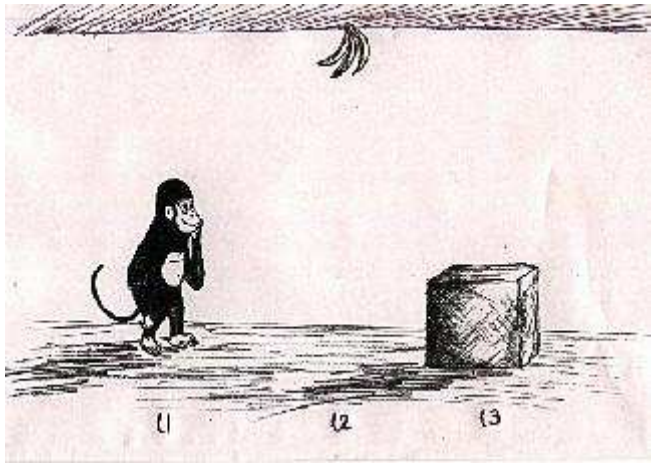
```
(define (domain jug-pouring)
    (:requirements :typing :fluents)
    (:types jug)
    (:functions
            (amount ?j - jug)
            (capacity ?j - jug))
    (:action pour
    :parameters (?jug1 ?jug2 - jug)
    :precondition
            (>=(-(capacity ?jug2)(amount ?jug2))(amount ?jug1))
    :effect (and    (assign (amount ?jug1) 0)
            (increase (amount ?jug2) (amount ?jug1))) )
```

```
(define (domain jug-pouring)
    (:requirements :typing :fluents)
    (:types jug)
    (:functions
            (amount ?j - jug)
            (capacity ?j - jug))
    (:action pour
    :parameters (?jug1 ?jug2 - jug)
    :precondition
            (>=(-(capacity ?jug2)(amount ?jug2))(amount ?jug1))
    :effect (and    (assign (amount ?jug1) 0)
            (increase (amount ?jug2) (amount ?jug1))) )
```

```
(define (problem Jarras1)
(:domain jug-pouring)
(:objects
            jarra1 jarra2 - jug
            )
(:init
            (= (capacity jarra1) 4)
            (= (capacity jarra2) 6)
            ...
)
(:goal (amount jarra2 3)... ))
```

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 también.
- El mono puede **desplazarse** de una ubicación a otra, **subir o bajarse** de un objeto y **coger o soltar** objetos.
- ¿Qué tiene que hacer el mono para coger los plátanos.



Algunas consideraciones

- Representación
  - ¿Cómo representamos el estado del mundo?
  - ¿Cómo representamos acciones?
  - ¿Hace nuestra representación fácil
    - la comprobación de precondiciones
    - representar el estado del mundo después de ejecutar cada acción
    - reconocer/alcanzar el estado final?

- Ontología (Objetos, propiedades y relaciones) del problema del Mono y los Plátanos
  - Objetos
    - Ubicaciones
  - Relaciones entre objetos:
  - Escoger predicados adecuados y argumentos:

- Ontología del problema del Mono y los Plátanos
  - Objetos:mono, caja, platanos,suelo
    - Ubicaciones: A,B,C,....
  - Relaciones entre objetos:
    - el mono está en una ubicación
    - el mono está o no en el suelo
    - los plátanos están: colgados o cogidos)
    - la caja está en la misma ubicación que los plátanos ...
  - Escoger predicados adecuados y argumentos:
    - `at(mono,?x)`
    - `on(mono,suelo)`
    - `status(platanos,colgados)`
    - `at(caja,?x), at(platanos,?x)`

Estado inicial y final

Cuando hayamos definido los predicados, podemos usarlos para definir estados iniciales y finales (el problema).

- Estado inicial:
  - on(mono, suelo)
  - on(caja,suelo)
  - at(mono, a)
  - at(caja,b)
  - at(platanos,c)
  - status(platanos, colgados)

- Estado final:
  - on(mono,caja)
  - on(caja,suelo)
  - at(mono,c)
  - at(caja,c)
  - at(platanos,c)
  - status(platanos,cogidos)

Objetivo

- El objetivo no tiene que corresponderse con un estado final completo.
- Basta con especificar qué predicados queremos que se hagan verdad en el estado final. Por ejemplo:
  - status(platanos,cogidos)

Representando operadores

- Nombre y parámetros.
  - Los parámetros deben estar en las precondiciones.
- Precondiciones:
  - qué predicados tienen que ser ciertos para que se aplique el operador (acción)
- Efectos : qué predicados representan el cambio del mundo representado por la acción
  - STRIPS:
    - Adición: qué predicados se añaden al mundo
    - Supresión: qué predicados son eliminados del mundo
  - En pddl una única expresión lógica,
    - Si queremos especificar que se suprime el predicado P → not(P)

El problema del marco

- ¿Qué aspectos del mundo cambian realmente cuando ejecutamos una acción?
  - Cuando representamos acciones asumimos que los únicos efectos (en la realidad) de nuestro operador son los especificados.
  - En el mundo real es una restricción muy fuerte, puesto que no podemos saber con absoluta seguridad cuales son los efectos de la acción
  - Esto es conocido com el Problema del Marco
- En los sistemas de planificación reales es difícil encontrar una solución de compromiso, y los dominios de planificación deben de adaptarse a medida que se va conociendo la ejecución en el mundo real.

Representación de acciones

Actions	Preconditions	Delete List	Add List
go(X,Y)	at(monkey,X)	at(monkey,X)	at(monkey,Y)
	on(monkey, floor)		
push(B,X,Y)	at(monkey,X)	at(monkey,X)	at(monkey,Y)
	at(B,X)	at(B,X)	at(B,Y)
	on(monkey,floor)		
	on(B,floor)		
climb_on(B)	at(monkey,X)	on(monkey,floor)	on(monkey,B)
	at(B,X)		
	on(monkey,floor)		
	on(B,floor)		
grab(B)	on(monkey,box)	status(B,hanging)	status(B,grabbed)
	at(box,X)		
	at(B,X)		
	status(B,hanging)		

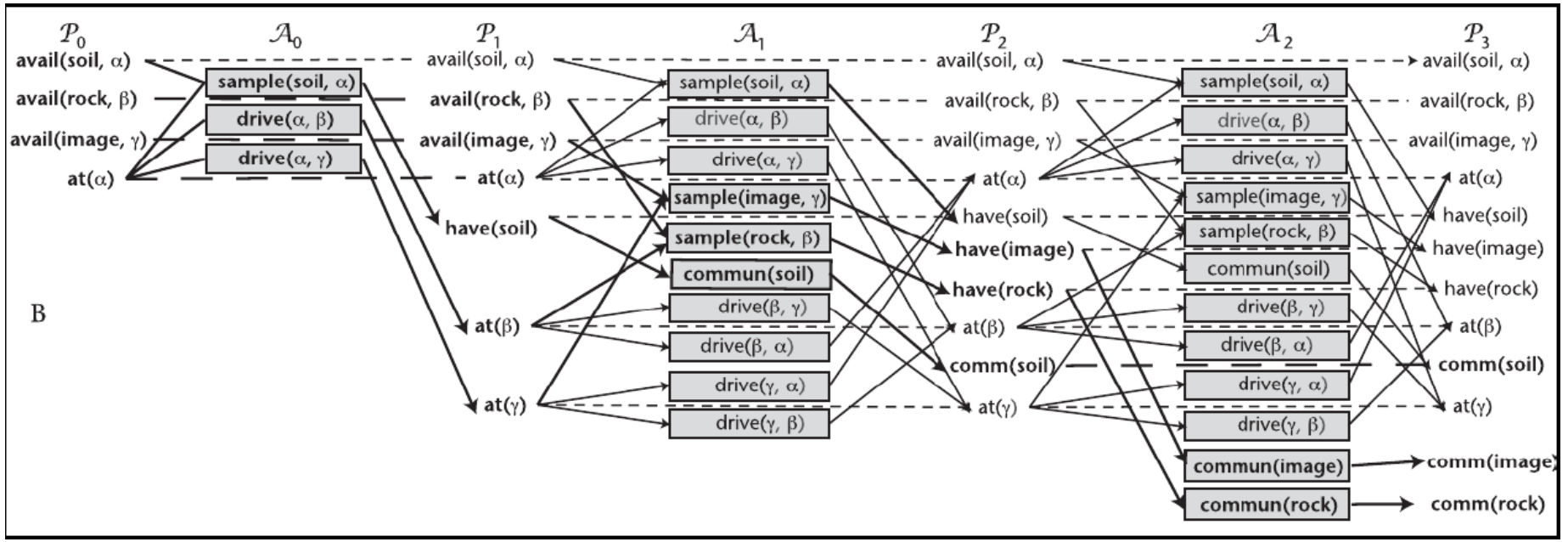
y en PDDL?

- Un fichero <dominio>.pddl en el que se describa el dominio de planificación
  - Objetos y sus tipos
  - Predicados que se van a usar en la descripción del estado inicial, de objetivos y de precondiciones y efectos de las acciones.
  - Las acciones.
- otro fichero <problema>.pddl en el que se describe el estado inicial y el objetivo.

Planificador FF (Fast Forward)

- Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research, 14, 253–302.
- Significó una revolución en las técnicas de planificación y es uno de los más usados y referenciados en la actualidad.
- Utiliza una búsqueda en espacio de estados "estándar" progresiva
  - Genera sucesores a partir del estado inicial, aplicando los operadores (acciones) permitidos en cada estado
  - Operador permitido en un estado:
    - Todas sus precondiciones son satisfechas por el estado.
  - Sucesores:
    - Estados representados como listas de hechos (listas de literales, listas de predicados instanciados).
  - Condición de parada
    - Todos los literales del objetivo están incluidos en el estado seleccionado.

Grafo del plan



- Expande el "**grafo relajado del plan**"
  - Aplicar de forma progresiva (forward) todas las posibles instanciaciones de acciones
    - sin listas de eliminación (e.d. sin efectos negados)
  - desde el estado inicial hasta que todos los objetivos están cubiertos.

Algoritmo general

- Etapa previa: generar el grafo del plan relajado para extraer una heurística admisible:
  - Obtener un plan por regresión (buscando en ese grafo desde los objetivos hacia atrás).
  - El tamaño de ese plan es un valor heurístico que nunca sobreestima la longitud total del plan (siempre desconocida a priori).
- Aplicar Escalada por la máxima pendiente "forzada", desde el estado inicial.
  - Una variante de escalada en la que, si no mejoran los sucesores (meseta, mínimo), se aplica BÚSQUEDA EN ANCHURA hasta que se encuentra un mejor sucesor.
  - El camino hasta ese sucesor es añadido al plan y la búsqueda por escalada continúa.
- Funciona bien porque los problemas de planificación de referencia tienen mesetas y mínimos "pequeños".