Tema 6: Planificación

Rosa María Maza Quiroga – rosammq@uma.es

Departamento de Lenguajes y Ciencias de la Computación

Universidad de Málaga

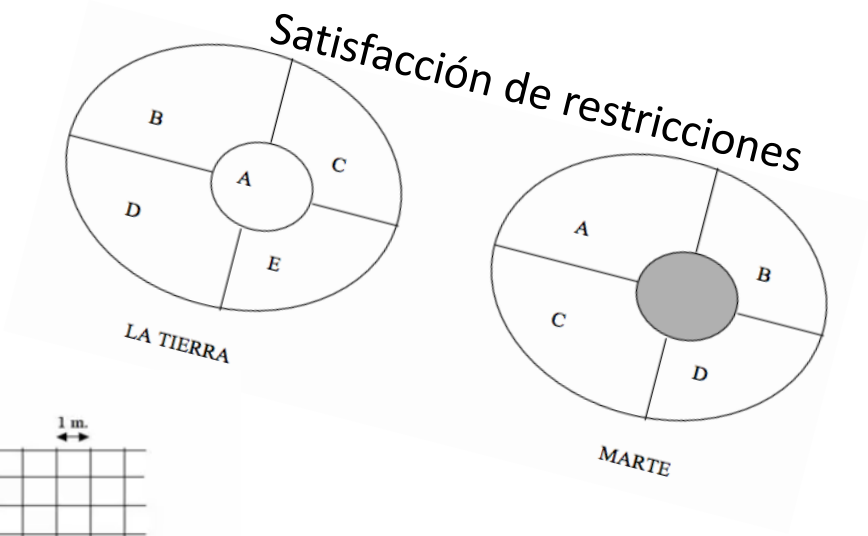
Contenido

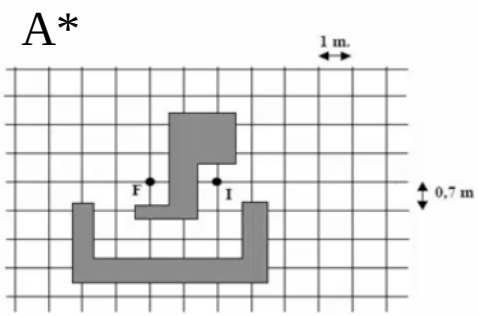
1. Introducción
2. Fundamentos
3. Algoritmo GRAPHPLAN
1. Conclusiones

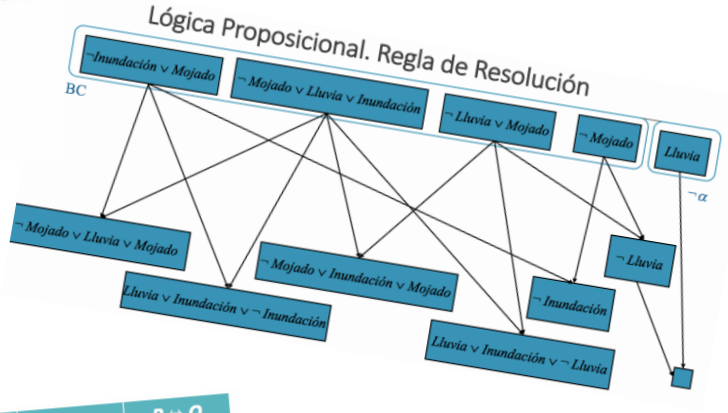
1. Introducción

Motivación

Generalidades

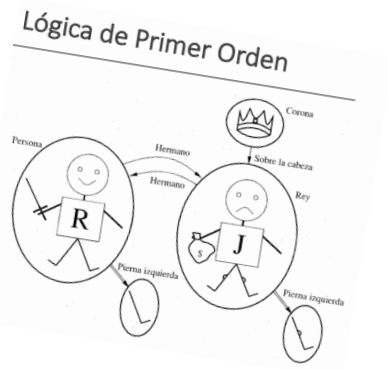






P	Q	P∧Q	P∨Q	P⇒Q	P⇔Q
V	V	V	V	V	V
V	F	F	V	F	F
F	V	F	V	V	F
F	F	F	F	V	V

P	¬P
V	F
F	V

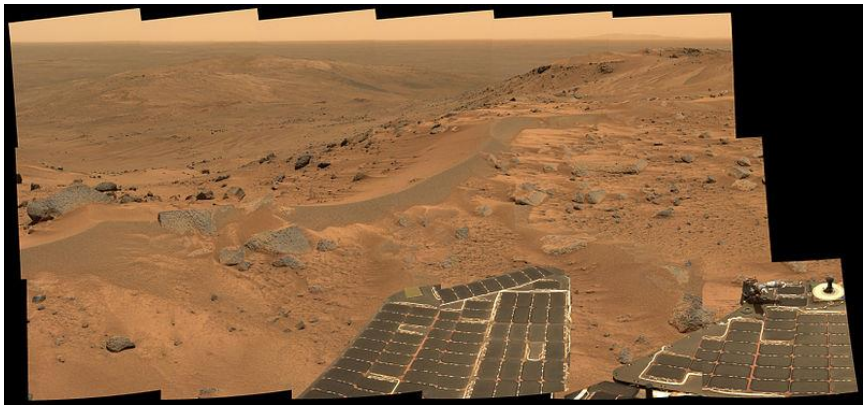


1. Introducción
Motivación

- Los edificios modernos tienen varios ascensores:
  - Los rascacielos tienen muchos grupos de ascensores, y se tarda varios minutos en llegar a la última planta.
- Sin una planificación adecuada, el viaje en momentos de máximo tráfico podría ser una pesadilla.

1. Introducción
Motivación

- En enero de 2004, dos vehículos robóticos de la NASA, llamados *Spirit* y *Opportunity*, llegaron a Marte con éxito
- Cada día había que generar un plan nuevo para cada vehículo
  - Cada plan tenía que satisfacer reglas de seguridad complejas, al mismo tiempo que conseguir tantos resultados científicos como fuera posible.
  - Se supuso que los vehículos no sobrevivirían mucho más de 90 días.



Panorámica "tipo postal" tomada por el *Spirit*: una meseta azotada por el viento, con rocas esparcidas, pequeños afloramientos de minerales, y dunas de arena

1. Introducción
Generalidades

- Modelar entornos complejos: la Lógica proposicional es demasiado simple y la Lógica de Primer Orden es demasiado compleja. Se crea la Planificación: nuevo lenguaje **PDDL**, algoritmos como **GRAPHPLAN**.

- La **planificación**: el desarrollo de un plan de acción para conseguir ciertos objetivos, es una parte crítica de la Inteligencia Artificial.

- Un agente puede usar la estructura de un problema para construir planes de acción complejos.

- Representación para los problemas de planificación (problemas grandes).

2. Fundamentos

- Representación
- Formalismo lógico
- Estados
- Acciones

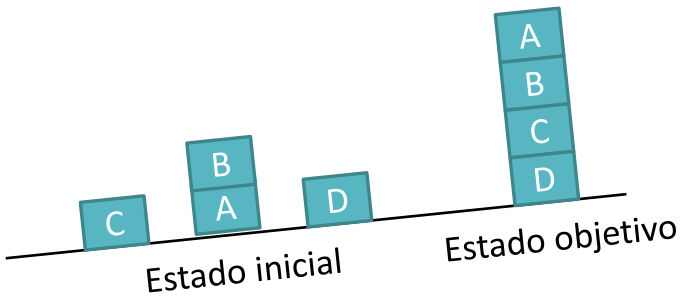
$Init(At(C_1,SFO) \land At(C_2,JFK) \land At(P_1,SFO) \land At(P_2,JFK) \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \land Airport(SFO) \land Airport(JFK))$

$Goal(At(C_1,JFK) \land At(C_2,SFO))$

$Action(Load(c,p,a),$
PRECOND: $At(c,a) \land At(p,a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $\neg At(c,a) \land In(c,p))$

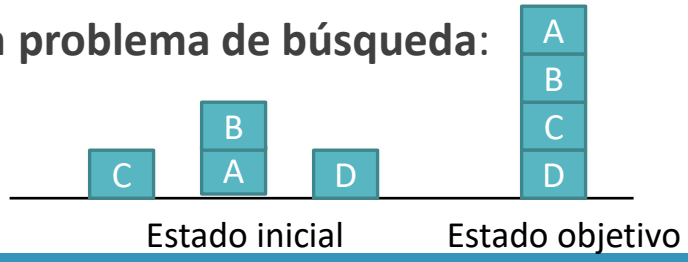
$Action(UnLoad(c,p,a),$
PRECOND: $In(c,p) \land At(p,a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $At(c,a) \land \neg In(c,p))$

$Action(Fly(p,from,to),$
PRECOND: $At(p,from) \land Plane(p) \land Airport(from) \land Airport(to)$
EFFECT: $\neg At(p,from) \land At(p,to))$



2. Fundamentos
Representación de problemas de planificación

- **Representación factorizada** de los estados: representaremos un estado del mundo mediante una colección de variables.
- Lenguaje de Definición de Dominios de Planificación (Planning Domain Definition Language, **PDDL**):
  - Basado en STRIPS. Cronología: STRIPS-ADL-PDDL.
- Trabajaremos con **entornos de Planificación clásica**: completamente observables, deterministas, finitos, estáticos (los cambios suceden sólo cuando los agentes actúan) y discretos (en tiempo, acciones, objetos y efectos).
- La inferencia supone unas restricciones:
  - Semántica de **Base de Datos**:
    - **Nombres únicos**: un objeto tiene un solo símbolo.
    - **Mundo cerrado**: todo lo que no se diga a la BC se supone falso.
  - **Clausura**: sólo existen los objetos que son definidos.
- PDDL describe cuatro componentes para definir un **problema de búsqueda**:
  - estado inicial.
  - acciones que están disponibles en un estado.
  - resultado de aplicar una acción.
  - comprobación del objetivo.

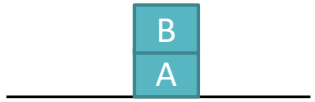


2. Fundamentos
Formalismo lógico: lenguaje PDDL

- Un lenguaje para representar problemas de planificación:
  - Constantes: objetos del mundo (en mayúsculas y cursiva).
  - Variables para representar cualquier objeto (en minúsculas y cursiva).
  - Símbolos de predicados: expresan propiedades de los objetos (cursiva) .
  - Símbolos de acciones: representan operadores (cursiva).
  - No hay funciones, porque pueden hacer que los algoritmos caigan en bucles infinitos.
- Terminología:
  - Átomos: por definición, fórmulas de la forma $P(o_1, \ldots, o_n)$, donde P es un símbolo de predicado y cada o_i es una constante o una variable. Ej: *Bloque*(*A*), *Sobre*(*x*,*y*)
  - Literales: átomos o negación de átomos (¬ , -).
  - Átomos y literales cerrados/simples/planos: sin variables.
- Estados: conjunción de átomos cerrados.
  - Hipótesis del mundo cerrado: los átomos que no se mencionan se suponen falsos.

❖ Ej. Mundo de bloques.

Bloque(*A*) ∧ *Bloque*(*B*) ∧ *Sobre*(*B*,*A*)



Estado inicial

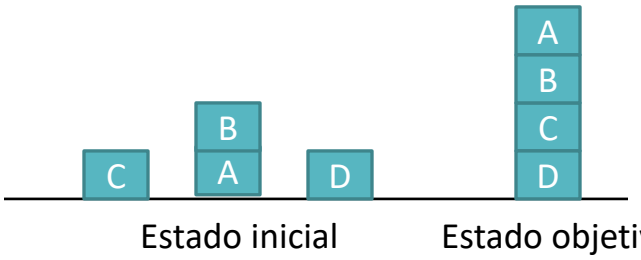
2. Fundamentos
Estados

- Flujo=Fluente: afirmaciones del mundo que pueden cambiar con el tiempo.
- Cada estado se representa como una **conjunción de flujos**:
  - Son literales simples. Dentro de un flujo no puede haber disyunciones, implicaciones etc. Son simplemente relaciones y propiedades.
  - No se usa la negación (al ser el mundo cerrado, todo lo que no sea, es falso).
  - No hay variables, tienen que ser objetos concretos (una constante por cada objeto distinto).
- Estado: conjunción de literales positivos.

❖ Ej. Mundo de los bloques: colocar varios bloques formando una torre.

Elementos que intervienen:

- Una mesa.
- Una serie de bloques cúbicos.
- Sólo se puede mover un bloque cada vez.
- Un bloque puede estar sobre la mesa o sobre otro bloque en orden alfabético.



2. Fundamentos
Estados

- Flujo=Fluente: afirmaciones del mundo que pueden cambiar con el tiempo.
- Cada estado se representa como una **conjunción de flujos**:
  - Son literales simples. Dentro de un flujo no puede haber disyunciones, implicaciones etc. Son simplemente relaciones y propiedades.
  - No se usa la negación (al ser el mundo cerrado, todo lo que no sea, es falso).
  - No hay variables, tienen que ser objetos concretos (una constante por cada objeto distinto).
- Estado: conjunción de literales positivos.

❖ Ej. Mundo de los bloques: colocar varios bloques formando una torre.

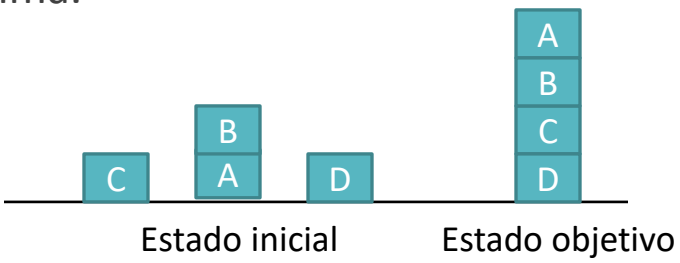
Predicados:

Despejado(*x*) el bloque *x* no tiene un bloque encima.

Sobre(*x,y*) el objeto *x* está encima del objeto *y*.
Atención, *y* puede ser bloque o mesa.

Bloque(*x*) indica que x es un bloque

Mesa(*x*) indica que x es una mesa



2. Fundamentos
Estados

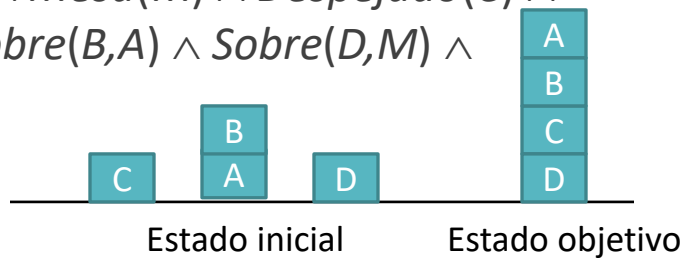
- Flujo=Fluente: afirmaciones del mundo que pueden cambiar con el tiempo.
- Cada estado se representa como una **conjunción de flujos**:
  - Son literales simples. Dentro de un flujo no puede haber disyunciones, implicaciones etc. Son simplemente relaciones y propiedades.
  - No se usa la negación (al ser el mundo cerrado, todo lo que no sea, es falso).
  - No hay variables, tienen que ser objetos concretos (una constante por cada objeto distinto).
- Estado: conjunción de literales positivos.

❖ Ej. Mundo de los bloques: colocar varios bloques formando una torre.

Descripción del **estado inicial**:

$Bloque(A) \wedge Bloque(B) \wedge Bloque(C) \wedge Bloque(D) \wedge Mesa(M) \wedge Despejado(C) \wedge Despejado(B) \wedge Despejado(D) \wedge Sobre(C,M) \wedge Sobre(B,A) \wedge Sobre(D,M) \wedge Sobre(A,M)$

☞ Ejercicio 3. Mundo de bloques. Finalice clase.



2. Fundamentos
Acciones

- **Esquema de acción**:
  - Nombre, seguido de las variables.
  - Precondición: conjunción de literales que debe cumplirse para realizar la acción (t).
  - Efectos: conjunción de literales que indica cómo cambia el mundo cuando se cumple la acción (t+1):
    - Añadir: literales positivos.
    - Borrar: literales negativos.

- **Mundo cerrado**: cualquier fluente que no sea mencionado es falso.

❖ Ej. Esquema de acción

Action(*Fly*(*p*,*from*,*to*),

PRECOND: $At(p,from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$

EFFECT: $\neg At(p,from) \wedge At(p,to)$)

2. Fundamentos
Acción aplicable

- Una **acción a es aplicable** en el **estado *s*** si *s* **satisface las precondiciones**:
  - Precondición **unifica** con los literales de la acción.

❖ Ejemplo: Consideremos un problema de transporte aéreo de mercancías que consiste en cargar y descargar mercancías entre aviones que vuelan entre diferentes destinos.

Acciones: *Load*, *Unload*, y *Fly*

Las acciones afectan a dos **predicados**:
- *In*(*c*,*p*) significa que la mercancía *c* está en el avión *p.*
- *At*(*x*,*a*) significa que el objeto *x* está en el aeropuerto *a.*

Action(*Load*(*c*,*p*,*a*),
PRECOND: $At(c,a) \wedge At(p,a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $\neg At(c,a) \wedge In(c,p)$)

Action(*UnLoad*(*c*,*p*,*a*),
PRECOND: $In(c,p) \wedge At(p,a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $At(c,a) \wedge \neg In(c,p)$)

Action(*Fly*(*p*,*from*,*to*),
PRECOND: $At(p,from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
EFFECT: $\neg At(p,from) \wedge At(p,to)$)

2. Fundamentos
Dominio y problema específico

- Un **dominio** de planificación viene dado por un conjunto de esquemas de acción.
- Para definir un problema específico dentro del dominio añadimos un **estado inicial** y un **objetivo**:
  - El estado inicial es una conjunción de literales positivos.
  - El objetivo es como una precondición. Puede definirse con objetos concretos o variables [supone que todas las variables están cuantificadas existencialmente (existe algún objeto para cada variable)].
  - La solución es una secuencia de acciones que ejecutada en el estado inicial da como resultado el estado final que satisface el objetivo.

❖ Ejemplo (continuación):

Objetos: $C_1, C_2, P_1, P_2, SFO, JFK.$

$Init(At(C_1,SFO) \land At(C_2,JFK) \land At(P_1,SFO) \land At(P_2,JFK) \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \land Airport(SFO) \land Airport(JFK))$

$Goal(At(C_1,JFK) \land At(C_2,SFO))$

2. Fundamentos
Ejemplo PDDL

- Ejemplo: Consideremos un problema de transporte aéreo de mercancías que consiste en cargar y descargar mercancías entre aviones que vuelan entre diferentes destinos.

Init($At(C_1,SFO) \land At(C_2,JFK) \land At(P_1,SFO) \land At(P_2,JFK) \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \land Airport(SFO) \land Airport(JFK)$)

Goal($At(C_1,JFK) \land At(C_2,SFO)$)

Action(*Load*(*c*,*p*,*a*),
PRECOND: $At(c,a) \land At(p,a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $\neg At(c,a) \land In(c,p)$)

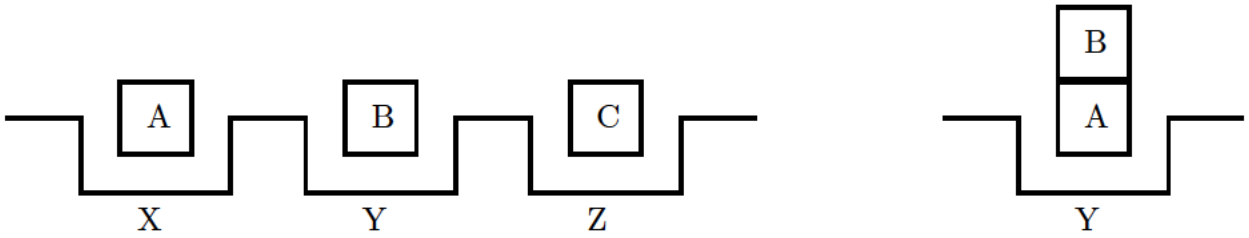
Action(*UnLoad*(*c*,*p*,*a*),
PRECOND: $In(c,p) \land At(p,a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $At(c,a) \land \neg In(c,p)$)

Action(*Fly*(*p*,*from*,*to*),
PRECOND: $At(p,from) \land Plane(p) \land Airport(from) \land Airport(to)$
EFFECT: $\neg At(p,from) \land At(p,to)$)

- El siguiente plan es una posible solución:
  - [$Load(C_1,P_1,SFO)$, $Fly(P_1,SFO,JFK)$, $UnLoad(C_1,P_1,JFK)$, $Load(C_2,P_2,JFK)$, $Fly(P_2,JFK,SFO)$, $UnLoad(C_2,P_2,SFO)$]

Ejercicio 6.7. Mundo de los bloques. Enunciado

Ejercicio 6.7 El problema del mundo de los bloques simplificado consiste en un robot que tiene la tarea de apilar varios bloques denominados a,b y c de una determinada manera, pero estos bloques sólo pueden situarse en determinadas posiciones fijas llamadas X,Y,Z o bien unos encima de otros. La disposición inicial y final de los bloques es la siguiente:

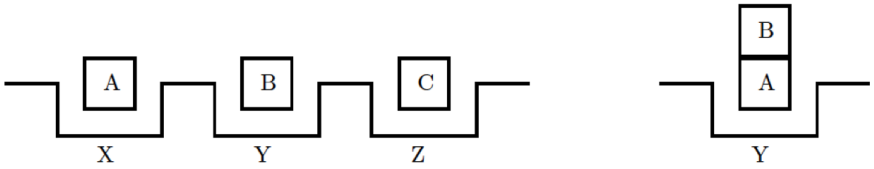


a) Describir el dominio del mundo de los bloques simplificado especificando las constantes, los predicados y las acciones (pista: sólo es necesaria una acción para resolver óptimamente el problema).

b) Especifica el estado inicial y objetivo descritos anteriormente con tu formulación PDDL del problema.

Ejercicio 6.7. Mundo de los bloques. Solución

a) Describir el dominio del mundo de los bloques simplificado especificando las constantes, los predicados y las acciones (pista: solo es necesaria una acción para resolver óptimamente el problema).



Constantes: Bloques: *A, B, C ;* Posiciones: *X, Y, Z*

Predicados:

Libre(*x*): el bloque *x* o la posición *x* no tienen a nadie encima.

En(*x,y*): el bloque *x* esta directamente sobre el bloque *y* o la posición *y*.

Acciones:

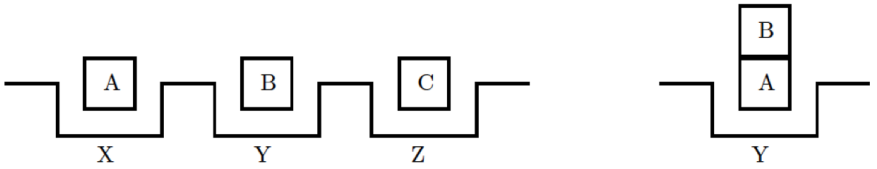
Acción(*Mover*(*bloq,orig,dest*),

PRECONDICION: *En*(*bloq,orig*) ^ *Libre*(*dest*) ^ *Libre*(*bloque*)

EFECTO: ¬ *En*(*bloq,orig*) ^ ¬ *Libre*(*dest*) ^ *En*(*bloq,dest*) ^ *Libre*(*orig*)

Ejercicio 6.7. Mundo de los bloques. Solución

b) Especifica el estado inicial y objetivo descritos anteriormente con tu formulación PDDL del problema.



Estado inicial:

Bloque(*A*) ^ *Bloque*(*B*) ^ *Bloque*(*C*) ^ *Posición*(*X*) ^ *Posición*(*Y*) ^ *Posición*(*Z*) ^ *En*(*A,X*) ^ *En*(*B,Y*) ^ *En*(*C,Z*) ^ *Libre*(*A*) ^ *Libre*(*B*) ^ *Libre*(*C*)

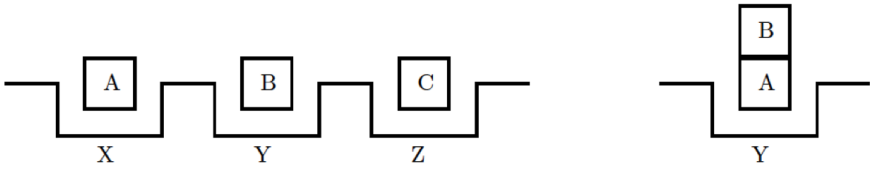
Estado objetivo:

En(*B,A*) ^ *En*(*A,Y*)

Ejercicio 6.7. Mundo de los bloques. Solución

Inicio(*Bloque*(*A*) ^ *Bloque*(*B*) ^ *Bloque*(*C*) ^ *Posición*(*X*) ^ *Posición*(*Y*) ^ *Posición*(*Z*) ^ *En*(*A,X*) ^ *En*(*B,Y*) ^ *En*(*C,Z*) ^ *Libre*(*A*) ^ *Libre*(*B*) ^ *Libre*(*C*))

Objetivo(*En*(*B,A*) ^ *En*(*A,Y*))



Acción(*Mover*(*bloq,orig,dest*),

PRECONDICION: *En*(*bloq,orig*) ^ *Libre*(*dest*) ^ *Libre*(*bloque*)

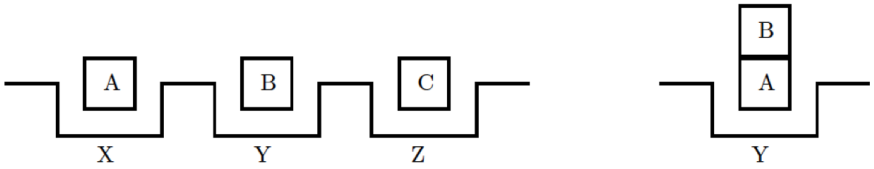
EFECTO: ¬ *En*(*bloq,orig*) ^ ¬ *Libre*(*dest*) ^ *En*(*bloq,dest*) ^ *Libre*(*orig*)

- Plan, posible solución:

Ejercicio 6.7. Mundo de los bloques. Solución

Inicio(*Bloque*(*A*) ^ *Bloque*(*B*) ^ *Bloque*(*C*) ^ *Posición*(*X*) ^ *Posición*(*Y*) ^ *Posición*(*Z*) ^ *En*(*A,X*) ^ *En*(*B,Y*) ^ *En*(*C,Z*) ^ *Libre*(*A*) ^ *Libre*(*B*) ^ *Libre*(*C*))

Objetivo(*En*(*B,A*) ^ *En*(*A,Y*))



Acción(*Mover*(*bloq,orig,dest*),

PRECONDICION: *En*(*bloq,orig*) ^ *Libre*(*dest*) ^ *Libre*(*bloque*)

EFECTO: ¬ *En*(*bloq,orig*) ^ ¬ *Libre*(*dest*) ^ *En*(*bloq,dest*) ^ *Libre*(*orig*)

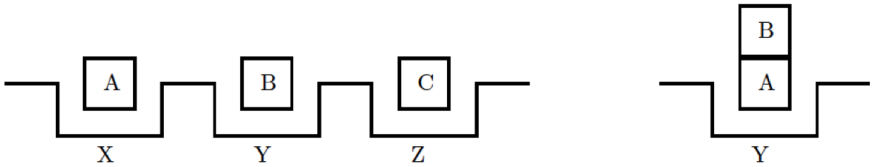
- Plan, posible solución:

[*Mover*(*B,Y,C*) ^ *Mover*(*A,X,Y*) ^ *Mover*(*B,C,A*)]

Ejercicio 6.7. Mundo de los bloques. Solución

Inicio(*Bloque*(*A*) ^ *Bloque*(*B*) ^ *Bloque*(*C*) ^ *Posición*(*X*) ^ *Posición*(*Y*) ^ *Posición*(*Z*) ^ *En*(*A,X*) ^ *En*(*B,Y*) ^ *En*(*C,Z*) ^ *Libre*(*A*) ^ *Libre*(*B*) ^ *Libre*(*C*))

Objetivo(*En*(*B,A*) ^ *En*(*A,Y*))



Acción(*Mover*(*bloq,orig,dest*),
PRECONDICION: *En*(*bloq,orig*) ^ *Libre*(*dest*) ^ *Libre*(*bloque*)
EFECTO: ¬ *En*(*bloq,orig*) ^ ¬ *Libre*(*dest*) ^ *En*(*bloq,dest*) ^ *Libre*(*orig*)

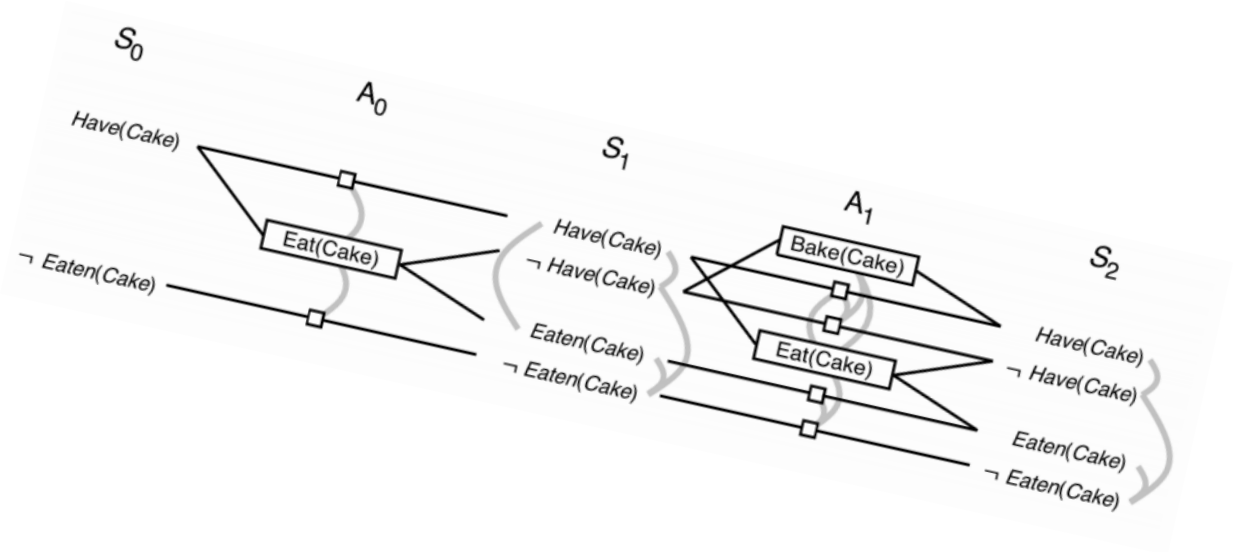
- Plan, posible solución:

[*Mover*(*B,Y,C*) ^ *Mover*(*A,X,Y*) ^ *Mover*(*B,C,A*)]

Acción(*Mover*(*B,Y,C*),
PRECONDICION: *En*(*B, Y*) ^ *Libre*(*C*) ^ *Libre*(*B*)
EFECTO: ¬ *En*(*B, Y*) ^ ¬ *Libre*(*C*) ^ *En*(*B, C*) ^ *Libre*(*Y*)

☞ Ejercicios de la relación.

3.- Algoritmo GRAPHPLAN

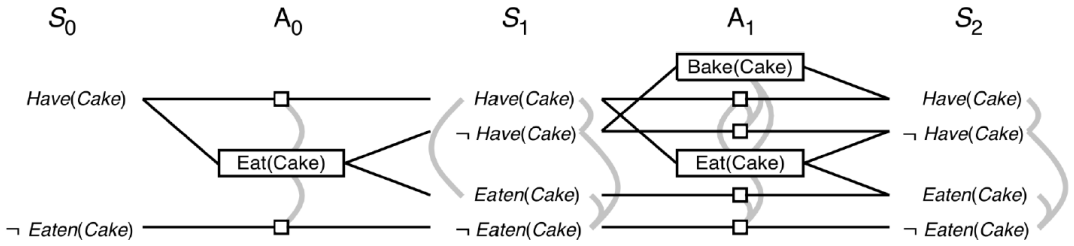


3. Algoritmo GRAPHPLAN

- Durante años se desarrollaron múltiples algoritmos y heurísticos independientes del dominio.
- **GRAPHPLAN**:
  - Algoritmo que mostró mejoras notables.
  - se basa en el desarrollo de un nuevo tipo de grafos, los **grafos de planificación**.
  - Sencillo.
  - Base de los algoritmos más eficientes.
  - A medio camino entre la búsqueda en espacio de estado y la búsqueda en espacio de planes:
    - Búsqueda sobre el espacio de estados: los estados de la búsqueda son situaciones concretas del mundo y los operadores son acciones que cambiaban esas situaciones.
    - Búsqueda sobre el espacio de planes: los estados de la búsqueda son planes (secuencias de acciones) y los operadores cambian estos planes (en general, incompletos) hasta conseguir un plan completo que resuelva el problema.

3. Algoritmo GRAPHPLAN
Grafo Planificación

- Recoge información del problema para facilitar su posterior resolución.
- **Grafo dirigido**: entre dos nodos hay un arco y ese arco tiene un sentido.
- **Grafo por niveles**:
  - S_i son los literales que podrían ser ciertos en el paso i.
  - A_i acciones que podrían ver sus precondiciones satisfechas en el paso i.
- Funciona sobre problemas de planificación proposicional (no hay variables):
  - Definición de acciones: una acción por cada combinación de objetos posible.



3. Algoritmo GRAPHPLAN
Grafo Planificación. Ejemplo pastel

Init(Have(Cake))

Goal(Have(Cake) $\wedge$ Eaten(Cake))

Action(Eat(Cake),

PRECOND: Have(Cake)

EFFECT: ¬Have(Cake) $\wedge$ Eaten(Cake))

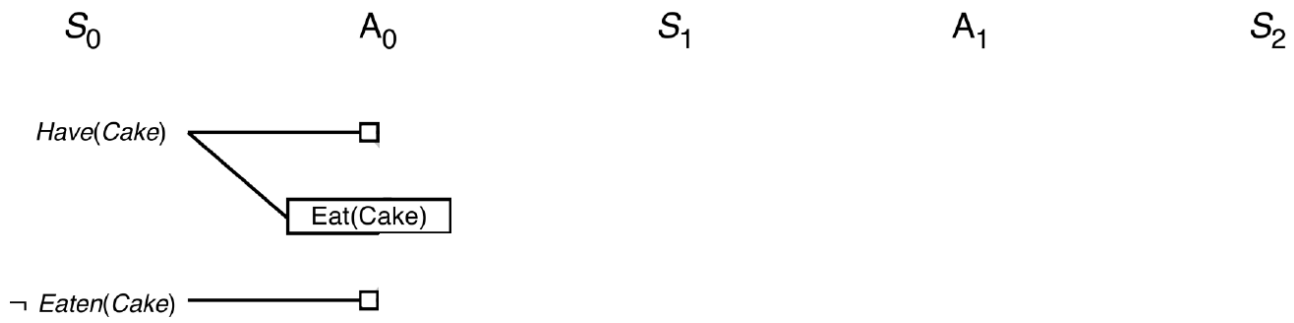
Action(Bake(Cake),

PRECOND: ¬Have(Cake)

EFFECT: Have(Cake))

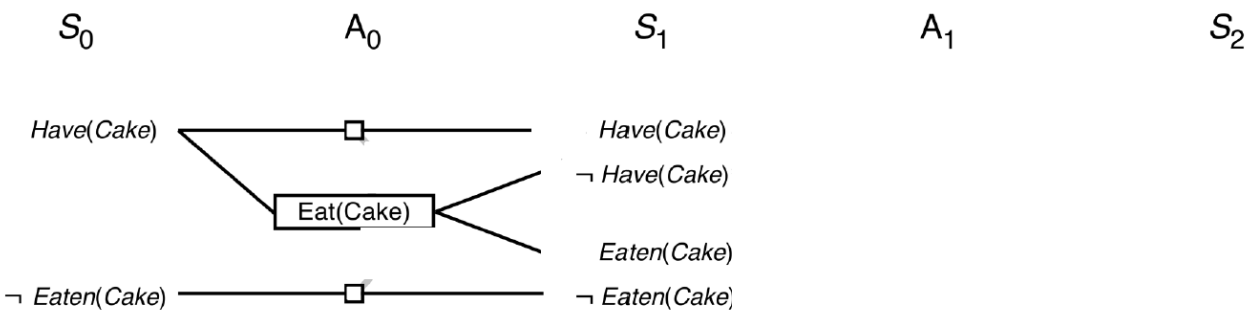
3. Algoritmo GRAPHPLAN
Grafo Planificación. Nivel 0

- S_0 tendrá un **nodo** por cada literal cierto en la situación inicial.
- A_0 tendrá:
  - Un **nodo** por cada acción cuyas precondiciones se puedan satisfacer en S0.
  - Un **nodo** por cada literal, que actúa como acción de persistencia: el literal seguirá siendo cierto si ninguna acción lo niega.
- Un **arco** entre cada acción y sus precondiciones.



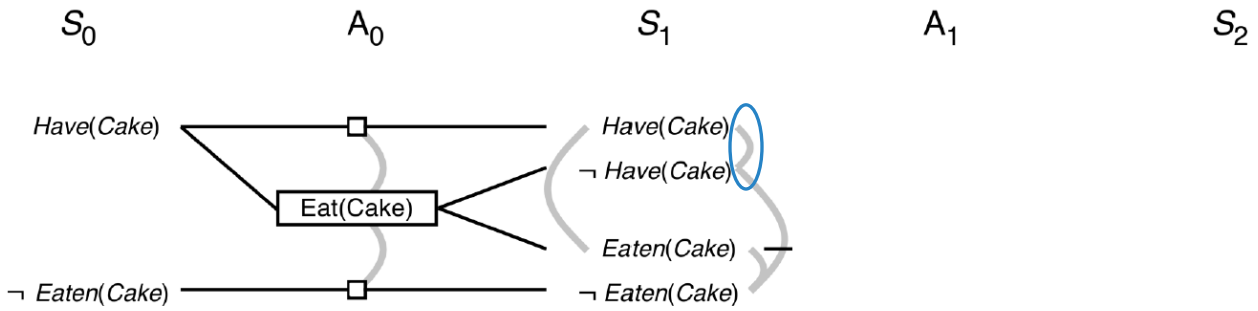
3. Algoritmo GRAPHPLAN
Grafo Planificación. Nivel 1

- S_1 tendrá un nodo por cada literal que aparezca en los efectos de las acciones de A_0.
- Un arco entre cada acción y sus efectos.



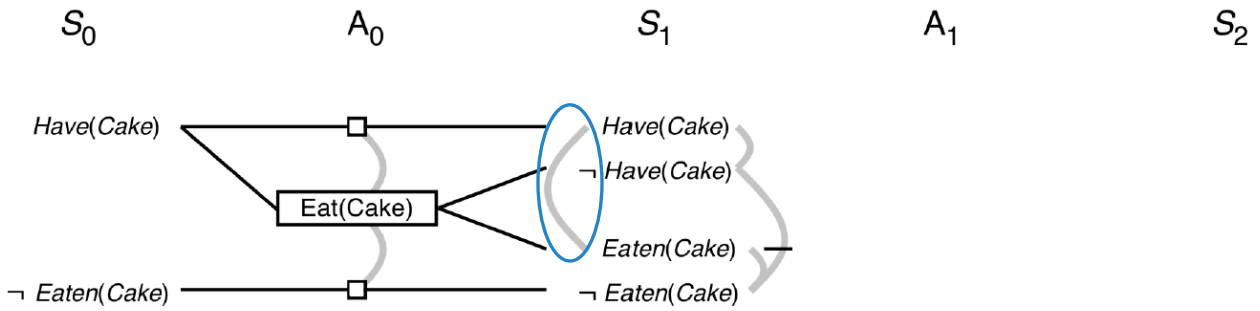
3. Algoritmo GRAPHPLAN
Grafo Planificación. Exclusión mutua

- **Enlace exclusión mutua entre literales de un mismo nivel que nunca podrían aparecer juntos**:
  - Un literal y su negación.
  - Soporte inconsistente: par de acciones que pudiera conseguir los dos literales es mutuamente excluyente.
- Enlace exclusión mutua entre acciones de un mismo nivel que nunca podrían ocurrir al mismo tiempo:
  - Efecto inconsistente: una acción niega los efectos de la otra (efectos contrarios).
  - Interferencia: un efecto de una negación niega la precondición de otra.
  - Necesidades que compiten: la precondición de una acción es mutuamente excluyente con la de la otra.



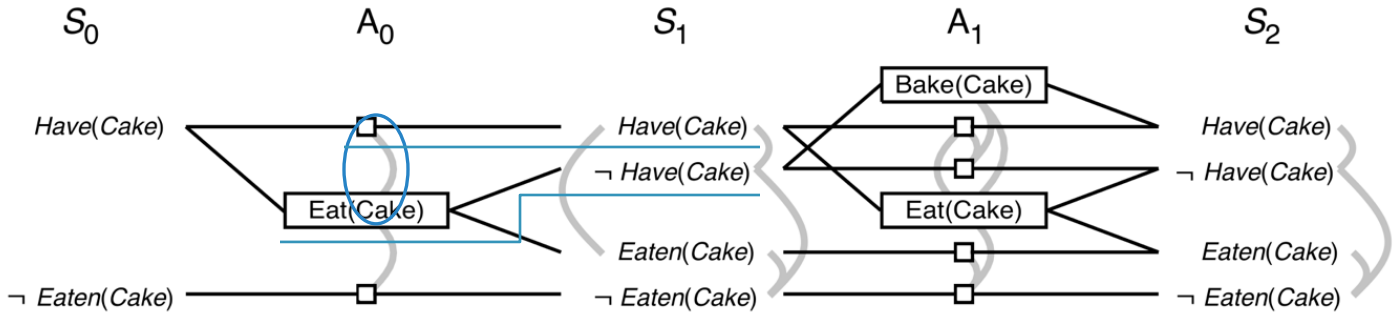
3. Algoritmo GRAPHPLAN
Grafo Planificación. Exclusión mutua

- **Enlace exclusión mutua entre literales de un mismo nivel que nunca podrían aparecer juntos**:
  - Un literal y su negación.
  - Soporte inconsistente: par de acciones que pudiera conseguir los dos literales es mutuamente excluyente.
- Enlace exclusión mutua entre acciones de un mismo nivel que nunca podrían ocurrir al mismo tiempo:
  - Efecto inconsistente: una acción niega los efectos de la otra (efectos contrarios).
  - Interferencia: un efecto de una negación niega la precondición de otra.
  - Necesidades que compiten: la precondición de una acción es mutuamente excluyente con la de la otra.



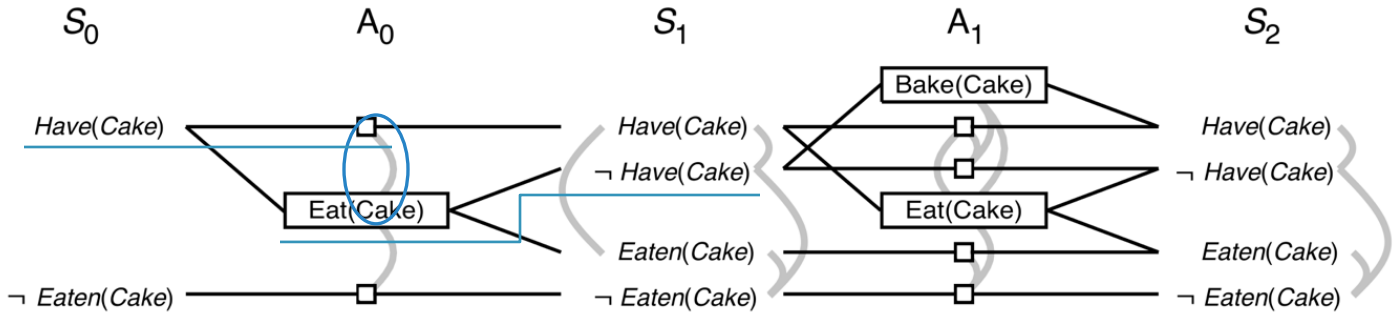
3. Algoritmo GRAPHPLAN
Grafo Planificación. Exclusión mutua

- Enlace exclusión mutua entre literales de un mismo nivel que nunca podrían aparecer juntos:
  - Un literal y su negación.
  - Soporte inconsistente: par de acciones que pudiera conseguir los dos literales es mutuamente excluyente.
- **Enlace exclusión mutua entre acciones de un mismo nivel que nunca podrían ocurrir al mismo tiempo**:
  - Efecto inconsistente: una acción niega los efectos de la otra (efectos contrarios).
  - Interferencia: un efecto de una acción niega la precondición de otra.
  - Necesidades que compiten: la precondición de una acción es mutuamente excluyente con la de la otra.



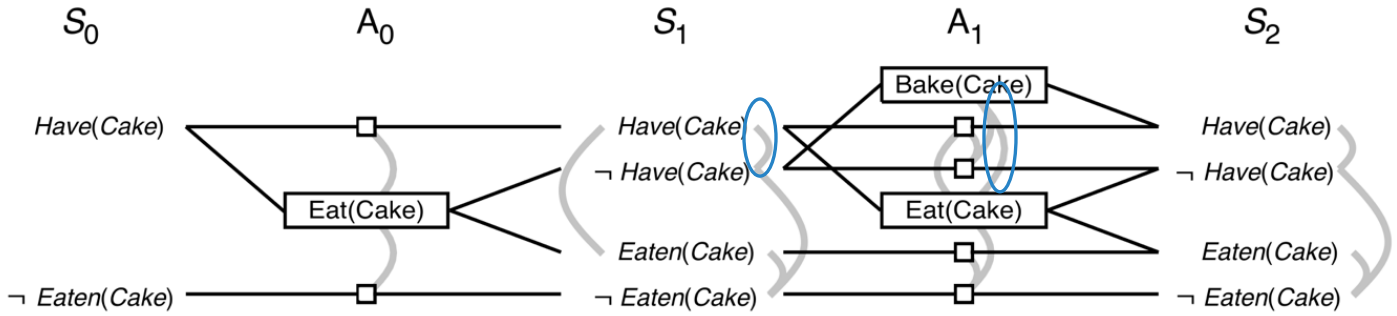
3. Algoritmo GRAPHPLAN
Grafo Planificación. Exclusión mutua

- Enlace exclusión mutua entre literales de un mismo nivel que nunca podrían aparecer juntos:
  - Un literal y su negación.
  - Soporte inconsistente: par de acciones que pudiera conseguir los dos literales es mutuamente excluyente.
- **Enlace exclusión mutua entre acciones de un mismo nivel que nunca podrían ocurrir al mismo tiempo**:
  - Efecto inconsistente: una acción niega los efectos de la otra (efectos contrarios).
  - Interferencia: un efecto de una acción niega la precondición de otra.
  - Necesidades que compiten: la precondición de una acción es mutuamente excluyente con la de la otra.



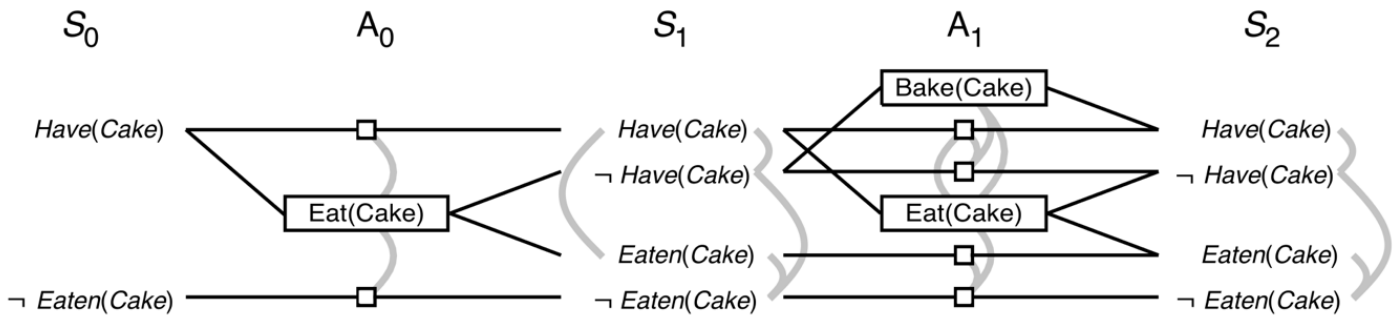
3. Algoritmo GRAPHPLAN
Grafo Planificación. Exclusión mutua

- Enlace exclusión mutua entre literales de un mismo nivel que nunca podrían aparecer juntos:
  - Un literal y su negación.
  - Soporte inconsistente: par de acciones que pudiera conseguir los dos literales es mutuamente excluyente.
- **Enlace exclusión mutua entre acciones de un mismo nivel que nunca podrían ocurrir al mismo tiempo**:
  - Efecto inconsistente: una acción niega los efectos de la otra (efectos contrarios).
  - Interferencia: un efecto de una acción niega la precondición de otra.
  - Necesidades que compiten: la precondición de una acción es mutuamente excluyente con la de la otra.



3. Algoritmo GRAPHPLAN
Grafo Planificación

- Si dos literales son *mutex* en S_i sus acciones de persistencia son *mutex* en A_i y generalmente esto último no se muestra gráficamente.
- Los grafos de planificación se puede utilizar:
  - Como heurística de algoritmos de planificación clasica.
  - Para crear un plan de acción a partir del algoritmo GRAPHPLAN.
- Comparativa: S_i representa estados múltiples como una búsqueda de regresión en el espacio de estados y los enlaces *mutex* son restricciones que definen el conjunto de posibles estados.



3. Algoritmo GRAPHPLAN
Algoritmo

- El algoritmo GRAPHPLAN trabaja de forma progresiva sobre el número de niveles.
- Se intenta encontrar un plan en el nivel i, en principio en el nivel 1.
- Si no se encontró plan en el nivel i, se genera el grafo hasta i+1 y se prueba de nuevo.
- Existe una **condición de terminación** para detectar si el problema no tiene solución posible:
  - El grafo de planificación puede proseguir hasta que dos niveles son iguales:
  - Falta algún objetivo o dos objetivos son *mutex*: GRAPHPLAN se detiene y devuelve el error.
  - Abarga todos los objetivos: GRAPHPLAN se detiene con éxito.

4. Conclusiones

- Los **sistemas de planificación** son algoritmos de resolución de problemas que operan sobre representaciones de estados y acciones.
- El lenguaje **PDDL** describe los estados inicial y objetivo como conjunciones de literales, y las acciones en términos de sus precondiciones y efectos.
- Los **problemas de planificación** se pueden ver como un caso particular de los **problemas de búsqueda**.
- Se puede abordar un problema de planificación mediante **algoritmos heurísticos de búsqueda generales** (búsqueda en profundidad, ávidos, A*) o mediante **algoritmos específicos para planificación** (GRAPHPLAN es el que se explica en este tema)
- Como encontrar buenas funciones heurísticas para planificación es difícil, en la práctica se suelen usar algoritmos específicos para planificación.
- Los **grafos de planificación** se pueden usar como heurística de algoritmos clásicos de planificación o con el algoritmos GRAPHPLAN para determinar el plan de acción.

4. Conclusión
Epílogo

La NASA desarrolló un programa de planificación automática para los vehículos robóticos de Marte

- Cuando se empleó en un entorno intenso y con plazos reducidos, consiguió un incremento del 10%-40% en los resultados científicos, en comparación con la operación sin ayuda de la IA
- *Spirit* funcionó hasta que quedó atascado en una trampa de arena en 2009, y *Opportunity* dejó de dar señal en 2018.

Los fabricantes de ascensores incorporan planificadores automáticos en sus productos

- Disminuyen la congestión y el tiempo de viaje, a la vez que ahorran energía

Gracias, Rosa ☺