

# Tema 3: Planificación

## 1. Introducción

## 2. Problema de planificación. Generalidades

## 3. Representación y formalización del dominio + problema. PDDL

*MODELADO*

## 4. Planificación como un problema de búsqueda

*TÉCNICAS DE  
RESOLUCIÓN*

## 5. Planificación heurística

## 6. Planificación para el mundo real

## 7. Conclusiones

## Bibliografía

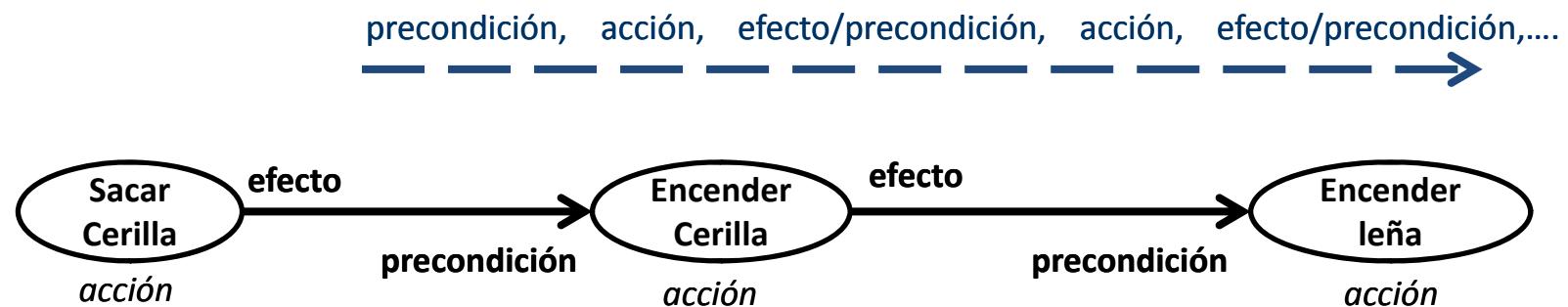
- Artificial Intelligence: A modern approach. Russel, Norvig. Prentice Hall (2010). Cap. 10, 11
- Inteligencia Artificial. Técnicas, métodos y aplicaciones. Varios autores. McGraw Hill (2008) Cap. 13
- Inteligencia Artificial. Una nueva síntesis. Nilsson. McGraw Hill (2001).
- Automated Planning. Theory and Practice. Ghallab, Nau, Traverso. Morgan Kaufmann (2004).

### 3.1.- Introducción

#### ¿Qué es la planificación?

- Es un proceso de *razonamiento inteligente* para seleccionar y organizar un conjunto de acciones que permita la consecución de unos objetivos en base a su causalidad
- Es un proceso deliberativo en el que tratamos de anticiparnos a los efectos de las acciones, por lo que es *previo a la ejecución* (actuación) de las acciones

#### Enlace Causal (causalidad)



Queremos encontrar una secuencia eficiente de acciones contemplando todas las restricciones. Esto implica una *toma compleja de decisiones*

## ¿Técnicas de Planificación Inteligente?

- Necesitamos encontrar una secuencia eficiente de acciones.

*No sirve cualquier combinación de acciones sino una suficientemente buena, o incluso la mejor.*

*Muchas posibles secuencias de acciones pueden obtener el mismo objetivo a partir del mismo estado inicial, pero a diferente coste, tiempo, etc.*

- Nos encontramos en un escenario con muchas restricciones.

*No todas las acciones son aplicables en cualquier momento, unas deben ir antes que otras, unas eliminan recursos o precondiciones que otras necesitan, unas pueden solaparse pero otras no, etc.*

- Se utiliza en escenarios que requieren una *toma compleja de decisiones* que involucra: un orden de ejecución (clara organización), un elevado número de ejecutores, recursos variados, múltiples restricciones que satisfacer y diversos objetivos que cumplir

*En general, tendremos que decidir entre muchas posibles acciones, y que pueden requerir complejas condiciones para su ejecución, al inicio, al final o durante toda su ejecución, así como la utilización de recursos compartidos entre ellas*

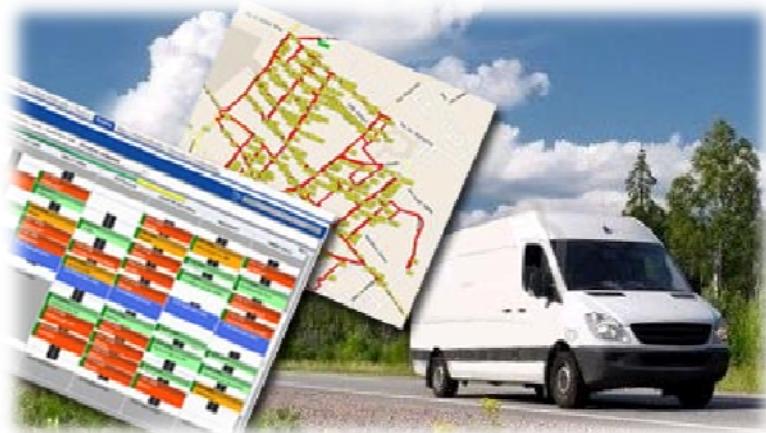
Existen multitud de aplicaciones del mundo real.

Después de todo, ¡planificamos diaria y constantemente sin darnos cuenta!

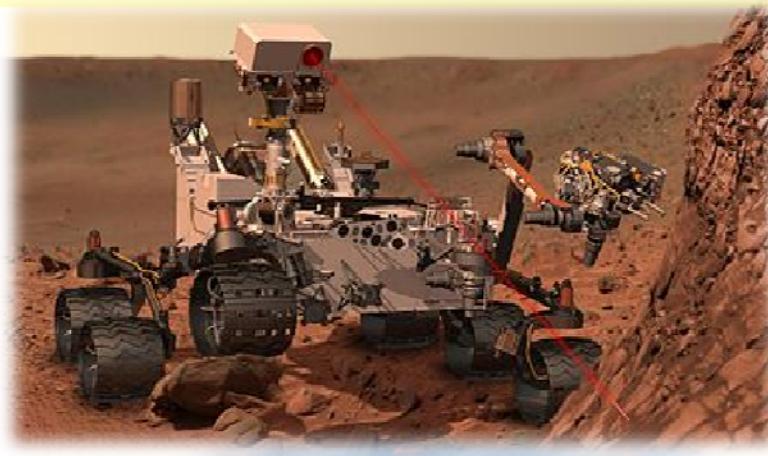
## Aplicaciones del mundo real



Planes de extinción de incendios



Planificación de rutas de transporte/reparto



Planificación de tareas en entornos hostiles



Planificación en aeropuertos



Aplicación a redes de comunicación, ámbitos médicos, e-learning, agentes con movimiento autónomo: ascensores, coches, satélites, personas/inmigración ....y **mucho más...**

## 3.2.- Problema de planificación. Generalidades

### Problema de planificación

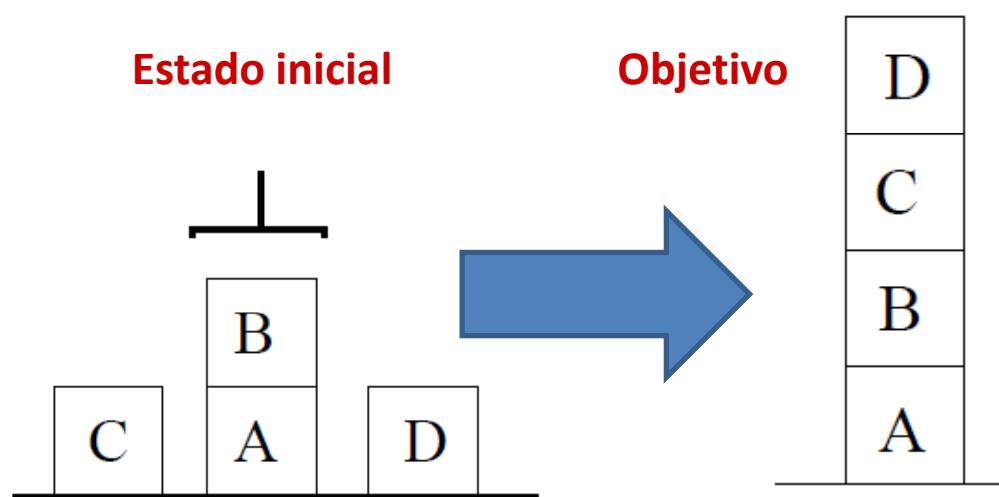
#### 1. Modelado: Describe

- un conjunto de acciones (*qué* se puede hacer?),
- un estado inicial del entorno (*desde dónde?*), y
- una descripción del objetivo a conseguir (*hasta dónde?*)

#### 2. Resolución: Obtiene las acciones + órdenes de ejecución que permiten convertir el estado inicial en un estado que satisfaga todos los objetivos.

Es decir, decidir *qué acciones planificar en cada momento (o secuencia posible)*.

**Ejemplo.** Problema para mover bloques mediante un brazo-grúa



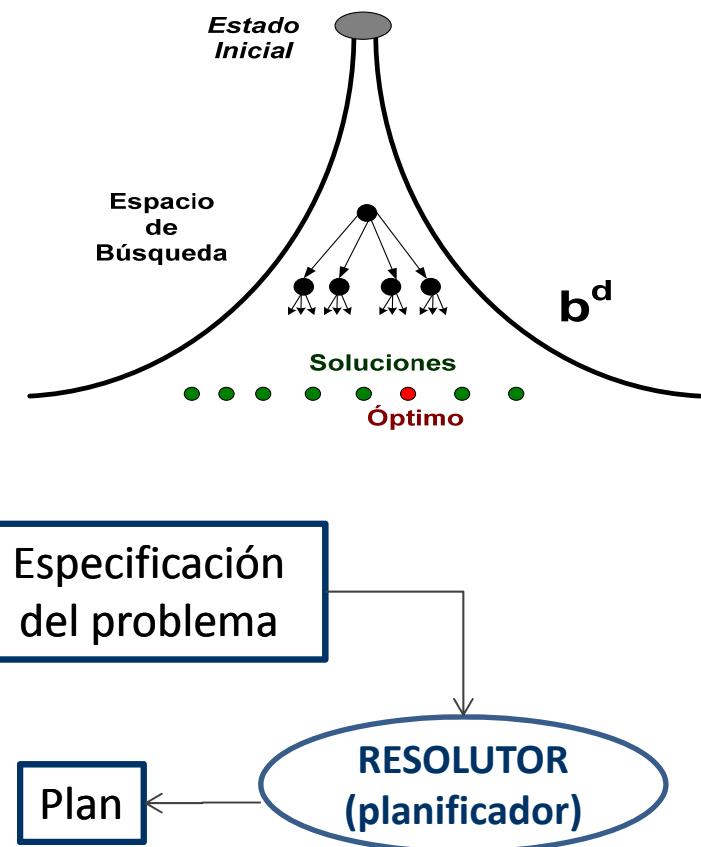
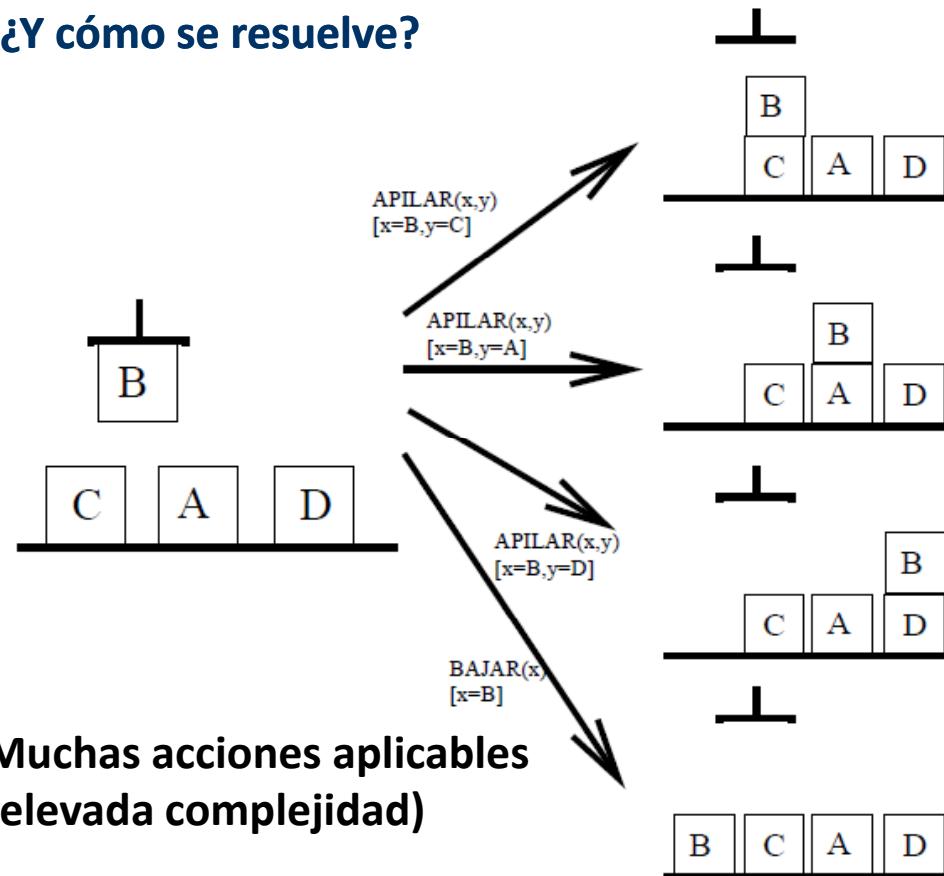
- **Objetos:** Elementos del problema (*objetos*) como los bloques y la mesa.
- **Predicados:** Estados de los objetos, como el hecho de que un bloque esté sobre otro, en la mesa o cogido (*predicados lógicos/booleanos*)

Las acciones *modifican* el estado de los *objetos (predicados)*  
Las acciones pueden utilizar *recursos*

Además de los objetos, estado inicial y objetivo, también hay que modelar:

- **Las acciones:** apilar, desapilar, dejar en la mesa y coger de la mesa
- **Recursos (por ejemplo, grúa/s para trabajar con los objetos):**  
*Dualidad recurso/objeto (!), Capacidades, Renovabilidad, Costes, etc.*
- **¿Y qué pasaría si cada bloque tuviera un peso distinto y las grúas distintas características?: Tipos**

¿Y cómo se resuelve?



### 3.3.- Representación y formalización del Dominio + Problema: PDDL

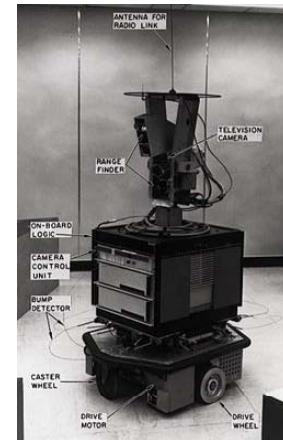
La formalización de un problema de planificación se basa en el modelo definido en STRIPS (*Stanford Research Institute Problem Solver*).

Partimos de:

Conjunto finito de objetos (agrupados o no en tipos)  $\Rightarrow$  **V**: variables de estado.

**Predicados**: Estados posibles de los objetos, con un enfoque proposicional T/F:

*Sobre (bloque, bloque), En\_Mesa (bloque), etc.*



Robot "Shakey" de Standford (1970)

**Problema de planificación clásico**: Se define como una tripleta **<I, G, A>**, donde

**I**: estado inicial con una **asignación completa sobre las variables V**

**G**: objetivos a conseguir con una **asignación parcial sobre las variables V**

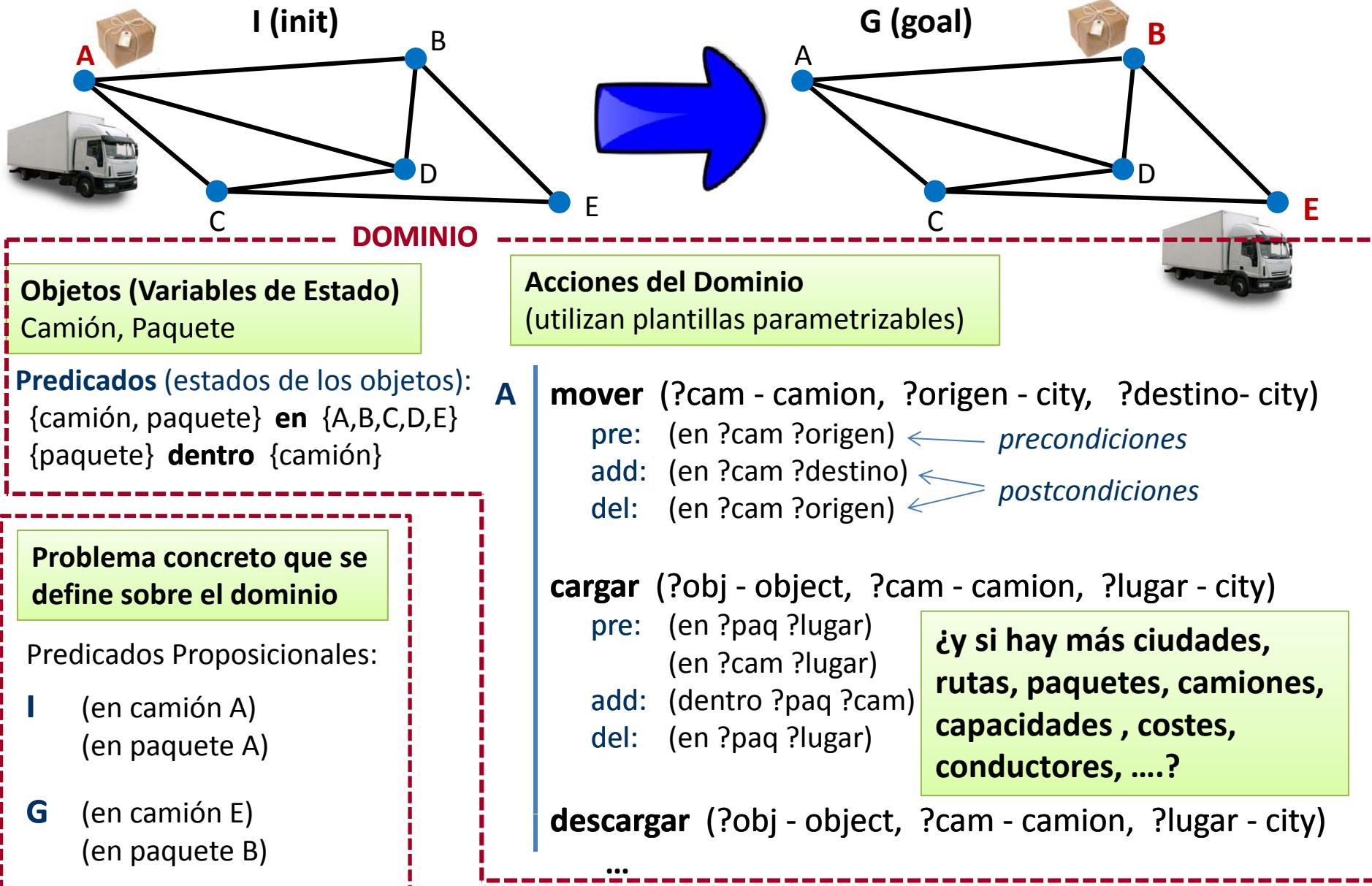
**A**: conjunto de acciones disponibles, en la forma **<pre, add, delete>** que modifican el estado tras la ejecución de cada acción y definen la secuencia de enlaces causales: **relación causa-efecto**.

**Plan**: secuencia de acciones **a<sub>1</sub>, a<sub>2</sub>... a<sub>n</sub>** que transforma **I** en un estado que satisface **G**

- el orden es importante pues una acción  $a_i$  puede eliminar algo que  $a_{i+1}$  necesite
- el plan puede estar total o parcialmente ordenado.

## Ejemplo (logística)

Variables (V), Tripletas: <I, G, A>



## Planificación Clásica: Se asumen ciertas propiedades

- **Conjunto finito de estados (variables de estado)** (*ej. el camión está en A o C, pero no en uno de los puntos intermedios infinitos entre A y C*). Los predicados se interpretan sobre cierto/falso (*no existen fluents/funciones reales*).
- **Mundo es observable**: tenemos una instantánea completa sobre cada estado, **sin incertidumbre**.  
*Ej. siempre sabremos dónde está el camión*
- Modelo **Estático**, sin influencias externas. *Ej. las ruedas del camión no se pinchan*
- Modelo **Determinista**: las acciones siempre se comportan igual, sin efectos imprevisibles.
- Acciones **instantáneas** (tiempo implícito), donde todas las acciones tardan lo mismo. *Ej. mover tarda lo mismo que cargar/descargar*
- Acciones **no descomponibles, ni interrumpibles** (definición atómica). *Ej. mover agrupa todos los pasos: arrancar, poner primera, iniciar la marcha, ...*
- Planificación **offline**: se planifica antes de comenzar con la ejecución de la primera acción. El plan está completo antes de su ejecución

Si se satisfacen todas estas asunciones hablamos de **planificación clásica**

Pero ya veremos que en el mundo real **se relajan** algunas suposiciones

## Problema de planificación. Formalización y semántica: PDDL

¿Cómo definimos un problema de planificación? ⇒ **PDDL, lenguaje estándar *de facto***

Planning Domain Definition Language establece dos ficheros de texto: Dominio y Problema

- **Dominio:** Principalmente, define el conjunto de Predicados (qué se puede cumplir) y Acciones (qué se puede hacer) del dominio, independientemente de cada problema concreto.
- **Problema:** identifica los Objetos concretos del problema, y define su Estado Inicial y sus Objetivos en base a los predicados del dominio

*Un Dominio puede tener muchos Problemas asociados, pero un Problema solo un Dominio*

```
(define (domain <domain name>)
  <PDDL code for predicates>
  <PDDL code for first action>
  [...]
  <PDDL code for last action>
)
```

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

Especificación PDDL:

<http://cs-www.cs.yale.edu/homes/dvm>  
con sus distintas versiones y revisiones  
(también en Polifomat)

Especificación PDDL  
<Dominio> + <Problema>

Plan

RESOLUTOR  
(planificador)

## PDDL: Sintaxis del Dominio.

```
(define (domain <name>)

  (:requirements  <:req1> ... <:reqn>) ; requisitos del dominio. Típicos son :strips :typing

  (:types   <subtype11> ... <subtype1m> – <type1> ; tipos de objetos que se usan
           <subtypen1> ... <subtypenm> – <typen> )

  (:constants <cons1> ... <consn>) ; definición de constantes (si se van a usar)

  (:predicates <p1> <p2> ... <pn>) ; definición de predicados (info proposicional)

  ; ----- acciones (u operadores) como plantillas parametrizables -----

  (:action <name_1>
    :parameters (?par1 – <subtype1> ?par2 – <subtype2> ...) ;parámetros (objetos)
    :precondition ([and] <condition1> ... <conditionn>) ;predicados precondiciones
    :effect  ([and] <effect1> ... <effectn>) ;predicados efectos
  )
  .....
  (:action <name_n> ....)
); fin del dominio
```

Distintas condiciones requieren and  
Distintos efectos requieren and  
Los efectos pueden ser:  
efectos <add>: (predicado ....)  
efectos <del>: (not (predicado ....))

## PDDL: Sintaxis del problema

(**define** (problem <name>)

  (:domain <name>) ; *nombre del dominio al que pertenece este problema*

  (:objects <obj<sub>1</sub>> - <type<sub>1</sub>> ... <obj<sub>n</sub>> - <type<sub>n</sub>>) ; *objetos del problema y sus tipos*

  (:init ; *estado inicial. Todos los objetos*

    (<predicate<sub>1</sub>> ... <predicate<sub>i</sub>>) ; *info proposicional estado inicial (no proposiciones negadas)*

  (:goal ; *objetivo*

    (and ((<predicate<sub>1</sub>> ... <predicate<sub>i</sub>>)) ; *objetivos proposicionales*

) ; *fin del problema*

*Las expresiones en INIT no requieren and*

*Las expresiones en GOAL requieren and*

*Todos los objetos del dominio deben estar asociados a un estado (predicado)*

*Guiones separados por espacios*

*No predicados negados en INIT, GOAL (not ...)*

## Un ejemplo de PDDL para un dominio de transporte/logística

```
(define (domain logistics) ; Define los tipos, predicados y operadores
(:requirements :strips :typing)
(:types ; tipos utilizados
    package location vehicle - object
    truck - vehicle
    city - location)
(:predicates
(at ?vehicle-or-package - (either vehicle package) ?location - location)
(in ?package - package ?vehicle - vehicle))
(:action load ;acción para cargar
:parameters (?obj - package ?truck - truck ?loc - location)
:precondition (and (at ?truck ?loc)
(at ?obj ?loc))
:effect (and (not (at ?obj ?loc)) ;efecto delete
(in ?obj ?truck))) ;efecto add
(:action unload ;acción para descargar
:parameters (?obj - package ?truck - truck ?loc - location)
:precondition (and (at ?truck ?loc)
(in ?obj ?truck)))
:effect (and (not (in ?obj ?truck))
(at ?obj ?loc)))
(:action move ... ;acción para mover
```

**La descripción del dominio es independiente del número de camiones, paquetes, ciudades (!)**



(define (problem pb1) ; Objetos, Inicial y Final

(:domain logistics)

(:objects pck1 - package  
pck2 - package  
pck3 - package  
truck1 - vehicle  
truck2 - vehicle  
madrid - city  
valencia - city  
barcelona - city ...)

Problema concreto del dominio

(:init (at truck1 valencia) (at truck2 madrid)  
(at pck1 barcelona) (at pck2 madrid)...)

Estado de todos los objetos

(:goal (and (at pck1 madrid)  
(at pck2 valencia)  
(at truck1 madrid)  
(at truck2 barcelona)))

Estado de algunos objetos

## Ejemplo de PDDL para un dominio de Mars rover

(<https://mars.jpl.nasa.gov/>)

```
(define (domain Rover) ;from NASA

(:types rover waypoint store camera mode lander objective)

(:predicates
  (at ?x - rover ?y - waypoint)
  (at_lander ?x - lander ?y - waypoint)
  (can_traverse ?r - rover ?x - waypoint ?y - waypoint)
  (equipped_for_soil_analysis ?r - rover)
  ...)

(:action navigate ;navegar entre dos puntos
:parameters ( ?x – rover ?y – waypoint ?z - waypoint)
:precondition (and (can_traverse ?x ?y ?z)
  (available ?x)
  (at ?x ?y) (visible ?y ?z))
:effect      (and (not (at ?x ?y)) (at ?x ?z)))

(:action sample_soil ;recoger muestra de tierra
:parameters ( ?x - rover ?s - store ?p - waypoint)
:precondition (and (at ?x ?p) (at_soil_sample ?p)
  (equipped_for_soil_analysis ?x)
  (store_of ?s ?x) (empty ?s))
:effect      (and (not (empty ?s)) (full ?s)
  (have_soil_analysis ?x ?p) (not (at_soil_sample ?p))))
...
)
```



```
(define (problem roverprob1234)
(:domain Rover)
(:objects
  general - lander
  rover0 - rover
  rover0store - store
  waypoint0 waypoint1 – waypoint
  ....)

(:init
  (visible waypoint1 waypoint0)
  (visible waypoint0 waypoint1)
  (visible waypoint2 waypoint0)
  (can_traverse rover0 waypoint3 waypoint0)
  (can_traverse rover0 waypoint0 waypoint3)
  (empty rover0store)
  (equipped_for_soil_analysis rover0)
  (equipped_for_rock_analysis rover0)
  ...)

(:goal (and
  (communicated_soil_data waypoint2)
  (communicated_rock_data waypoint3))))
```

## Extensiones en PDDL. Hacia una representación más realista

Ciertas suposiciones iniciales demasiado estrictas para modelar el mundo real, que hay que relajar

**Tiempo:** *Las acciones tienen una duración (no instantáneas, ni igual para todas)*

- Se rompe la uniformidad en los pasos de planificación y se añade una dimensión temporal (*scheduling*)
- Modelo de tiempo no conservativo (*acciones durativas*):

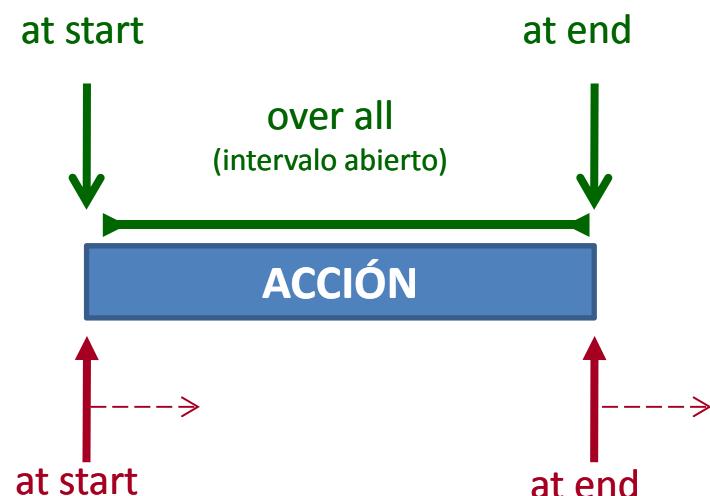
*Precondiciones:* Al **principio** (*at start*), cuando la acción se aplica,

Al **final** (*at end*), cuando finaliza la acción y se realizan los efectos,

**Durante** toda la acción (*over all*); intervalo abierto

*Efectos:* Al **principio** (*at start*), efecto inmediato al inicio del intervalo,

al **final** (*at end*), efecto retrasado al final del intervalo



Formalmente, '*over all*' es un intervalo abierto. Si se quiere especificar que una condición  $<p>$  se mantiene en el intervalo cerrado durante la duración de una acción durativa, se requiere:

$(at start <p>)$ ,  $(over all <p>)$  y  $(at end <p>)$

Aunque la implementación típica de los planificadores, asume '*over all*' como intervalo cerrado, incluyendo start/end.

## Extensiones en PDDL. Hacia una representación más realista

### Condiciones Numéricas. Variables Numéricas (fluents, functions), Recursos

Las acciones usan **recursos** con información numérica (no booleana) para **modelar capacidades**.

- Recursos: Renovables o No-Renovables, con información numérica (capacidad, coste, ....).
- Se utilizan variables numéricas (fluents) con dominios reales (infinitos), definidas en el dominio como '*functions*' (*funciones numéricas*).

Ej. (:functions (fuel\_level ?t - tank) ..... ) ;define el 'nivel' de un tanque

- Las precondiciones/efectos pueden utilizar predicados/funciones aritméticas sobre las variables fluents:

Ej. pre: (> (fuel\_level ?tank) 420) ;>, >=, <, <=, =

eff: (decrease (fuel\_level ?tank) 50) ;assign, increase, decrease, scale up y scale down  
(:=, +=, -=, \*=, /=)

Puede establecerse un **coste/beneficio** asociado sobre el valor final de los fluents.

- Permite aplicar criterios de optimización para evaluar la **calidad** del plan :metric minimize/maximize

Ejemplo: (:metric minimize (+ (\* 0.2 (total-time)) (\* 0.5 (fuel\_level ?tank))))

(total-time) es una función ya predefinida.

## Problema de planificación. Formalización y semántica: PDDL 2.1 (posterior al nivel 3)

- Se incluyen acciones durativas con un modelo temporal más expresivo.
- Se pueden incluir fluents (costes, beneficios, etc.), representando información numérica

```
(define (domain <name>)
  (:requirements :typing :durative-actions :fluents) <:req 1> ... <:req n>) ; requisitos
  (:types <subtype1>... <subtype n> - <type1> <typen>) ; tipos que se usan
  (:constants <cons1> ... <cons n>) ; definición de constantes (si se van a usar)
  (:predicates <p1> <p2>... <p n>) ; definición de predicados (info proposicional)
  (:functions <f1> <f2>... <fn>) ; definición de funciones (variables numéricas, fluents)
    ; ejemplo: (fuel_level ?truck - truck)
    ; en el problema se deben indicar el valor inicial de todos los fluents!
```

; Acciones durativas

```
(:durative-action 1 ; acción con duración
  :parameters (?par1 - <subtype1> ?par2 - <subtype2> ...)
  :duration <value> ; número o expresión que incluya functions
  :condition ( [and] <condition1> ... <conditionn>) ; "at start", "over all", "at end"
  :effect ( [and] <effect1> ... <effectn>) ; "at start", "at end"
)
```

:duration 5

:duration (= ?duration (/ (distance ?c1 ?c2) (speed ?a)))

## Ejemplo de PDDL durativo para Mars rovers

```
(define (domain Rover) ;from NASA
  (:requirements :typing :durative-actions :fluents)
  (:types rover waypoint store camera mode lander objective)

  (:predicates
    (at ?x - rover ?y - waypoint)
    (available ?x - rover)
    (can_traverse ?x - rover ?y - waypoint ?z - waypoint)
    (visible ?y - waypoint ?z - waypoint)
    ...)

  (:functions (energy ?r - rover) (recharge-rate ?x - rover))

  (:durative-action navigate ;navegar entre dos puntos
    :parameters ( ?x - rover ?y - waypoint ?z - waypoint)
    :duration (= ?duration 5)
    :condition (and (at start (available ?x))
                    (at start (at ?x ?y))
                    (at start (>= (energy ?x) 8))
                    (over all (can_traverse ?x ?y ?z))
                    (over all (visible ?y ?z)))
    :effect (and (at start (decrease (energy ?x) 8))
                  (at start (not (at ?x ?y))))
                  (at end (at ?x ?z))))
    ...
  )
```



Funciones numéricas (*fluents*), sobre las que se pueden establecer condiciones.

La acciones tienen duraciones, tal que:

- Las condiciones se pueden requerir al **principio**, durante **toda** la ejecución o solo al **final**.
- Los efectos se pueden generar al **principio** o al **final** (no durante toda la ejecución, es decir no hay prorrateo)

## IMPORTANTE!!!!

- Es necesario inicializar **todas** las funciones numéricas (fluent), definidas en el dominio, en el estado inicial (init) del problema (*para cada objeto en la que se definen las functions*)

```
(:functions ;dominio
  (distancia ?c1 - city ?c2 - city)
  (capacidad ?c - jarra)
  (combustible-total))
```

```
(:init ;problema
  (= (distancia valencia madrid) 350) ; para todo par ciudades
  (= (capacidad jarra1) 50) ; para toda jarra
  (= (combustible-total) 100))
```

- Se pueden expresar condiciones sobre las funciones numéricas en la meta (goal)
- Puede indicarse una métrica a optimizar sobre las funciones numéricas

```
(:functions (energy ?r - rover) (recharge-rate ?x - rover))
```

```
(:durative-action navigate ;navegar entre dos puntos
:parameters ( ?x - rover ?y - waypoint ?z - waypoint)
```

```
:duration (= ?duration 5)
```

```
:condition (and (at start (available ?x))
  (at start (at ?x ?y))
  (at start (>= (energy ?x) 8))
  (over all (can_traverse ?x ?y ?z))
  (over all (visible ?y ?z)))
```

```
:effect (and
  (at start (decrease (energy ?x) 8))
  (at start (not (at ?x ?y)))
  (at end (at ?x ?z))))
```

...

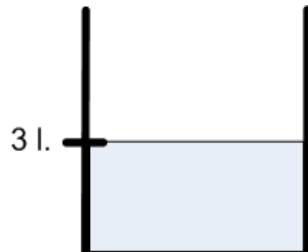
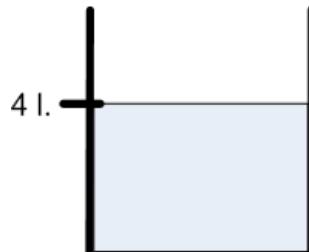
```
rover0 - rover
rover0store - store
....)
```

```
(:init
  (visible waypoint1 waypoint0)
  (visible waypoint0 waypoint1)
  (visible waypoint2 waypoint0)
  (= (energy rover0) 50)
  (= (recharge-rate rover0) 11)
  ...)
```

```
(:goal (and
  (communicated_soil_data waypoint2)
  (> (energy rover0) 30)
  ...))
```

**(:metric minimize (+ (total-time) (energy rover0)))**

## Ejemplo Fluents: Jarras de agua



```
(define (domain jug-pouring)
  (:requirements :typing :durative-actions :fluents)
  (:types jug)
  (:functions
    (amount ?j - jug)
    (capacity ?j - jug) )
  (:durative-action EmptyFrom1To2
    :parameters (?jug1 ?jug2 - jug)
    :duration 10
    :condition (at start (>= (- (capacity ?jug2) (amount ?jug2)) (amount ?jug1)))
    :effect (and (at end (assign (amount ?jug1) 0))
      (at end (increase (amount ?jug2) (amount ?jug1))))
    ...
  )
```

Funciones numéricas  
establecer condiciones  
;Precondiciones: >, >=, <, <=, =  
;Efectos: assign, increase, decrease, scale up y scale down

```
(define (2jug )
(:domain jug-pouring)
(:objects
  jugx - jug
  jugy - jug)
(:init
  (= (amount jugx) 0)
  (= (amount jugy) 0)
  (= (capacity jugx) 4)
  (= (capacity jugy) 3)
)
```

condiciones/asignaciones sobre fluentes

## Ejemplo Acciones Durativas- 1

(:durative-action load-truck ;Load 'cargo' in 'truck' with 'crane', (grúa puede simultanear cargas)

:parameters (?t - truck)  
                  (?l - location)  
                  (?o - cargo)  
                  (?c - crane)

:duration (= ?duration 5)

:condition (and  
              (at start (at ?t ?l)) ; truck en ?l  
              (over all (at ?t ?l)) ; truck en ?l  
              (at start (at ?o ?l)) ; cargo en ?l  
              (at start (empty ?c))) ; grúa disponible (y compatible)

:effect (and  
              (at start (not (at ?o ?l))) ; cargo no en ?l  
              (at end (in ?o ?t))) ; cargo en truck



*Si se quiere especificar que una condición <p> se mantiene en el intervalo cerrado durante la duración de una acción durativa, entonces se requieren tres condiciones:*

*(at start <p>), (over all <p>) y (at end <p>)*



## Ejemplo Acciones Durativas- 2

(:durative-action load-truck ;Load 'cargo' in 'truck' with 'crane', no simultaneidad de grúa por 'cargos'

:parameters	(?t - truck) (?l - location) (?o - cargo) (?c - crane)
:duration	(= ?duration ( / (weight ?c) (powercrane))) ; duración dependiente peso carga y potencia grúa
:condition (and	(at start (at ?t ?l)) ; truck en ?l (over all (at ?t ?l)) ; truck en ?l (at start (at ?o ?l)) ; cargo en ?l (at start (empty ?c)) ; grúa vacía (no disponible)
:effect (and	(at end (empty ?c)) ; no cargo en ?l (at start (not (empty ?c))) ; al inicio grúa no vacía (at end (empty ?c)) ; al final grúa vacía (at end (in ?o ?t)) ; cargo en truck

En Domain:  
(:functions  
  (weighth ?o - cargo)  
  (powercrane)  
.....)

En Problema:  
(init  
  (= (weighth O1) 70)  
  (= (powercrane) 10)  
.....)



## Ejemplo Acciones Durativas- 3

(:durative-action load-truck ;Load 'cargo' in 'truck' with 'crane', **no simultaneidad** de grúa por 'cargos'  
y capacidad en truck

:parameters	(?t - truck) (?l - location) (?o - cargo) (?c - crane)
:duration	(= ?duration ( / (weight ?c) (powercrane))) ; duración dependiente peso carga y potencia grúa
:condition (and	(at start (at ?t ?l)) ; truck en ?l (over all (at ?t ?l)) ; truck en ?l (at start (at ?o ?l)) ; cargo en ?l (at start (empty ?c)) ; grúa vacía (at start (< (contenido ?t) (capacidad ?t))) ; capacidad disponible
:effect (and	(at start (not (at ?o ?l))) ; no cargo en ?l (at start (not (empty ?c))) ; al inicio grúa no vacía (at end (empty ?c)) ; al final grúa vacía (at end (in ?o ?l)) ; cargo en truck (at start (increase (contenido ?t) 1))) ; incremento contenido

En Domain:

(:functions  
    (capacidad ?t - truck) ;número, peso, volumen, etc  
    (contenido ?t - truck)  
.....)

En Problema:

(init  
    (= (capacidad truck1) 10)  
    (= (contenido truck1) 0)  
.....)

Especificación PDDL  
<Dominio> + <Problema>

### 3.3.- Modelado (PDDL)

RESOLUTOR  
(planificador)

### 3.4.- Resolución (Búsqueda)

- En un Espacio de Estados. Direccionamiento.
- En un Espacio de Planes (Planificación Orden Parcial)
- Planificación Heurística (relajación, abstracción, etc.)

Plan

### 3.4.- Planificación como un problema de búsqueda

#### Búsqueda en un espacio de estados

La alternativa **más simple y directa** para planificar es realizar una **búsqueda en un espacio de estados**, habitualmente representado como un árbol

- Nodo del árbol (*estado*): conjunto de variables de estado asignadas (proposiciones) que definen el estado del mundo en ese momento
- Arco/transición: aplicación de una acción, que transforma un estado previo (donde se deben cumplir las precondiciones de la acción) en un estado resultante (donde se han aplicado los efectos positivos y negativos de la acción)
- El plan es una secuencia de acciones, correspondiente a las transiciones entre nodos del espacio de búsqueda
- El plan se construye secuencialmente y será *totalmente ordenado*. Su ejecución será *siempre hacia delante*, pero su generación podrá ser *hacia delante o hacia atrás*

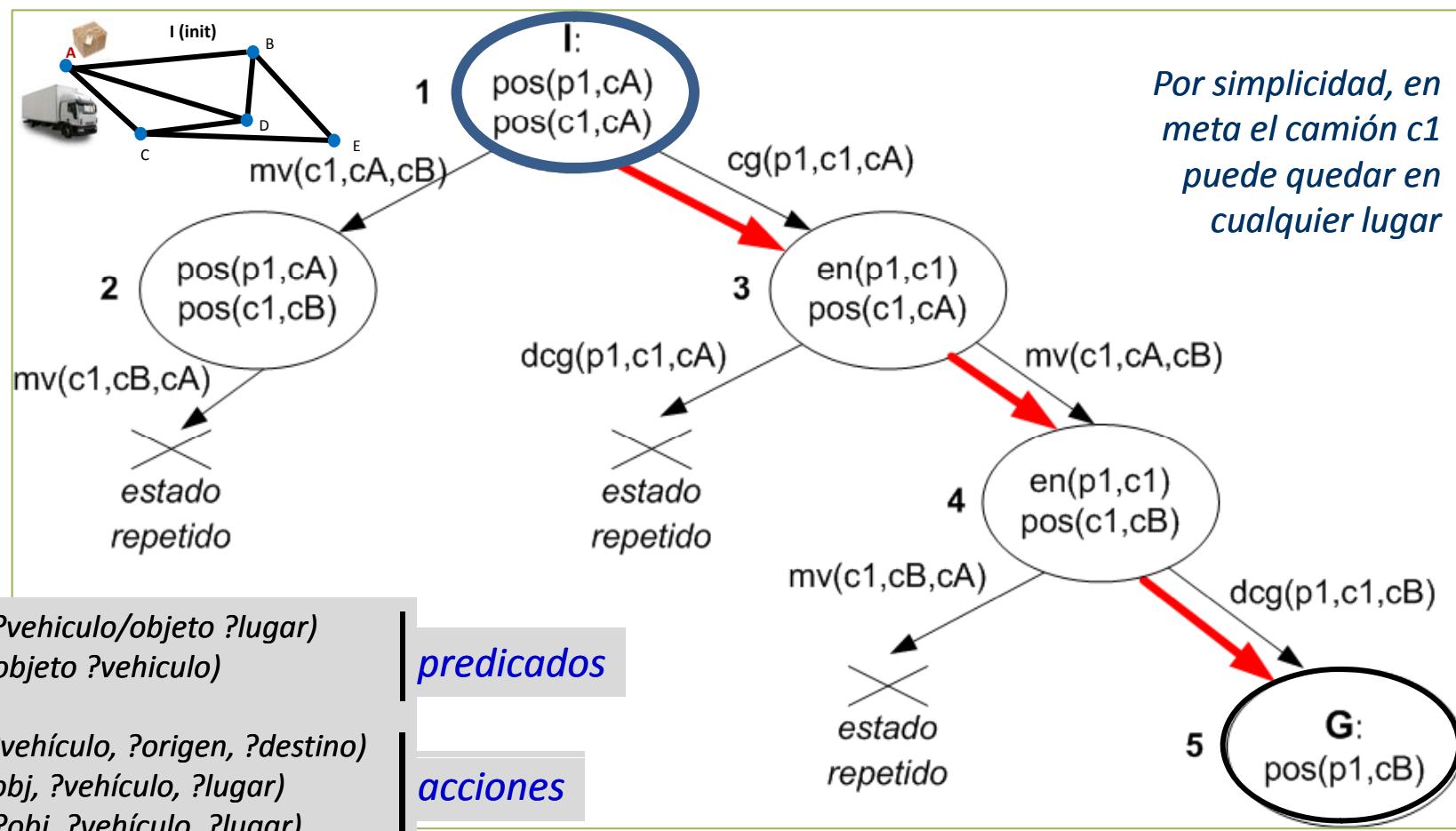
#### Direccionamiento

**Búsqueda hacia delante o progresiva.** El nodo raíz es el estado inicial del problema. Cada nodo se expande a partir de todas las acciones aplicables en dicho estado (si sus precondiciones se satisfacen), generándose un conjunto de sucesores

**Búsqueda hacia atrás o regresiva.** Se parte de los objetivos del problema. En cada nodo se aplican inversamente las acciones del dominio, generando así nuevos estados subobjetivos como nodos predecesores

## Búsqueda hacia delante o progresiva (*guiado por condiciones*)

- Algoritmo muy sencillo. El plan se construye añadiendo acciones (transiciones)
- Se generan tantos nodos sucesores como acciones aplicables, evitando nodos que representan estados ya visitados
- Cada nodo representa un estado completo y alcanzable
- Útil para determinar **alcanzabilidad** (*¿qué puede obtenerse desde un estado dado?*)



(*pos ?vehiculo/objeto ?lugar*)  
 (*en ?objeto ?vehiculo*)

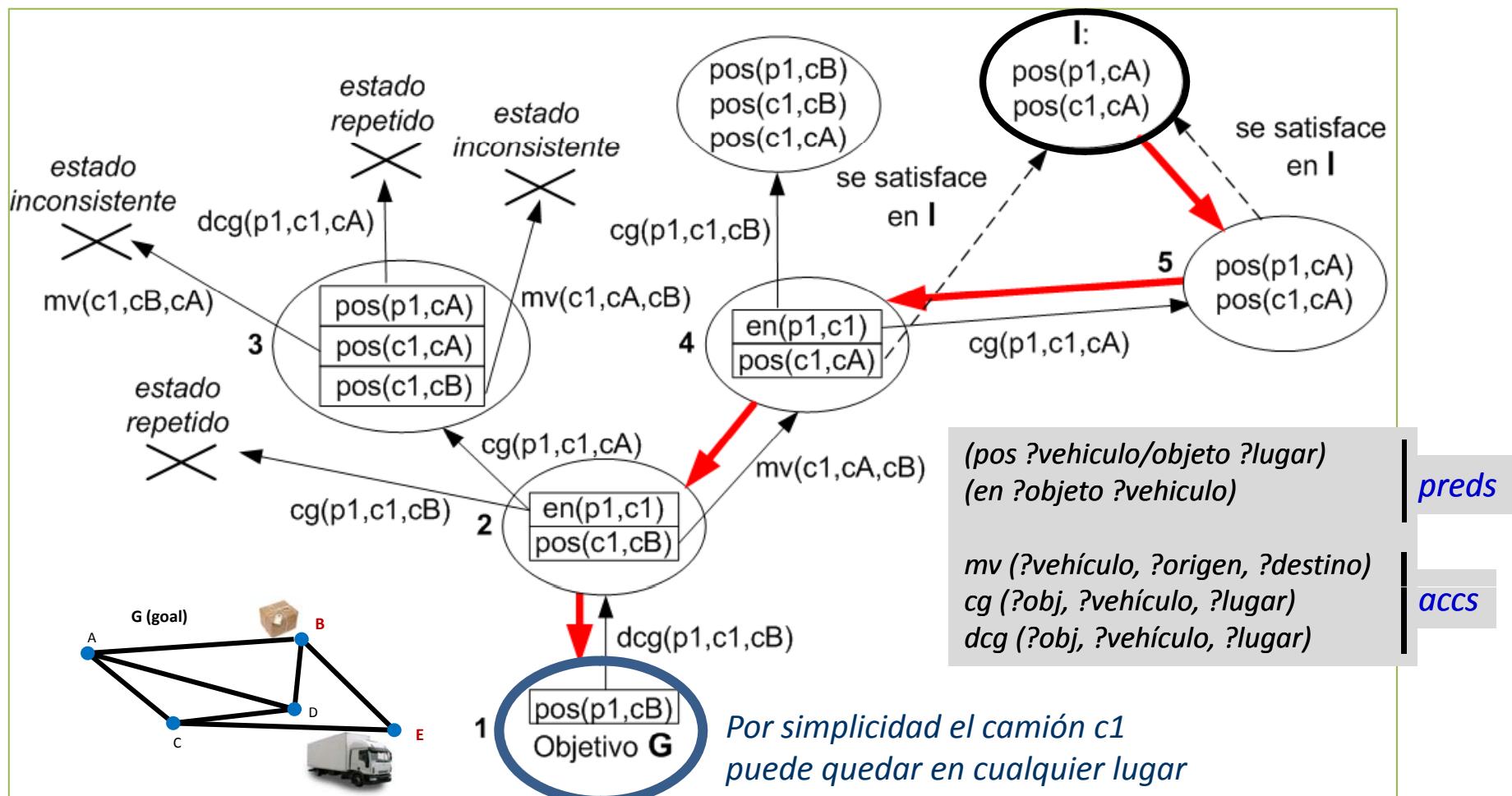
**predicados**

*mv (?vehículo, ?origen, ?destino)*  
*cg (?obj, ?vehículo, ?lugar)*  
*dgc (?obj, ?vehículo, ?lugar)*

**acciones**

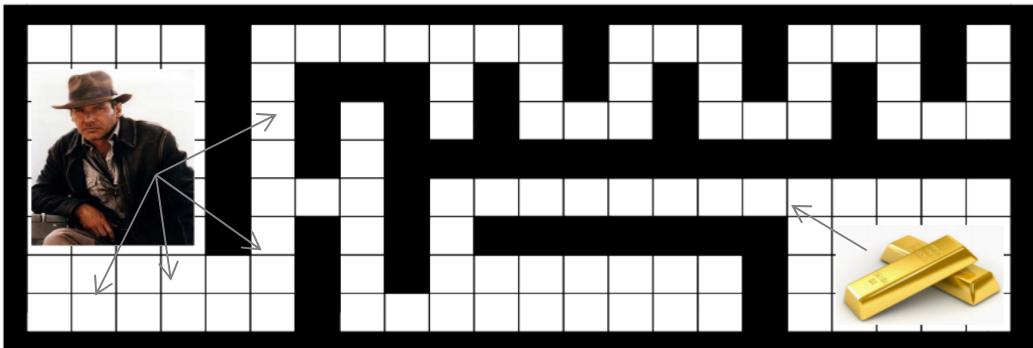
## Búsqueda hacia atrás o regresiva (guiado por efectos)

- Algoritmo guiado por el objetivo (el plan será el recorrido inverso de cómo se genera)
- Se generan solo los nodos predecesores necesarios para la consecución de los objetivos
- Cada nodo **no** es un estado completo, sino parcial. No representa un estado *real*
- Útil para determinar **relevancia** (*¿qué es útil para alcanzar el objetivo?*)



## En resumen: búsqueda hacia delante vs. búsqueda hacia atrás

Nodos: Estados Arcos: Acciones	Hacia delante	Hacia atrás
Tipo de algoritmo	Muy sencillo	Más elaborado
Acciones (y nodos en el espacio de búsqueda)	<ul style="list-style-type: none"> <li>Se pueden generar acciones irrelevantes para conseguir los objetivos.</li> <li>Cada nodo es un estado posible.</li> <li>Se <i>arrastra</i> la información innecesaria.</li> </ul>	<ul style="list-style-type: none"> <li>Solo se generan acciones útiles para conseguir los objetivos.</li> <li>Cada nodo es un estado parcial (no real).</li> <li>Se <i>elimina</i> la información innecesaria.</li> </ul>
Factor de ramificación	Muy elevado	Más limitado
Eficiencia	Menos eficiente: expansión <b>ciega</b> de todo lo alcanzable	Más eficiente: expansión de solo lo imprescindible

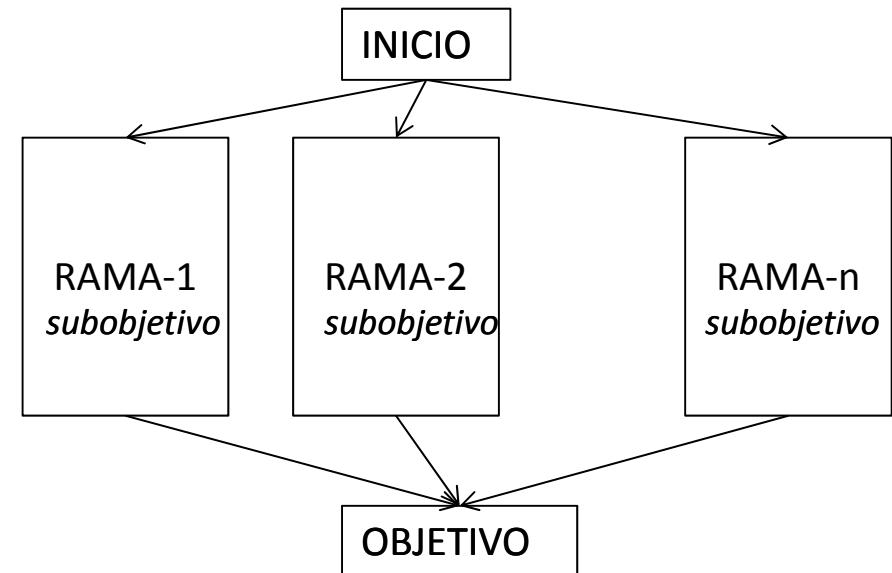


¿Intuitivamente, qué es **más fácil**?

## Búsqueda en planificación. Planificación de Orden Parcial (POP).

**Motivación:** ¿por qué hay que construir el plan utilizando un único direccionamiento (siempre hacia delante o siempre hacia atrás)?

- Construcción de un plan donde se van satisfaciendo subobjetivos arbitrariamente: *construcción del plan “por partes/ramas” (no secuencial, no de orden total)*
- Se plantea un **orden parcial** entre las acciones de las distintas ramas. Estos órdenes se van refinando poco a poco mediante precedencias entre acciones
- Si se necesita, al final del proceso se puede obtener un orden total teniendo en cuenta la interacción entre las acciones: Pueden existir muchos órdenes totales factibles a partir de un único orden parcial
- Adecuado para problemas complejos con **múltiples subobjetivos** donde puede haber varias **ramas independientes**  
*(Ej. Para transportar distintos paquetes en distintos camiones se puede planificar primero una rama/paquete, luego un trozo de otra rama, luego pasar a otra, y así sucesivamente)*

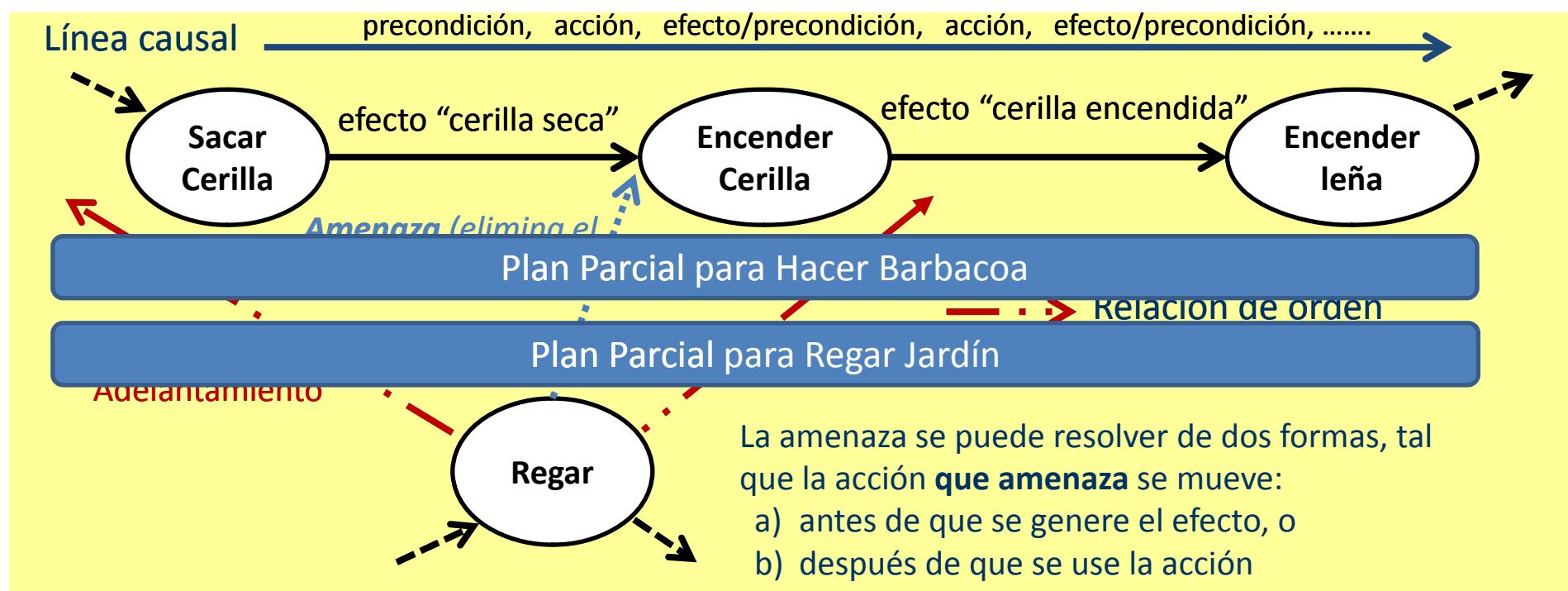
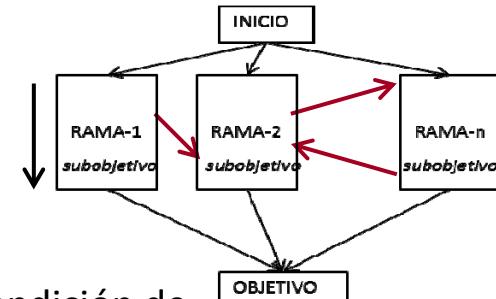


## Planificación de Orden Parcial. POP

Dado un plan de orden parcial

- Los **nodos** representan acciones.
- Los **arcos** pueden representar:

- a) Un enlace causal ( $a_i, e, a_j$ ): la acción  $a_i$  consigue el efecto  $e$ , que es precondición de la acción  $a_j$ . *Los enlaces causales SIEMPRE implican una restricción de orden entre  $a_i$  y  $a_j$ :  $a_i \rightarrow a_j$*
- b) Una restricción de orden ( $\rightarrow$  precedencia) entre dos acciones ( $A_1 < A_2$ ), indicando que  $A_1$  debe ejecutarse antes (no necesario inmediatamente antes) que  $A_2$ . Entre  $A_1$  y  $A_2$  no se requiere que haya relación causal. *No debe haber ciclos*.



## Planificación de Orden Parcial $\Rightarrow$ Planificación basada en un espacio de planes

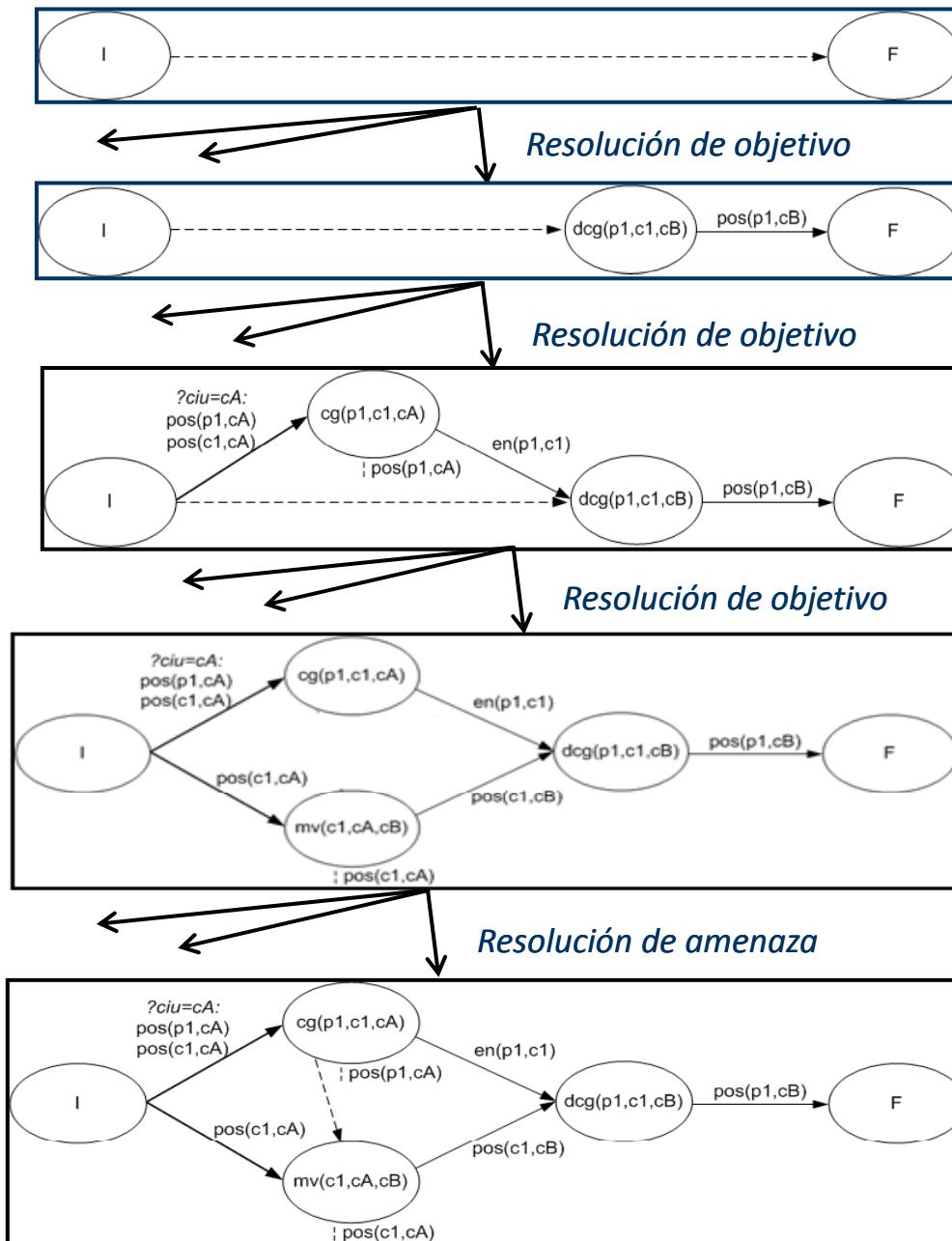
- La Planificación de Orden Parcial requiere una **Representación Basada en Planes**.
- Cada **nodo representa un plan parcial (no un estado)**.
- Se realiza una búsqueda en un espacio de planes parciales (cada nodo del espacio de búsqueda es un plan parcial) y vamos seleccionando el nodo (plan parcial) que más convenga (heurística).

### Proceso

- La generación del plan no es ni hacia delante ni hacia atrás, sino arbitraria: **Se va refinando poco a poco.**
- **Guiado por los (sub)objetivos** que van apareciendo por precondiciones de las acciones planificadas (y que resuelven un sub-objetivo previo).
- Se generan nodos sucesores (es decir, nuevos planes parciales) ante dos situaciones:
  1. Resolución de **subobjetivos pendientes** (*open conditions*), mediante:
    - a) Insertando una nueva acción, con un nuevo enlace causal
    - b) Reutilizando una acción existente, y se crea un nuevo enlace causal
  2. Resolución de **amenazas** (*acción que invalida el enlace causal (efecto-precondición) entre dos acciones del plan*).  
Se resuelve añadiendo una **restricción de orden** que evita la amenaza (adelantamiento o retraso, también conocido como *promotion/demotion*)

## Ejemplo Planificación de Orden Parcial

Nodo inicial: solo sabemos que I va antes que F



(*pos ?vehiculo/objeto ?lugar*)  
(*en ?objeto ?vehiculo*)

*mv (?vehículo, ?origen, ?destino)*  
*cg (?obj, ?vehículo, ?lugar)*  
*dgc (?obj, ?vehículo, ?lugar)*

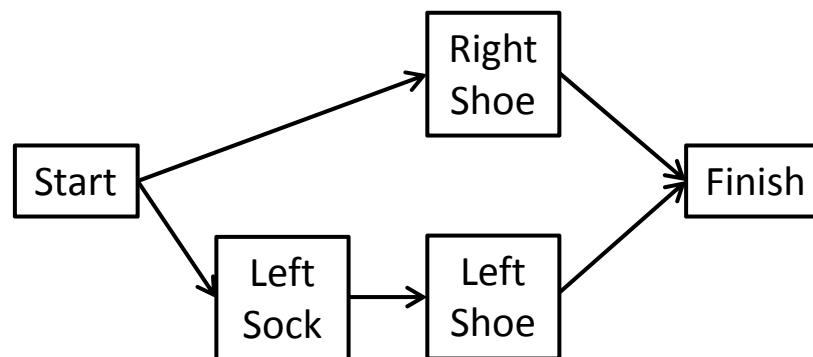
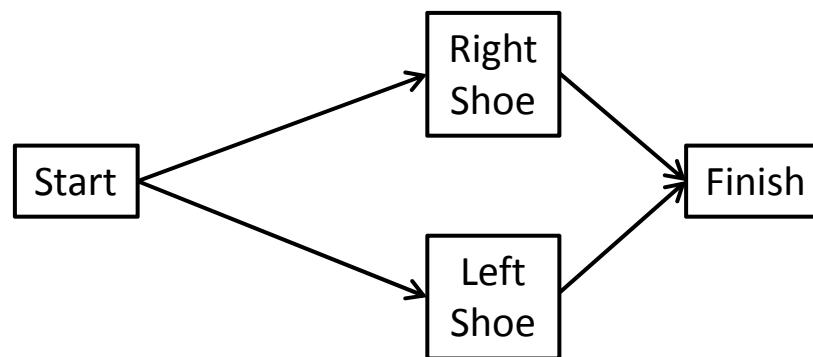
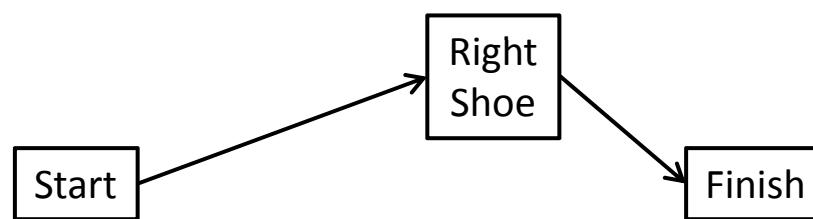
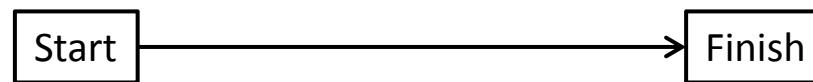
*Inicial: pos (p1, CA), pos (c1, CA)*  
*Objetivo: pos (p1, CB)*

Por simplicidad el camión  
*c1* puede quedar en  
cualquier lugar

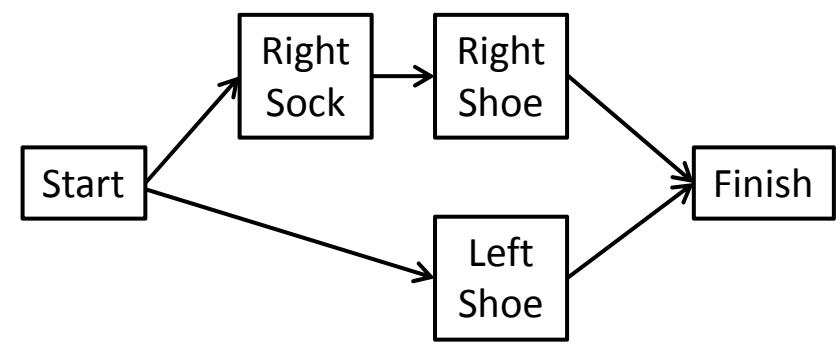
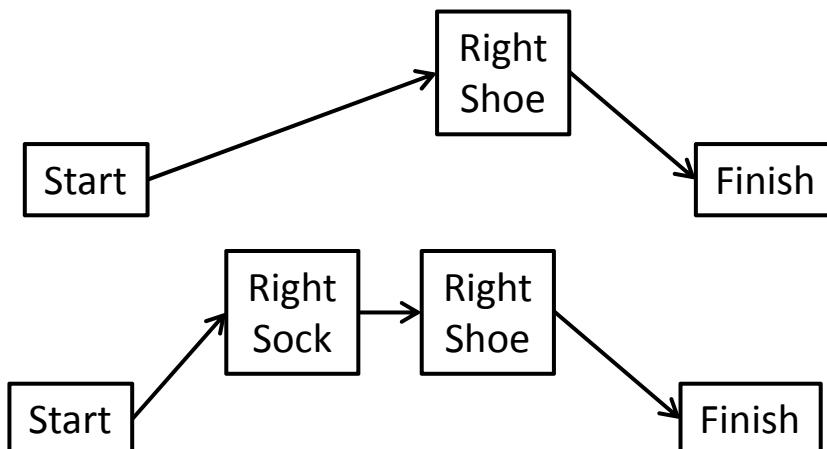
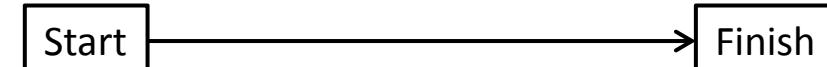
-----> Relación de orden  
→ Enlace causal

En este nodo se ha resuelto la  
amenaza que crea  $mv(c1,cA,CB)$  a  
 $cg(p1,c1,CA)$ : se retrasa la acción  
de  $mv$  después de  $cg$

## Ejemplo Planificación de Orden Parcial. (Objetivo: ponerse calcetines y zapatos en los dos pies)

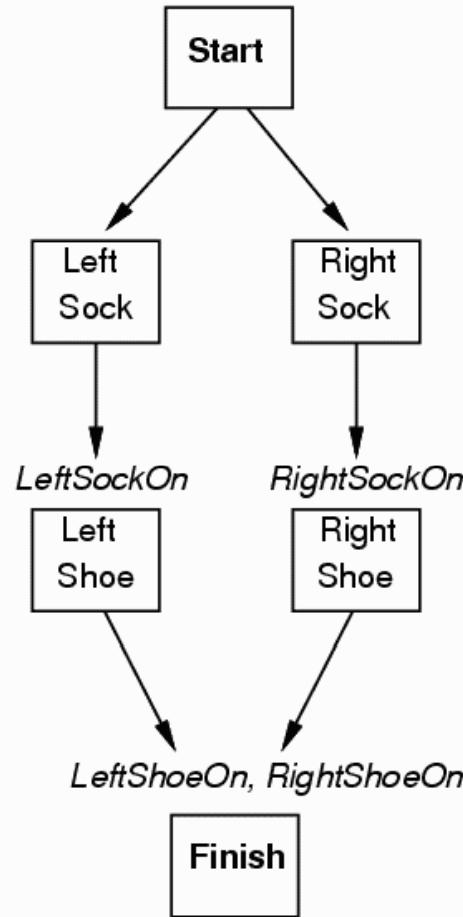


Otra alternativa...  
(cualquier otra combinación sería válida)

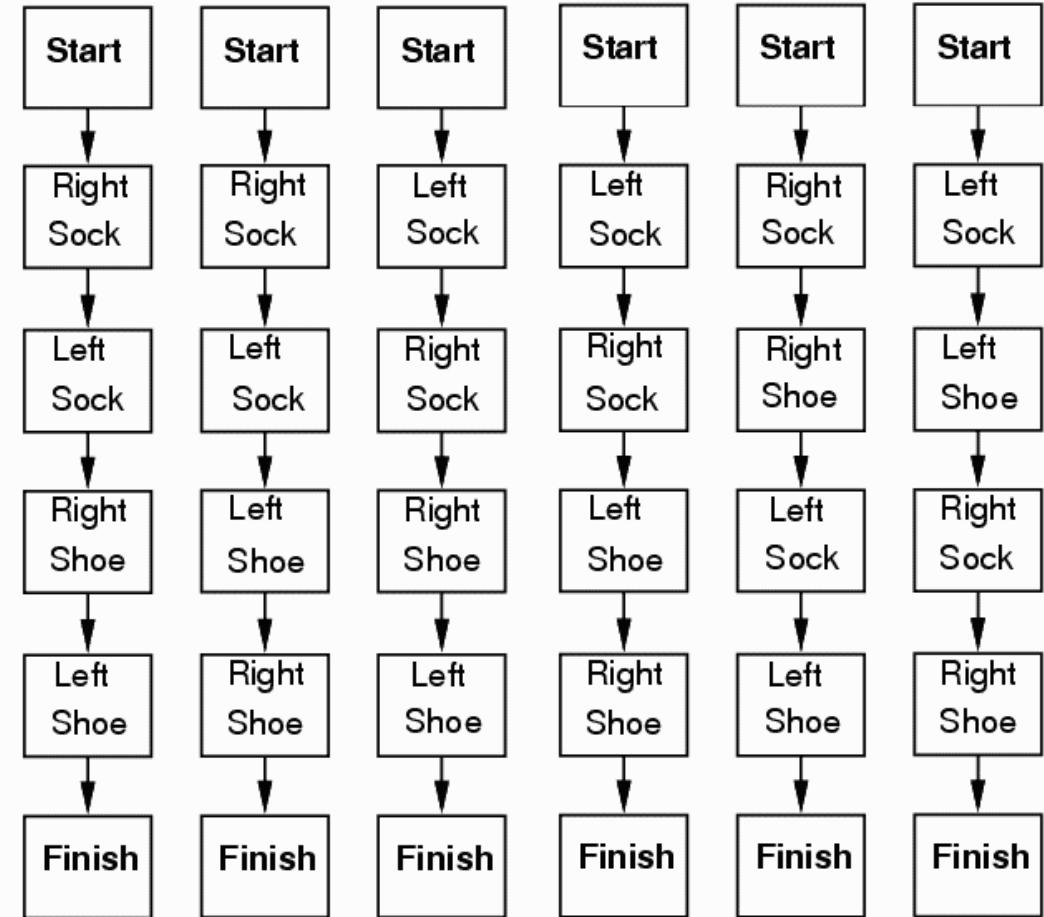


## Comparativa Plan de Orden Parcial vs. Orden Total

Partial Order Plan:



Total Order Plans:



*Hay muchos órdenes posibles (total order plans),  
pero lo IMPORTANTE es que el calcetín se ponga antes que el zapato (relación causal).*

## En resumen...

Búsqueda en Espacio de Estados (progresiva/regresiva) **VERSUS** Búsqueda en Espacio de Planes (POP)

El tipo de representación y tipo de búsqueda van íntimamente ligados.

a) La generación de nuevos nodos en un *Espacio de Estados* es muy sencilla:

Simplemente consiste en **aplicar todo lo que se pueda (especialmente en un tipo de búsqueda hacia adelante o progresivo)**

- La idea de estado intermedio está explícita en una búsqueda en un espacio de estados
- El plan se construye como una secuencia de acciones (totalmente ordenadas).
- Típicamente planes de orden total

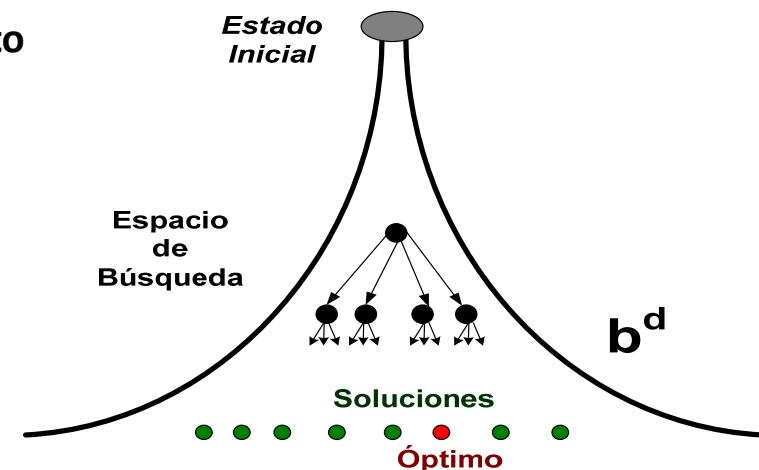
a) La generación de nuevos nodos (planes) en un *Espacio de Planes* requiere más trabajo:

- 1) decidir qué subobjetivo vamos a resolver,
- 2) cómo lo vamos a resolver, y
- 3) añadir nuevos nodos con nuevos pasos (enlaces causales) o nuevas relaciones de orden (adelantamiento/retraso)

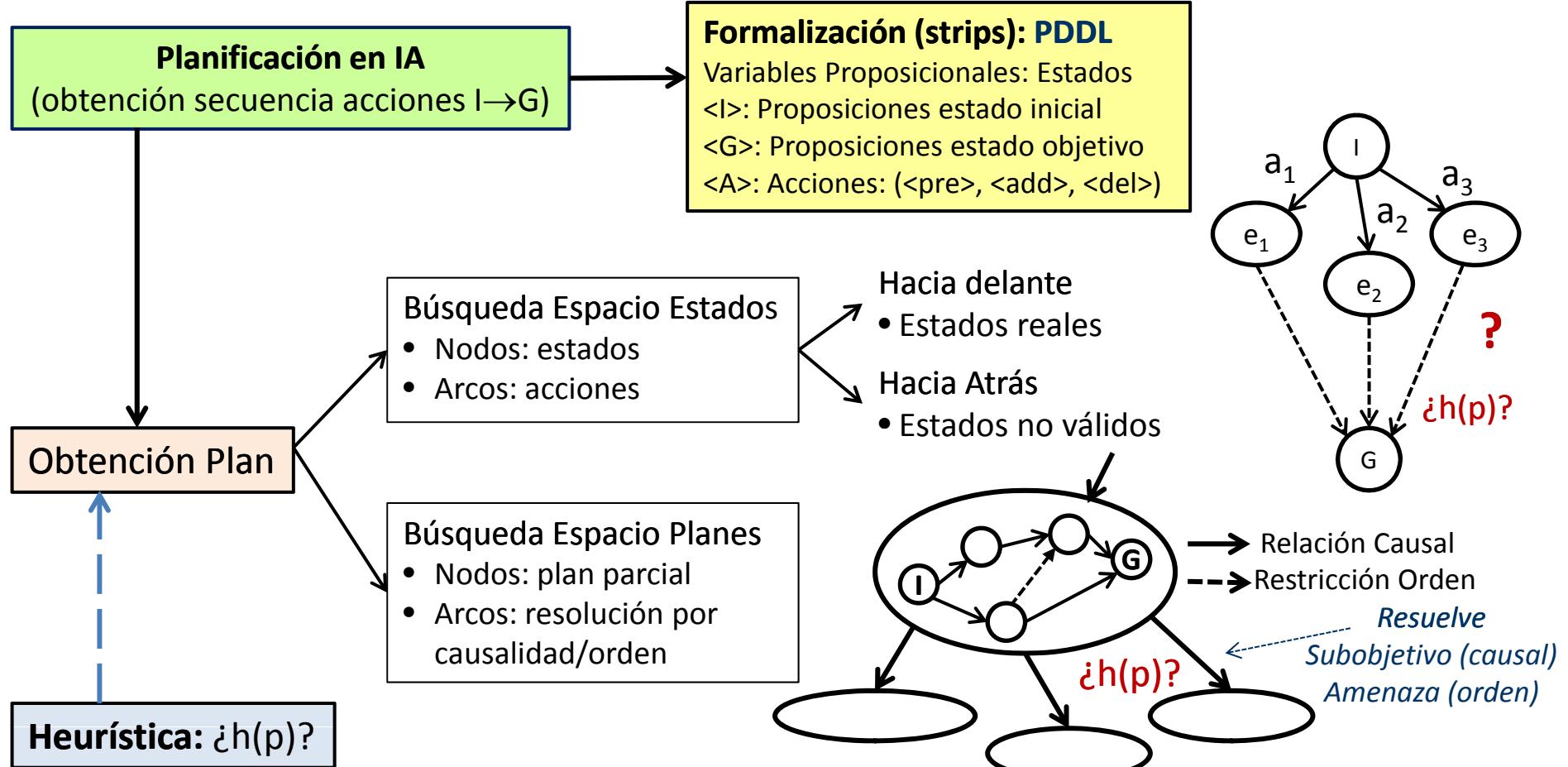
- La idea de estado intermedio no existe en un espacio de planes
- El plan se construye como acciones que tienen órdenes entre sí, pero no son necesariamente una secuencia (planificación de orden parcial)
- Un plan de orden parcial puede representar muchos planes de orden total – mayor flexibilidad

## En resumen...

- Aplicar un proceso de búsqueda tradicional en IA para resolver un problema de planificación es, a priori, una aproximación sencilla:  
Se pueden utilizar algoritmos de tipo A, IDA, etc.
- No obstante, si queremos resolver problemas complejos dicho proceso se vuelve rápidamente inmanejable
  - En el problema de logística, ¿qué pasa si hay muchos camiones, muchas ciudades y muchos paquetes?
- Resolver un problema de planificación es **NP-completo**
- Resolverlo de forma óptima es **NP-duro**



- Es necesario la **utilización de heurísticas**

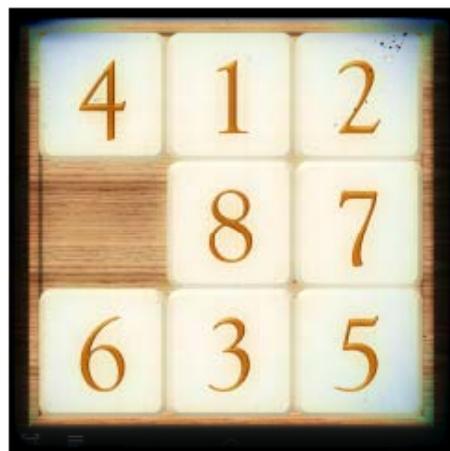


- Relajación
- Abstracción
- Landmarks

## 3.5.- Planificación heurística

### Motivación

- La más costosa, la más cara tarea de planificación requiere tomar decisiones sobre un gran número de posibles acciones alternativas (vía estados o planes)
- Las heurísticas ayudan a seleccionar la decisión más adecuada en base a una métrica: la más corta, la más beneficiosa, la más rápida, etc.



No solo aplicamos  
heurísticas en planificación

Heurísticas en el 8-puzzle:  
distancia Euclídea,  
distancia de Manhattan,  
piezas descolocadas, etc.

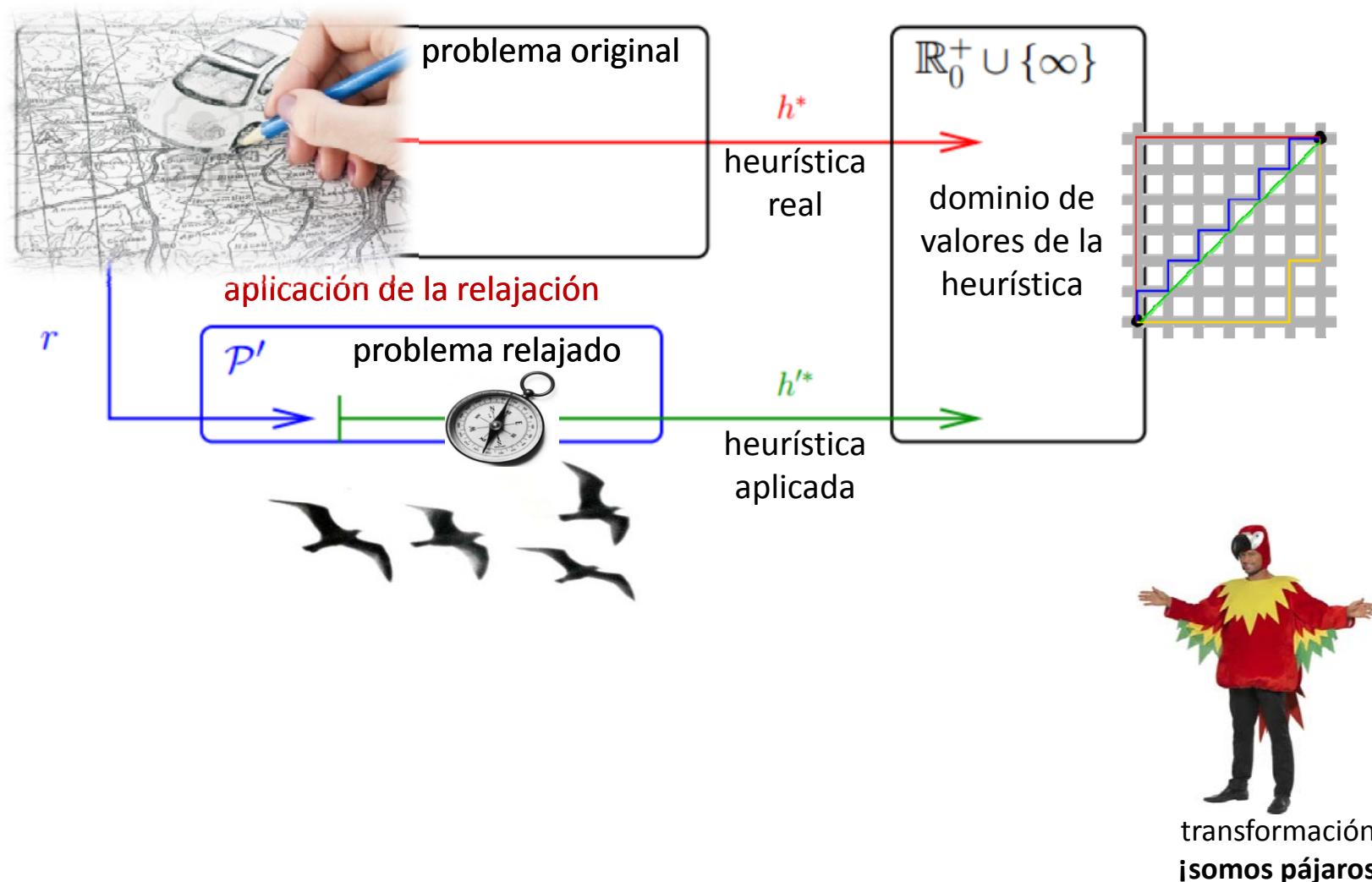
La base para definir heurísticas en planificación es **la misma** que en otros problemas de búsqueda:

relajar ciertas restricciones para trabajar con un problema más sencillo.

Todos los planificadores actuales utilizan algún tipo de heurística

## Heurísticas de Relajación, de forma gráfica

La idea subyacente es la de **simplificar** el problema original para hacer el problema más **fácilmente abordable**



## Heurística por Relajación: Relajación de efectos delete

- Una de las heurísticas por relajación más usadas en planificación se basa en la **relajación de los efectos delete**: *una vez se consigue algo nunca se borra*
- Heurística muy sencilla que, además, **elimina todo tipo de amenazas**: *no será necesario reponer nada de los efectos ya generados, permite asumir una cantidad infinita de recursos, etc.*

Por ejemplo, en el problema de logística previo, la relajación supondría:

*la acción mover no elimina la proposición de que el vehículo esté en la ciudad origen;  
la acción de descargar sigue manteniendo el objeto en el vehículo)*

**mover(?vehículo, ?origen, ?destino)**  
pre: ?vehículo en ?origen  
add: ?vehículo en ?destino  
~~del: ?vehículo en ?origen~~

**descargar(?objeto, ?vehículo, ?lugar)**  
pre: ?objeto en ?vehículo  
?vehículo en ?lugar  
add: ?objeto en ?lugar  
~~del: ?objeto en ?vehículo~~

Más útil cuanto menos se requiera repetir la misma acción en el plan o reponer *deletes*

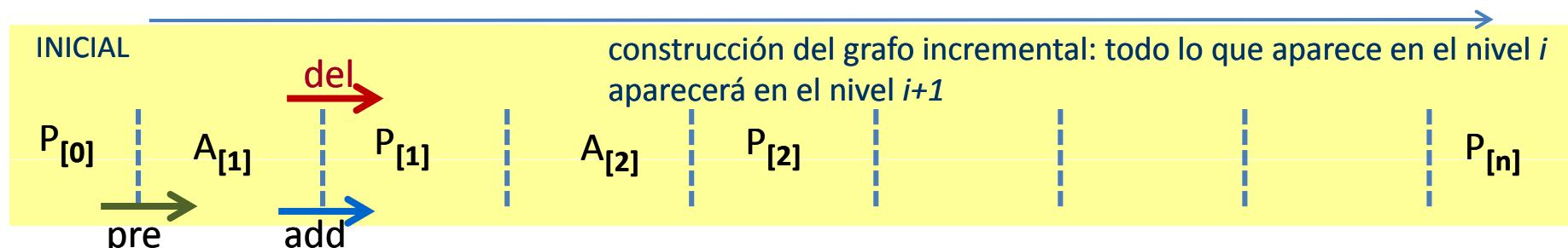
## Aplicación de la Heurística por Relajación

- Aplicación mediante **Grafo de Planificación Relajado**:  
La heurística permitirá estimar la **distancia (pasos)** de un estado dado al objetivo indicado.  
Utilizada como base por **multitud de planificadores (LPG, mips-xxl, sgplan, etc.)**
- La heurística no sobreestima el valor real de alcanzar el objetivo: *Heurística Admisible*

## Aplicación heurística por relajación: Grafo de Planificación

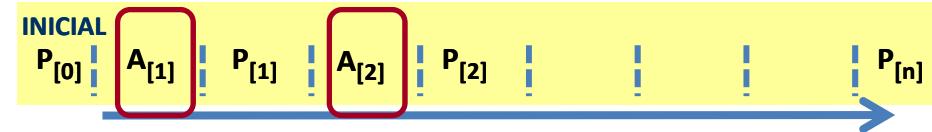
Una forma sencilla para aplicar la heurística de relajación es utilizar **Grafos de Planificación** (introducidos por el planificador *Graphplan*)

- Estructura de tamaño polinómico: muy **eficiente**
- Grafo dirigido multinivel formado por dos tipos de niveles:
  - Nivel de Proposición: solo contiene nodos proposición,
  - Nivel de Acción: solo contiene nodos acción.
- Por cada proposición existe una acción **no-operación (no-op)** que propaga y mantiene la proposición intacta en los sucesivos niveles (*persistencia*).
- Existen tres tipos de aristas entre niveles: De nivel de proposición a nivel de acción (**aristas-pre**), y de nivel de acción a nivel de proposición (**aristas-add**, **aristas-del**).
- El grafo va alternando:
  - **Nivel de proposición:** inicialmente con las proposiciones del estado inicial. Posteriormente, con las proposiciones (efectos) que se generan por la ejecución de las acciones del nivel actual de acción
  - **Nivel de acción:** acciones que se pueden ejecutar a partir de las proposiciones (precondiciones) del nivel anterior de proposición



## Construcción Grafo de Planificación. Acciones y Proposiciones Mutex

- El grafo de planificación se va construyendo alternando niveles de proposición / acción.
- Entre *niveles distintos* de acción existe un *orden total*, pero *dentro del mismo nivel* las acciones tienen un *orden parcial*



Un nivel de acción (**paso** de planificación) puede contener a muchas acciones en paralelo.

En un mismo nivel las acciones pueden ejecutarse simultáneamente, excepto las *acciones mutex* que no podrán darse a la vez en **ningún** plan válido.

### Acciones y Proposiciones mutex

Dos proposiciones/acciones en un nivel son mutuamente excluyente (**mutex**) cuando no pueden darse a la vez:

**Acciones Mutex:** una acción borra la precondición de la otra, o las acciones tienen precondiciones o efectos contradictorios (imposibles de darse a la vez).

Ej: *Mover(ciudad1, ciudad2)* y *Mover(ciudad1, ciudad3)* son mutex por efectos contradictorios

**Proposiciones Mutex:** una proposición no puede darse a la vez que otra.

Ej: “*(En persona1 Valencia)*” y “*(En persona1 Madrid)*” no pueden darse simultáneamente.

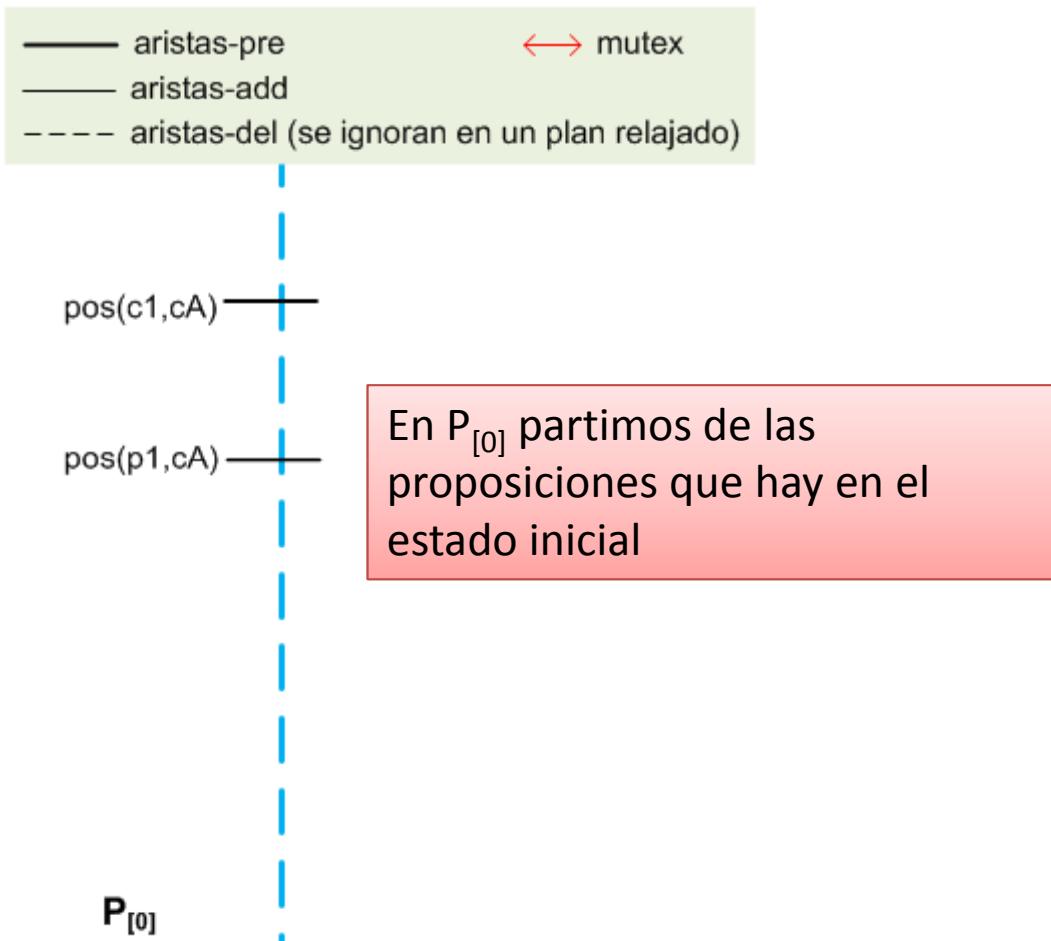
Ej: Si dos proposiciones “*p*” y “*q*” **solo** se obtienen por dos acciones mutex, entonces dichas proposiciones también son mutex.

La información de mutex se va **propagando** a lo largo del grafo: **Si dos proposiciones mutex son precondición de una acción, no se permite realizar acción.**

## Ejemplo: Construcción de un Grafo de Planificación.

Inicial: pos (p1, CA), pos (c1, CA)

Objetivo: pos (p1, CB)

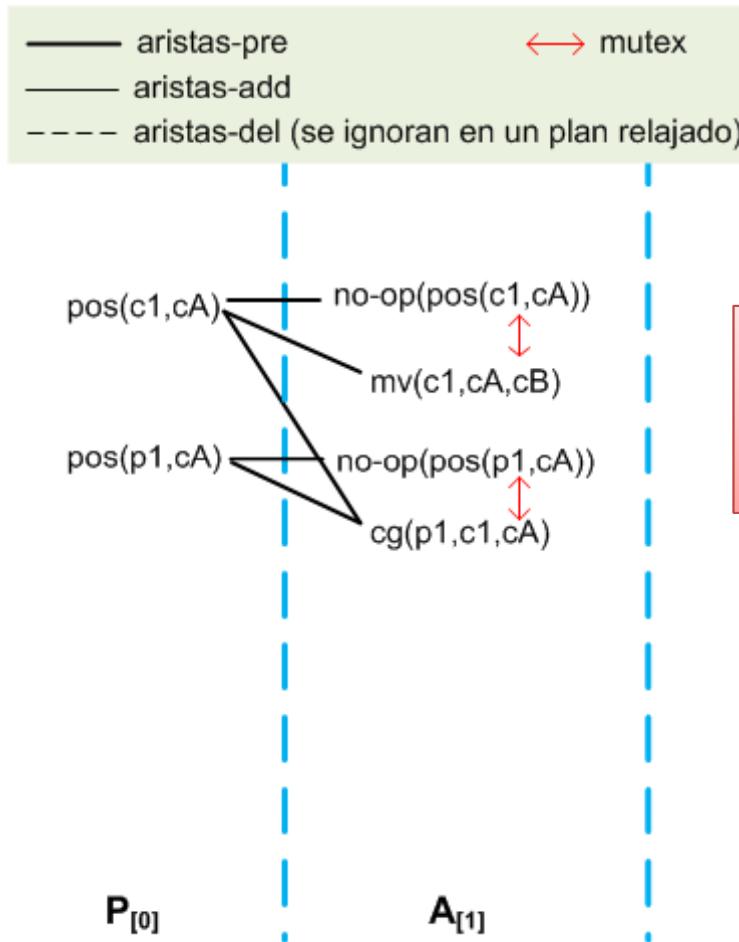


- El grafo se construye incrementalmente a partir del estado inicial.
  - En  $A_t$ , aparecen todas las acciones (*incluidas no-op*) cuyas precondiciones están presentes en  $P_{t-1}$ .
  - En  $P_t$ , aparecen todas las proposiciones alcanzadas a lo largo del grafo de planificación (*relajación*)
- Acciones no-op (*no-operación*): No hace nada con la proposición: idea de persistencia (axioma marco).
- Termina cuando, en un Nivel de Proposición, todos los objetivos se satisfacen y no son mutex.

## Ejemplo: Construcción de un Grafo de Planificación.

Inicial: pos (p1, CA), pos (c1, CA)

Objetivo: pos (p1, CB)



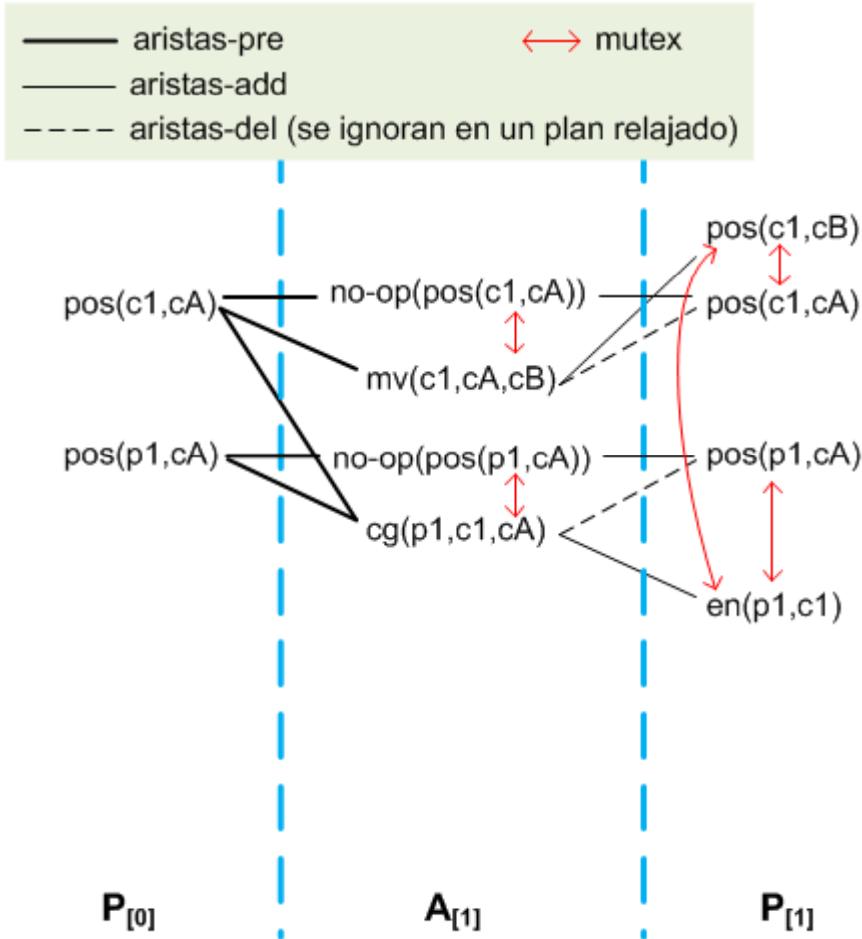
Las **acciones mutex** no se podrán ejecutar a la vez

Por ejemplo,  $\text{mv}(c1, cA, cB)$  es mutex con  $\text{no-op}(\text{pos}(c1, cA))$  porque sus efectos son contradictorios

- El grafo se construye incrementalmente a partir del estado inicial.
  - En  $A_t$ , aparecen todas las acciones (*incluidas no-op*) cuyas precondiciones están presentes en  $P_{t-1}$ .
  - En  $P_t$ , aparecen todas las proposiciones alcanzadas a lo largo del grafo de planificación (*relajación*)
- Acciones no-op (*no-operación*): No hace nada con la proposición: idea de persistencia (axioma marco).
- Termina cuando, en un Nivel de Proposición, todos los objetivos se satisfacen y no son mutex.

## Ejemplo: Construcción de un Grafo de Planificación.

Inicial: pos (p1, CA), pos (c1, CA)  
 Objetivo: pos (p1, CB)



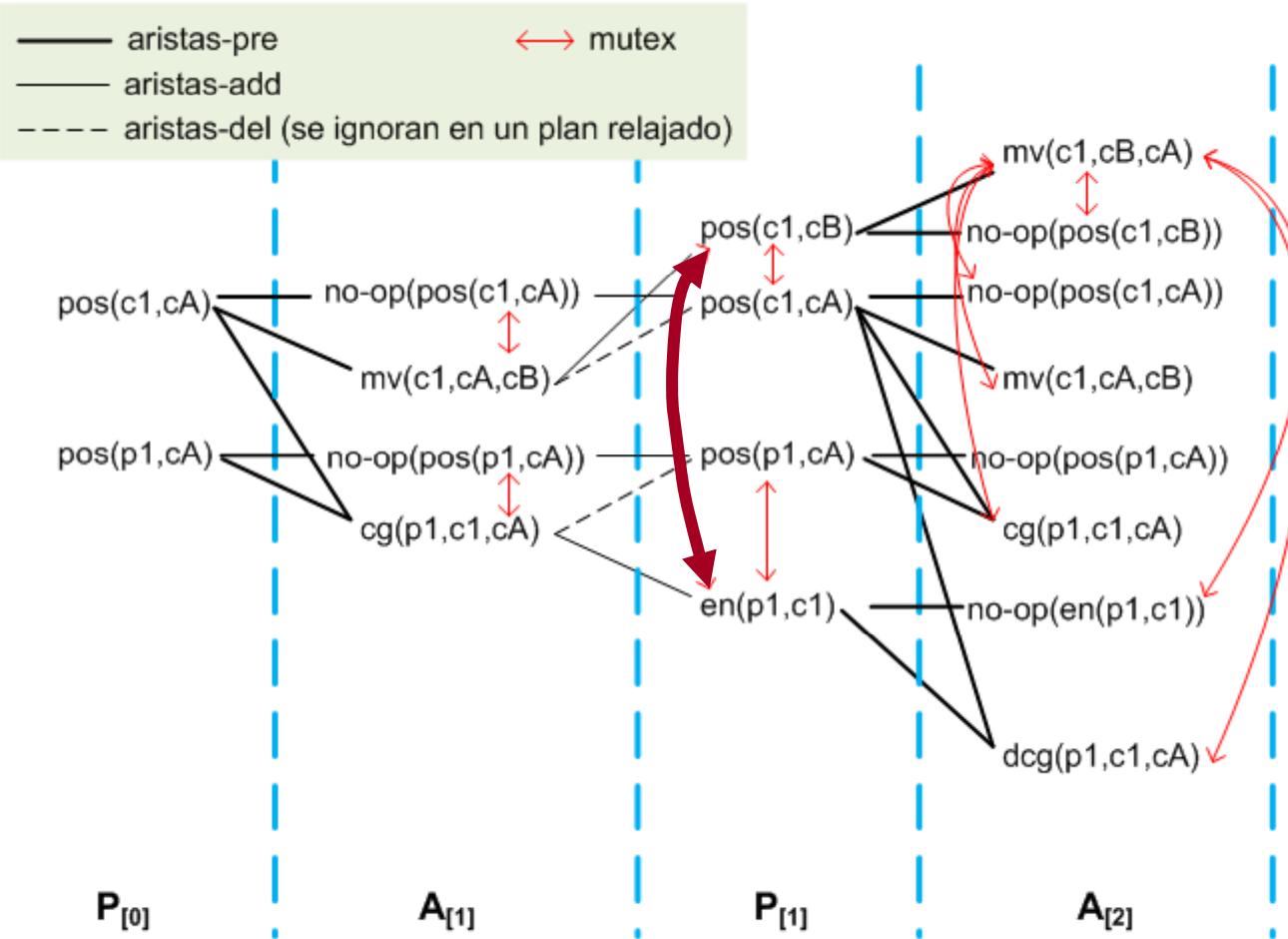
Las **proposiciones mutex** no se podrán dar a la vez

Ej.  $pos(c1, cB)$  es mutex con  $en(p1, c1)$  porque las acciones que las generan en el nivel previo son mutex

- El grafo se construye incrementalmente a partir del estado inicial.
  - En  $A_t$ , aparecen todas las acciones (*incluidas no-op*) cuyas precondiciones están presentes en  $P_{t-1}$ .
  - En  $P_t$ , aparecen todas las proposiciones alcanzadas a lo largo del grafo de planificación (*relajación*)
- Acciones no-op (*no-operación*): No hace nada con la proposición: idea de persistencia (axioma marco).
- Termina cuando, en un Nivel de Proposición, todos los objetivos se satisfacen y no son mutex.

## Ejemplo: Construcción de un Grafo de Planificación.

Inicial: pos (p1, CA), pos (c1, CA)  
 Objetivo: pos (p1, CB)



Por simplicidad solo se muestran los mutex con mv(c1,cB,cA)

¿Por qué no aparece la acción dcg(p1,c1,cB) en  $A_{[2]}$ ?

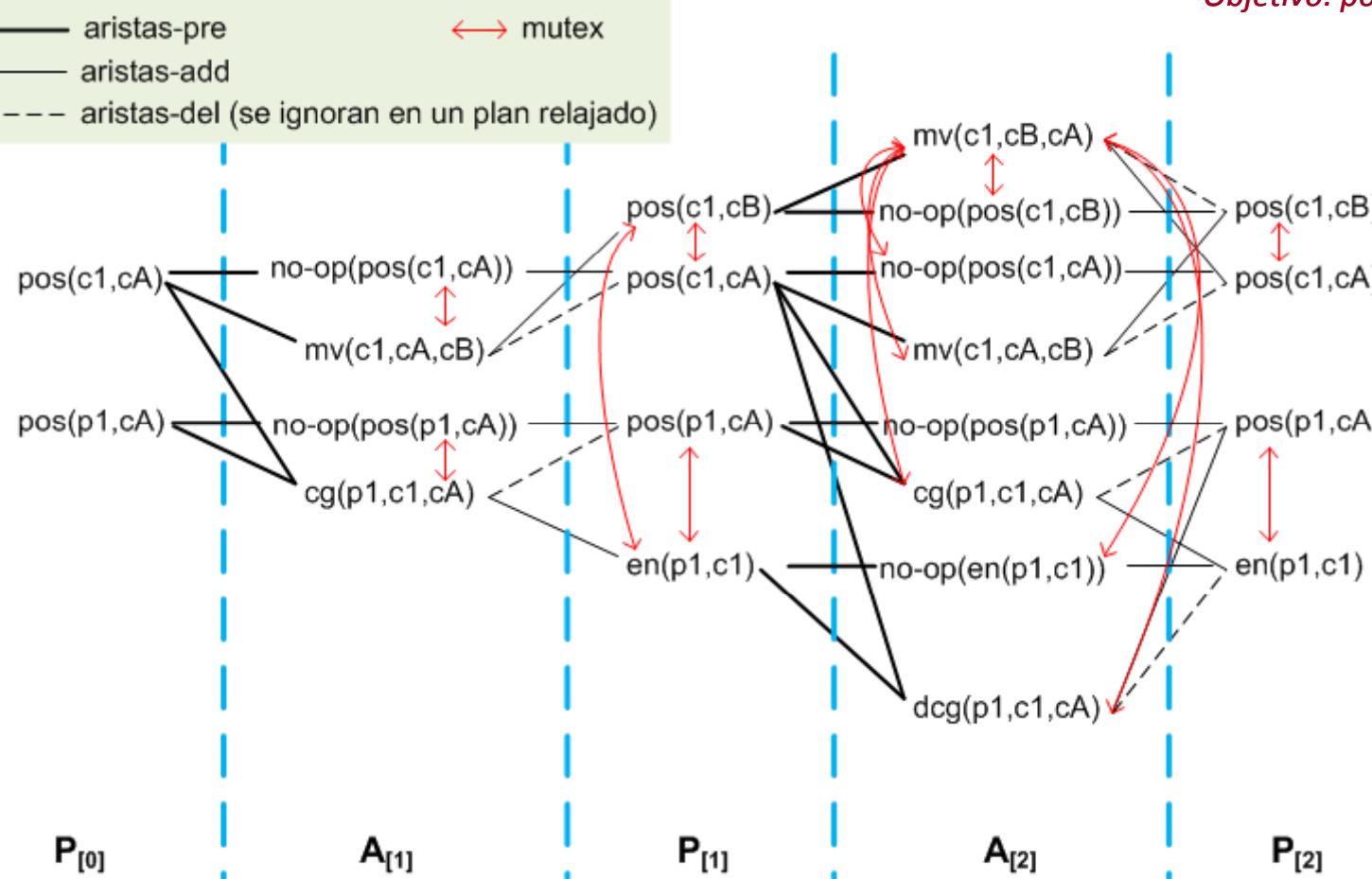
Proposiciones mutex

- El grafo se construye incrementalmente a partir del estado inicial.
  - En  $A_t$ , aparecen todas las acciones (*incluidas no-op*) cuyas precondiciones están presentes en  $P_{t-1}$ .
  - En  $P_t$ , aparecen todas las proposiciones alcanzadas a lo largo del grafo de planificación (*relajación*)
- Acciones no-op (*no-operación*): No hace nada con la proposición: idea de persistencia (axioma marco).
- Termina cuando, en un Nivel de Proposición, todos los objetivos se satisfacen y no son mutex.

## Ejemplo: Construcción de un Grafo de Planificación.

*Inicjal: pos (p1, CA), pos (c1, CA)*

## *Objetivo: pos (p1, CB)*



El grafo continúa hasta que aparecen los objetivos y no son mutex...

¿Cuántos niveles más habría que construir hasta alcanzar la proposición pos(p1,cB)?

- El Grafo de Planificación termina cuando, en un Nivel de Proposición, todos los Objetivos se satisfacen y no son mutex.
  - *El algoritmo Graphplan obtiene un plan válido a partir del grafo de planificación (aplica algoritmo IDA).*

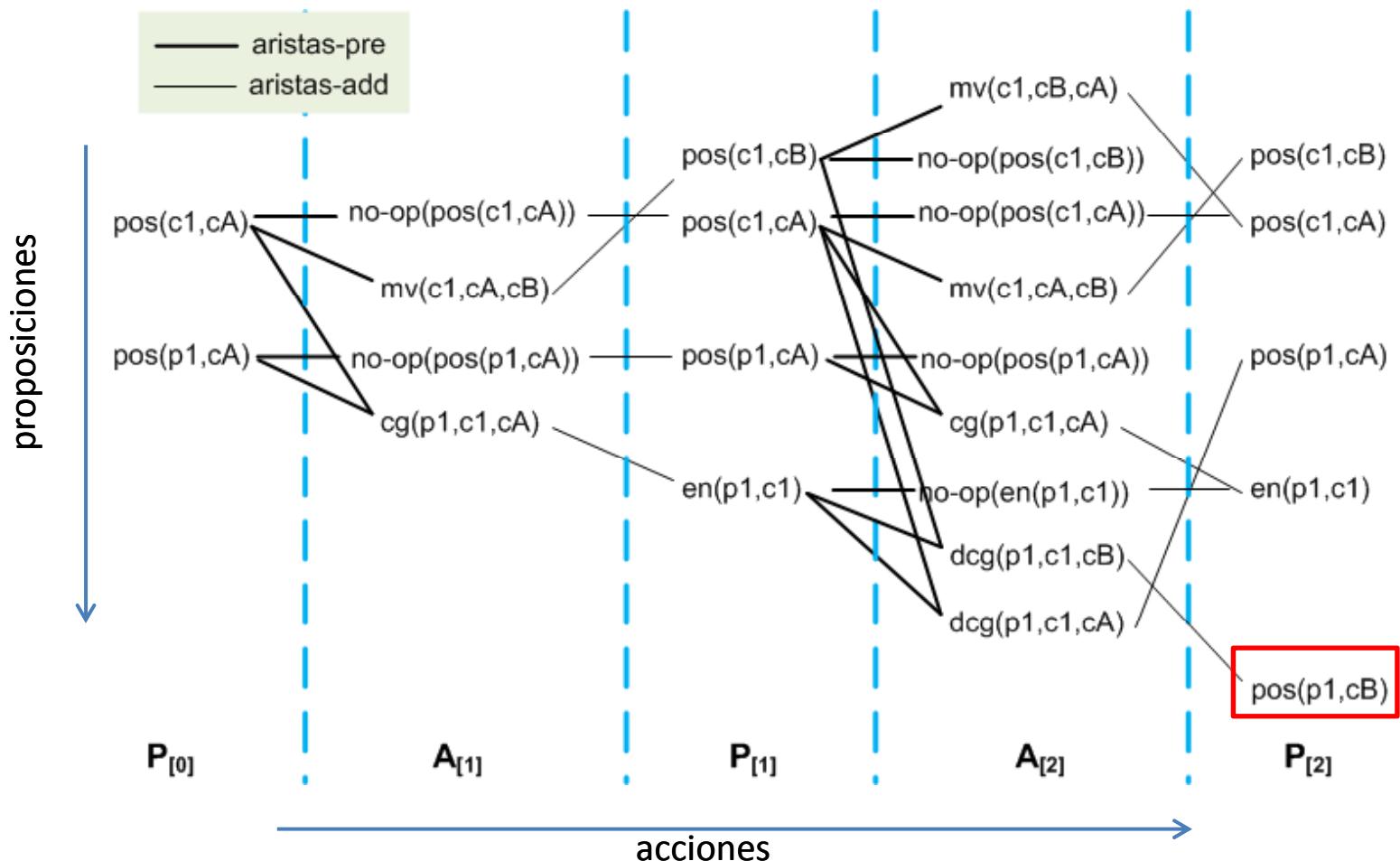
⇒ **Grafo de Planificación Relajado: *NO contempla aristas-del***

## Grafo de Planificación Relajado

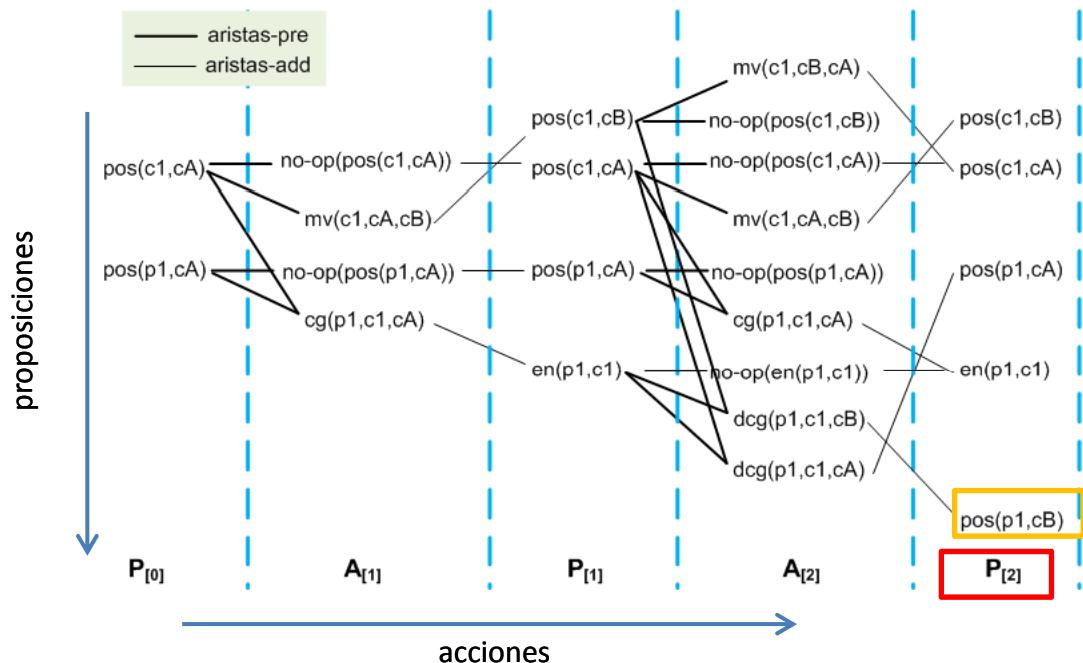
Permiten aplicar la heurística por relajación de efectos delete y estimar  $h(p)$  de cada nivel.

Se construye **exactamente igual** que un grafo de planificación, pero **sin considerar aristas-del** (se relajan los efectos delete). Esto implica que **no habrá mutex**.

*La estimación del número de pasos desde un estado  $P[i]$  a meta es la estimación heurística en  $P[i]$ .*



- El grafo de planificación relajado representa los predicados alcanzables mediante los pasos del grafo  
 → *permite estimar el número de pasos requeridos para alcanzar un estado desde otro estado.*

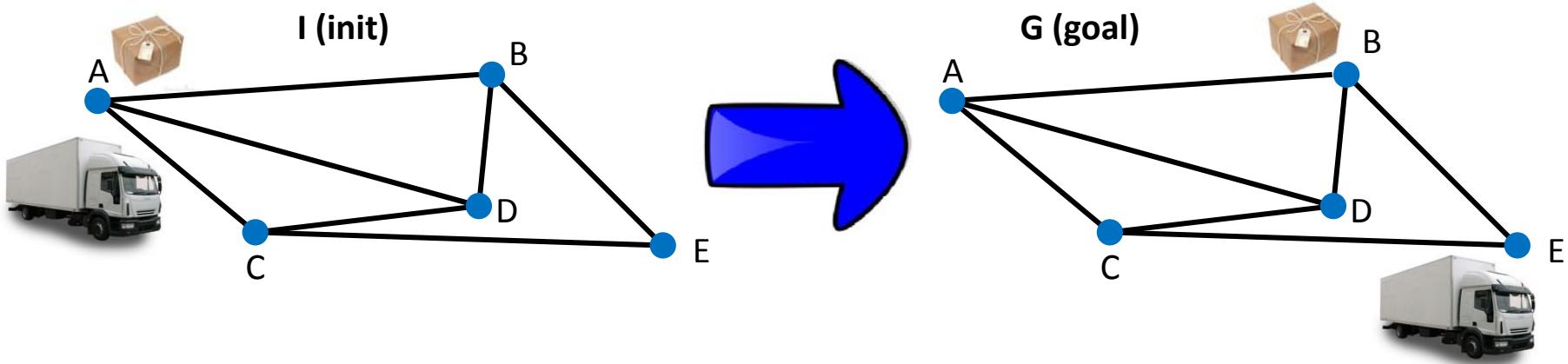


No efectos delete, no mutex ⇒  
 Complejidad de la construcción de  
 un grafo relajado es polinomial:  
**|proposiciones| x |acciones|**

*Objetivo conseguido en dos pasos de  
 planificación desde el estado inicial.  
 ¿Es esto real?*

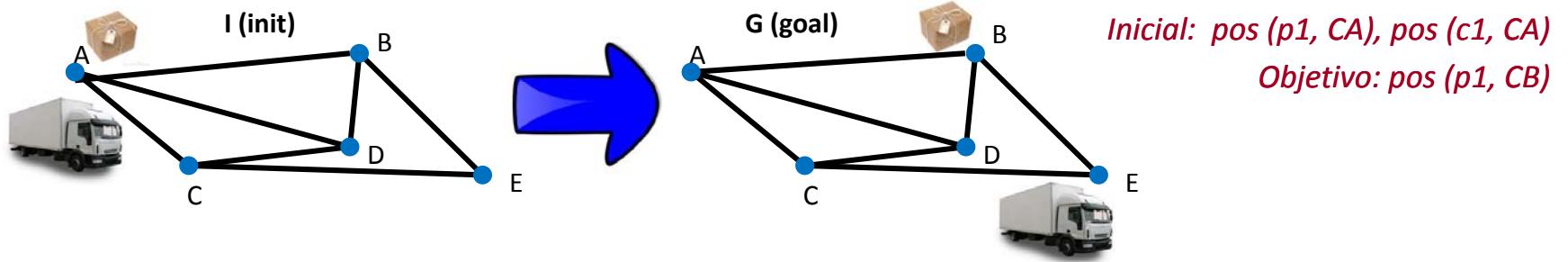
- Estimación conservadora: posiblemente harán falta nuevas acciones/pasos para restaurar los efectos delete no considerados, y secuenciar las acciones/proposiciones mutex.
- Si existe un plan válido de  $t$  pasos (¡pasos, no acciones!), dicho plan existe como un subgrafo (de  $t$  niveles) del árbol de planificación correspondiente al problema.
- Al igual que un grafo de planificación, las proposiciones y acciones son monótonamente crecientes:
  - Si una proposición/acción no aparece en un nivel  $t$ , es porque no existe una combinación factible que permita alcanzar/ejecutar dicha proposición/acción en  $t$  pasos de planificación.
  - Una vez una proposición/acción aparece en un nivel  $t$ , aparece en todos los niveles sucesivos.

## Ejercicio. Grafo de planificación relajado



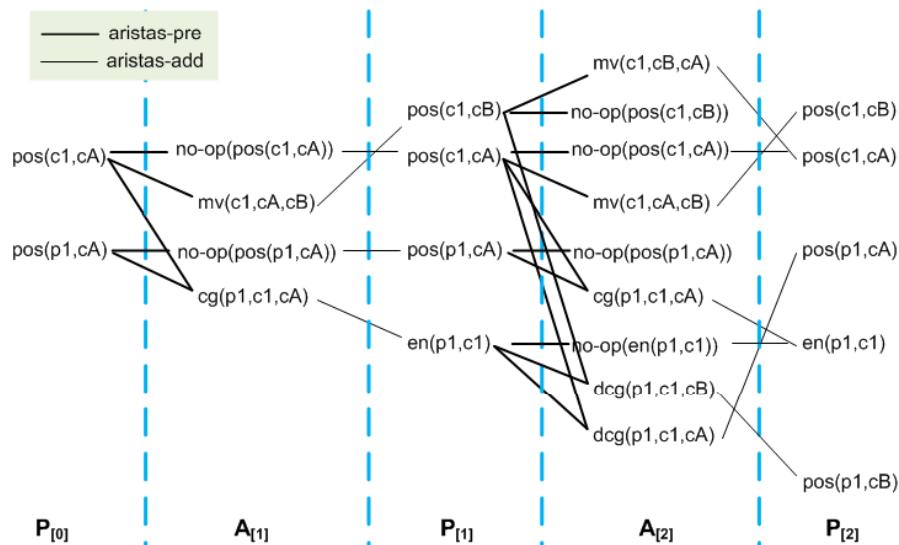
Construir un grafo de planificación relajado **para estimar**:

- El número de pasos de planificación para conseguir el objetivo desde el estado inicial
- El número de pasos de planificación para conseguir que el camión esté en E desde el estado en el que se ha conseguido por primera vez que el paquete esté en B
- Si queremos estimar el número de acciones necesarias, ¿coincide este valor con el número de pasos de planificación?
- ¿Qué complejidad tienen las operaciones anteriores?
- ¿Cuál es la complejidad de obtener el plan relajado óptimo de menor número de acciones?



## Planificación heurística. Relajación y otras variantes.

- Un grafo de planificación proporciona una estructura útil para aplicar una gran **familia de heurísticas** cuando se desea conseguir varios objetivos.
- **Heurística por Relajación:** El método es construir, desde cada estado, un *grafo de planificación* hasta un objetivo y extraer a partir de él un *plan relajado* para calcular  $h$ : número de acciones, coste, utilidad, etc.
  - Ejemplo: Estimación del coste de alcanzar una proposición  $p$ :  $h(p) =$  número de pasos necesarios en el grafo relajado hasta que aparezca dicha proposición.



Ahora  $h(p)$  se lee como la estimación para alcanzar la proposición  $p$  desde el estado actual

Ej.: Desde el estado inicial,

$$h(\text{pos}(c1, CA))= 0$$

$$h(\text{pos}(p1, CB))= 2$$

$$h(G=\text{objetivo}) = h(\text{pos}(p1, B)) = 2$$

$$h_{\text{sum}} \{ \{\text{pos}(c1, CA), (\text{pos}(p1, B)\} \}= 0+2$$

$$h_{\text{máx}} \{ \{\text{pos}(c1, CA), (\text{pos}(p1, B)\} \}= \max(0,2) = 2$$

### Variantes: $h(\{\text{varios objetivos}\})$ :

- $h_{\text{sum}}$ : suma de las estimaciones individuales de cada proposición objetivo (no admisible)
- $h_{\text{máx}}$ : máximo de las estimaciones individuales (admisible, pero poco informada)

## En resumen:

- La heurística de **relajación** se puede aplicar tanto en la búsqueda basada en espacio de estados como en un espacio de planes. *Muy útil para aplicar heurísticas admisibles y no admisibles.*
  - Para cada nodo frontera en la búsqueda, se genera su grafo de planificación relajado y se calcula su  $h(G)$ , o su  $h_{\text{sum}}$ ,  $h_{\text{max}}$ , o  $h_{\text{max\_k}}$
  - Los valores  $h(G)$  de los nodos frontera se utilizan para guiar la búsqueda
- Heurística ***utilizada como base por muchos planificadores actuales***
- Se pueden **relajar tantos aspectos como nos interese** (efectos delete, valores numéricos, utilización de recursos, tiempo/duraciones, etc.), pero es interesante que se haga a partir del elementos del modelo PDDL (dominio+problema), ya que:
  - así se consigue **independencia de cada dominio** y mayor **aplicabilidad**
  - aunque a costa de perder precisión al no aprovechar las particularidades del problema concreto: *independencia del dominio vs. dependencia del dominio*
- El **grafo de planificación** proporciona una estructura útil para aplicar una gran familia de heurísticas basadas en relajación. Hay **otras formas** de relajación

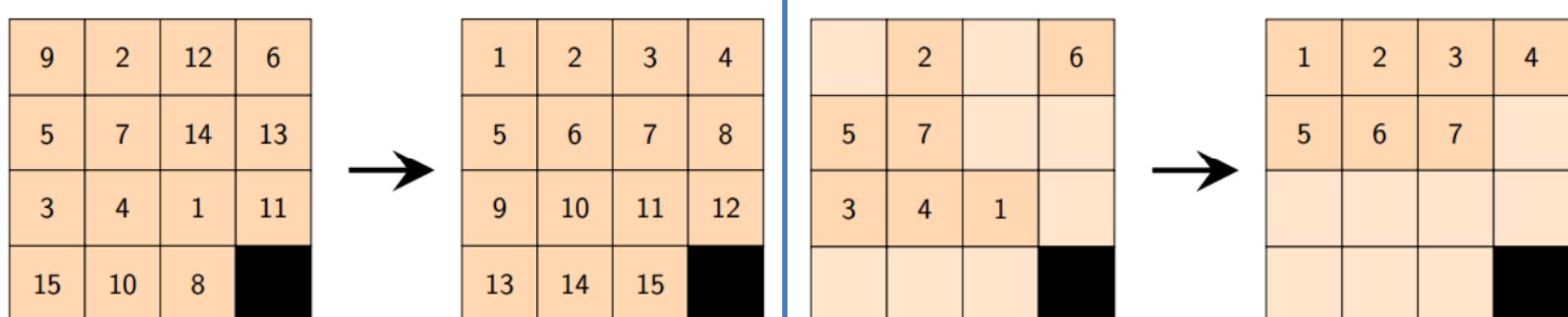
# Planificación heurística. Abstracción

## Definición de Abstracción

Es un mecanismo de relajación que elimina ciertas distinciones entre estados para obtener una representación más **compacta** tratando de preservar el comportamiento de las transiciones (acciones) tanto como sea posible.

- Es como un mapping que comprime varios estados iniciales en uno “proyectado”.
- La abstracción debe definir qué estados se considerarán iguales y cuáles no.
- Se calculará la estimación sobre el nuevo estado reducido más **compacto** basado en variables y sus dominios asociados. Es **possible** que se necesiten nuevas estructuras

## Intuitivamente en el problema del 15-puzzle



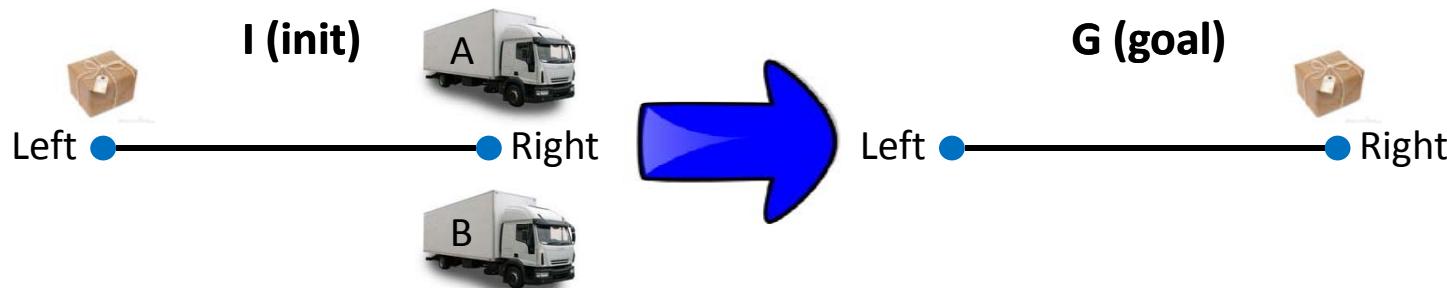
Espacio de estados:

$$16! = 20922789888000 > 2 \cdot 10^{13} \text{ estados}$$

Abstracción: proyección que ignora la posición de las celdas 8..15  
espacio de estados:

$$16!/8! = 518918400 > 5 \cdot 10^8 \text{ estados}$$

## Abstracción. Ejemplo de logística simplificado



**Variables:** {paquete, camiónA, camiónB}

**Dominio(paquete) = {A,B,L,R}, Dominio(camiónA)=Dominio(camiónB)={L,R}**

**I:** paquete=L, camiónA=camiónB=R;      **G:** paquete=R

**A:**

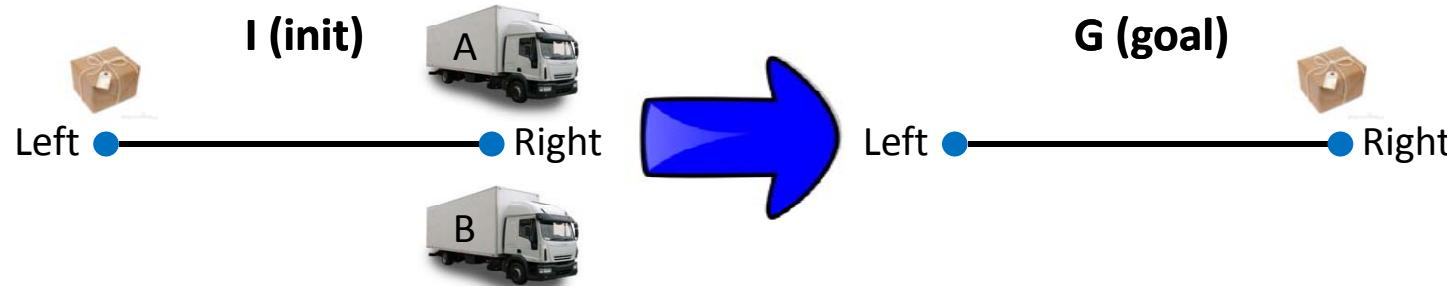
- mover(?vehículo, ?origen, ?destino) con transición  
?vehículo = ?origen → ?vehículo = ?destino
- cargar(?objeto, ?vehículo, ?lugar) con transición  
?objeto = ?lugar, ?vehículo = ?lugar → ?objeto = ?vehículo
- descargar(?objeto, ?vehículo, ?lugar) con transición  
?objeto = ?vehículo, ?vehículo = ?lugar → ?objeto = ?lugar

Utilización de variables de estado de forma no proposicional . Ofrece una representación más compacta (formalismo SAS<sup>+</sup>):

*Forma proposicional: At/In (A, R).*  
*Forma Variables-Estado: A=R*

- Ya no hace falta la definición de efectos add y delete (las variables toman un valor)
- Permite construir **otras estructuras útiles** como los *Domain Transition Graphs* (DTGs) usadas en los **planificadores**

## Abstracción. Ejemplo de logística simplificado

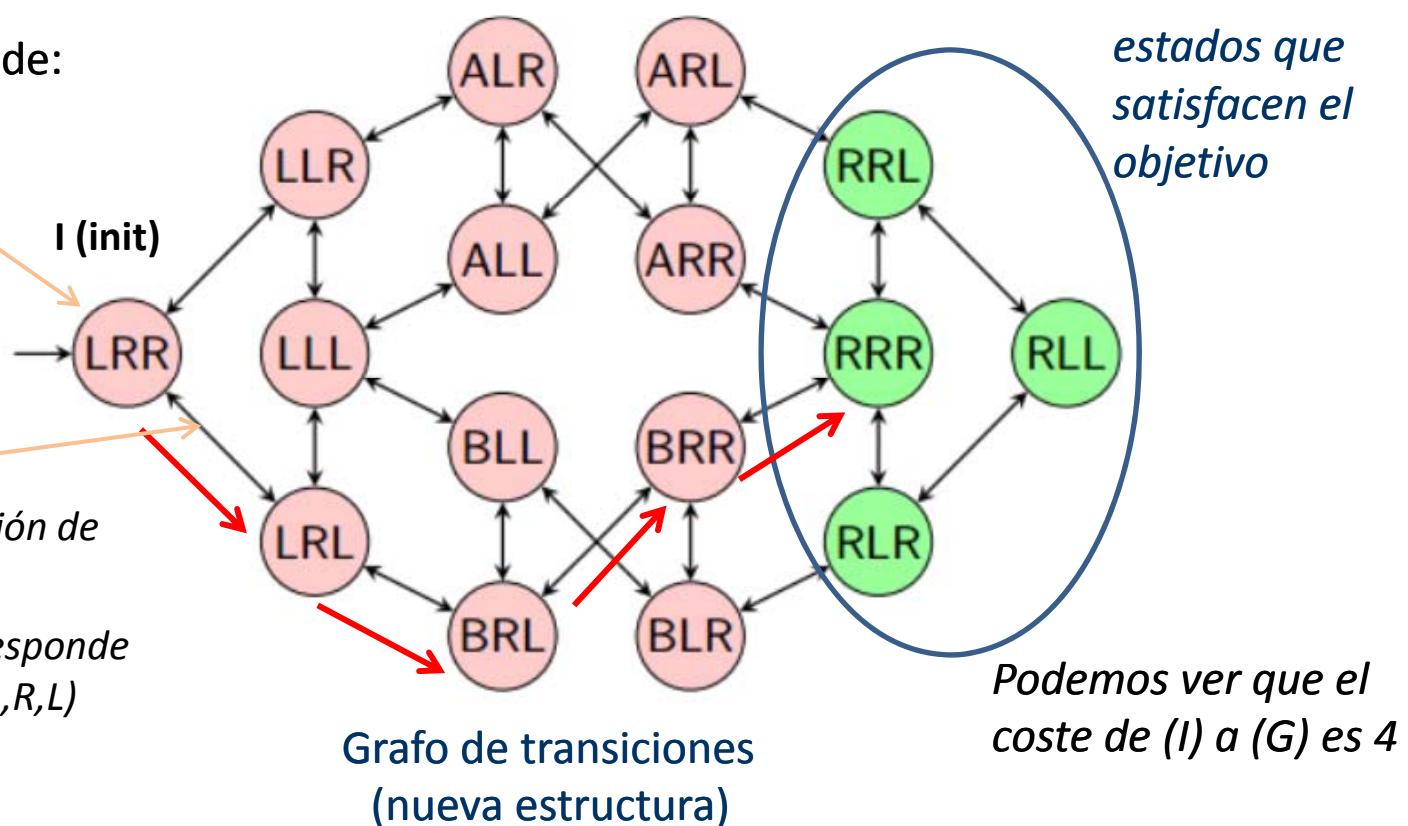


Representa el valor de:

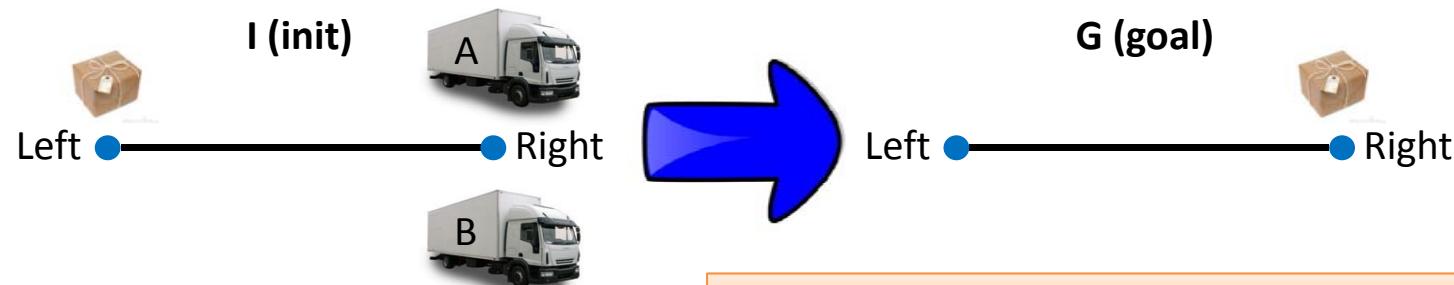
- paquete = L
- camiónA = R
- camiónB = R

Transiciones  
(representan la ejecución de una sola acción).

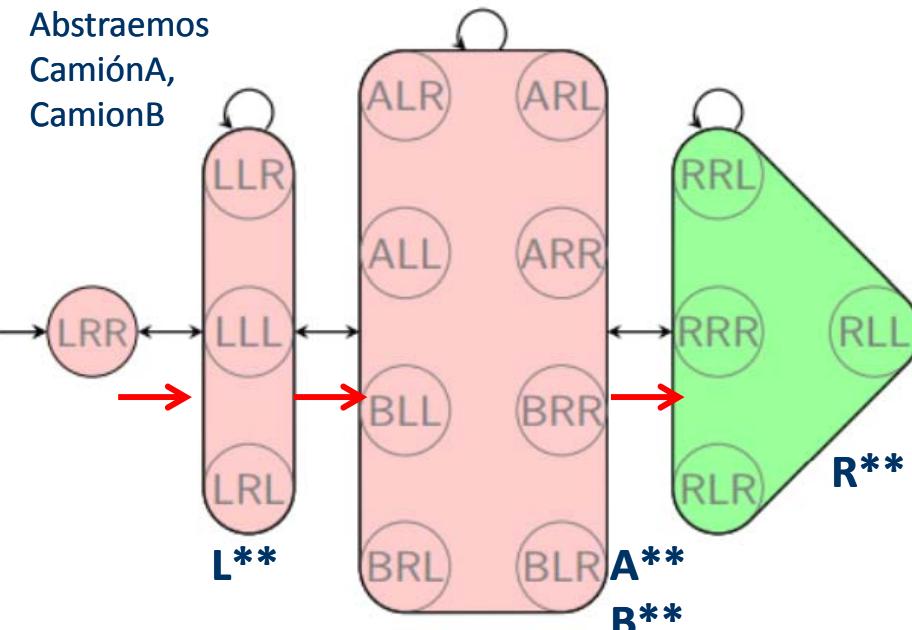
Ej.  $LRR \rightarrow LLR$  se corresponde con  $move(camiónA, R, L)$



## Abstracción. Ejemplo de logística simplificado

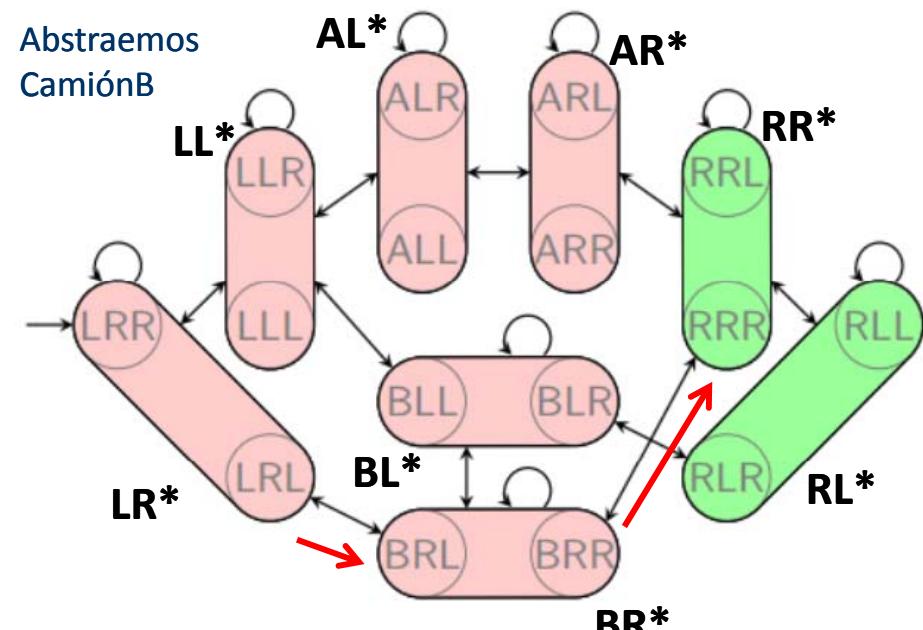


### Possibles abstracciones



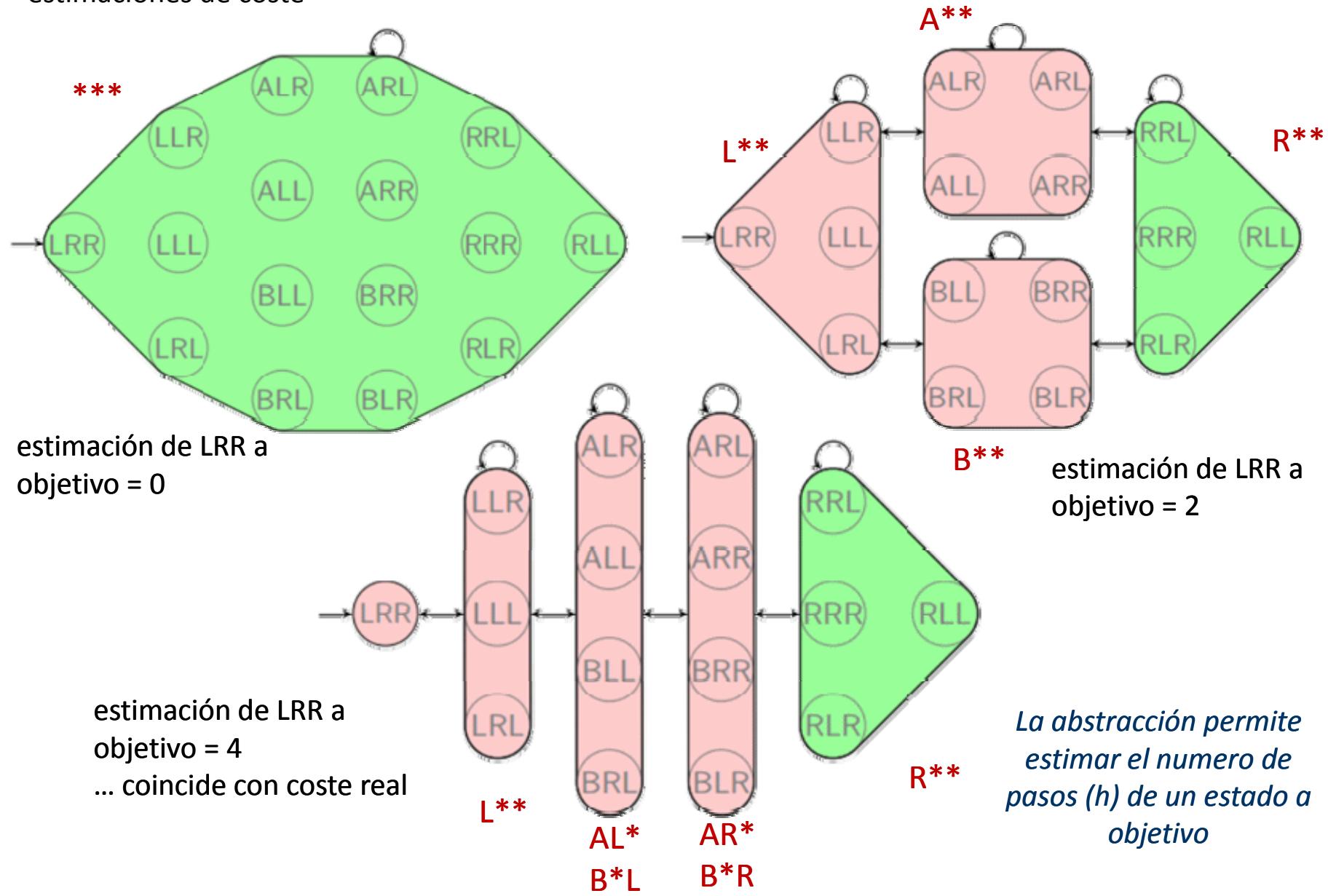
estimación de LRR a objetivo = 3,  
aunque el coste real es 4

Debido a la abstracción de estados, ahora una transición representa una o varias acciones!

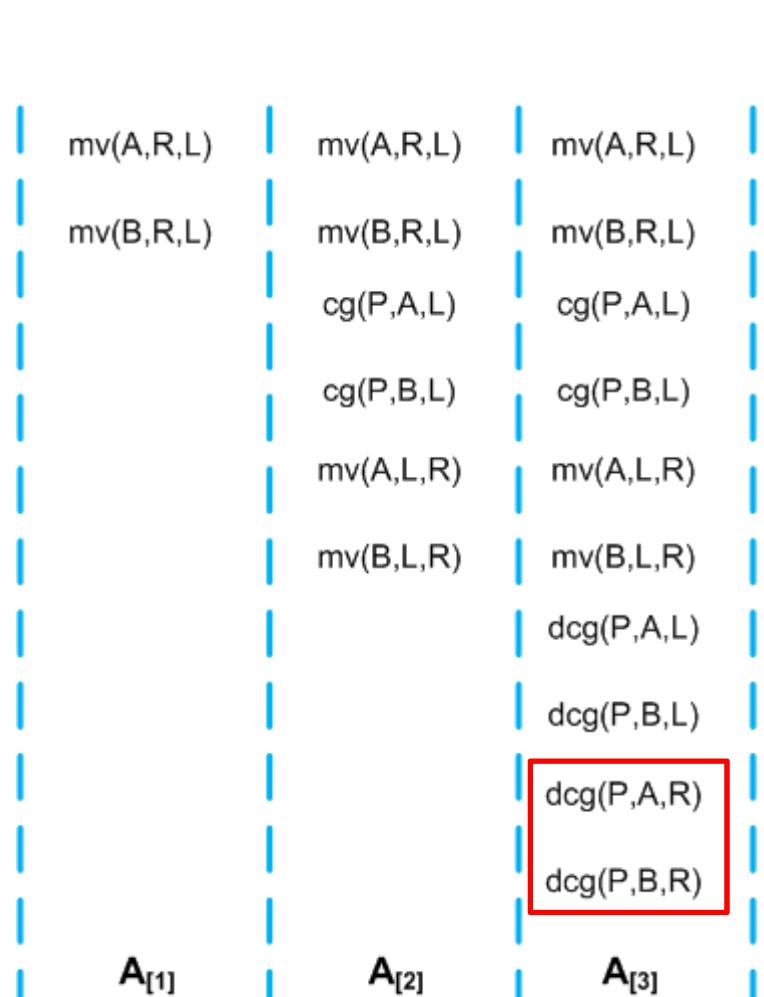


estimación de LRR a objetivo = 2  
el coste real sigue siendo 4

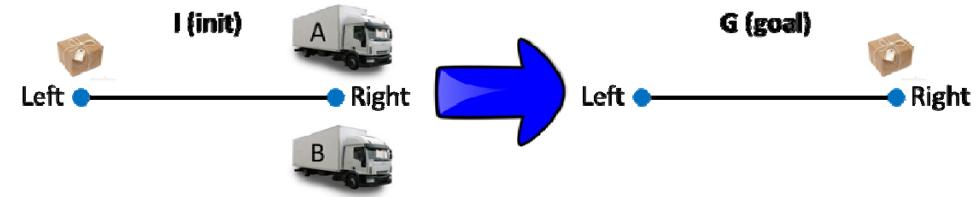
Por supuesto, hay muchas **más posibles abstracciones**... que conducen a mejores o peores estimaciones de coste



En comparación con un grafo de planificación relajado...



Grafo de planificación relajado  
(solo se muestran los niveles de acción)



- $mv(?vehículo, ?origen, ?destino)$
- $cg(?paquete, ?vehiculo, ?lugar)$
- $d cg(?paquete, ?vehiculo, ?lugar)$

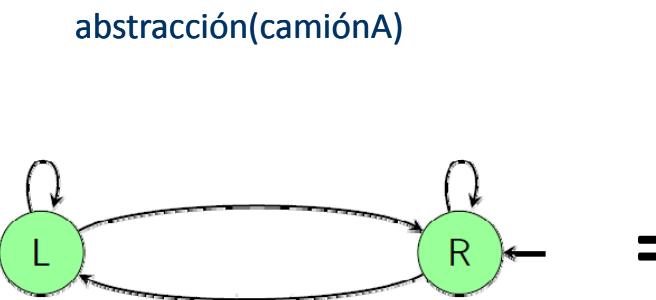
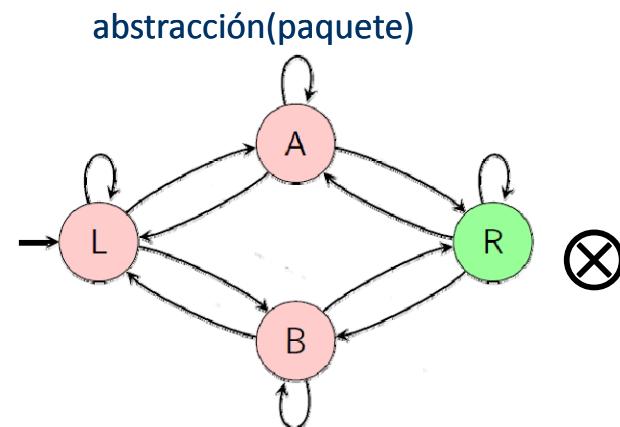
Como se observa, el mecanismo de abstracción nos ofrece **varios niveles de abstracción** que conducen a **distintas estimaciones** (4, 3, 2, 0)

Por el **contrario**, el grafo de planificación relajado estima la distancia al objetivo en este problema **siempre** en 3

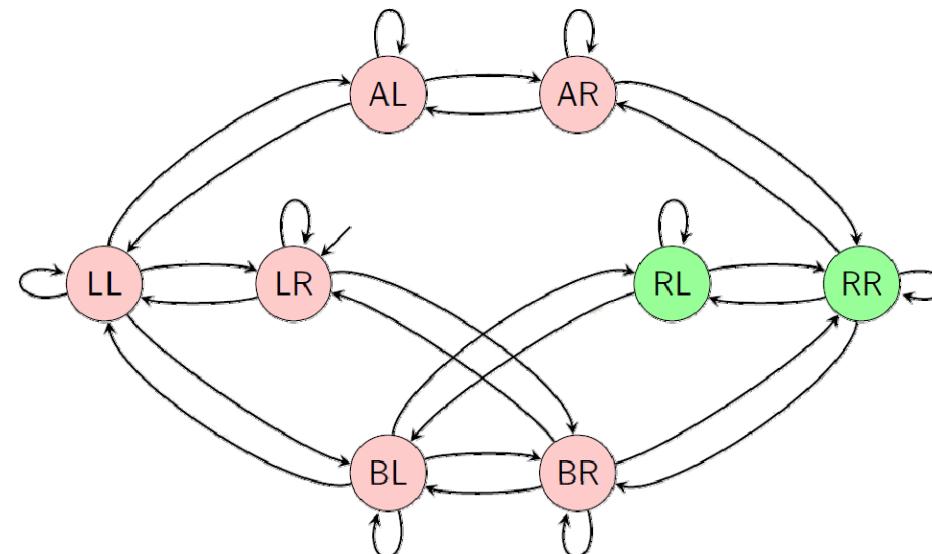
Cada planificador usa distintas heurísticas

## Aplicación Heurística Abstracción: Método Merge & shrink.

- Realizamos abstracciones sobre variables individuales (es decir, proyecciones atómicas)
- Realizamos el producto cartesiano  $\otimes$  sobre estas variables individuales
  - Explosión combinatoria a medida que aumenta el número de variables



aproximación de abstracción (paquete, camiónA)



$LL^*$   
 $LR^*$   
 $AL^*$   
 $AR^*$   
 $BL^*$   
 $BR^*$   
 $RL^*$   
 $RR^*$

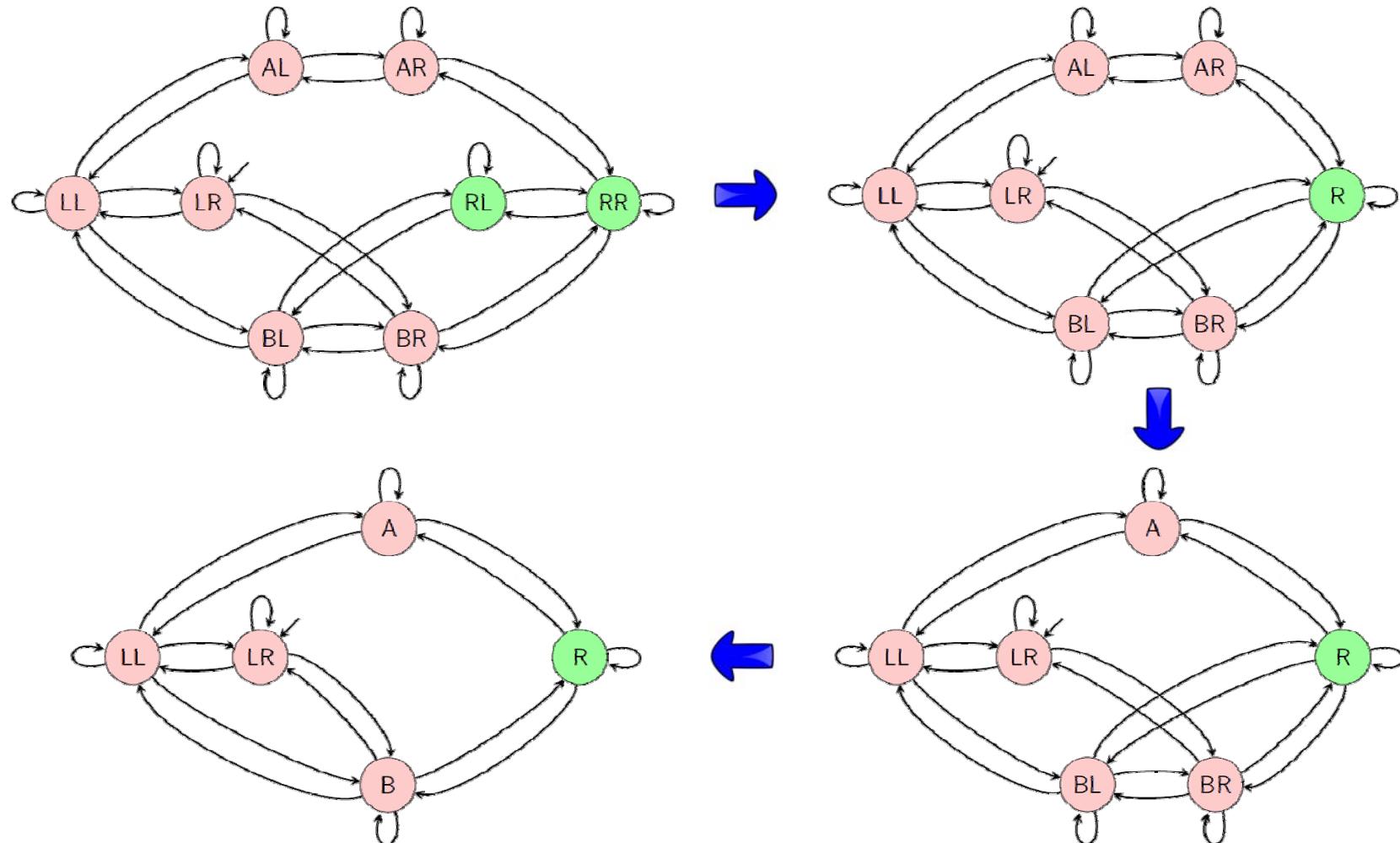
Idea:

1. Fusionar (merge:  $\otimes$ ) las proyecciones individuales
2. Reducir (shrink) el resultado de la fusión para que el tamaño no explote realizando nuevas abstracciones (se define un límite  $n$  en el nº de estados)

2. Se reduce (**shrink**) la fusión previa, realizando nuevas abstracciones:

Ejemplo intuitivo con n=5

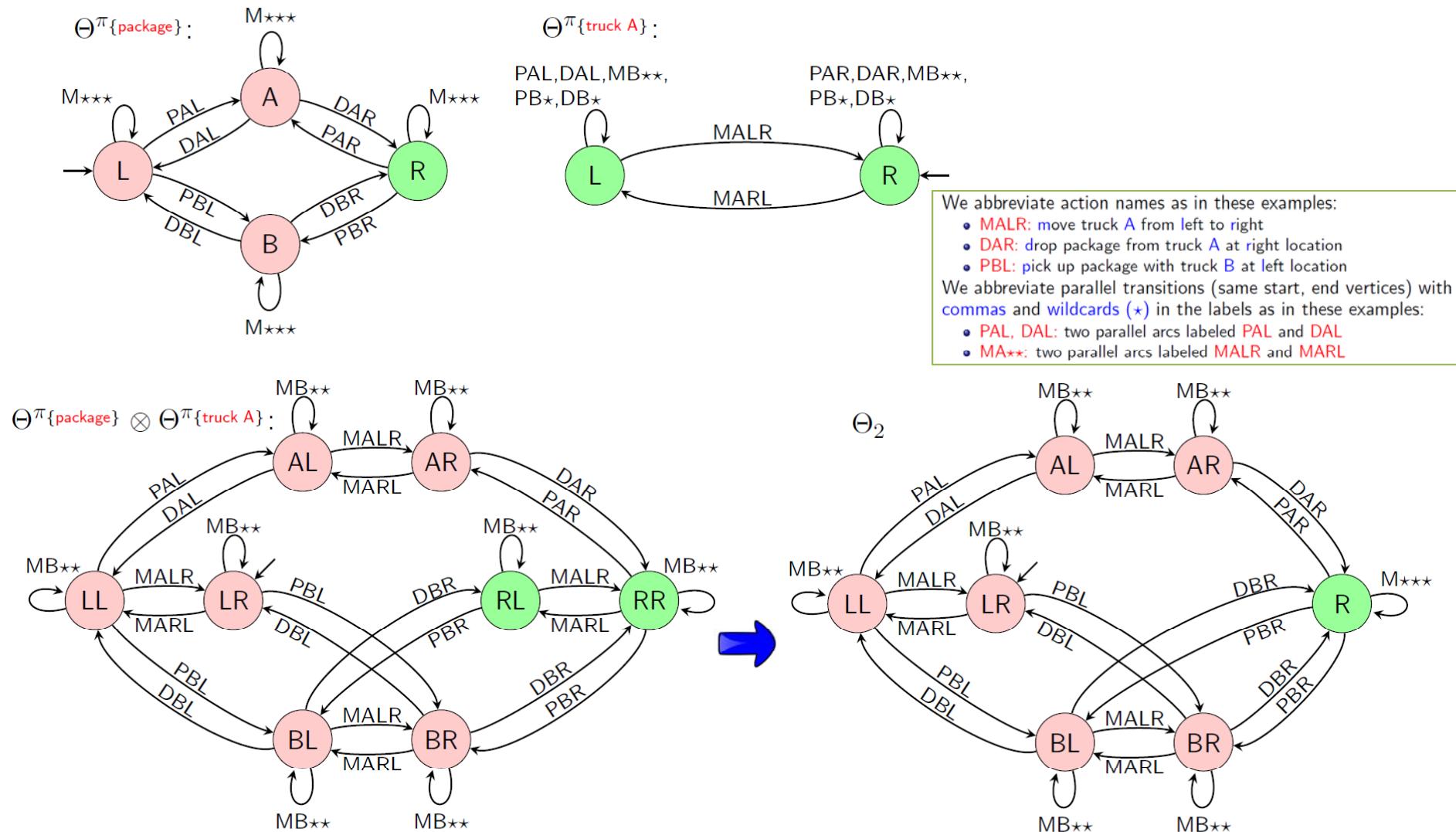
Nº de transiciones:  
estimación nº acciones,  $h(p)$



Sobre estos conjuntos se van fusionando progresivamente otros...

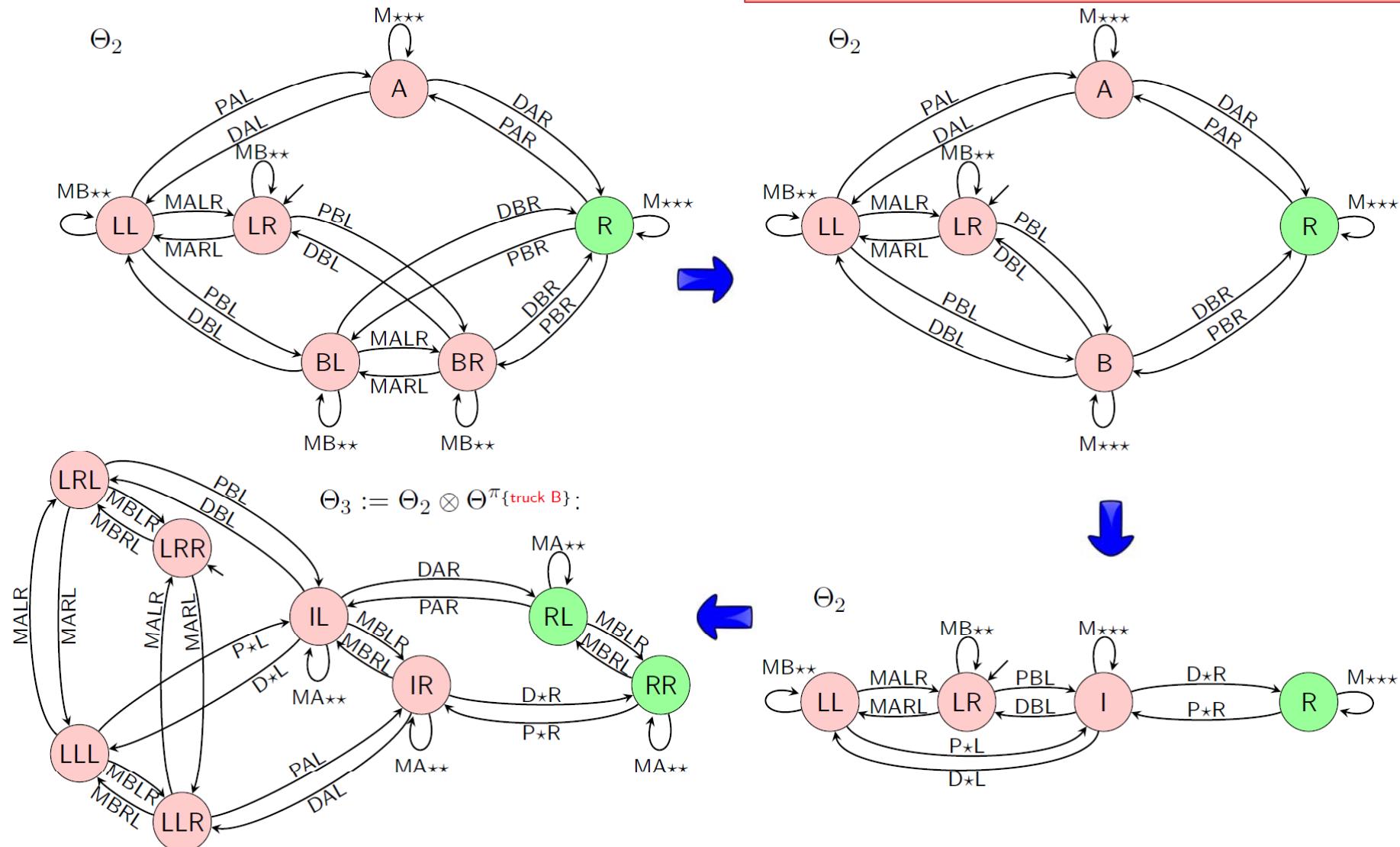
## Abstracción. Merge & shrink. Lo que hace realmente el algoritmo (y con las transiciones exactas)

Y ahora el procedimiento entero en detalle...  
[Jörg Hoffmann]



## Abstracción. Merge & shrink. Ejemplo con más detalle y con las transiciones exactas

Y ahora el procedimiento entero en detalle...  
[Jörg Hoffmann]



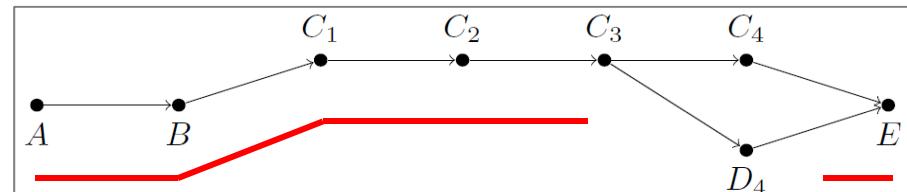
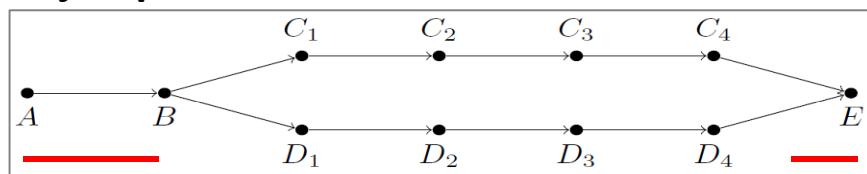
## Planificación heurística. Landmarks ( $\approx$ hito)

- Inicialmente introducidos como un método para descomposición de problemas y establecer órdenes razonables en los planificadores
- Un landmark es algo que todo plan válido **debe satisfacer** en algún punto

*Por ejemplo: En el caso de utilizar un único camión, seguro que en algún punto del plan:*

- *habrá que cargar y descargar cada paquete en/desde el camión,* (\*acción\*)
- *el paquete debe estar cargado en el camión,* (\*proposición\*)
- Los objetivos del problema son **siempre** landmarks
- Landmark de acciones disjuntas: al menos una acción debe aparecer en el plan.
  - Ej: si hay dos camiones, cargar(paquete, camiónA, left) o cargar(paquete, camiónB, left)
- Orden en landmarks  $\Rightarrow$  Permite definir una amplia variedad de órdenes entre estados
  - Posteriormente se han **extendido y mejorado**:
    - $x$  debe obtenerse (justo) antes que  $y$
    - $x$  debe obtenerse antes que  $y$ , o en caso contrario el plan será más largo

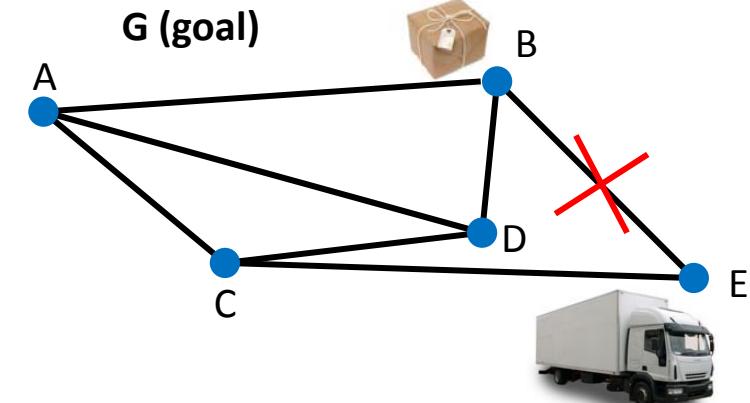
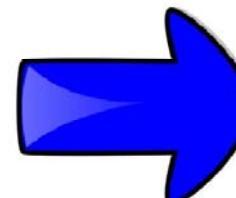
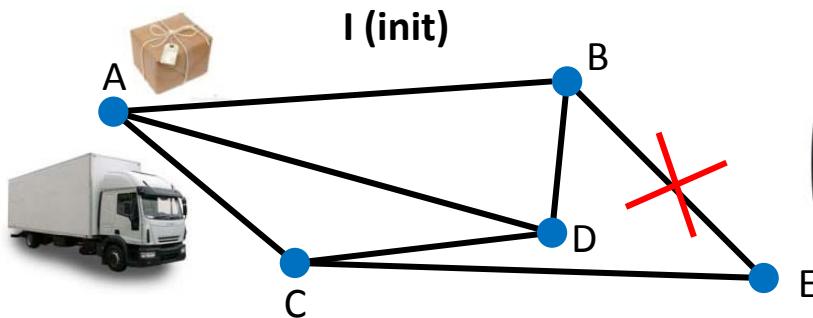
### Ejemplos:



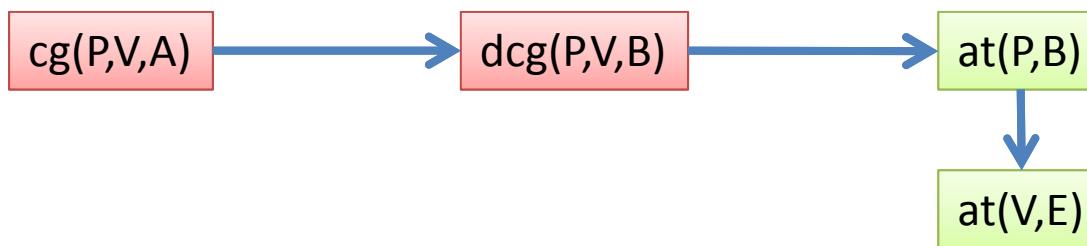
En primer caso “A-B...E” siempre deben estar. En segundo caso “A-B-C1-C2-C3...E” debe estar presentes

Similitudes con técnicas de **camino crítico**. Se construye un **esqueleto** del plan

## Planificación heurística. Landmarks ( $\approx$ hito). Ejemplos



Solo hay un paquete y un vehículo

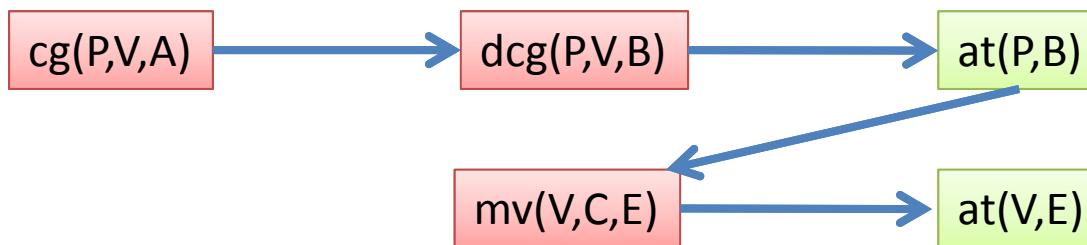


- $mv(\text{?vehículo}, \text{?origen}, \text{?destino})$
- $cg(\text{?paquete}, \text{?vehículo}, \text{?lugar})$
- $dcg(\text{?paquete}, \text{?vehículo}, \text{?lugar})$

acción

proposición

No existe la conexión (carretera) entre B y E,  
solo se puede llegar a E por C



¿Y si hubiera muchos  
paquetes y muchos  
vehículos?

### 3.6.- Planificación para el mundo real

La aplicación de planificación a problemas del mundo real está aumentando a un ritmo vertiginoso y los planificadores se tienen que volver más complejos

- Ahora se pueden generar planes con miles de acciones
- Pero para ello hay que **seguir relajando** las asunciones iniciales
- Requiere **cambios en la representación** y en las **técnicas de resolución**



#### Modelo no determinista y observabilidad limitada

- **Planificación probabilística**

- Los efectos de una acción tienen **probabilidades**, y la duración sigue una **determinada distribución**.

Ej. una acción de transporte tiene una probabilidad de 0.9 de llegar al destino, puede no llegar, o tener mayor duración (retrasos, averías, etc.): esto condicionará el éxito del plan y de sus objetivos

- El mundo no es totalmente conocido: existe **incertidumbre**
- El mundo es **dinámico**, cambia aunque no queramos debido a eventos exógenos.  
Ej. *el sol sale a una determinada hora* → importante si usamos paneles solares.  
Esto se define como **Timed Initial Literals**, TILs, desde PDDL2.2 en el archivo del problema
- En los dos casos anteriores habrá que planificar acciones de sensorización para conocer el estado del mundo: **planificación conforme** y **planificación contingente**.

# Planificación en un mundo dinámico. Planificación conforme y contingente

## Dos aproximaciones

- **Planificación Conforme (*conformant planning*)**

- Se generan planes para contemplar múltiples posibilidades del mundo a pesar de la incertidumbre del estado inicial y efectos no deterministas de las acciones.

*Un plan conforme es una secuencia de acciones que alcanza el objetivo sin importar la incertidumbre en las condiciones iniciales ni los efectos no deterministas de las acciones*

- Hay que añadir la condicionalidad en las acciones (en PDDL se conoce como efectos condicionales, que utilizan la cláusula *when*) lo que dificulta el modelado y la resolución
- La condicionalidad puede ser **muy elevada**

- **Planificación Contingente (*contingent planning*)**

- Conformant planning extendido con acciones de sensorización/monitorización que hay que planificar → planificación preparada para contingencias/emergencias

- El plan contingente se adapta según la información de los sensores, definiendo planes condicionales **y/o reactivos:**

*ante esta situación, ¡haz esto!*

- Mientras se está planificando es imposible saber qué efectos ocurrirán por una acción en tiempo de ejecución (*en el vídeo no sé a priori si el bote está lleno o vacío*)

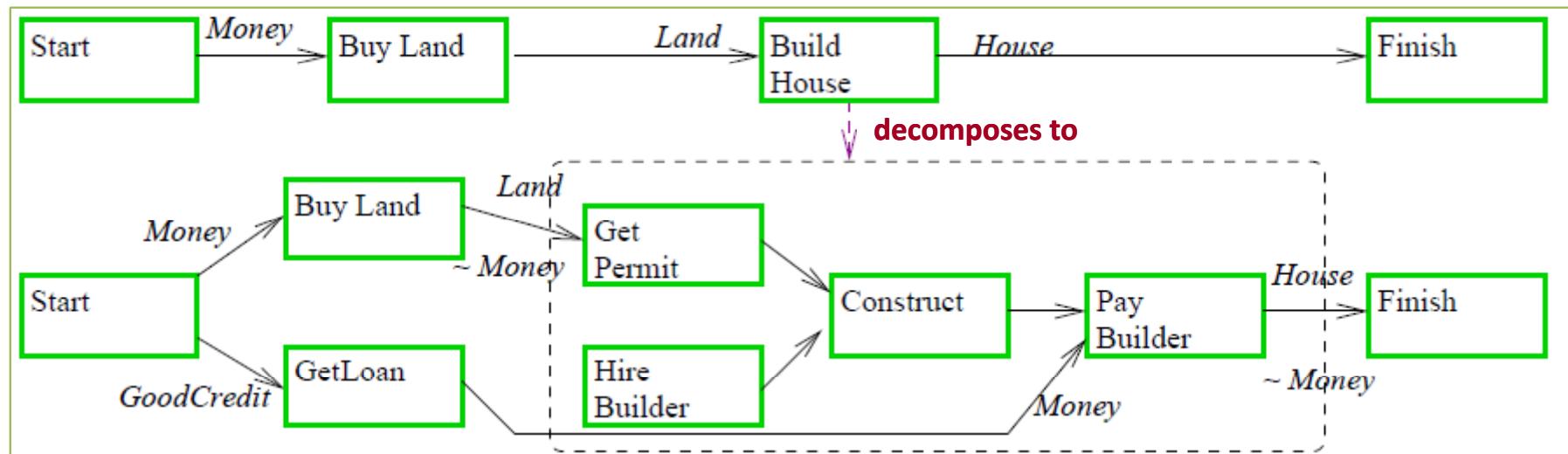


# Planificación en un entorno jerárquico. Planificación jerárquica

## Uso de jerarquías, Hierarchical Task Network (HTN)

- Se establece una jerarquía de acciones en la **representación del modelo**
- Permite una formalización **más natural y real**: nadie planifica desde lo más general hasta el más mínimo detalle. Ej. al planificar un tour por el extranjero, inicialmente no se piensa en si llegar al aeropuerto de origen en metro, autobús, coche, etc.
- La **técnica de resolución** realiza una descomposición desde lo más abstracto hasta lo más primitivo (acciones ejecutables) – planificación a varios niveles
  - Se descomponen las tareas en subtareas
  - Se manejan las restricciones internas (enlaces causales) y se resuelven las interacciones; si es necesario, probar otra descomposición

*Ejemplo típico para construir una casa de forma jerárquica*



## Planificación y ejecución. Planificación online

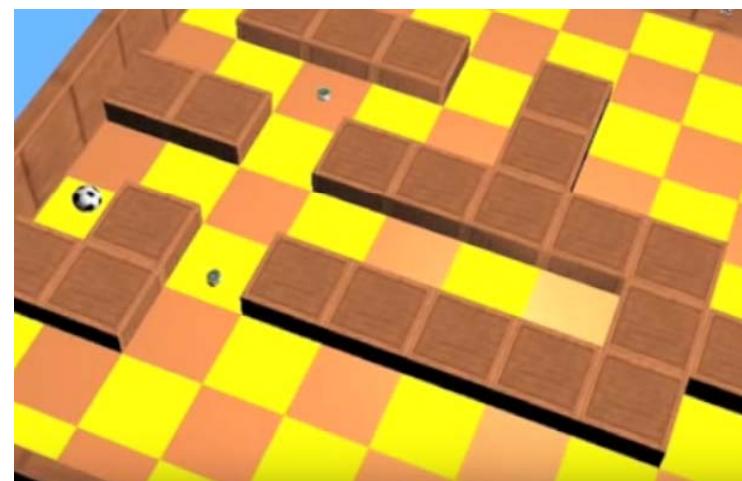


### Online planning: planificación + ejecución

- Hay situaciones donde la planificación y la ejecución están intercaladas
- El objetivo final se desconoce o cambia frecuentemente, debido a la naturaleza del problema o a algún adversario
- No tiene sentido (o simplemente no se puede) generar un plan entero desde el principio que alcance los objetivos
- *Planificación para juegos (planning games)* y/o planificación estratégica contra un adversario: ¿para qué generar un plan completo si cuando juegue el adversario lo más probable es que tenga que cambiarlo?
- *Razonamiento sobre objetivos (goal reasoning)* y objetivos emergentes (como en el vídeo, ¿pueden aparecer nuevos objetivos durante la ejecución del plan?)

### Dynamic Environment PDDL Planning

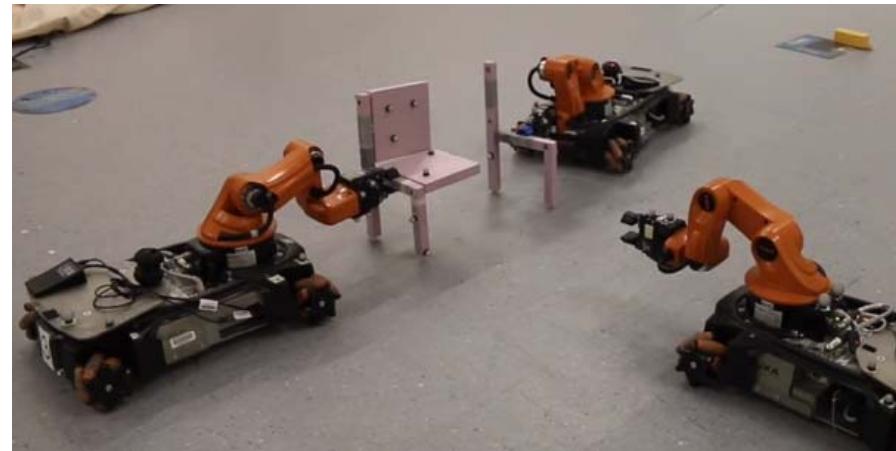
*El movimiento se planifica paso a paso en función de lo que vaya haciendo el adversario*



## Planificación para múltiples ejecutores. Planificación multi-agente

- *Planificación multi-agente*, donde es posible que haya que colaborar para conseguir un objetivo común, competir por el uso de recursos o por la consecución de objetivos privados, llegar a acuerdos y/o situaciones de equilibrio, etc.
- Y todo ello desde un punto de vista **centralizado o distribuido**

[Multi-Robot Grasp Planning for Sequential Assembly Operations](#)



- *Replanificación*: Durante la ejecución se pueden producir problemas inesperados que impidan la ejecución de las siguientes acciones: necesidad de **monitorización constante**
  - Hay que **reparar** el plan (reutilizando el plan existente) o se **replanifica** desde cero. Ej. si el robot se queda sin batería, ¿desechamos el plan existente y usamos otro robot o simplemente recargamos la batería?
  - Se ha demostrado que reparar el plan puede ser tan complejo como generar un nuevo plan. Ej. ¿qué ocurre si hay que traer el robot de repuesto desde muy lejos?
  - Se usan **técnicas de aprendizaje y planificación basada en casos** (Case-Based Planning)

## Repositorios, Herramientas, Entornos y Planificadores

- En **International Planning Competition** hay mucha información sobre planificación, repositorios, PDDL, dominios y problemas de ejemplo, y planificadores: <http://ipc.icaps-conference.org/>
  - A collection of tools for working with planning domains: <http://planning.domains/>
  - PDDL editor on-line: <http://editor.planning.domains/>
  - An automated planner in the cloud (planning solver as a service): <http://solver.planning.domains/>
  - **Planificadores para Linux (algunos disponibles en Polifomat) que soportan distintas versiones de PDDL:**
    - LPG (<http://zeus.ing.unibs.it/lpg/>)
    - mips-xxl (<http://sjabbar.com/mips-xxl-planner>)
    - SGPlan (<https://wah.cse.cuhk.edu.hk/wah/programs/SGPlan/>)
    - TFD (Temporal Fast Downward) (<http://gki.informatik.uni-freiburg.de/tools/tfd/>)
    - Optic (<https://nms.kcl.ac.uk/planning/software/optic.html>)
    - FF y metric-FF (Fast Forward) (<https://fai.cs.uni-saarland.de/hoffmann/ff.html>)
    - FD (Fast Downward) (<http://www.fast-downward.org/>)
- Y muchos más...

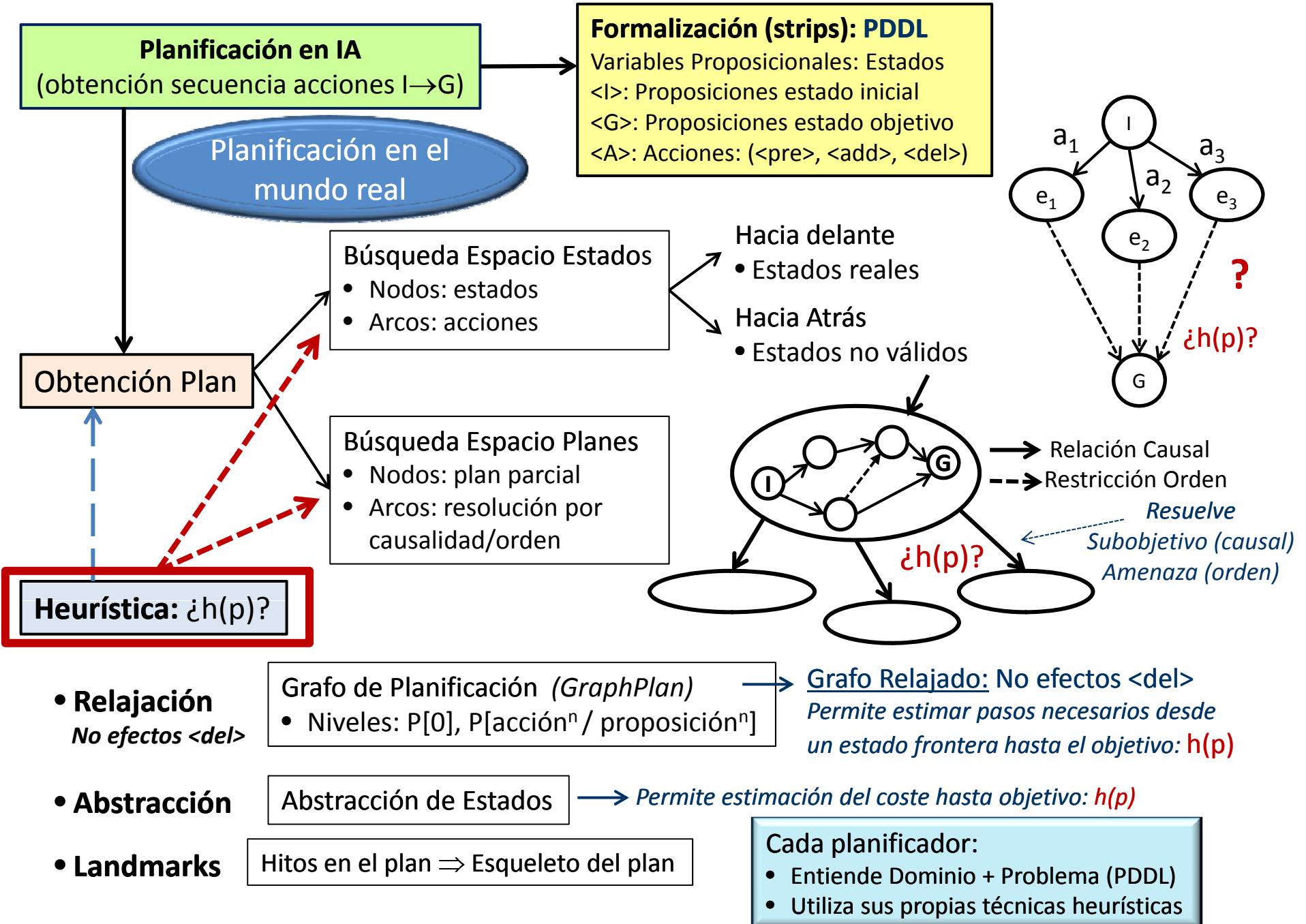
### 3.7.- Conclusiones

#### La planificación es una disciplina de la IA con multitud de técnicas y aplicaciones

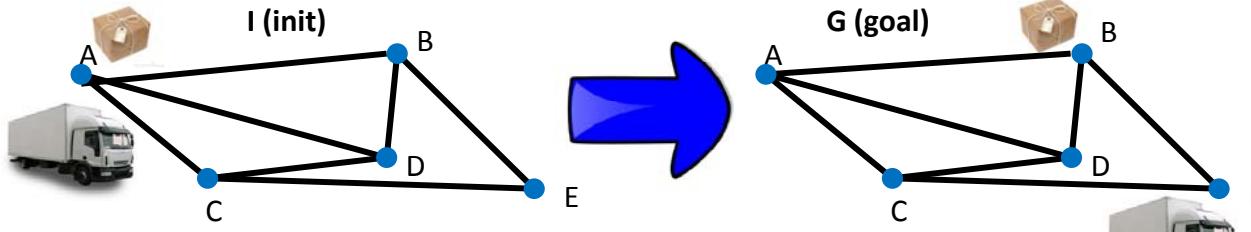
- Consiste en un proceso **formal** de resolución de problemas mediante búsqueda y técnicas de demostración de problemas
  - Objetivo: obtener secuencia de acciones para obtener una meta.
  - La definición y uso de heurísticas es fundamental y marca el éxito de un planificador
  - La planificación inteligente se centra más en encontrar una solución que en encontrar la mejor solución (planificación óptima es mucho más compleja y cara)
  - También se trabaja en encontrar lenguajes más expresivos: sucesivas mejoras en PDDL que enriquecen el lenguaje
- Se suele combinar con otras técnicas: problema de satisfacción de restricciones (CSPs), scheduling e incluso con técnicas de investigación operativa y optimización
- **No hay dos planificadores iguales:** cada uno usa/combina técnicas de distinta forma
- Su aplicación está muy extendida: medicina, logística, procesos de negocio, enseñanza...

#### Más info:

- International Conference on Automated Planning and Scheduling, ICAPS:  
<http://www.icaps-conference.org/>



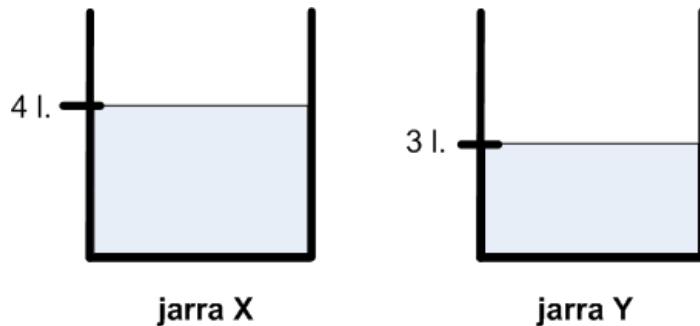
## Solución al problema Grafo Relajado



<b>P[0]</b>	<b>A[1]</b>	<b>P[1]</b>	<b>A[2]</b>	<b>P[2]</b>
<b>Pos(c1, cA)</b>	<b>No-op ()</b>	<b>Pos(c1, cA)</b>	<b>No-op ()</b>	<b>Pos(c1, cA)</b>
<b>Pos (p1, cA)</b>	<b>Mv (c1, cA, cB)</b>	<b>Pos (p1, cA)</b>	<b>Mv (c1, cA, cB)</b>	<b>En(p1, c1)</b>
	<b>Mv (c1, cA, cC)</b>		<b>Mv (c1, cA, cC)</b>	
	<b>Mv (c1, cA, cD)</b>	<b>Pos (c1, cB)</b>	<b>Mv (c1, cA, cD)</b>	
		<b>Pos (c1, cC)</b>	<b>Mv (c1, cB, cD)</b>	
	<b>cg (p1, c1, cA)</b>	<b>Pos (c1, cD)</b>	<b>Mv (c1, cB, cE)</b>	
			<b>Mv (c1, cB, cA)</b>	
			<b>Mv (c1, cC, cD)</b>	
			<b>Mv (c1, cC, cE)</b>	
			<b>Mv (c1, cC, cA)</b>	
				<b>Dcg (p1, c1, cA)</b>
				<b>Dcg (p1, c1, cB)</b>
				<b>Dcg (p1, c1, cC)</b>
				<b>Dcg (p1, c1, cD)</b>
				<b>Dcg (p1, c1, cE)</b>
				<b>Pos (p1, cE)</b>
				<b>Pos (p1, cE)</b>

- Número de pasos de planificación para conseguir el objetivo desde el estado inicial: **2. Se cumple en P[2].**
- Número de pasos de planificación para conseguir que el camión esté en E desde el estado en el que se ha conseguido por primera vez que el paquete esté en B: **0. Se cumple en P[2]**
- Si queremos estimar el número de acciones necesarias, ¿coincide este valor con el número de pasos de planificación? **No, un paso de planificación incluye varias acciones supuestas simultáneas**
- ¿Qué complejidad tienen las operaciones anteriores?: **Polinomial: |acciones| x |proposiciones|**
- ¿Cuál es la complejidad de obtener el plan relajado óptimo de menor número de acciones?: **Exponencial**

## Ejemplo Fluents: Jarras de agua. SOLUCIONES



```
(define (domain jug-pouring)
  (:requirements :typing :durative-actions :fluents)
  (:types jug)
  (:functions
    (amount ?j - jug)
    (capacity ?j - jug) )
  (:durative-action EmptyFrom1To2
    :parameters (?jug1 ?jug2 - jug)
    :duration 10
    :condition (at start (>= (- (capacity ?jug2) (amount ?jug2)) (amount ?jug1)))
    :effect (and (at end (assign (amount ?jug1) 0))
      (at end (increase (amount ?jug2) (amount ?jug1)))))
  ...
  )
```

```
(define (2jug )
  (:domain jug-pouring)
  (:objects
    jugx - jug
    jugy - jug)
  (:init
    (= (amount jugx) 0)
    (= (amount jugy) 0)
    (= (capacity jugx) 4)
    (= (capacity jugy) 3)
  )
  (:goal (= (amount jugx) 2)))
  )
```

Funciones numéricas (fluents), sobre las que se pueden establecer condiciones. *Asimilables a variables numéricas.*  
*;Precondiciones:*  $>$ ,  $\geq$ ,  $<$ ,  $\leq$ ,  $=$   
*;Efectos:* *assign*, *increase*, *decrease*, *scale up* y *scale down*

## Ejemplo Acciones Durativas- 1. Soluciones

(:durative-action load-truck ;Load 'cargo' in 'truck' with 'crane', (grúa puede simultanear cargas)

```
:parameters      (?t - truck)
                  (?l - location)
                  (?o - cargo)
                  (?c - crane)

:duration        (= ?duration 5)

:condition (and  (at start (at ?t ?l))          ; truck en ?l
              (over all (at ?t ?l))           ; truck en ?l
              (at start (at ?o ?l))          ; cargo en ?l
              (at start (empty ?c)))         ; grúa disponible

:effect   (and  (at start (not (at ?o ?l)))       ; cargo no en ?l
                  (at end   (in ?o ?t))))        ; cargo en truck
```

Si se quiere especificar que una condición  $<p>$  se mantiene en el intervalo cerrado durante la duración de una acción durativa, entonces se requieren tres condiciones:

(at start  $<p>$ ), (over all  $<p>$ ) y (at end  $<p>$ )



## Ejemplo Acciones Durativas- 2

(:durative-action load-truck ;Load 'cargo' in 'truck' with 'crane', no simultaneidad de grúa por 'cargos'

```
:parameters      (?t - truck)
                (?l - location)
                (?o - cargo)
                (?c - crane)

:duration       (= ?duration ( / (weight ?c) (powercrane))) ; duración dependiente
                           peso carga y potencia grúa

:condition (and (at start (at ?t ?l)) ; truck en ?l
                  (over all (at ?t ?l)) ; truck en ?l
                  (at start (at ?o ?l)) ; cargo en ?l
                  (at start (empty ?c)) ; grúa vacía

:effect   (and (at start (not (at ?o ?l))) ; no cargo en ?
                  (at start (not (empty ?c))) ; al inicio grúa no vacía
                  (at end (empty ?c))) ; al final grúa vacía
                  (at end (in ?o ?t))) ; cargo en truck
```

En Domain:

```
(:functions
  (weighth ?o - cargo)
  (powercrane)
  ....)
```

En Problema:

```
(init
  (= (weighth O1) 70)
  (= (powercrane) 10)
  ....)
```



## Ejemplo Acciones Durativas- 3

(:durative-action load-truck ;Load 'cargo' in 'truck' with 'crane', **no simultaneidad** de grúa por 'cargos'  
y capacidad en truck

```
:parameters      (?t - truck)
                (?l - location)
                (?o - cargo)
                (?c - crane)

:duration        (= ?duration ( / (weight ?c) (powercrane))) ; duración dependiente
                                                               peso carga y potencia grúa

:condition (and  (at start (at ?t ?l)) ; truck en ?l
                  (over all (at ?t ?l)) ; truck en ?l
                  (at start (at ?o ?l)) ; cargo en ?l
                  (at start (empty ?c)) ; grúa vacía
                  (at start (< (contenido ?t) (capacidad ?t))) ; capacidad disponible

:effect   (and  (at start (not (at ?o ?l))) ; no cargo en ?l
                  (at start (not (empty ?c))) ; al inicio grúa no vacía
                  (at end (empty ?c))) ; al final grúa vacía
                  (at end (in ?o ?t)) ; cargo en truck
                  (at start (increase (contenido ?t) 1)))) ;incremento contenido
```

En Domain:

```
(:functions
  (capacidad ?t - truck) ;número, peso, volumen, etc
  (contenido ?t - truck)
  .....)
```

En Problema:

```
(init
  (= (capacidad truck1) 10)
  (= (contenido truck1) 0)
  .....)
```