

Tema 6: Planificación

José Luis Ruiz Reina
Francisco J. Martín Mateos
Miguel A. Gutiérrez Naranjo

Departamento de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

Inteligencia Artificial

Índice

Representación de problemas como espacios de estados

Técnicas básicas de búsqueda en espacios de estados

Búsqueda informada mediante técnicas heurísticas

Introducción a la planificación

El formalismo PDDL

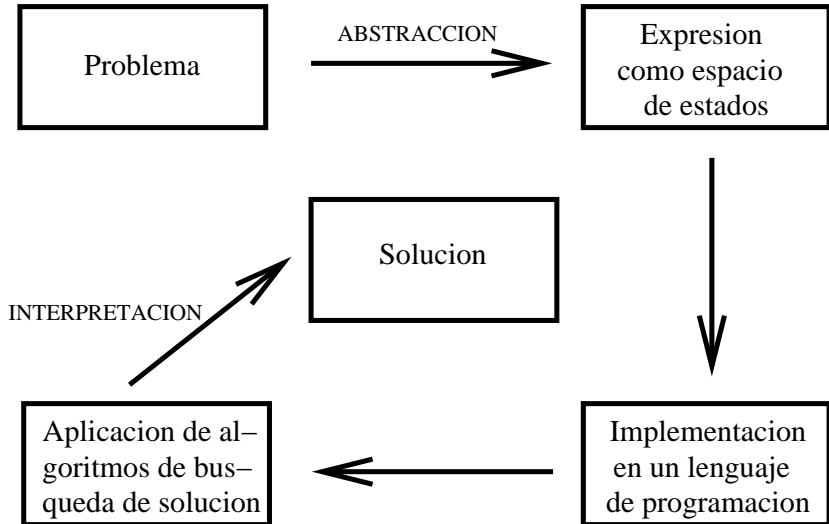
Búsqueda hacia adelante

Búsqueda hacia atrás

Heurísticas para planificación

Planificación de orden parcial: POP

Método de solución de problemas



Definición de un problema como espacio de estados

- Paso previo a la búsqueda de soluciones de un problema:
 - Especificación del problema
- Elementos del problema:
 - ¿Cuál la situación *inicial* desde la que se parte?
 - ¿Cuál es el *objetivo final*?
 - ¿Cómo describir las diferentes situaciones o *estados* por los que podríamos pasar?
 - ¿Qué pasos elementales o *acciones* para cambiar de estado y cómo actúan?
- Especificar un problema como *espacio de estados* consiste en describir de manera clara cada uno de estos componentes
 - Ventaja: procedimientos generales de búsqueda de soluciones
 - *Independientes* del problema

Planteamiento del problema del 8-puzle

- Un tablero cuadrado (3x3) en el que hay situados 8 bloques cuadrados numerados (con lo cual se deja un hueco del tamaño de un bloque). Un bloque adyacente al hueco puede deslizarse hacia él. El juego consiste en transformar una posición inicial en la posición final mediante el deslizamiento de los bloques. En particular, consideramos el estado inicial y final siguientes:

| | | |
|---|---|---|
| 2 | 8 | 3 |
| 1 | 6 | 4 |
| 7 | | 5 |

Estado inicial

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 8 | | 4 |
| 7 | 6 | 5 |

Estado final

Representación de estados

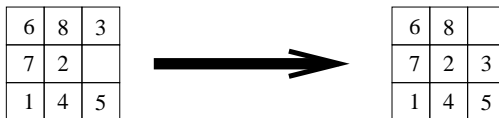
- Estado: descripción de una posible situación en el problema
 - Abstracción de propiedades
- Importancia de una buena representación de los estados
 - Sólo considerar información relevante para el problema
 - La representación escogida influye en el número de estados y éste en los procedimientos de búsqueda de soluciones
- Ejemplo: 8-puzzle: Elementos de la representación:
 - relevante: localización de cada bloque y del hueco;
 - irrelevante: tipo de material de los bloques, colores de los bloques,...

Acciones

- Acciones:
 - Representan un conjunto finito de operadores básicos que transforman unos estados en otros
- Elementos que describen una acción
 - Aplicabilidad: precondition y postcondición
 - Estado resultante de la aplicación de una acción (aplicable) a un estado
- Criterio para elegir acciones.
 - Depende de la representación de los estados
 - Preferencia por representaciones con menor número de acciones
- Ejemplo: Acciones del 8-puzzle:
 - Según los movimientos de los bloques: $32 = (8 \text{ bl.} \times 4 \text{ mov.})$
 - Según los movimientos del hueco: 4.

Acciones

- Acciones en el 8 puzle
 - Mover el hueco hacia arriba
 - Mover el hueco hacia abajo
 - Mover el hueco hacia la derecha
 - Mover el hueco hacia la izquierda
- Descripción de la acción “Mover el hueco hacia arriba”
 - Aplicabilidad: es aplicable a estados que no tengan el hueco en la primera fila
 - Resultado de aplicarlo: intercambiar las posiciones del hueco y del bloque que está encima de éste



- Los tres restantes, análogamente

Estado inicial

- Estado inicial
 - Un estado que describe la situación de partida
- Estado inicial en el ejemplo del 8-puzle

| | | |
|---|---|---|
| 2 | 8 | 3 |
| 1 | 6 | 4 |
| 7 | | 5 |

Estados finales

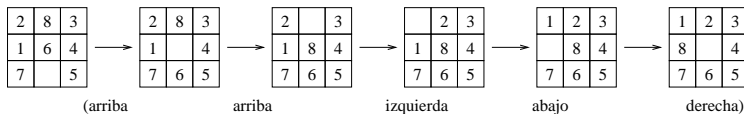
- Descripción del objetivo
 - Usualmente, un conjunto de estados, que llamaremos *finales*
 - A veces, aunque no necesariamente, un único estado final
- Ejemplo del 8-puzzle (un único estado final)

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 8 | | 4 |
| 7 | 6 | 5 |

- Formas de describir los estados finales:
 - Enumerativa.
 - Declarativa.

Soluciones de un problema

- Definición de solución de un problema.
 - Secuencia de acciones a realizar para conseguir el objetivo
 - Secuencia de acciones cuya aplicación desde el estado inicial obtiene un estado final
- Ejemplo: Una solución del 8-puzle:

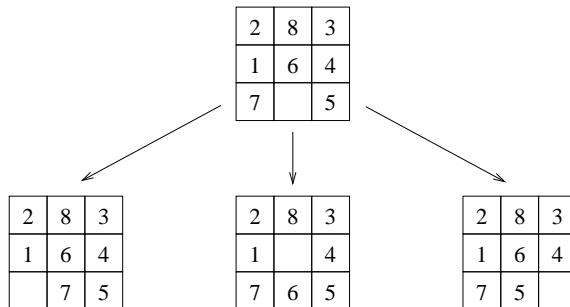


Soluciones de un problema

- Tipos de problemas:
 - Buscar una solución.
 - Determinar si existe solución y encontrar un estado final.
 - Buscar cualquier solución lo más rápidamente posible.
 - Buscar todas las soluciones.
 - Buscar la solución más corta.
 - Buscar la solución menos costosa.

Espacio de estados como un grafo

- Un espacio de estados se puede ver como un grafo dirigido
 - Los vértices de dicho grafo son los estados
 - Sucesores de un estado: aquellos obtenidos a partir del estado aplicando una acción aplicable
- Ejemplo en el 8-puzle



Planteamiento del problema del granjero

- Enunciado:
 - Un granjero está con un lobo, una cabra y una col en una orilla de un río.
 - Desea pasarlos a la otra orilla.
 - Dispone de una barca en la que sólo puede llevar una cosa cada vez.
 - El lobo se come a la cabra si no está el granjero.
 - La cabra se come la col si no está el granjero.
- Información de los estados: orilla en la que está cada elemento
 - La orilla de la barca es redundante (está en la orilla donde esté el granjero)

Formulación del problema del granjero

- Representación de estados: (x y z u) en $\{i,d\}^4$.
 - Número de estados: 16.
- Estado inicial: (i i i i).
- Estado final (único): (d d d d).
- Acciones:
 - Pasa el granjero solo.
 - Pasa el granjero con el lobo.
 - Pasa el granjero con la cabra.
 - Pasa el granjero con la col.

Formulación del problema del granjero

- Aplicabilidad de las acciones
 - Precondición (en los tres últimos): los dos elementos que pasan han de estar en la misma orilla
 - Poscondición: en el estado resultante no deben estar el lobo y la cabra, o la cabra y la col, en la misma orilla sin que el granjero esté en esa misma orilla
- Estado resultante de aplicar la acción
 - Cambiar la orilla de los elementos que pasan por la orilla opuesta

Planteamiento del problema de las jarras

- Enunciado:
 - Se tienen dos jarras, de 4 y 3 litros respectivamente.
 - Ninguna de ellas tiene marcas de medición.
 - Se tiene una bomba que permite llenar las jarras de agua.
 - Averiguar cómo se puede lograr tener exactamente 2 litros de agua en la jarra de 4 litros de capacidad.
- Representación de estados: (x, y) con x en $\{0, 1, 2, 3, 4\}$ e y en $\{0, 1, 2, 3\}$.
- Número de estados: 20.

Formulación del problema de las jarras

- Estado inicial: (0 0).
- Estados finales: todos los estados de la forma (2 _).
- Acciones:
 - Llenar la jarra de 4 litros con la bomba.
 - Llenar la jarra de 3 litros con la bomba.
 - Vaciar la jarra de 4 litros en el suelo.
 - Vaciar la jarra de 3 litros en el suelo.
 - Llenar la jarra de 4 litros con la jarra de 3 litros.
 - Llenar la jarra de 3 litros con la jarra de 4 litros.
 - Vaciar la jarra de 3 litros en la jarra de 4 litros.
 - Vaciar la jarra de 4 litros en la jarra de 3 litros.

Planteamiento del problema de las jarras

- Aplicación de acciones a un estado (x, y)
- Acción “Llenar jarra de 3”
 - Aplicabilidad: $y < 3$ (precondición)
 - Estado resultante: $(x, 3)$
- Acción “Llenar jarra de 4 con jarra de 3”
 - Aplicabilidad: $x < 4, y > 0, x + y > 4$ (precondición)
 - Estado resultante: $(4, x + y - 4)$
- Acción “Vaciar jarra de 3 en jarra de 4”
 - Aplicabilidad: $y > 0, x + y \leq 4$ (precondición)
 - Estado resultante: $(x + y, 0)$
- Análogamente las demás acciones

Problemas de la vida real

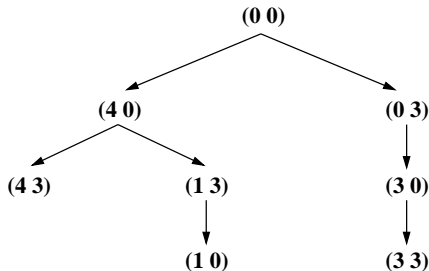
- Problemas de la vida real que se pueden plantear y resolver como espacios de estados:
 - Búsqueda de rutas en redes informáticas
 - Rutas aéreas para viajar
 - Problema del viajante
 - Diseño de microchips
 - Ensamblaje de componentes
 - Desplazamiento de robots

Búsqueda de soluciones en espacios de estados

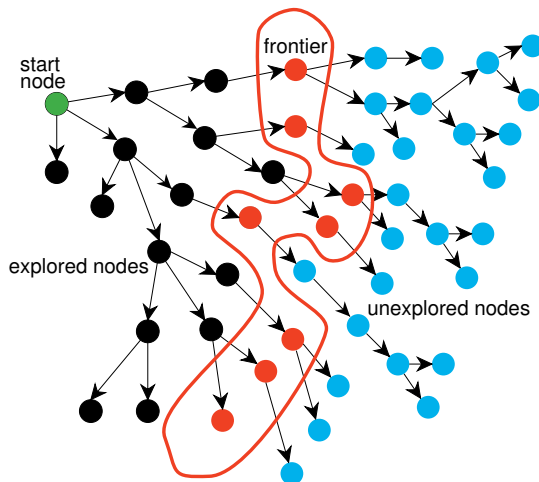
- Objetivo: encontrar una secuencia de acciones que, partiendo del estado inicial, obtenga un estado final
- Idea básica: exploración del *grafo* del espacio de estados
 - En cada momento se analiza un estado actual (en un principio, el inicial)
 - Si el estado actual es final, acabar (recopilando la sucesión de acciones)
 - En caso contrario, obtener los sucesores del estado actual (*expandir*)
 - *Elegir* un nuevo estado actual, *dejando* los restantes para analizarlos posteriormente (si fuera necesario)
 - Repetir el proceso mientras haya estados por analizar
- La elección del estado actual en cada momento determina una *estrategia* de búsqueda

Árboles de búsqueda

- El proceso anterior puede verse como la construcción incremental de un árbol de búsqueda
- Ejemplo en el problema de las jarras:



Árboles de búsqueda, intuición gráfica



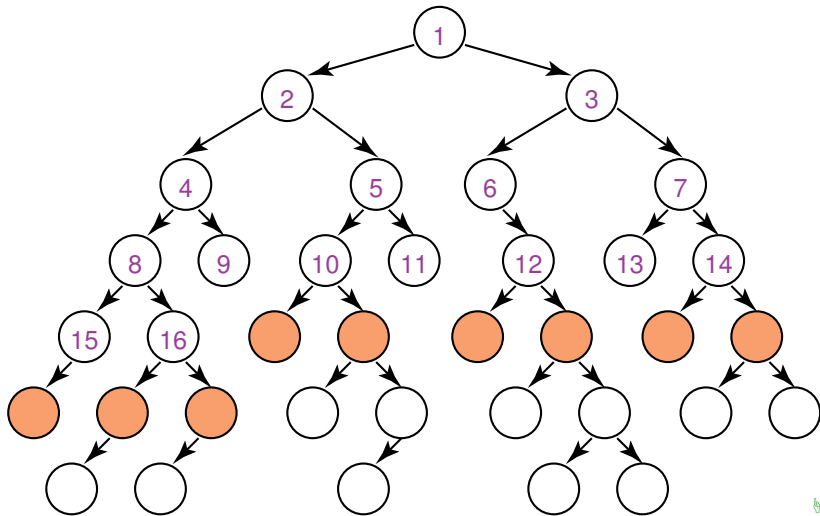
Implementación de la búsqueda en anchura

- En la búsqueda en anchura, ABIERTOS se gestiona como una cola (FIFO):

FUNCION BUSQUEDA-EN-ANCHURA()

1. Hacer ABIERTOS la cola formada por el nodo inicial (el nodo cuyo estado es *ESTADO-INICIAL*)
Hacer CERRADOS vacío
2. Mientras que ABIERTOS no esté vacía,
 - 2.1 Hacer ACTUAL el primer nodo de ABIERTOS
 - 2.2 Hacer ABIERTOS el resto de ABIERTOS
 - 2.3 Poner el nodo ACTUAL en CERRADOS.
 - 2.4 Si ES-ESTADO-FINAL(ESTADO(ACTUAL)),
 - 2.4.1 devolver el nodo ACTUAL y terminar.
 - 2.4.2 en caso contrario,
 - 2.4.2.1 Hacer NUEVOS-SUCESORES la lista de nodos de SUCESORES(ACTUAL) cuyo estado no está ni en ABIERTOS ni en CERRADOS
 - 2.4.2.2 Hacer ABIERTOS el resultado de incluir NUEVOS-SUCESORES al final de ABIERTOS
3. Devolver FALLO.

Búsqueda en anchura



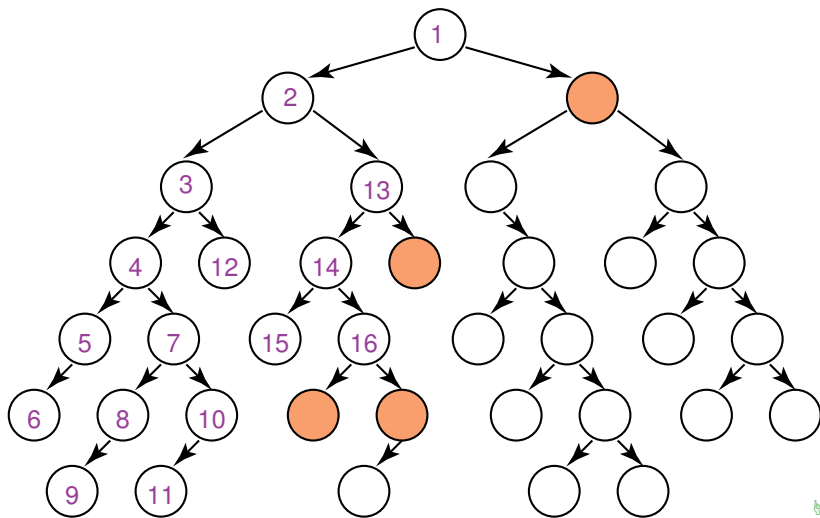
Implementación de la búsqueda en profundidad

- Búsqueda en profundidad, ABIERTOS se gestiona como una pila (LIFO):

FUNCION BUSQUEDA-EN-PROFUNDIDAD()

1. Hacer ABIERTOS la pila formada por el nodo inicial (el nodo cuyo estado es *ESTADO-INICIAL*)
Hacer CERRADOS vacío
2. Mientras que ABIERTOS no esté vacía,
 - 2.1 Hacer ACTUAL el primer nodo de ABIERTOS
 - 2.2 Hacer ABIERTOS el resto de ABIERTOS
 - 2.3 Poner el nodo ACTUAL en CERRADOS.
 - 2.4 Si ES-ESTADO-FINAL(ESTADO(ACTUAL)),
 - 2.4.1 devolver el nodo ACTUAL y terminar.
 - 2.4.2 en caso contrario,
 - 2.4.2.1 Hacer NUEVOS-SUCESORES la lista de nodos de SUCESORES(ACTUAL) cuyo estado no está ni en ABIERTOS ni en CERRADOS
 - 2.4.2.2 Hacer ABIERTOS el resultado de incluir NUEVOS-SUCESORES al principio de ABIERTOS
3. Devolver FALLO.

Búsqueda en profundidad



Búsqueda en profundidad acotada

- Podemos paliar en cierto modo la incompletitud de la búsqueda en profundidad
- Idea: no explorar caminos más allá de una determinada longitud.
- Problema: tampoco es completa, si la cota es menor de la longitud de la solución más corta
- Aunque en muchos casos se puede conocer de antemano una cota adecuada.

Implementación de la búsqueda en profundidad acotada

FUNCION BUSQUEDA-EN-PROFUNDIDAD-ACOTADA(COTA)

- 1. Hacer ABIERTOS la pila formada por el nodo inicial (el nodo cuyo estado es *ESTADO-INICIAL*);
Hacer CERRADOS vacío**
- 2. Mientras que ABIERTOS no esté vacía,**
 - 2.1 Hacer ACTUAL el primer nodo de ABIERTOS**
 - 2.2 Hacer ABIERTOS el resto de ABIERTOS**
 - 2.3 Poner el nodo ACTUAL en CERRADOS.**
 - 2.4 Si ES-ESTADO-FINAL(ESTADO(ACTUAL)),**
 - 2.4.1 devolver el nodo ACTUAL y terminar.**
 - 2.4.2 en caso contrario, si PROFUNDIDAD(ACTUAL) es menor que la cota,**
 - 2.4.2.1 Hacer NUEVOS-SUCESORES la lista de nodos de SUCESORES(ACTUAL) cuyo estado no está ni en ABIERTOS ni en CERRADOS**
 - 2.4.2.2 Hacer ABIERTOS el resultado de incluir NUEVOS-SUCESORES al principio de ABIERTOS**
- 3. Devolver FALLO.**

Búsqueda en profundidad iterativa

- Cuando no se conoce la cota, una opción para evitar la incompletitud es realizar búsquedas acotadas, incrementando la cota gradualmente
- Implementación de la búsqueda en profundidad iterativa

FUNCION BUSQUEDA-EN-PROFUNDIDAD-ITERATIVA(COTA-INICIAL)

1. Hacer $N = \text{COTA-INICIAL}$

2. Si $\text{BUSQUEDA-EN-PROFUNDIDAD-ACOTADA}(N)$ no devuelve fallo,

2.1 devolver su resultado y terminar.

2.2 en caso contrario, hacer N igual a $N+1$ y volver a 2

Comparación de procedimientos

| | Anchura | Profundidad | Profundidad acotada | Profundidad iterativa |
|----------|--------------|-------------|------------------------|--------------------------|
| Tiempo | $O(r^{p+1})$ | $O(r^m)$ | $O(r^c)$ | $O(r^p)$ |
| Espacio | $O(r^{p+1})$ | $O(rm)$ | $O(rc)$ | $O(rp)$ |
| Completa | Sí | No | Sí, si $c \geq p$ | Sí |
| Mínima | Sí | No | No | Sí |

- r : factor de ramificación.
- p : profundidad de la solución.
- m : máxima profundidad de la búsqueda.
- c : cota de la profundidad.

Búsqueda informada

- Búsqueda ciega o no informada (anchura, profundidad, . . .): no cuenta con ningún conocimiento sobre cómo llegar al objetivo.
- Búsqueda informada: aplicar *conocimiento* al proceso de búsqueda para hacerlo más eficiente.
- El conocimiento vendrá dado por una función que *estima* la “bondad” de los estados:

Concepto de heurística

- Heurística:
 - Del griego *heuriskein*, descubrir: ¡*Eureka!*
 - Según el DRAE: “técnica de la indagación y del descubrimiento”.
 - Otro significado: método para resolver problemas que no garantiza la solución, pero que en general funciona bien.
 - En nuestro caso, una heurística será una función numérica sobre los estados.
- Función heurística, **heurística (estado)** :
 - Estima la “distancia” al objetivo.
 - Siempre mayor o igual que 0.
 - Valor en los estados finales: 0.
 - Admitimos valor ∞ .
- Todo el conocimiento específico que se va a usar sobre el problema está codificado en la función heurística.

Recordar: problema del 8-puzle

- Un tablero cuadrado (3x3) en el que hay situados 8 bloques cuadrados numerados (con lo cual se deja un hueco del tamaño de un bloque). Un bloque adyacente al hueco puede deslizarse hacia él. El juego consiste en transformar una posición inicial en la posición final mediante el deslizamiento de los bloques. En particular, consideramos el estado inicial y final siguientes:

| | | |
|---|---|---|
| 2 | 8 | 3 |
| 1 | 6 | 4 |
| 7 | | 5 |

Estado inicial

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 8 | | 4 |
| 7 | 6 | 5 |

Estado final

- Estados: cada una de las posibles configuraciones del tablero
- Acciones: arriba, abajo, izquierda, derecha (movimientos del hueco)

Primera heurística en el problema del 8-puzle

- Problema del 8-puzle (primera heurística):
 - **heurística (estado)**: número de piezas descolocadas respecto de su posición en el estado final
- Ejemplo:

| | | |
|---|---|---|
| 2 | 8 | 3 |
| 1 | 6 | 4 |
| 7 | | 5 |

$$H = 4$$

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 8 | | 4 |
| 7 | 6 | 5 |

$$H = 0$$

Segunda heurística en el problema del 8-puzzle

- Problema del 8-puzzle (segunda heurística):
 - **heurística (estado)**: suma de las distancias Manhattan de cada pieza a donde debería estar en el estado final
- Ejemplo:

| | | |
|---|---|---|
| 2 | 8 | 3 |
| 1 | 6 | 4 |
| 7 | | 5 |

$$H = 5$$

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 8 | | 4 |
| 7 | 6 | 5 |

$$H = 0$$

Idea de la búsqueda por primero el mejor

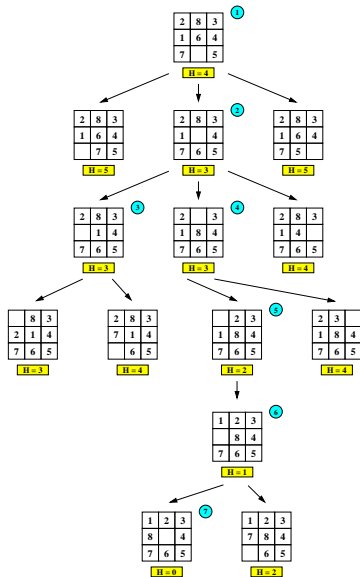
- Búsqueda por primero el mejor:
 - Analizar preferentemente los nodos con heurística más baja.
 - Ordenar la cola de abiertos por heurística, de menor a mayor
- También llamada búsqueda *voraz* o *codiciosa* (del inglés “*greedy*”)
 - Porque siempre elige expandir *lo que estima* que está más “cerca” del objetivo
- Su rendimiento dependerá de la bondad de la heurística usada.

Implementación de la búsqueda por primero el mejor

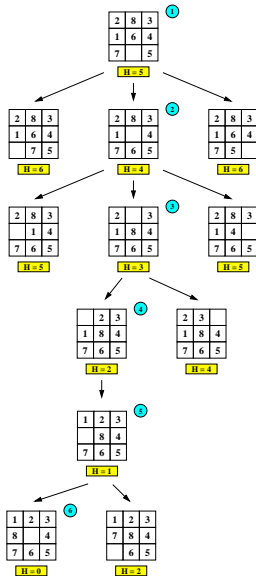
FUNCION BUSQUEDA-POR-PRIMERO-EL-MEJOR()

1. Hacer ABIERTOS la cola formada por el nodo inicial (el nodo cuyo estado es *ESTADO-INICIAL*);
Hacer CERRADOS vacío
2. Mientras que ABIERTOS no esté vacía,
 - 2.1 Hacer ACTUAL el primer nodo de ABIERTOS
 - 2.2 Hacer ABIERTOS el resto de ABIERTOS
 - 2.3 Poner el nodo ACTUAL en CERRADOS.
 - 2.4 Si ES-ESTADO-FINAL(ESTADO(ACTUAL)),
 - 2.4.1 devolver el nodo ACTUAL y terminar.
 - 2.4.2 en caso contrario,
 - 2.4.2.1 Hacer NUEVOS-SUCESORES la lista de nodos de SUCESORES(ACTUAL) cuyo estado no está ni en ABIERTOS ni en CERRADOS
 - 2.4.2.2 Hacer ABIERTOS el resultado de incluir NUEVOS-SUCESORES en ABIERTOS y ordenar en orden creciente de las heurísticas de los estados de los nodos
3. Devolver FALLO.

8-puzzle por primero el mejor: heurística 1



8-puzzle por primero-el-mejor: heurística 2



Planificación en IA

- Planificar: encontrar una secuencia de acciones que alcanzan un determinado objetivo si se ejecutan desde un determinado estado inicial.
- Plan: secuencia de acciones que consiguen el objetivo
- Aplicaciones del mundo real:
 - Robótica
 - Fabricación mediante ensamblado de componentes
 - Misiones espaciales
- *Planning vs Scheduling*
 - *Planning* se refiere a la secuenciación de acciones para conseguir un objetivo y *Scheduling* pone más énfasis en la utilización eficiente de los recursos disponibles, por parte de las acciones

El problema de la planificación en IA

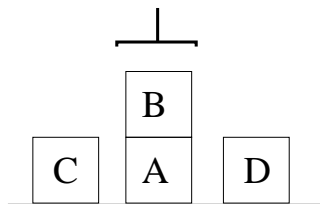
- Posibles cuestiones a abordar:
 - **Representación del mundo y de las acciones que lo transforman**
 - **Algoritmos de búsqueda de planes**
 - Minimizar los recursos consumidos por el plan
 - Tiempo en el que se realiza cada acción
 - Monitorizar la ejecución del plan, revisándolo en caso de errores o contingencias
- Por simplificar, en este tema supondremos:
 - Un número finito de estados
 - Completamente observables
 - Acciones deterministas, totalmente definidas por su especificación
 - Tiempo implícito: las acciones actúan *sin duración*
 - La planificación se realiza *a priori*
 - Hipótesis del mundo cerrado

Planificación y búsqueda en espacio de estados

- Problema abordable con búsqueda en espacio de estados, pero en problemas de escala real hay que abordar además:
 - Descripción de los estados en el mundo real extremadamente compleja
 - Gran cantidad de posibles acciones, muchas de ellas irrelevantes para la consecución del objetivo final
 - Las acciones sólo cambian una pequeña porción del mundo (*el problema del marco*)
 - Necesitamos heurísticas *independientes* del dominio
 - La necesidad de una acción puede detectarse sin necesidad de que se haya decidido las acciones previas (*compromiso mínimo*)
 - A veces es aconsejable *descomponer* en subproblemas más simples
- Idea: usar la lógica para representar estados, acciones y objetivos, y algoritmos que operan sobre esta representación

Ejemplo: el mundo de los bloques

- Ejemplo clásico en planificación.
- Elementos que intervienen:
 - Una superficie plana.
 - Una serie de bloques cúbicos.
 - Un brazo robotizado, que puede coger un bloque cada vez.
 - Un bloque puede estar sobre la mesa o apilado sobre otro bloque.

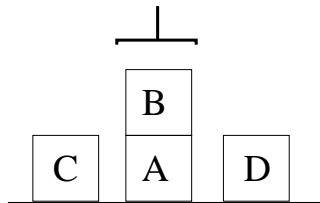


Formalismo lógico: el lenguaje PDDL

- Un lenguaje para representar problemas de planificación:
 - Constantes: objetos del mundo (en mayúsculas)
 - Variables para representar objetos genéricos (en minúsculas)
 - Símbolos de predicados (para expresar propiedades de los objetos)
 - Símbolos para representar las acciones
- Terminología:
 - Átomos: fórmulas de la forma $P(o_1, \dots, o_n)$, donde P es un símbolo de predicado y cada o_i es una constante o una variable (no hay símbolos de función)
 - Literales: átomos o negación de átomos (usaremos el símbolo $-$ para la negación)
 - Átomos y literales cerrados: sin variables
- Estados: conjunción de átomos cerrados
 - Hipótesis del mundo cerrado: los átomos que no se mencionan se suponen falsos

Representación de estados en el mundo de los bloques

- Descripción de un estado:



`DESPEJADO(B)`, `DESPEJADO(C)`, `DESPEJADO(D)`, `BRAZOLIBRE()`,
`SOBRE(B,A)`, `SOBRELAMESA(C)`, `SOBRELAMESA(D)`, `SOBRELAMESA(A)`

- Predicados lógicos usados en esta representación:

`DESPEJADO(x)`, el bloque `x` está despejado.

`BRAZOLIBRE()`, el brazo no agarra ningún bloque.

`SOBRELAMESA(x)`, el bloque `x` está sobre la mesa.

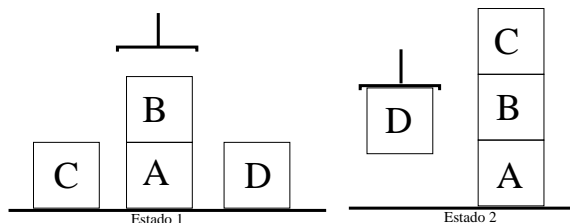
`SOBRE(x,y)`, el bloque `x` está sobre el `y`.

`AGARRADO(x)`, el bloque `x` está sujeto por el brazo.

Representación de objetivos

- Objetivos: descripción de los estados finales.
- Los objetivos se representan como conjunción de literales (con posibilidad de usar variables)
 - Las variables en los objetivos se interpretan como existencialmente cuantificadas
- Satisfacer un objetivo:
 - Un estado satisface un objetivo si es posible sustituir las variables del objetivo por constantes de manera que el estado contiene todos los literales positivos del *objetivo instanciado* y ninguno de los negativos
 - Un estado es estado final si satisface el objetivo requerido
- Importante: verificar si un estado satisface un objetivo es un cálculo meramente simbólico

Ejemplo de objetivos en el mundo de los bloques



- Ejemplos de objetivos:
 - **SOBRE (B, A)** , **SOBRELAMESA (A)** , **-SOBRE (C, B)** es satisfecho por el estado 1 pero no por el estado 2
 - **SOBRE (x, A)** , **DESPEJADO (x)** , **BRAZOLIBRE ()** es satisfecho por el estado 1 pero no por el estado 2
 - **SOBRE (x, A)** , **SOBRE (y, x)** no es satisfecho por el estado 1 pero sí por el estado 2
 - El objetivo **SOBRE (x, A)** , **-SOBRE (C, x)** es satisfecho por el estado 1 pero no por el estado 2

Descripción de esquemas de acciones

- Para intentar solucionar el *problema del marco*, sólo se especifica lo que *cambia* por la acción.
- Una acción se describe mediante:
 - Su nombre y *todas* las variables involucradas:
 $O(x_1, \dots, x_n)$
 - Precondición: lista de literales que deben cumplirse para poder aplicar la acción.
 - Efectos: lista de literales que indica los cambios que se producirán cuando se aplique.

Descripción de esquemas de acciones

- En la lista de efectos distinguimos:
 - Efectos positivos (o lista de *adición*): átomos que pasarán a ser ciertos
 - Efectos negativos (o lista de *borrado*): átomos que dejarán de ser ciertos
- El uso de variables hace que una acción usualmente represente, en realidad, un *esquema de acción*:
 - Por cada manera de sustituir las variables por constantes, tenemos una *acción* concreta (una *instancia* de la acción)

Ejemplo: acciones en el mundo de los bloques

- Colocar un bloque sobre otro:

APILAR(x, y)

Prec.: DESPEJADO(y), AGARRADO(x)

Efec.: -DESPEJADO(y), -AGARRADO(x),
BRAZOLIBRE(), SOBRE(x, y), DESPEJADO(x)

- Quitar un bloque que estaba sobre otro:

DESAPILAR(x, y)

Prec.: SOBRE(x, y), DESPEJADO(x), BRAZOLIBRE()

Efec.: -SOBRE(x, y), -DESPEJADO(x), -BRAZOLIBRE(),
AGARRADO(x), DESPEJADO(y)

- Agarrar un bloque con el robot:

AGARRAR(x)

Prec.: DESPEJADO(x), SOBRELAMESA(x), BRAZOLIBRE()

Efec.: -DESPEJADO(x), -SOBRELAMESA(x), -BRAZOLIBRE(),
AGARRADO(x)

- Bajar un bloque hasta la superficie:

BAJAR(x)

Prec.: AGARRADO(x)

Efec.: -AGARRADO(x),
SOBRELAMESA(x), BRAZOLIBRE(), DESPEJADO(x)

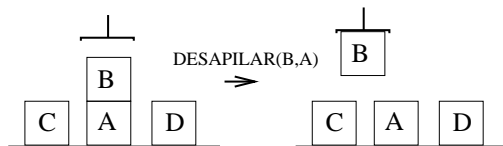
Aplicabilidad de una acción

- Una acción es *aplicable* a un estado si éste satisface su precondition
- Si aparecen variables en la precondition, la aplicabilidad se define *respecto de la sustitución* θ usada para satisfacer la precondition
- Por abreviar, a veces la sustitución usada aparecerá implícita al hablar de la acción
- Por ejemplo, hablaremos de **DESAPILAR**(**A**, **B**) para referirnos a **DESAPILAR**(**x**, **y**) con la sustitución [**x**/**A**, **y**/**B**]
- Un mismo esquema de acción puede dar lugar a distintas acciones que son aplicables al mismo estado

Resultado de aplicar una acción

- El resultado de aplicar una acción aplicable (respecto de una sustitución θ) a un estado E es el estado resultante de:
 - Eliminar de E los átomos, instanciados por θ , correspondiente a la lista de efectos negativos (si estuvieran)
 - Añadir a E los átomos, instanciados por θ , correspondientes a la lista de efectos positivos (si no estuvieran)

Ejemplo de aplicación de acción (I)



* Estado antes de aplicar DESAPILAR(B,A):

$E = \{ \text{DESPEJADO}(B), \text{DESPEJADO}(C), \text{DESPEJADO}(D), \text{BRAZOLIBRE}(), \\ \text{SOBRE}(B,A), \text{SOBRELAMESA}(C), \text{SOBRELAMESA}(D), \text{SOBRELAMESA}(A) \}$

* Precondiciones de DESAPILAR(B,A):

$\text{Prec} = \{ \text{SOBRE}(B,A), \text{DESPEJADO}(B), \text{BRAZOLIBRE}() \}$

----- Condiciones satisfechas en el estado -----
----- (acción aplicable) -----

* Efectos de DESAPILAR(B,A):

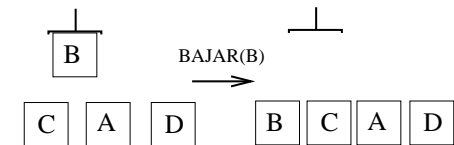
$\text{Efec-} = \{ \text{SOBRE}(B,A), \text{DESPEJADO}(B), \text{BRAZOLIBRE}() \}$

$\text{Efec+} = \{ \text{AGARRADO}(B), \text{DESPEJADO}(A) \}$

* Estado después de aplicar DESAPILAR(B,A):

$E' = (E - \text{Efec-}) \cup \text{Efec+} = \\ \{ \text{DESPEJADO}(C), \text{DESPEJADO}(D), \text{SOBRELAMESA}(C), \\ \text{SOBRELAMESA}(D), \text{SOBRELAMESA}(A), \text{AGARRADO}(B), \text{DESPEJADO}(A) \}$

Ejemplo de aplicación de acción (II)



* Estado antes de aplicar BAJAR(B):

$E = \{ \text{DESPEJADO}(C), \text{DESPEJADO}(A), \text{DESPEJADO}(D), \text{SOBRELAMESA}(C), \text{SOBRELAMESA}(A), \text{SOBRELAMESA}(D), \text{AGARRADO}(B) \}$

* Precondiciones de BAJAR(B):

$\text{Prec} = \{ \text{AGARRADO}(B) \}$

----- Condiciones satisfechas en el estado -----
 ----- (acción aplicable) -----

* Efectos de BAJAR(B):

$\text{Efec-} = \{ \text{AGARRADO}(B) \}$

$\text{Efec+} = \{ \text{SOBRELAMESA}(B), \text{BRAZOLIBRE}(), \text{DESPEJADO}(B) \}$

* Estado después de aplicar BAJAR(B):

$E' = (E - \text{Efec-}) \cup \text{Efec+} =$

$\{ \text{DESPEJADO}(C), \text{DESPEJADO}(A), \text{DESPEJADO}(D), \text{SOBRELAMESA}(C), \text{SOBRELAMESA}(A), \text{SOBRELAMESA}(D), \text{SOBRELAMESA}(B), \text{BRAZOLIBRE}(), \text{DESPEJADO}(B) \}$

Planes y soluciones

- Plan: secuencia de acciones
 - La primera aplicable al estado inicial y cada una de ellas aplicable al resultado de la anterior
- Solución: plan que a partir del estado inicial obtiene un estado que satisface el objetivo

Ejemplo: cambio de rueda pinchada

- Lenguaje:
 - Objetos: RUEDA-REPUESTO, RUEDA-PINCHADA, EJE, MALETERO, SUELO
 - Predicado: $EN(-, -)$
- Estado inicial:
 $EN(RUEDA-PINCHADA, EJE), EN(RUEDA-REPUESTO, MALETERO)$
- Estado final:
 $EN(RUEDA-REPUESTO, EJE)$

Acciones en el cambio de rueda pinchada

- Sacar la rueda de repuesto del maletero:

QUITAR(RUEDA-REPUESTO, MALETERO)

Prec.: EN(RUEDA-REPUESTO, MALETERO)

Efec.: EN(RUEDA-REPUESTO, SUELO), -EN(RUEDA-REPUESTO, MALETERO)

- Quitar la rueda pinchada del eje:

QUITAR(RUEDA-PINCHADA, EJE)

Prec.: EN(RUEDA-PINCHADA, EJE)

Efec.: -EN(RUEDA-PINCHADA, EJE), EN(RUEDA-PINCHADA, SUELO)

- Colocar la rueda de repuesto en el eje:

PONER(RUEDA-REPUESTO, EJE)

Prec.: -EN(RUEDA-PINCHADA, EJE), EN(RUEDA-REPUESTO, SUELO)

Efec.: -EN(RUEDA-REPUESTO, SUELO), EN(RUEDA-REPUESTO, EJE)

- Dejar el coche solo hasta la mañana siguiente:

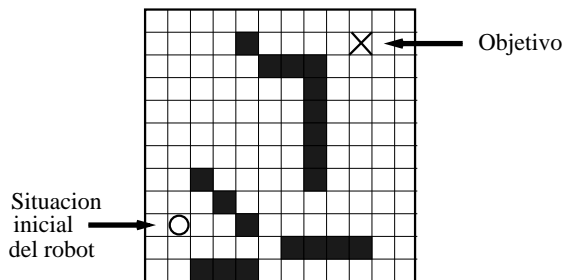
DEJARSOLO()

Prec.: {}

Efec.: -EN(RUEDA-REPUESTO, SUELO), -EN(RUEDA-REPUESTO, EJE),
-EN(RUEDA-REPUESTO, MALETERO) -EN(RUEDA-PINCHADA, EJE),
-EN(RUEDA-PINCHADA, SUELO)

Ejemplo: movimiento de robot por una rejilla

- Un robot ha de desplazarse por una rejilla, desde una posición inicial a una final
 - 8 movimientos posibles: N, S,E,O,NO,NE,SO,SE
 - En algunas de las rejillas existen obstáculos no franqueables



Representación del problema del movimiento de robot

- Lenguaje:
 - Constantes: números que indican coordenadas horizontales y verticales
 - Predicados: **ROBOT-EN**(-, -) y **LIBRE**(-, -)
- Estado inicial (casillas sin obstáculos y posición del robot):

LIBRE(1, 1), ..., **LIBRE**(6, 2), **LIBRE**(11, 2), ..., **LIBRE**(12, 12),
ROBOT-EN(2, 3) .

- Objetivo: **ROBOT-EN**(10, 11)
- Acciones (sólo una, las siete restantes son análogos):

MOVER-SE(x, y)

Prec.: **ROBOT-EN**(x, y), **LIBRE**(x+1, y-1)

Efec.: **-ROBOT-EN**(x, y), **-LIBRE**(x+1, y-1),
ROBOT-EN(x+1, y-1), **LIBRE**(x, y)

- En este caso, necesitamos símbolos de función (+ y -), y que el test de satisfacibilidad maneje las nociones de número siguiente y anterior (extendemos la semántica)

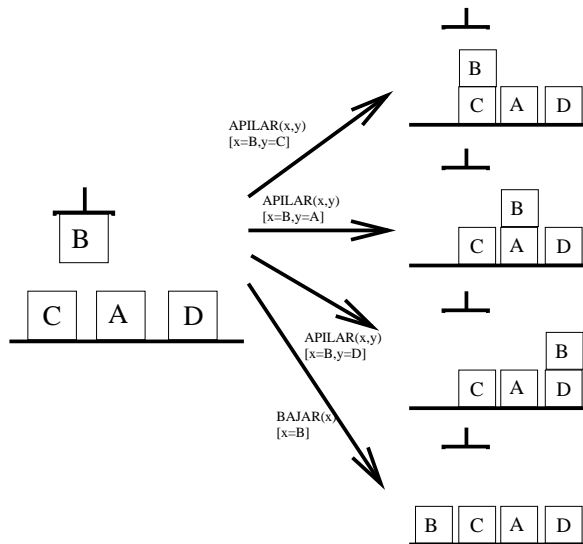
Extensiones al formalismo PDDL

- Existen sistemas de planificación que usan un lenguaje de representación más expresivo
- Por ejemplo:
 - Variables con tipos (no es una extensión que aumente la capacidad expresiva)
 - Uso de símbolos de función
 - Manejo del símbolo de igualdad
 - Evaluación de funciones al aplicar acciones
 - Disyunciones en las precondiciones
- En general existe un compromiso entre expresividad del lenguaje y simplicidad de los algoritmos que manejan la representación

Búsqueda en el espacio de estados

- Un problema de planificación se puede plantear como un problema de espacio de estados:
 - Estados descritos mediante listas de átomos cerrados.
 - Acciones como listas de precondiciones y efectos.
 - Función `es-estado-final` descrita por un objetivo.
- La búsqueda de planes podría hacerse usando los algoritmos de búsqueda ya vistos en los temas anteriores: anchura, profundidad, primero el mejor, A^* ,...
- El uso del formalismo lógico permite el uso de heurísticas *independientes del dominio*
 - Por ejemplo, el número de literales en el objetivo que quedan por satisfacer en un estado

Cálculo de sucesores



Cálculo de sucesores

- Ejemplo:

$E = \{ \text{DESPEJADO}(C), \text{DESPEJADO}(A), \text{DESPEJADO}(D), \text{SOBRELAMESA}(C), \text{SOBRELAMESA}(A), \text{SOBRELAMESA}(D), \text{AGARRADO}(B) \}$

Cuatro posibles acciones aplicables:

- 1), 2) y 3): $O = \text{APILAR}(x, y)$ con $\text{THETA} = [x=B, y=C]$,
 $\text{THETA} = [x=B, y=A]$ y $\text{THETA} = [x=B, y=D]$, resp.
- 4): $O = \text{DEJAR}(x)$ con sustitución $\text{THETA} = [x=B]$.

Sucesores:

$E1 = \{ \text{DESPEJADO}(A), \text{DESPEJADO}(D), \text{SOBRELAMESA}(C), \text{SOBRELAMESA}(A), \text{SOBRELAMESA}(D), \text{BRAZOLIBRE}(), \text{DESPEJADO}(B), \text{SOBRE}(B, C) \}$

$E2 = \{ \text{DESPEJADO}(C), \text{DESPEJADO}(D), \text{SOBRELAMESA}(C), \text{SOBRELAMESA}(A), \text{SOBRELAMESA}(D), \text{BRAZOLIBRE}(), \text{DESPEJADO}(B), \text{SOBRE}(B, A) \}$

$E3 = \{ \text{DESPEJADO}(A), \text{DESPEJADO}(C), \text{SOBRELAMESA}(C), \text{SOBRELAMESA}(A), \text{SOBRELAMESA}(D), \text{BRAZOLIBRE}(), \text{DESPEJADO}(B), \text{SOBRE}(B, D) \}$

$E4 = \{ \text{DESPEJADO}(A), \text{DESPEJADO}(C), \text{DESPEJADO}(D), \text{SOBRELAMESA}(C), \text{SOBRELAMESA}(A), \text{SOBRELAMESA}(D), \text{BRAZOLIBRE}(), \text{DESPEJADO}(B), \text{SOBRELAMESA}(B) \}$

Búsqueda hacia adelante en profundidad con heurística

```
FUNCION BUSQUEDA-EN-PROFUNDIDAD-H(ESTADO-INICIAL, OBJETIVO, ACCIONES)
  Devolver BEP-H-REC({}, {}, ESTADO-INICIAL, OBJETIVO, ACCIONES)
```

```
FUNCION BEP-H-REC(PPLAN, VISITADOS, ACTUAL, OBJ, ACCIONES)
  1. Si ACTUAL satisface OBJ, devolver PPLAN
  2. Hacer APLICABLES igual a la lista de acciones que sean instancias
    de una acción de ACCIONES, que sean aplicables a ACTUAL y cuya
    aplicación no resulte en un estado de VISITADOS
  3. Hacer ORD-APLICABLES igual a ORDENA-POR-HEURISTICA(APLICABLES)
  4. Para cada ACCION en ORD-APLICABLES
    4.1 Hacer E' el resultado de aplicar ACCION a ACTUAL
    4.2 Hacer RES igual a
      BEP-H-REC(PPLAN·ACCION, VISITADOS U {E'}, E', OBJ, ACCIONES)
    4.3 Si RES no es FALLO, devolver RES y terminar
  5. Devolver FALLO
```

- En el algoritmo anterior, queda por especificar la función
ORDENA-POR-HEURISTICA

- En general, se define a través de una heurística h sobre los estados, que estima el número de pasos hasta el objetivo

Búsqueda hacia adelante en profundidad (propiedades)

- Es un algoritmo tipo *Backtracking* en el que se ordenan por heurística los sucesores del estado actual
- Propiedades:
 - Correcto, completo y siempre termina. No se garantiza la solución más corta
 - Complejidad en tiempo: exponencial
 - La complejidad en espacio es lineal respecto de la máxima profundidad del árbol de búsqueda (esto lo hace preferible a primero el mejor y a A^* , salvo que se busque la solución más corta y se disponga de una h admisible)
- En la práctica, su eficiencia depende de la bondad de la heurística
 - Comentaremos más sobre heurísticas posteriormente

Búsqueda hacia atrás

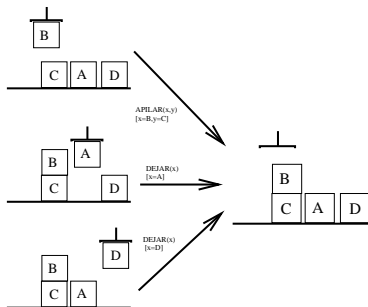
- Si la heurística no es buena, la búsqueda hacia adelante tiene el problema de la excesiva ramificación
 - Muchas acciones aplicables, pero la mayoría de ellas irrelevante para el objetivo
- Alternativa a la búsqueda hacia adelante: buscar hacia atrás, dirigidos por el objetivo
 - Se empieza en el objetivo, se aplican las acciones a la inversa y se trata de llegar al estado inicial
 - Nodos del árbol de búsqueda: objetivos
- La clave: sólo aplicar hacia atrás las acciones relevantes para el objetivo

Acción relevante para un objetivo

- Sea **G** un objetivo *sin variables* (más adelante trataremos el caso con variables). Decimos que una acción **A** es *relevante* para **G** si:
 - Al menos, uno de los efectos (positivo o negativo) de la acción está en **G** con el mismo "signo"
 - Ninguno de los efectos (positivo o negativo) de la acción aparece en **G** con distinto "signo"
- Intuitivamente: una acción es relevante si podría ser la última de un plan que llevara al objetivo

Acciones relevantes: ejemplo

- Objetivo: { **DESPEJADO (A)** , **DESPEJADO (B)** , **DESPEJADO (D)** , **SOBRE (B, C)** }
- Acciones relevantes: **APILAR (B, C)** , **DEJAR (A)** y **DEJAR (D)**



- Ejemplo de acción no relevante para el mismo objetivo:
APILAR (A, D) (aunque tiene a **DESPEJADO (A)** como efecto positivo sin embargo tiene como efecto negativo **-DESPEJADO (D)**)

Cálculo de predecesores

- Si **A** es una acción relevante para un objetivo **G** (sin variables), entonces el *objetivo predecesor* de **G** respecto a **A** es $(G - \text{efectos}(A)) \cup \text{precond}(A)$

- Ejemplo:

$G = \{\text{DESPEJADO}(A), \text{DESPEJADO}(B), \text{DESPEJADO}(D), \text{SOBRE}(B, C)\}$

Predecesor respecto de la acción relevante APILAR(B,C)

$$\begin{aligned} G' &= (G - \{-\text{DESPEJADO}(C), -\text{AGARRADO}(B), \text{BRAZOLIBRE}(), \\ &\quad \text{SOBRE}(B, C), \text{DESPEJADO}(B)\}) \\ &\quad \cup \{\text{DESPEJADO}(C), \text{AGARRADO}(B)\} \\ &= \{\text{DESPEJADO}(A), \text{DESPEJADO}(D), \text{DESPEJADO}(C), \text{AGARRADO}(B)\} \end{aligned}$$

- Nótese que el formalismo lógico nos permite calcular predecesores de manera sencilla

Búsqueda hacia atrás con heurística (sin variables)

```
FUNCION BUSQUEDA-HACIA-ATRÁS-H(ESTADO-INICIAL, OBJ, ACCIONES)
  Devolver BHA-H-REC({}, {}, ESTADO-INICIAL, OBJ, ACCIONES)
```

```
FUNCION BHA-H-REC(PLAN, VISITADOS, ESTADO-INICIAL, G-ACTUAL, ACCIONES)
1. Si ESTADO-INICIAL satisface G-ACTUAL, devolver PLAN
2. Hacer RELEVANTES igual a la lista de acciones que sean instancias
   de una acción de ACCIONES, que sean relevantes para G-ACTUAL
   y tal que el predecesor de G-ACTUAL respecto de la acción
   no sea un objetivo que contiene a alguno de VISITADOS
3. Hacer RELEVANTES-ORDENADOS
   igual a ORDENA-POR-HEURISTICA(RELEVANTES)
4. Para cada ACCION en RELEVANTES-ORDENADOS
   4.1 Hacer G' el objetivo predecesor de G-ACTUAL respecto a ACCION
   4.2 Hacer RES igual a
       BHA-H-REC(ACCION·PLAN, VISITADOS U {G'},
                 ESTADO-INICIAL, G', ACCIONES)
   4.4 Si RES no es FALLO, devolver RES y terminar
5. Devolver FALLO
```

- En el algoritmo anterior, queda por especificar la función **ORDENA-POR-HEURISTICA**
 - Se va a definir a través de una heurística h sobre los objetivos, que estima el número de pasos desde el estado inicial al objetivo

Búsqueda hacia atrás con heurística (propiedades)

- Es un algoritmo de búsqueda en profundidad en un espacio de estados (el de los objetivos)
 - Incorpora heurística para ordenar objetivos
- Propiedades
 - Correcto, completo y siempre termina. No se garantiza la solución más corta
 - Complejidad teórica: como en la búsqueda hacia adelante
- En la práctica, y nuevamente debido a la elevada ramificación, su eficiencia depende de la heurística usada
 - Comentaremos más sobre esto, más adelante
- Además, en el caso de la búsqueda hacia atrás podemos reducir la ramificación, instanciando sólo parcialmente las acciones
 - Para ello, usaremos unificación (similar a PROLOG)

Ejemplo de la conveniencia de no instanciar completamente

- Supongamos el siguiente problema:
 - $n + 2$ bloques: $A, A_1, A_2, \dots, A_n, B$
 - Estado inicial:

 $\{ \text{DESPEJADO}(A), \text{SOBRE}(A, A_1), \text{SOBRE}(A_1, A_2), \dots, \text{SOBRE}(A_{n-1}, A_n),$

 $\text{DESPEJADO}(B), \text{BRAZOLIBRE}() \}$
 - Objetivo: $\neg \text{DESPEJADO}(B), \text{BRAZOLIBRE}() \}$
- Si aplicamos el procedimiento anterior, tenemos entre otras las siguientes $n + 2$ acciones relevantes:

 $\{ \text{APILAR}(A, B), \text{APILAR}(A_1, B), \dots, \text{APILAR}(A_n, B), \text{APILAR}(B, B) \}$
- Idea: podríamos dejar el primer argumento de **APILAR** sin especificar, para reducir la ramificación en la búsqueda
 - Pero nos obligará a tratar con objetivos en los que aparecen variables
 - Para ello, usaremos *unificadores de máxima generalidad*

Unificación de máxima generalidad

- Unificador de dos términos s y t de la lógica de primer orden:
 - Sustitución σ (de las variables de s y de t) tales que $\sigma(s) = \sigma(t)$
- Unificador de máxima generalidad (*umg*)
 - Un unificador más general que cualquier otro unificador
 - Intuitivamente, por *más general* entendemos que las variables se concretan lo mínimo necesario para hacer iguales los dos términos
- Ejemplo: unificación de $P(x_1, A, x_2, x_3)$ y de $Q(x_4, x_5, B, x_5)$
 - $[x_1/B, x_2/B, x_3/A, x_4/B, x_5/A]$ es unificador pero no es el más general
 - $[x_1/x_4, x_2/B, x_3/A, x_5/A]$ es un umg

Ejemplo de búsqueda hacia atrás con unificación (I)

- Apliquemos esta idea al problema anterior
- Hacia atrás con unificación, paso 1:
 - El efecto $\text{-DESPEJADO}(y1)$ de la acción $\text{APILAR}(x1, y1)$ unifica con el literal $\text{-DESPEJADO}(B)$ del objetivo (y es relevante para G)
 - Un umg es la sustitución $\sigma_1 = [y1/B]$ (es decir, $x1$ no se instancia)
 - Por tanto se obtiene un *esquema de acción* relevante:
 $\text{APILAR}(x1, B)$
 - Nuevo objetivo G_1 :

$$(G - \text{efectos}(\text{APILAR}(x1, B))) \cup \text{precond}(\text{APILAR}(x1, B)) =$$

$$\{\text{DESPEJADO}(B), \text{AGARRADO}(x1)\}$$
 - El estado inicial no satisface G_1 , así que seguimos

Ejemplo de búsqueda hacia atrás con unificación (II)

- Hacia atrás con unificación, paso 2:
 - El efecto **AGARRADO**(**x2**) de la acción **DESAPILAR**(**x2**, **y2**) unifica con el literal **AGARRADO**(**x1**) del objetivo (y es relevante para **G₁**)
 - Un umg es la sustitución $\sigma_2 = [\mathbf{x2/x1}]$
 - Por tanto, un esquema de acción relevante sería:
DESAPILAR(**x1**, **y2**)
 - Nuevo objetivo **G₂**:

$$(\mathbf{G1} - \text{efectos}(\mathbf{DESAPILAR}(\mathbf{x1}, \mathbf{y2})) \cup \text{precond}(\mathbf{DESAPILAR}(\mathbf{x1}, \mathbf{y2})) = \{\mathbf{DESPEJADO(B)}, \mathbf{SOBRE(x1, y2)}, \mathbf{DESPEJADO(x1)}, \mathbf{BRAZOLIBRE()}\})$$
 - El estado inicial satisface **G₂** (con la sustitución $[\mathbf{x1/A}, \mathbf{y2/A1}]$)
- Componiendo todas las sustituciones obtenemos el plan:
DESAPILAR(**A**, **A1**) , **APILAR**(**A**, **B**)
- Nota: cada vez que se considera una acción, es necesario renombrarla con variables nuevas (*estandarización*)

Búsqueda hacia atrás con heurística (con unificación)

```

FUNCION BUSQUEDA-HACIA-ATRÁS-H-U (ESTADO-INICIAL, OBJ, ACCIONES)
  Devolver BHA-H-U-REC( {}, {}, ESTADO-INICIAL, OBJ, ACCIONES)

```

```

FUNCION BHA-H-U-REC (PLAN, VISITADOS, ESTADO-INICIAL, G-ACTUAL, ACCIONES)

```

1. Si ESTADO-INICIAL satisface G-ACTUAL, devolver PLAN
2. Hacer RELEVANTES igual a la lista de pares (A, SIGMA) tales que:
 - * A es una acción (estandarizada) de ACCIONES, relevante para ESTADO
 - * SIGMA es umg entre los efectos de A que hacen relevante a la acción respecto de los correspondientes literales de G-ACTUAL
 - * El predecesor de ACTUAL respecto de la acción no es un objetivo con un subconjunto de literales que unifica con alguno de los objetivos VISITADOS
3. Hacer RELEVANTES-ORDENADOS igual a ORDENA-POR-HEURISTICA (RELEVANTES)
4. Para cada (A, SIGMA) en RELEVANTES-ORDENADOS
 - 4.1 Hacer G' el objetivo predecesor de G-ACTUAL respecto a (A, SIGMA); es decir,
 $G' = (\text{SIGMA}(\text{G-ACTUAL}) - \text{Efectos}(\text{SIGMA}(A))) \cup \text{Precondiciones}(\text{SIGMA}(A))$
 - 4.2 Hacer RES igual a
 $\text{BHA-H-U-REC}((A, \text{SIGMA}) \cdot \text{SIGMA}(\text{PLAN}), \text{VISITADOS} \cup \{G'\}, \text{ESTADO-INICIAL}, G', \text{ACCIONES})$
 - 4.3 Si RES no es FALLO, devolver RES y terminar
5. Devolver FALLO

Heurísticas para planificación basada en espacios de estados

- Una componente fundamental para la eficiencia práctica de los anteriores algoritmos es la heurística usada para ordenar los estados o los objetivos
- Esta heurística debe estimar distancia entre estados y objetivos (número de acciones necesarias)
 - En búsqueda hacia adelante: para cada estado, distancia hasta el objetivo
 - En búsqueda hacia atrás: para cada objetivo, distancia hasta el estado inicial
- Si esta estimación está por debajo del mínimo número de acciones real, diremos que la heurística es *admissible*
- Buscamos heurísticas independientes del dominio
 - Basadas en la representación lógica de estados, objetivos y acciones

Heurísticas basadas en relajar el problema

- Heurísticas obtenidas relajando algunas restricciones del problema y calculando el número de acciones necesarias en ese problema relajado
- Algunas ideas para relajar el problema:
 - Ignorar las precondiciones y/o efectos negativos de las acciones
 - Suponer que cada literal de un objetivo se alcanza de manera independiente: como si el número de acciones necesarias para alcanzar un objetivo fuera la *suma* del número de pasos necesario para alcanzar cada literal del objetivo
 - Suponer que el número de acciones necesarias para alcanzar un objetivo es el número de pasos *máximo* necesario para alcanzar *uno* de sus literales
 - Ignorar todas las precondiciones de las acciones
 - Ignorar determinados predicados

La heurística Δ_0

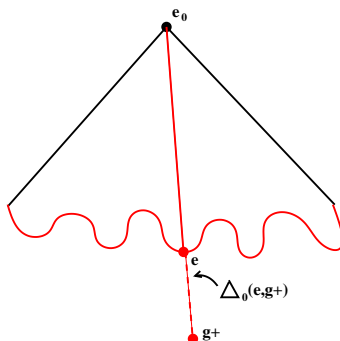
- Vamos a definir a continuación una heurística Δ_0 basada en las dos primeras ideas anteriores
- Dado un estado e , un átomo p y un objetivo g que sólo tiene literales positivos sin variables, definimos recursivamente $\Delta_0(e, p)$ y $\Delta_0(e, g)$ de la siguiente manera:
 - Si p aparece en e , $\Delta_0(e, p) = 0$
 - Si p no aparece en e ni en los efectos positivos de ninguna acción, $\Delta_0(e, p) = +\infty$
 - En otro caso, $\Delta_0(e, p) = \min_A \{1 + \sum_{q \in \text{precond}^+(A)} \Delta_0(e, q) \mid p \in \text{efectos}^+(A)\}$
 - $\Delta_0(e, g) = \sum_{p \in g} \Delta_0(e, p)$
- Notación: $\text{precond}^+(A)$ y $\text{efectos}^+(A)$ denotan respectivamente los literales positivos de la precondition y de los efectos de una acción A

La heurística Δ_0 (propiedades)

- Intuitivamente:
 - $\Delta_0(\mathbf{e}, \mathbf{p})$ cuenta el menor número de pasos necesarios para que se verifique \mathbf{p} a partir de \mathbf{e} , suponiendo que las acciones no tienen precondiciones negativas ni efectos negativos
 - $\Delta_0(\mathbf{e}, \mathbf{g})$ es la suma de las anteriores estimaciones para cada átomo $\mathbf{p} \in \mathbf{g}$
- Nótese que en general, la estimación que realiza Δ_0 no es admisible
 - Aunque en cualquier caso puede funcionar bien en la práctica
 - Y además no la usaremos con A^* , sino con búsqueda en profundidad
- Dado un \mathbf{e} , los valores $\Delta_0(\mathbf{e}, \mathbf{p})$ para cada átomo \mathbf{p} se pueden calcular con un algoritmo similar al algoritmo de caminos mínimos de Dijkstra

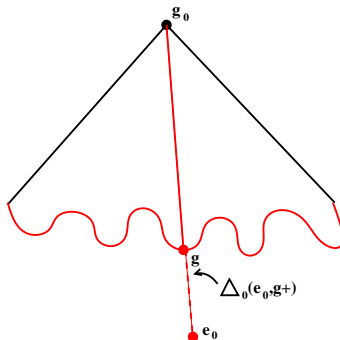
Usando la heurística Δ_0 en las búsquedas (I)

- En la búsqueda hacia adelante:
 - A cada acción A aplicable al estado actual se le asigna el valor heurístico $\Delta_0(e, g+)$, donde $g+$ es el conjunto de literales positivos del objetivo y e el estado que resulta al aplicar la acción A



Usando la heurística Δ_0 en las búsquedas (II)

- En la búsqueda hacia atrás:
 - A cada acción **A** relevante respecto del objetivo actual se le asigna el valor $\Delta_0(\mathbf{e}_0, \mathbf{g}+)$, donde \mathbf{e}_0 es el estado inicial, \mathbf{g} es el predecesor correspondiente a la acción **A** y $\mathbf{g}+$ es el conjunto de literales positivos de \mathbf{g}



Usando la heurística Δ_0 en las búsquedas (III)

- En el caso de que haya variables en los objetivos:
 - Si p es un átomo con variables, $\Delta_0(e, p)$ se define como el mínimo de entre todos los $\Delta_0(e, \sigma(p))$, siendo $\sigma(p)$ instancia sin variables de p
 - Si g es un objetivo con variables, $\Delta_0(e, g)$ es la suma de $\Delta_0(e, p)$ para cada literal positivo p de g

Planes parcialmente ordenados

- Recordar: en el tema anterior se han visto algoritmos para resolver problemas de planificación que hacen búsqueda en el espacio de los estados
- Vemos ahora una aproximación a la búsqueda de planes totalmente diferente a la búsqueda en el espacio de los estados:
 - En lugar de buscar en el espacio de estados o de objetivos, se busca en un espacio de planes, no completamente especificados (planes parcialmente ordenados)
- *Plan parcialmente ordenado*: un plan en el que sólo se especifican algunas de las precedencias entre sus acciones

Ejemplo: aprobar IA1

- Lenguaje:

- Objetos: **IA1**, **ETSII**, **CASA**
- Predicado: **EN (-)**, **ESTUDIADO (-)**, **APROBADO (-)**

- Estado inicial:

EN (CASA)

- Objetivo:

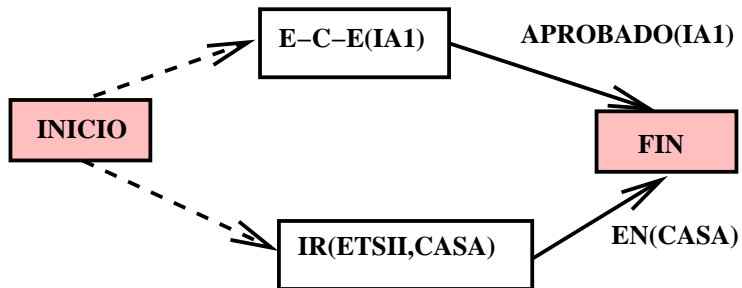
EN (CASA) , APROBADO (IA1)

- Acciones:

| | | |
|----------------------------|-------------------------|--------------------------------------|
| IR (x, y) | ESTUDIAR (x) | EXAMINARSE-CON-EXITO (x) |
| P: EN (x) | P: { } | P: EN (ETSII) , ESTUDIADO (x) |
| E: -EN (x) , EN (Y) | E: ESTUDIADO (x) | E: APROBADO (x) |

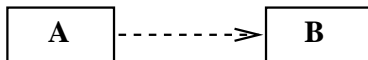
- Descartaremos acciones del tipo **IR (x, x)**

Ejemplo de plan parcial



Planes parcialmente ordenados: componentes (I)

- Nodos: *Acciones*, que constituyen los pasos que el plan lleva a cabo, de entre las acciones del problema, cada una con sus precondiciones y efectos
 - Dos acciones especiales: INICIO (sin precondiciones y cuyo efecto es el estado inicial) y FIN (sin efectos y cuyas precondiciones son el objetivo final)
- Arcos: *restricciones de orden* $\mathbf{A} \prec \mathbf{B}$ entre acciones del plan



(continúa...)

Planes parcialmente ordenados: componentes (II)

- Un caso particular de arcos: *enlaces causales*, $A \xrightarrow{p} B$, especificando la consecución, por parte de una acción, de una de las precondiciones de otra acción del plan (lleva implícita una restricción de orden)



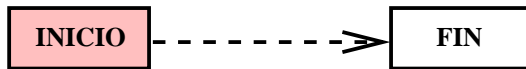
- Un conjunto de *precondiciones abiertas*: aquellas que aún no tienen enlaces causales que las consigan

Planes parciales inicial y final

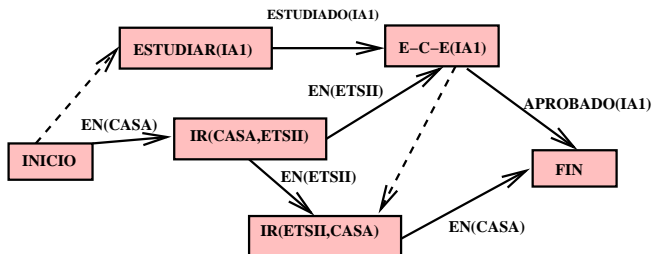
- Plan parcial inicial: plan cuyas únicas acciones son INICIO y FIN, con la restricción $\text{INICIO} \prec \text{FIN}$, sin enlaces causales y con todas las precondiciones de FIN abiertas
- Planes parciales finales: planes parciales **sin conflictos** entre los enlaces causales, **sin ciclos** entre las restricciones de orden y **sin precondiciones abiertas**
 - Una acción **C** entra en *conflicto con* (o *amenaza*) un enlace causal $A \xrightarrow{p} B$, si **C** tiene a $\neg p$ en su lista de efectos y según las restricciones de orden, **C** *podría ir* después de **A** y antes que **B**

Ejemplos de planes parciales

- Ejemplo de plan parcial inicial:

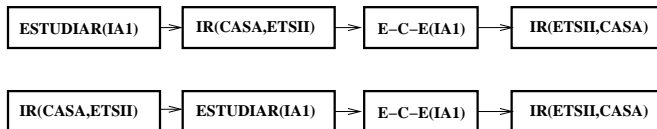


- Ejemplo de plan parcial final:



Secuenciaciones de planes parciales solución

- Secuenciación de un plan parcial: colocar las acciones una detrás de otra, sin contradecir ninguna restricción de orden que se deduzca del plan parcial
- Punto clave: cualquier secuenciación de un plan parcial solución supone una solución al problema original
- En el ejemplo, dos posibilidades (ambas soluciones al problema original):



Planificación como búsqueda en el espacio de planes

- ¿Cómo diseñar un algoritmo para encontrar planes parciales finales?
 - Idea POP: comenzar en el plan inicial y aplicar transformaciones a los planes parciales, refinándolos
 - Básicamente, estas transformaciones consisten en resolver precondiciones abiertas o resolver amenazas
- En cada punto habrá varias alternativas de refinamiento, y no todas conducen hacia un plan parcial final
- Problema: encontrar la secuencia de refinamiento que partiendo del plan parcial inicial, llegue a un plan parcial solución (es decir, sin ciclos, sin amenazas y sin precondiciones abiertas)
 - ¡Es una búsqueda en espacios de estados!
- Pero ahora *los estados son los planes parciales y los operadores son los refinamientos*

Algoritmo POP para el ejemplo de aprobar IA1

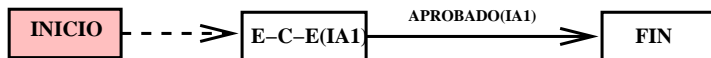
- Convenios en la representación gráfica del ejemplo, por claridad en la misma:
 - Algunas restricciones de orden no aparecerán, en particular la asociadas a enlaces causales
 - Las precondiciones y efectos no aparecerán. En particular, no aparecerán las precondiciones abiertas
 - Cuando una acción no tenga precondiciones abiertas, ésta aparecerá sombreada
- Paso 1: plan inicial



- Precondiciones abiertas: **APROBADO (IA1)** , **EN (CASA)** ; vamos a resolver primero **APROBADO (IA1)**
- Una única posibilidad: acción nueva **EXAMINARSE-CON-EXITO(IA1)**

Algoritmo POP para el ejemplo de aprobar IA1

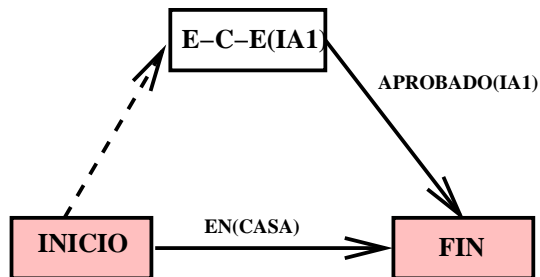
- Paso 2:



- Resolvemos ahora la precondition **EN (CASA)** de FIN
- Dos posibilidades: INICIO ó acción nueva IR(ETSII,CASA)
- Consideraremos la primera de ella en primer lugar, intentando la segunda en caso de fallo
- Cuando se usa (para cerrar una precondition) una acción que ya está en el plan, se denomina *establecimiento simple*

Algoritmo POP para el ejemplo de aprobar IA1

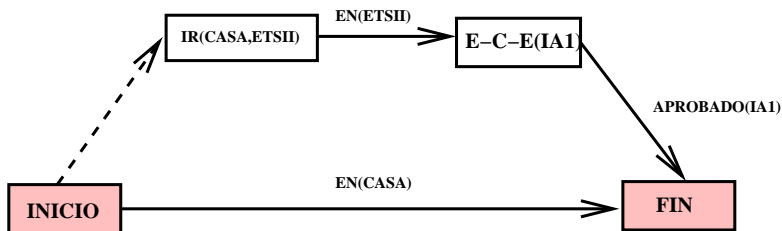
- Paso 3:



- Resolvemos ahora la precondition **EN(ETSII)** de **EXAMINARSE-CON-EXITO(IA1)**
- Sólo una posibilidad: acción nueva **IR(CASA,ETSII)**

Algoritmo POP para el ejemplo de aprobar IA1

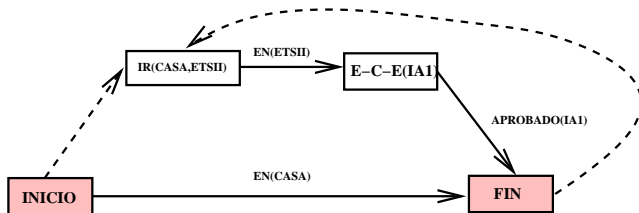
- Paso 4:



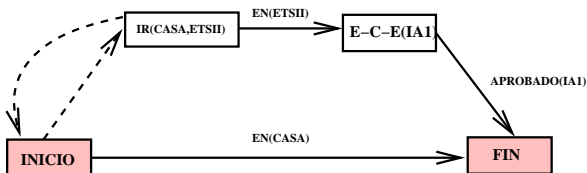
- Aparece una amenaza: de IR(CASA,ETSII) sobre el enlace causal entre INICIO y FIN
- Intentamos resolver la amenaza
 - obligando a que IR(CASA,ETSII) vaya después de FIN (*promoción*), o bien
 - obligando a que IR(CASA,ETSII) vaya antes que INICIO (*degradación*)

Algoritmo POP para el ejemplo de aprobar IA1

- Pero en ambos casos, se crearía un ciclo:
 - Promoción:

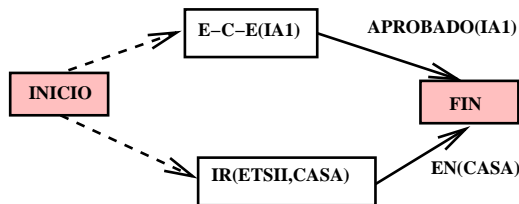


- Degradación:



Algoritmo POP para el ejemplo de aprobar IA1

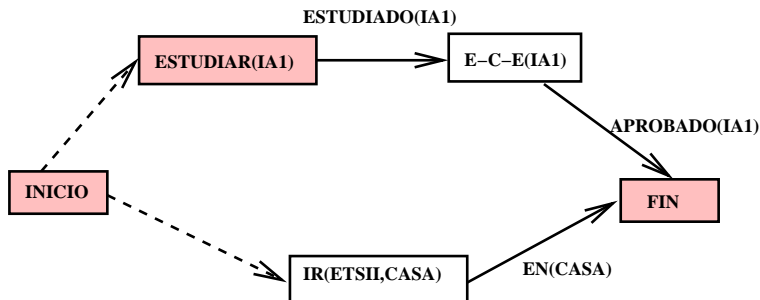
- Por tanto, hay FALLO y hemos de retroceder al anterior punto de alternativa, que estaba en el paso 2
 - Y elegir la otra alternativa: acción nueva IR(ETSII,CASA) para resolver la precondition **EN (CASA)** de FIN
- Paso 5:



- Elegimos ahora la precondition **ESTUDIADO (IA1)** de **EXAMINARSE-CON-EXITO(IA1)**
 - La única posibilidad para resolverla es acción nueva **ESTUDIAR(IA1)**

Algoritmo POP para el ejemplo de aprobar IA1

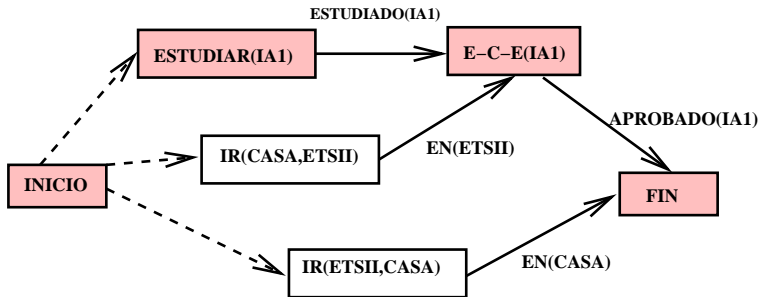
- Paso 6:



- Elegimos ahora la precondición **EN(ETSII)** de EXAMINARSE-CON-EXITO(IA1)
 - La única posibilidad para resolverla es acción nueva IR(CASA,ETSII)

Algoritmo POP para el ejemplo de aprobar IA1

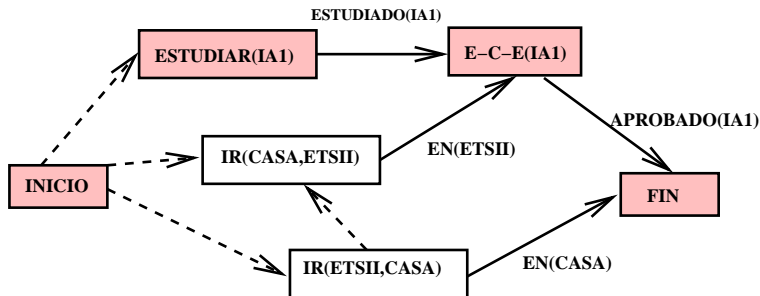
- Paso 7:



- Ahora existen amenazas : por ejemplo, de IR(ETSII,CASA) sobre el enlace causal entre IR(CASA,ETSII) y EXAMINARSE-CON-EXITO(IA1)
- Intentamos resolver esa amenaza por degradación; si no diera resultado, intentamos promoción

Algoritmo POP para el ejemplo de aprobar IA1

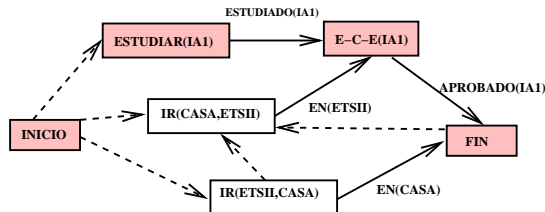
- Paso 8 (degradación):



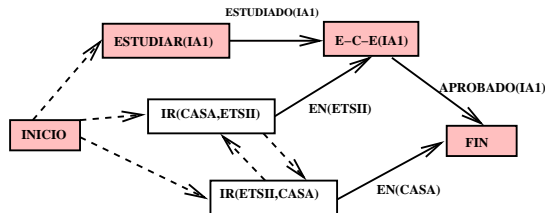
- Otra amenaza: de IR(CASA,ETSII) sobre el enlace causal entre IR(ETSII,CASA) y FIN
- Intentamos resolver la amenaza por promoción o por degradación

Algoritmo POP para el ejemplo de aprobar IA1

- Pero en ambos casos, se crearía un ciclo:
 - Promoción:

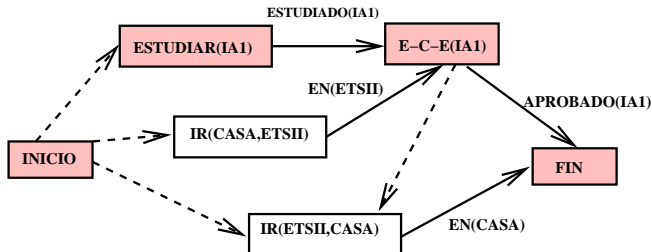


- Degradación:



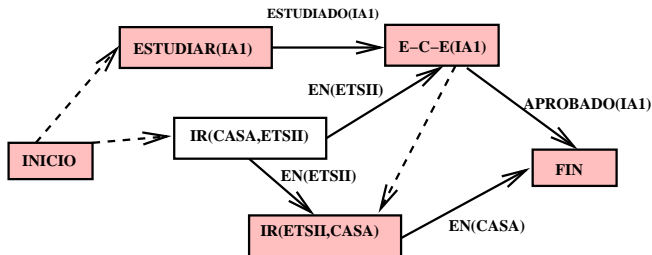
Algoritmo POP para el ejemplo de aprobar IA1

- Por tanto, hay FALLO
 - Hemos de retroceder al anterior punto de ramificación (paso 7), y elegir la otra alternativa: resolver la amenaza de IR(ETSII,CASA) sobre el enlace causal entre IR(CASA,ETSII) y EXAMINARSE-CON-EXITO(IA1), mediante promoción
- Paso 9:



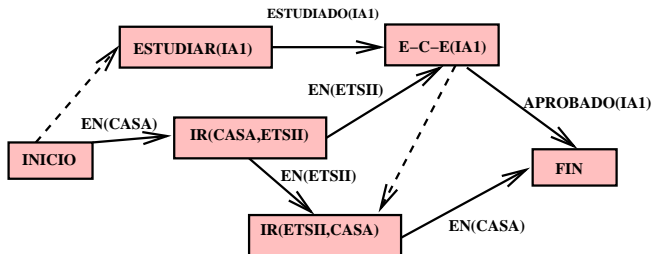
Algoritmo POP para el ejemplo de aprobar IA1

- Elegimos ahora la precondition **EN**(ETSII) de IR(ETSII,CASA)
 - Dos alternativas: establecimiento simple con IR(CASA,ETSII) o acción nueva con IR(CASA,ETSII).
 - Elegimos la primera, reconsiderándolo si fuera necesario
- Paso 10:



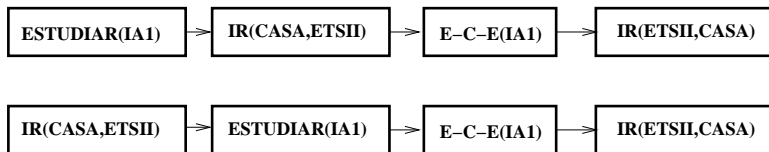
Algoritmo POP para el ejemplo de aprobar IA1

- Elegimos la precondition **EN (CASA)** de IR(CASA,ETSII)
 - Tres alternativas: establecimiento simple con INICIO, establecimiento simple con IR(ETSII,CASA) o acción nueva con IR(ETSII,CASA).
 - Elegimos la primera, reconsiderándolo si fuera necesario
- Paso 11 (Plan parcial final):



Algoritmo POP para el ejemplo de aprobar IA1

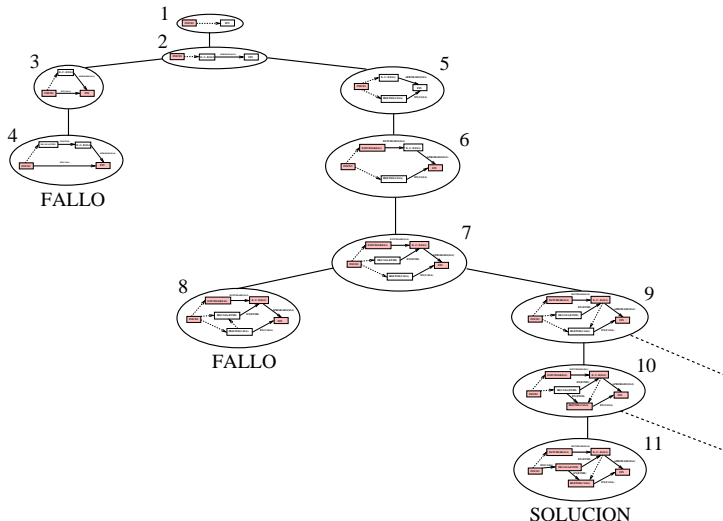
- Finalmente, secuenciamos (dos posibilidades):



Planificación de orden parcial: búsqueda

- Como se ha visto en el ejemplo anterior, es posible plantear un algoritmo de planificación como una búsqueda
 - Es decir, como búsqueda de una secuencia de transformaciones de refinamiento de planes que a partir del plan inicial obtiene un plan parcial final
- Un algoritmo POP es simplemente un algoritmo que aplica un algoritmo de búsqueda (en profundidad, por ejemplo) en el espacio de los planes parciales

Búsqueda POP para el ejemplo de aprobar IA1



Transformaciones de la búsqueda (refinamiento de planes parciales)

- Resolución de precondiciones abiertas: dada una precondición abierta p de una acción B del plan, por cada acción A que tiene a p como efecto, se puede obtener un plan sucesor (*refinar*) aplicando alguno de los siguientes pasos (siempre que el plan resultante no tenga ciclos):
 - Establecimiento simple: si la acción A ya aparece en el plan, añadir la restricción $A \prec B$ y el enlace causal $A \xrightarrow{p} B$
 - Acción nueva: añadir la acción A al plan, las restricciones $A \prec B$, INICIO $\prec A$ y $A \prec$ FIN, y el enlace causal $A \xrightarrow{p} B$ (es posible incluso introducir como nueva una acción igual a una que ya estuviera, entendiéndose que se trata de un *paso* diferente dentro del plan)

Transformaciones (refinamiento de planes parciales)

- Resolución de conflictos: dado un conflicto entre el enlace causal $\mathbf{A} \xrightarrow{P} \mathbf{B}$ y la acción \mathbf{C} , se puede obtener un plan sucesor aplicando alguno de los siguientes pasos (siempre que el plan resultante no tenga ciclos):
 - Añadir la restricción $\mathbf{B} \prec \mathbf{C}$ (promoción)
 - O añadir la restricción $\mathbf{C} \prec \mathbf{A}$ (degradación)

POP como búsqueda

- Arbol de búsqueda:
 - Nodos: planes parciales
 - Ramificación por la elección de acciones que resuelven precondiciones abiertas
 - Ramificación por las dos maneras de resolver las amenazas (promoción y degradación)
- No es necesario considerar un sucesor por cada precondición abierta del plan, ni por cada amenaza existente, basta considerarlas en pasos posteriores, siguiendo un orden dado
 - Aunque el orden en que se haga esto sí influye en la eficiencia de la búsqueda
- La búsqueda en profundidad podría entrar en una rama infinita
 - Por eso hemos de hacer una búsqueda acotada (por un número máximo de acciones en el plan)

Algoritmo POP (versión recursiva)

FUNCION POP (ESTADO-INICIAL, OBJ, ACCIONES, COTA)

1. Hacer POP-REC (PLAN-INICIAL (ESTADO-INICIAL, OBJ), ACCIONES, COTA)

FUNCION POP-REC (PLAN, ACCIONES, COTA)

1. Si PLAN es final, devolver SECUENCIACION (PLAN) y terminar.

2. Si existe en PLAN una amenaza de la acción C sobre A --p-->B, entonces

2.1 Sean PLANPR obtenido de PLAN mediante promoción en la amenaza anterior

2.2 Si PLANPR tiene ciclos, sea RESULTADO igual a FALLO; en caso contrario, sea RESULTADO igual a POP-REC (PLANPR, COTA).

2.3 Si RESULTADO no es FALLO, devolver RESULTADO y terminar; en otro caso:

2.3.1 Sea PLANDEG obtenido de PLAN mediante degradación en la amenaza anterior

2.3.2 Si PLANDEG tiene ciclos, sea RESULTADO igual a FALLO; en caso contrario, sea RESULTADO igual a POP-REC (PLANDEG, COTA).

2.3.3 Devolver RESULTADO y terminar.

(continúa...)

Algoritmo POP (versión recursiva, cont.)

3. Sea P una precondition abierta en $PLAN$
4. Para cada acción A de $ACCIONES$ (nueva o ya existente) que tiene a P como efecto, hacer:
 - 4.1 Hacer $PLANEXT$ el resultado de cerrar la precondition P en $PLAN$ mediante un enlace causal con A (si fueran acción nueva, añadir también la acción al $PLAN$)
 - 4.2 Si $PLANEXT$ no tiene ciclos y un número de acciones menor o igual que $COTA$, entonces:
 - 4.2.1 Sea $RESULTADO$ igual $POP-REC(PLANEXT, ACCIONES, COTA)$
 - 4.2.2 Si $RESULTADO$ no es FALLO, devolver $RESULTADO$ y terminar
5. Devolver FALLO

POP: algunas consideraciones

- Hemos presentado una versión simplificada del algoritmo. Falta:
 - Uso de unificación
 - Algunas variables podrían quedar sin instanciar (principio de mínimo compromiso)
 - Restricciones de desigualdad
- El algoritmo es correcto y completo (siempre que se tome una cota suficientemente grande)
- El pseudocódigo presentado resuelve primero las amenazas que las precondiciones abiertas, pero no necesariamente tiene que ser así
 - Ni tampoco necesariamente hay que intentar primero promoción y luego degradación

POP: uso de heurísticas

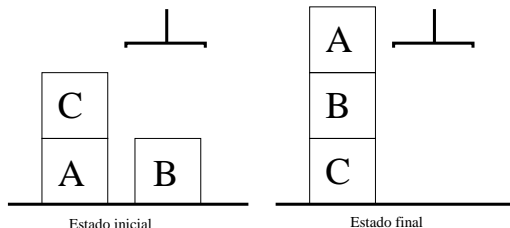
- Es difícil estimar la “cercanía” de un plan parcial a una solución
- Hay determinadas elecciones en el algoritmo que, bien tomadas, pueden influir positivamente en la eficiencia:
 - En cada paso, ¿qué amenaza o qué precondition abierta se refina?
 - Si se elige cerrar una precondition ¿en qué orden se intentan las acciones que la cierran?
 - Si se elige arreglar una amenaza ¿en qué orden se intenta promoción y degradación?

POP: algunas heurísticas

- Heurística para seleccionar el siguiente paso de refinamiento
 - Seleccionar la amenaza o precondition abierta que puede ser resuelta con el *menor* número de alternativas
 - Recuérdese que una amenaza tiene dos posibles alternativas de refinamiento
 - Y una precondition abierta, tantas como acciones (nuevas o ya existentes) la tengan como efecto
- Heurística para el orden de las acciones que cierran una precondition abierta
 - Ordenarla por el tamaño de sus listas de precondition (no muy buena)
 - Usar el *grafo de planificación*

La anomalía de Sussman

- Problema de planificación que mostraba las carencias de los primeros algoritmos de planificación (años 70):



- Estados:

Estado inicial = { SOBRELAMESA(A), SOBRELAMESA(B),
 DESPEJADO(B), DESPEJADO(C), BRAZOLIBRE(), SOBRE(C,A) }
Objetivo = { SOBRE(A,B), SOBRE(B,C) }

La anomalía de Sussman

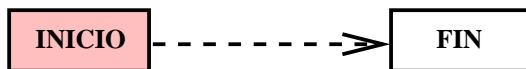
- La anomalía de Sussman fue un ejemplo pionero que mostraba que planificar no era una tarea trivial
- Por ejemplo, un planificador que primero buscara un subplan solución para el primer literal del objetivo **SOBRE (A, B)** y a continuación le concatenara una solución para **SOBRE (B, c)** sería fallido
 - Ya que el segundo subplan deshacería lo conseguido por el primero
- Fue un ejemplo que señalaba la no conveniencia de los planificadores *no entrelazados* (como el planificador *STRIPS*)

Anomalía de Sussman en POP

- Veremos cómo la planificación de orden parcial resuelve la anomalía de Sussman
 - Nota importante: por simplificar la presentación, en cada momento escogeremos transformaciones de refinamiento adecuadas
 - Es decir, nuestra “búsqueda” no tendrá fallos, aunque en la práctica esto es irreal
 - También por simplificar, a veces se aplicarán varios pasos de una vez

Anomalía de Sussman en POP

- Plan inicial:

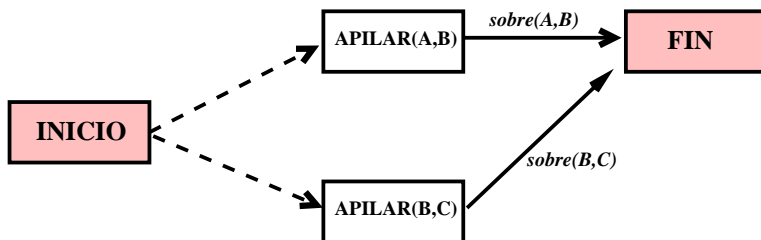


- Resolver la precondition $sobre(A,B)$ de FIN mediante la acción nueva APILAR(A,B) (es la única opción):



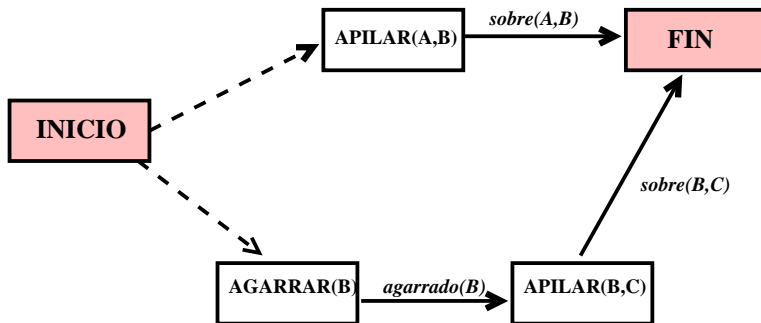
Anomalía de Sussman en POP

- Resolver la precondition $sobre(B,C)$ de FIN mediante la acción nueva APILAR(B,C) (es la única opción):



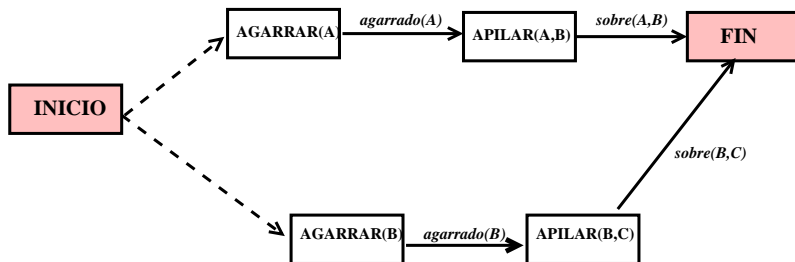
Anomalía de Sussman en POP

- Resolver la precondition $agarrado(B)$ de $APILAR(B,C)$ mediante la acción nueva $AGARRAR(B)$
 - También se podría usar $DESAPILAR$, opción que habría que considerar si la elección realizada fallara (búsqueda)



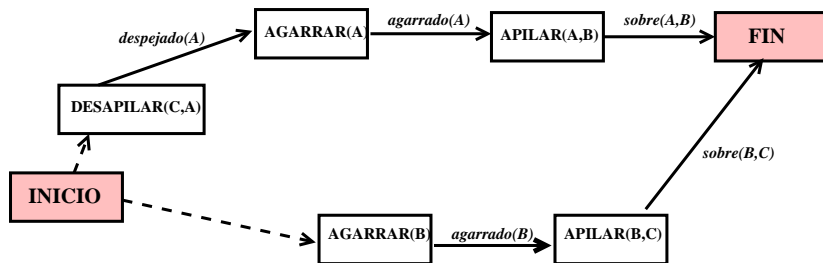
Anomalía de Sussman en POP

- Resolver la precondition $agarrado(A)$ de $APILAR(A,B)$ mediante la acción nueva $AGARRAR(A)$
 - Como en el caso anterior, también se podría usar $DESAPILAR$ (explorar mediante búsqueda)



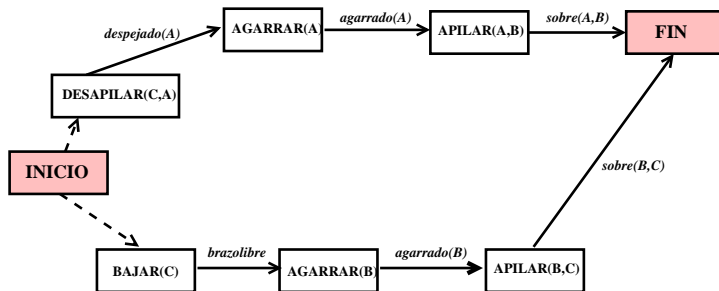
Anomalía de Sussman en POP

- Resolver la precondition $despejado(A)$ de AGARRAR(A) mediante la acción nueva desapilar(C,A):



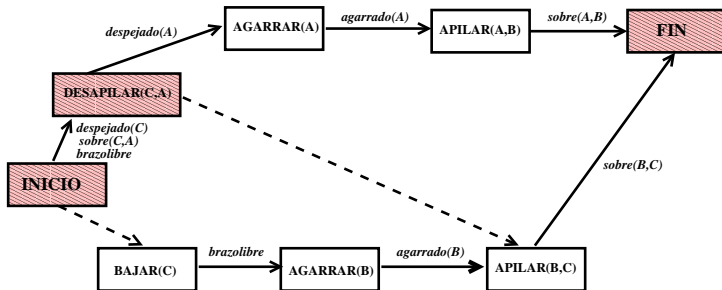
Anomalía de Sussman en POP

- Resolver la precondition *brazolibre* de AGARRAR(B) mediante la acción nueva bajar(C) (existen más opciones):



Anomalía de Sussman en POP

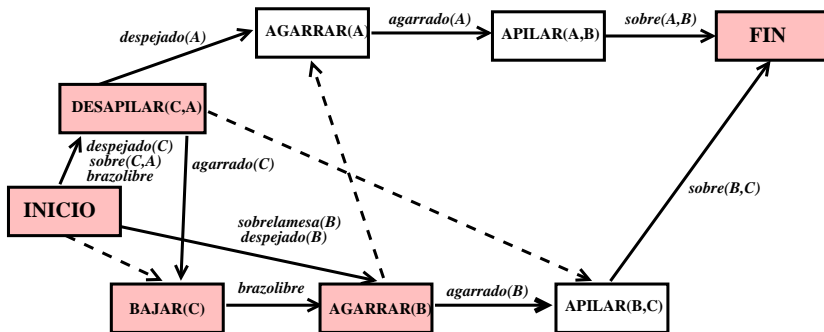
- Usar la acción INICIO para, mediante establecimiento simple, resolver todas las precondiciones de DESAPILAR(C,A) (existen otras opciones):
 - El enlace causal entre INICIO y DESAPILAR(C,A) que asegura *despejado(C)*, se ve amenazado por APILAR(B,C): resolver mediante promoción (única opción), colocando la restricción de que APILAR(B,C) debe ir tras DESAPILAR(C,A)



Anomalía de Sussman en POP

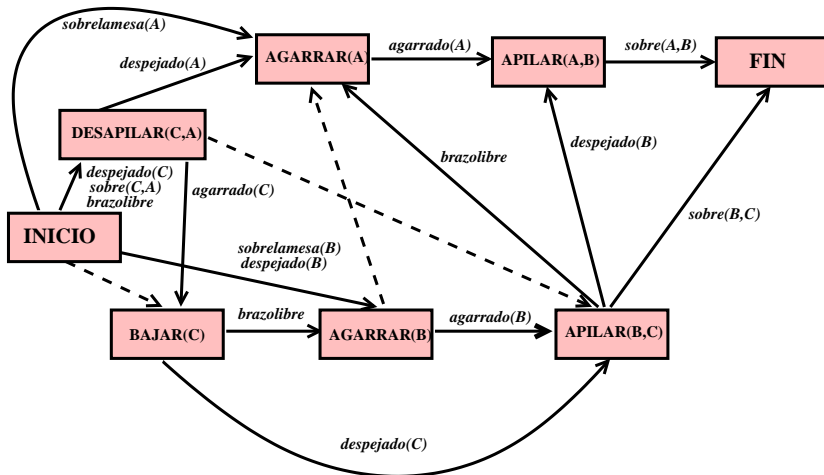
- Usar la acción INICIO para, mediante establecimiento simple, resolver las restantes precondiciones de AGARRAR(B); igualmente mediante establecimiento simple, usar DESAPILAR(C,A) para resolver la precondición *agarrado(C)* de BAJAR(C)
 - El enlace causal entre BAJAR(C) y AGARRAR(B) que asegura *brazolibre*, se ve amenazado por AGARRAR(A): resolver mediante promoción, colocando la restricción de que AGARRAR(A) debe ir tras AGARRAR(B)
 - De esta manera, también se resuelve la amenaza de APILAR(A,B) al enlace causal entre INICIO y AGARRAR(B), ya que la anterior restricción implica que APILAR(A,B) ocurre después de AGARRAR(B)

Anomalía de Sussman en POP



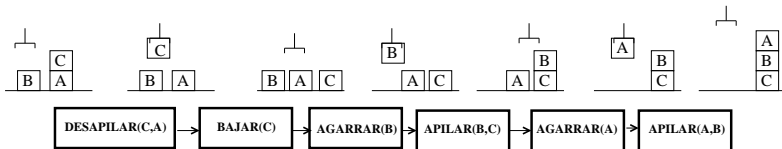
Anomalía de Sussman en POP

- Las restantes condiciones abiertas se pueden resolver mediante establecimiento simple, como indica la figura



Anomalía de Sussman en POP

- El plan parcial resultante es una solución: no tiene conflictos, no tiene ciclos y no hay precondiciones abiertas
- Secuenciación del plan:



- Nótese que se intercalan los pasos necesarios para solucionar los dos objetivos originales

Otras aproximaciones a la planificación

- SATPLAN, uso de técnicas basadas en lógica proposicional
- GRAPHPLAN
- Planificación como PSR
- Planificación con recursos limitados
- Planificación con tiempo explícito
- Planificación mediante descomposición jerárquica
- Planificación en entornos con incertidumbre
- Ejecución del plan: vigilancia y replanificación
- Planificación continua
- Planificación multiagente

Bibliografía

- Russell, S. y Norvig, P. *Artificial Intelligence (A Modern Approach)* (Prentice–Hall, 2010). Third Edition
 - Cap. 10: “Planning”.
- Russell, S. y Norvig, P. *Inteligencia Artificial (un enfoque moderno)* (Pearson Educación, 2004). Segunda edición
 - Cap. 11: “Planificación”.
- Ghallab M., Nau D. y Traverso P. *Automated Planning: theory and practice* (Elsevier, 2004)
 - Sec. 2.3 “Classical representation”.
 - Cap. 4 “State space planning”.
 - Cap. 5 “Plan space planning”.
 - Cap. 9 “Heuristics in planning”.