



UNIVERSIDAD  
DE GRANADA



# Técnicas de los Sistemas Inteligentes

## Tema 3: Sistemas de Planificación en Inteligencia Artificial

**E.T.S. de Ingenierías Informática y de Telecomunicación**  
**Universidad de Granada**

*Este documento está protegido por la Ley de Propiedad Intelectual ([Real Decreto Ley 1/1996 de 12 de abril](#)).  
Queda expresamente prohibido su uso o distribución sin autorización del autor.*

© Antonio González

[A.Gonzalez@decsai.ugr.es](mailto:A.Gonzalez@decsai.ugr.es)

Departamento de Ciencias de la  
Computación e Inteligencia Artificial  
<http://decsai.ugr.es>

# Objetivos

- Conocer los **sistemas de planificación en Inteligencia Artificial** como herramientas que permiten resolver problemas en distintos ámbitos.
- Analizar la **complejidad de los problemas reales** y la dificultad de resolverlos con técnicas de búsqueda sin el uso eficiente del conocimiento del problema.
- Estudiar algunos sistemas de planificación por **progresión y por regresión**.
- Estudio de otros modelos de planificación como la **planificación de orden parcial o la planificación jerárquica**.
- Conocer y manejar en problemas reales los **estándares de representación de problemas de planificación** a través del lenguaje PDDL.

# Bibliografía complementaria

- Nils J. Nilsson, “Inteligencia Artificial: Una nueva síntesis”, Ed. Mc Graw Hill, 2000.
- S. Russell, P. Norvig, Artificial Intelligence: A modern Approach, Tercera Edición, Ed. Pearson, 2010.
- S. Russell, P. Norvig, Artificial Intelligence: A modern Approach, Cuarta Edición, Ed. Pearson, 2022.

# Contenido



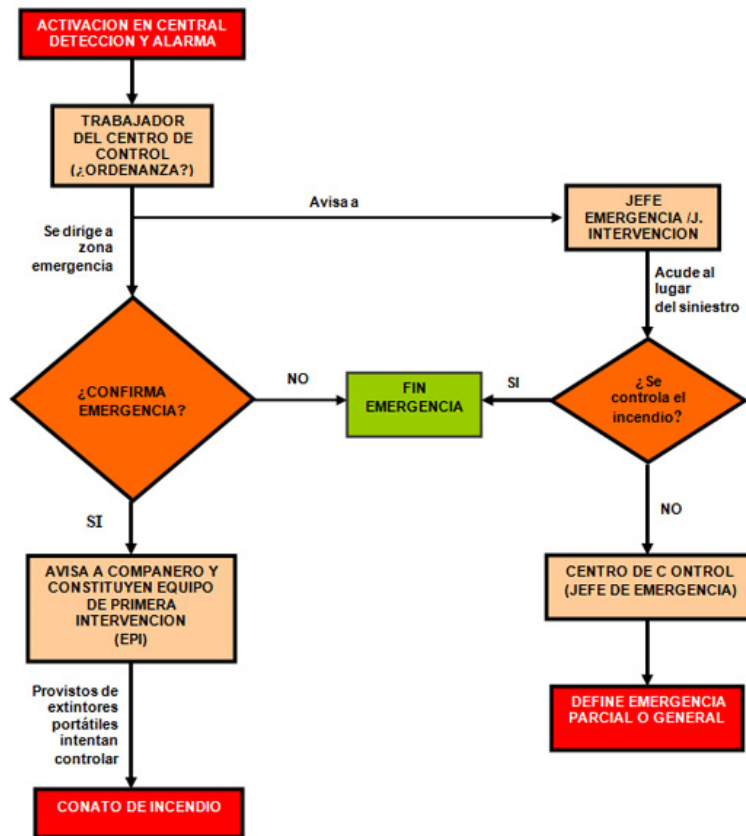
- Razonamiento sobre acciones y planificación clásica
- Lenguaje de planificación PDDL
- Planificación como búsqueda en un espacio de estados: STRIPS, HSP, FF
- Planificación como búsqueda en un espacio de planes
- Planificación jerárquica
- Representación para planes

# Razonamiento sobre acciones y planificación clásica

- ¿Qué es planificar?
- ¿Qué es un plan?
- Búsqueda y planificación



# Ejemplo: control de emergencias

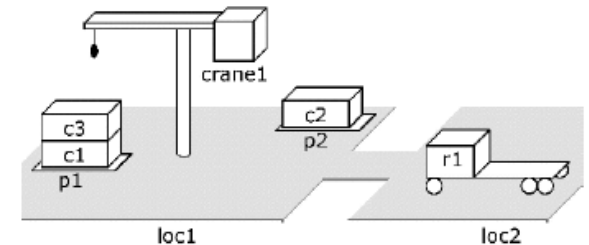


## Control de emergencias



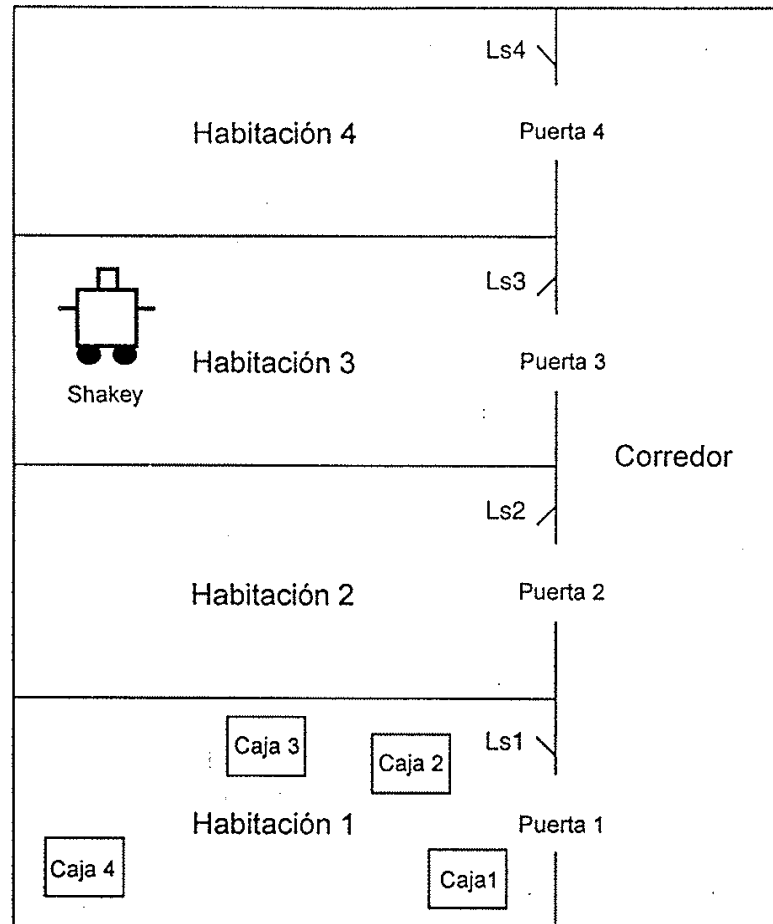
# Ejemplo: dominio logística

- 3 ciudades
- En cada ciudad hay una camión
- En cada ciudad hay un centro de carga/descarga de camiones, un centro de la ciudad y un aeropuerto
- Los camiones se mueven dentro de la ciudad
- Los aviones se mueven entre ciudades



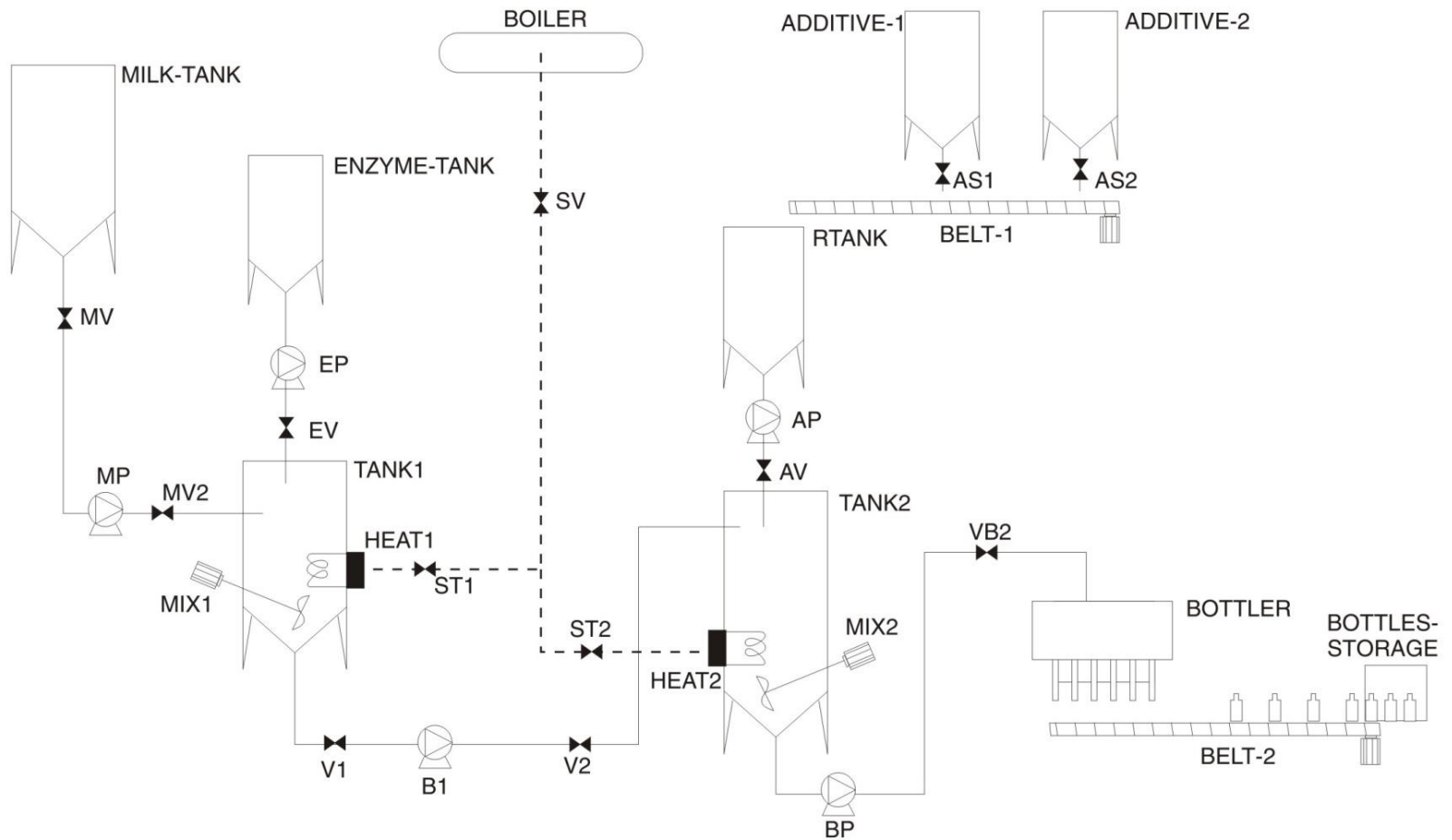
- 
- Inicio
    - Hay 3 paquetes en el centro de la ciudad 1
    - Hay 2 aviones en el aeropuerto de la ciudad 1
  - Final
    - Los paquetes 1 y 3 deben de estar en el centro de la ciudad 2
    - El paquete 2 debe de estar en el centro de la ciudad 3
-

# Ejemplo: Un robot móvil





# Ejemplo: Una planta industrial



# Problemas de la planificación

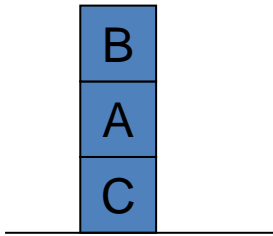
- Complejidad del mundo real
  - Búsqueda
  - Modelos basados en la lógica
- Otros problemas:
  - Problema del marco
  - Efectos dependientes del contexto
  - Problema de la cualificación



# Modelos previos a la planificación

- El cálculo de situaciones
- Reducción de diferencias

# El cálculo de situaciones



Deseamos que el agente desarrolle un plan para conseguir colocar algún bloque sobre el bloque B

Sobre(B,A)  
Sobre(A,C)  
Sobre(C,Suelo)  
Libre(B)  
Libre(Suelo)

$$(\exists x)Sobre(x, B)$$

# El cálculo de situaciones

- El cálculo de situaciones (Green 1969) es una formalización de los conceptos de estados, acciones y efectos de las acciones sobre los estados
- ¿Existe algún estado que satisfaga ciertas propiedades (objetivo) y, si es así, cómo se puede transformar, mediante acciones, el estado actual en ese estado?

# Variable de estado

$$Sobre(B, A) \wedge Sobre(A, C) \wedge Sobre(C, Suelo) \wedge \dots$$

¿cómo se puede representar el cambio de verdad en los predicados, cuando cambia el estado?

$$Sobre(B, A, s_0) \wedge Sobre(A, C, s_0) \wedge Sobre(C, Suelo, s_0) \wedge \dots$$

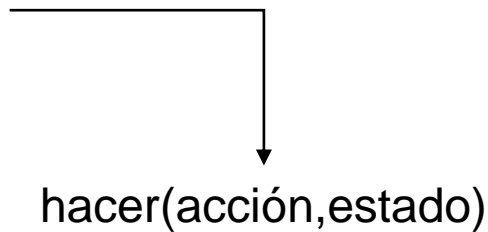
También es conveniente tener proposiciones ciertas para todos los estados

$$(\forall x, y, s)[Sobre(x, y, s) \wedge \neg(y = Suelo) \supset \neg Libre(y, s)]$$

$$(\forall s) Libre(Suelo, s)$$

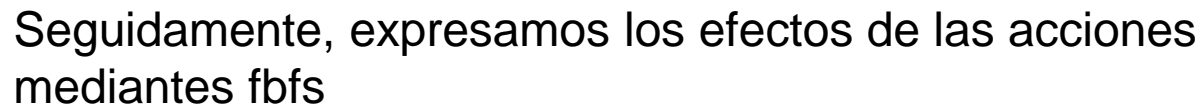
# Acciones

¿Cómo representar las acciones?  
Por ejemplo, mover(B,A,Suelo)



```
graph TD; A[¿Cómo representar las acciones?  
Por ejemplo, mover(B,A,Suelo)] --> B[hacer(acción,estado)];
```

hacer(acción,estado)



```
graph TD; B[hacer(acción,estado)] --> C[Seguidamente, expresamos los efectos de las acciones  
mediantes fbfs];
```

Seguidamente, expresamos los efectos de las acciones  
mediantes fbfs



# Efectos positivos y negativos

$$(\forall x, y, z, s)[Sobre(x, y, s) \wedge Libre(x, s) \wedge Libre(z, s) \wedge (x \neq z) \\ \supset Sobre(x, z, hacer(mover(x, y, z), s))]$$

$$(\forall x, y, z, s)[Sobre(x, y, s) \wedge Libre(x, s) \wedge Libre(z, s) \wedge (x \neq z) \\ \supset \neg Sobre(x, y, hacer(mover(x, y, z), s))]$$

fbfs para el par (mover, Sobre)

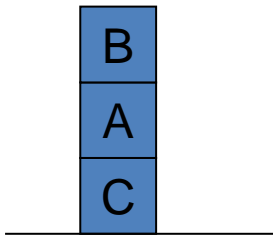
# Nuevos axiomas

$$(\forall x, y, z, s)[Sobre(x, y, s) \wedge Libre(x, s) \wedge Libre(z, s) \wedge (x \neq z) \wedge (y \neq z) \\ \supset Libre(y, hacer(mover(x, y, z), s))]$$

$$(\forall x, y, z, s)[Sobre(x, y, s) \wedge Libre(x, s) \wedge Libre(z, s) \wedge (x \neq z) \wedge (z \neq Suelo) \\ \supset \neg Libre(z, hacer(mover(x, y, z), s))]$$

fbfs para el par (mover, Libre)

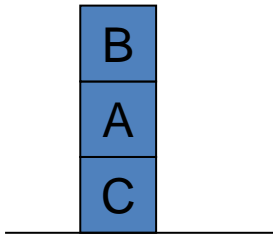
# Razonamiento



Sobre(B,A)  
Sobre(A,C)  
Sobre(C,Suelo)  
Libre(B)  
Libre(Suelo)

Sobre(B,Suelo,hacer(mover(B,A,Suelo),s0))  
 $\neg$ Sobre(B,A,hacer(mover(B,A,Suelo),s0))  
Libre(A,hacer(mover(B,A,Suelo),s0))

# Axiomas del marco



Sobre(B,A)  
Sobre(A,C)  
Sobre(C,Suelo)  
Libre(B)  
Libre(Suelo)

¿Si muevo B sobre el Suelo, en donde estará C?

Las acciones producen efectos locales y dejan muchos otros literales sin cambiar

# Axiomas del marco

$$Sobre(x, y, s) \wedge (x \neq u) \supset Sobre(x, y, hacer(mover(u, v, z), s))$$

$$\neg Sobre(x, y, s) \wedge [(x \neq u) \vee (y \neq z)] \supset \neg Sobre(x, y, hacer(mover(u, v, z), s))$$

Axiomas para el par (mover, Sobre)

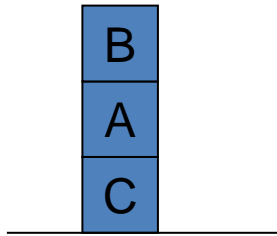
# Axiomas del marco

$$Libre(u, s) \wedge (u \neq z) \supset Libre(u, hacer(mover(x, y, z), s))$$

$$[\neg Libre(u, s) \wedge (u \neq y) \supset \neg Libre(u, hacer(mover(x, y, z), s))]$$

Axiomas para el par (mover, Libre)

# Ejemplo

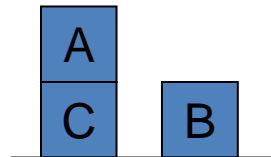


Sobre(B,A,s0)  
Sobre(A,C,s0)  
Sobre(C,Suelo,s0)  
Libre(B,s0)  
Libre(Suelo,s0)

s0

Mover(B,A,Suelo)

s1=hacer(mover(B,A,Suelo),s0))



# Ejemplo

- Inferido a partir de los axiomas de efecto:
  - $\text{Sobre}(\text{B}, \text{Suelo}, \text{hacer}(\text{mover}(\text{B}, \text{A}, \text{Suelo}), s_0))$
  - $\neg \text{Sobre}(\text{B}, \text{A}, \text{hacer}(\text{mover}(\text{B}, \text{A}, \text{Suelo}), s_0))$
  - $\text{Libre}(\text{A}, \text{hacer}(\text{mover}(\text{B}, \text{A}, \text{Suelo}), s_0))$
- Inferido a partir de los axiomas del marco
  - $\text{Sobre}(\text{A}, \text{C}, \text{hacer}(\text{mover}(\text{B}, \text{A}, \text{Suelo}), s_0))$
  - $\text{Sobre}(\text{C}, \text{Suelo}, \text{hacer}(\text{mover}(\text{B}, \text{A}, \text{Suelo}), s_0))$
  - $\text{Libre}(\text{B}, \text{hacer}(\text{mover}(\text{B}, \text{A}, \text{Suelo}), s_0))$
- Verdadero en todos los estados:
  - $(\text{forall } s) \text{ Libre}(\text{Suelo}, s)$



# Cualificaciones

- Acción mover
  - $\neg \text{No\_pesa\_demasiado}(x,s)$
  - $\neg \text{Pegado\_por\_debajo}(x,s)$
  - $\neg \text{No\_brazo\_roto}(s)$
  - etc.

# Generación de planes

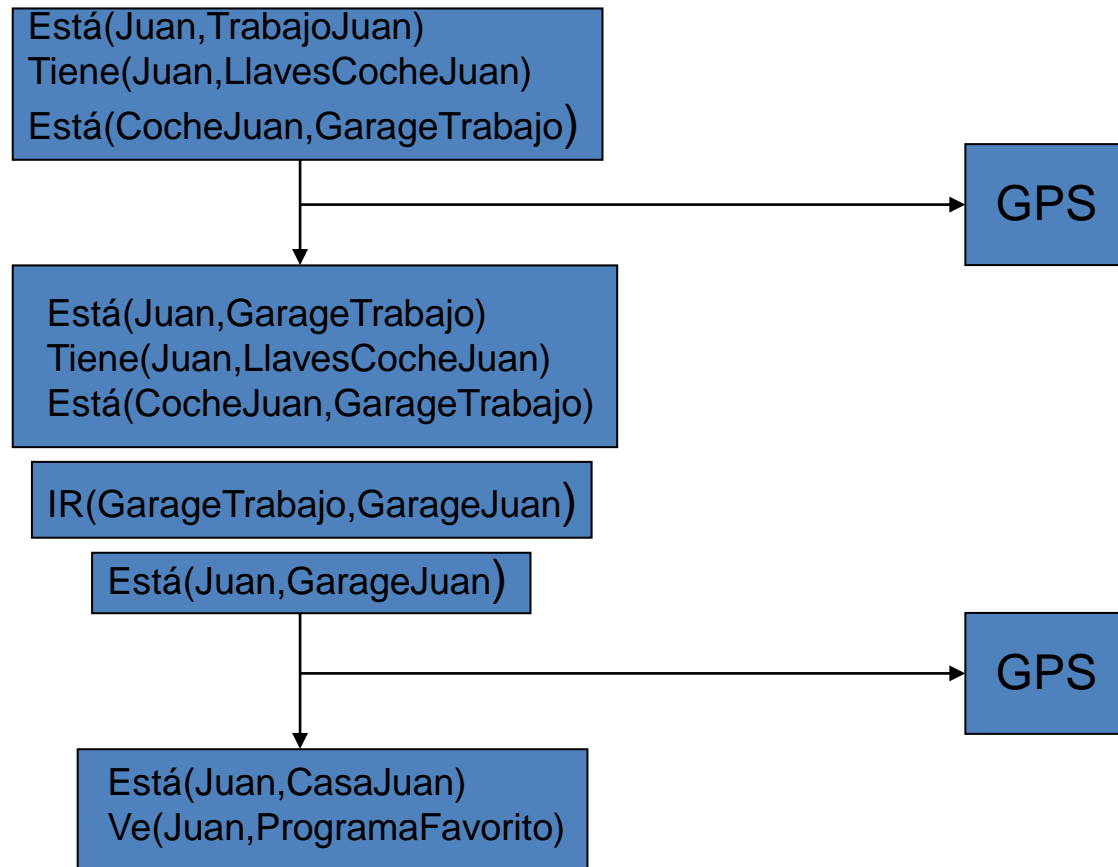
- Expresar el objetivo como una fbf

$$(\exists s)\gamma(s)$$

- B esté sobre el Suelo

$$(\exists s)\textit{Sobre}(B, \textit{Suelo}, s)$$

# Reducción de Diferencias



# Análisis Medios-Fines

- Encontrar una diferencia entre el objeto inicial y el objeto final. Si no hay diferencias, el problema está resuelto.
- Tomar la primera diferencia y encontrar un operador apropiado para reducirla.
- Comparar las precondiciones del operador seleccionado con el objeto inicial, encontrando diferencias, si no hay ninguna, se aplica de forma directa el operador, si hay, se tratará de reducirlas.
- Repetir el proceso tomando como objeto inicial el objeto producido por la aplicación del operador y el objeto final.

# Planificación clásica

1. El sistema tiene un número finito de estados (situaciones).
2. El sistema es completamente observable, es decir, se tiene un conocimiento completo del estado del sistema.
3. El sistema es determinista, es decir, la aplicación de una acción a un estado conduce siempre a un mismo estado.
4. El sistema es estático, es decir, el sistema permanece en el mismo estado hasta que se aplique una acción.
5. Los objetivos son conocidos antes de comenzar la planificación, es decir, los objetivos no cambian.
6. Un plan solución de un problema de planificación es una secuencia de acciones finita y linealmente ordenada.
7. No se contempla el razonamiento temporal y numérico, por lo que la calidad del plan se determina por el número de acciones del mismo.
8. La tarea de planificación consiste en construir un plan completo que satisfice el objetivo antes de la ejecución de cualquier parte del mismo.

# Un ejemplo para reflexionar

¿Podríamos considerar la tarea a la que se enfrenta los CSI en cada capítulo como un problema de planificación?

- Estado inicial
- Acciones
- Objetivo

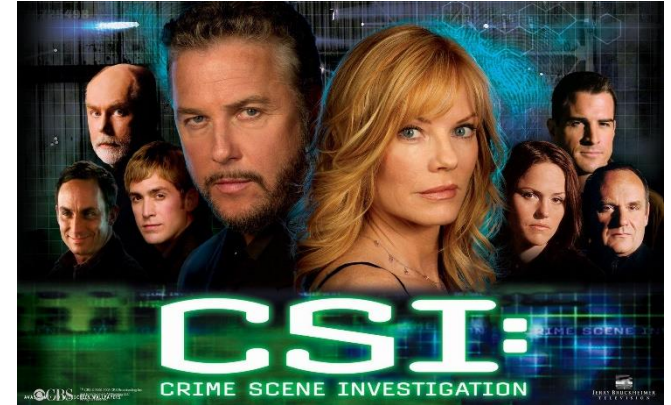


Imagen tomada de <https://wallpaperaccess.com/>

¿Sería un problema de planificación clásica?

# Planificación clásica

- Un problema de planificación clásica se formula a través de siguientes elementos:
  1. Un conjunto de fórmulas atómicas, denominadas hechos o literales, que representa la información relevante del problema.
  2. Un conjunto de operadores definidos en el dominio del problema.
  3. Un conjunto inicial de hechos que forman la situación inicial del problema.
  4. Un conjunto final de hechos que deben formar parte de la situación final del problema.
- El sistema STRIPS ha condicionado la mayoría de trabajos de planificación desde los comienzos de los años 70

# Shakey



<https://www.sri.com/hoi/shakey-the-robot/>

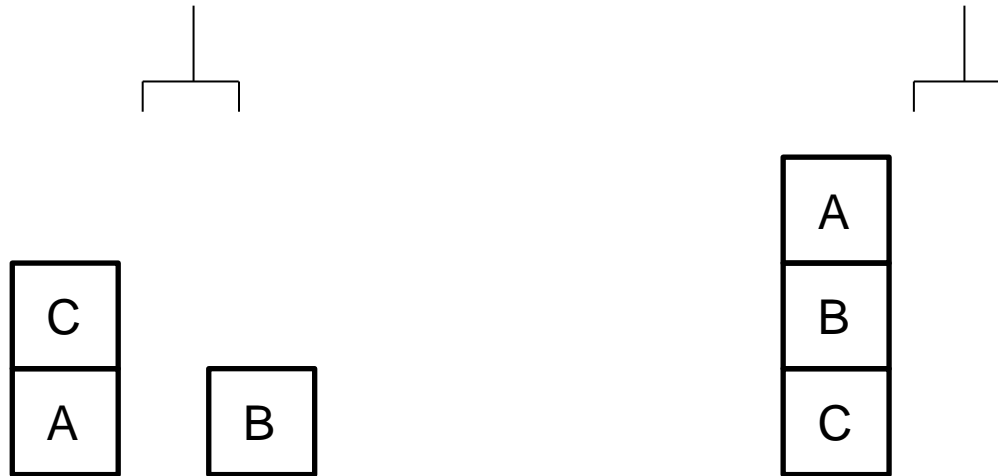
[http://en.wikipedia.org/wiki/Shakey\\_the\\_Robot](http://en.wikipedia.org/wiki/Shakey_the_Robot)



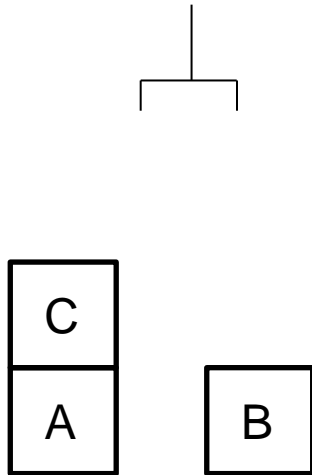
# STRIPS

- **Stanford Research Institute Problem Solver**  
(Fikes & Nilsson, 1971)
- Representación del conocimiento
  - Estados
    - Restricción
    - Hipótesis del Mundo Cerrado
  - Objetivos
    - Restricción
  - Operadores
    - Modelo de regla tipo STRIPS (esquema de acción)
- Resolución de problemas

# Estado/Objetivo



# Estado

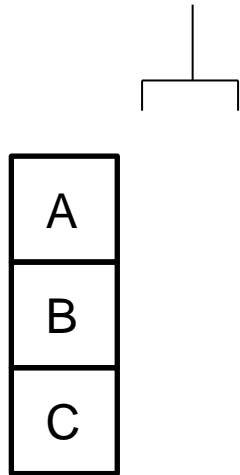


Predicados relevantes:

LIBRE(X)  
SOBRE(X,Y)  
SOBREMESA(X)  
MANOVACIA  
COGIDO(X)

$\text{LIBRE}(B) \wedge \text{SOBRE}(C,A) \wedge \text{SOBREMESA}(A) \wedge \text{LIBRE}(C) \wedge \text{MANOVACIA} \wedge \text{SOBREMESA}(B)$

# Objetivo



$\text{SOBRE}(A,B) \wedge \text{SOBRE}(B,C)$

# Modelo de regla tipo STRIPS

Nombre de la acción y variables usadas

Fórmula de Precondición

Lista de supresión

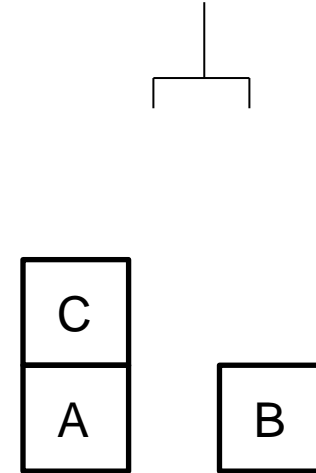
Lista de Adición

$$(a \in \text{ACTIONS}(s)) \Leftrightarrow s \models \text{PRECOND}(a)$$

$$\text{RESULT}(s, a) = (s - \text{DEL}(a)) \cup \text{ADD}(a)$$

# Modelo de regla tipo STRIPS

- Coger(x)
- FP: SOBREMESA(x), LIBRE(x), MANOVACIA
- LA: COGIDO(x)
- LS: SOBREMESA(x), LIBRE(x), MANOVACIA



$\text{LIBRE}(B) \wedge \text{SOBRE}(C,A) \wedge \text{SOBREMESA}(A) \wedge \text{LIBRE}(C) \wedge \text{MANOVACIA} \wedge \text{SOBREMESA}(B)$

$\text{LIBRE}(B) \wedge \text{SOBRE}(C,A) \wedge \text{SOBREMESA}(A) \wedge \text{LIBRE}(C) \wedge \text{MANOVACIA} \wedge \text{SOBREMESA}(B)$

$\text{SOBRE}(C,A) \wedge \text{SOBREMESA}(A) \wedge \text{LIBRE}(C) \wedge \text{COGIDO}(B)$

# El mundo de bloques

- Coger(x)
  - FP: SOBREMESA(x), LIBRE(x), MANOVACIA
  - LA: COGIDO(x)
  - LS: SOBREMESA(x), LIBRE(x), MANOVACIA
- Dejar(x)
  - FP: COGIDO(x)
  - LA: SOBREMESA(x), LIBRE(x), MANOVACIA
  - LS: COGIDO(x)
- Apilar(x,y)
  - FP: COGIDO(x), LIBRE(y)
  - LA: MANOVACIA, SOBRE(x,y), LIBRE(x)
  - LS: COGIDO(x), LIBRE(y)
- Desapilar(x,y)
  - FP: MANOVACIA, LIBRE(x), SOBRE(x,y)
  - LA: COGIDO(x), LIBRE(y)
  - LS: MANOVACIA, LIBRE(x), SOBRE(x,y)

# Lenguaje de planificación PDDL

- PDDL (Planning Domain Definition Language) es un intento de estandarizar los lenguajes para describir los problemas y los dominios de planificación.
- Fue desarrollado para permitir las competiciones entre planificadores.
- PDDL contiene lenguajes como STRIPS y ADL.



# Lenguaje de planificación PDDL

- PDDL (Planning Domain Definition Language) es un intento de estandarizar los lenguajes para describir los problemas y los dominios de planificación.
- Fue desarrollado para permitir las competiciones entre planificadores.
- PDDL contiene lenguajes como STRIPS y ADL.

# Lenguaje de planificación PDDL

- Una definición en PDDL consiste en:
  - Una definición de dominio.
  - Una definición del problema.
- Requerimientos:
  - :strips
  - :equality
  - :typing
  - :adl (uso de disyunciones y cuantificadores en precondiciones y objetivos, efectos condicionales,...)

# Definición del dominio

```
(define (domain DOMAIN_NAME)
  (:requirements [:strips] [:equality] [:typing] [:adl])
  (:predicates (PREDICATE_1_NAME [?A1 ?A2 ... ?AN])
               (PREDICATE_2_NAME [?A1 ?A2 ... ?AN])
               ...)
  (:action ACTION_1_NAME
    [:parameters (?P1 ?P2 ... ?PN)]
    [:precondition PRECOND_FORMULA]
    [:effect EFFECT_FORMULA]
  )
  (:action ACTION_2_NAME
    ...)
  ...)
```

# Definición del problema

```
define (problem PROBLEM_NAME) (:domain  
    DOMAIN_NAME)  
    (:objects OBJ1 OBJ2 ... OBJ_N)  
    (:init ATOM1 ATOM2 ... ATOM_N)  
    (:goal CONDITION_FORMULA) )
```

# Ejemplo

```
(define (problem hard1)
  (:domain strips-sliding-tile)
  (:objects t1 t2 t3 t4 t5 t6 t7 t8 p1 p2 p3)
  (:init
    (tile t1) (tile t2) (tile t3) (tile t4) (tile t5) (tile t6)
    (tile t7) (tile t8) (position p1) (position p2) (position p3)
    (inc p1 p2) (inc p2 p3) (dec p3 p2) (dec p2 p1)
    (blank p1 p1) (at t1 p2 p1) (at t2 p3 p1) (at t3 p1 p2)
    (at t4 p2 p2) (at t5 p3 p2) (at t6 p1 p3) (at t7 p2 p3)
    (at t8 p3 p3))
  (:goal
    (and (at t8 p1 p1) (at t7 p2 p1) (at t6 p3 p1)
          (at t4 p2 p2) (at t1 p3 p2)
          (at t2 p1 p3) (at t5 p2 p3) (at t3 p3 p3)))
  )
```

# Otras características de PDDL

- Acciones con duración.
- Expresiones y variables numéricas.
- Métricas del problema.
- Ventanas temporales.
- Restricciones duras y blandas sobre el plan.

<http://www.ida.liu.se/~TDDC17/info/labs/planning/2004/writing.html>

# Ejemplo

```
(define (domain DominioMaletín)
  (:requirements :strips :equality :typing :conditional-effects)
  (:types localizacion objetofisico)
  (constants (Maletin – objetofisico)
  (predicates (en ?x - objetofisico ?l - localizacion)
    (dentro ?x ?y - objetofisico))
  ...

  (:action muevemaletin
    :parameters (?m ?l – localizacion)
    :precondition (and (en Maletin ?m) (not (= ?m ?l)))
    :effect (and (en Maletin ?l) (not (en Maletin ?m))
      (forall (?z)
        (when (and (dentro ?z Maletin) (not (= ?z Maletin)))
          (and (en ?z ?l) (not (en ?z ?m))))))) )
```

# Transporte aéreo de cargas

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

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

$Action(Load(c, p, a),$

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



# El problema del neumático de repuesto

*Init*(*Tire*(*Flat*)  $\wedge$  *Tire*(*Spare*)  $\wedge$  *At*(*Flat*, *Axle*)  $\wedge$  *At*(*Spare*, *Trunk*))

*Goal*(*At*(*Spare*, *Axle*))

*Action*(*Remove*(*obj*, *loc*),

PRECOND: *At*(*obj*, *loc*)

EFFECT:  $\neg$  *At*(*obj*, *loc*)  $\wedge$  *At*(*obj*, *Ground*))

*Action*(*PutOn*(*t*, *Axle*),

PRECOND: *Tire*(*t*)  $\wedge$  *At*(*t*, *Ground*)  $\wedge$   $\neg$  *At*(*Flat*, *Axle*)

EFFECT:  $\neg$  *At*(*t*, *Ground*)  $\wedge$  *At*(*t*, *Axle*))

*Action*(*LeaveOvernight*,

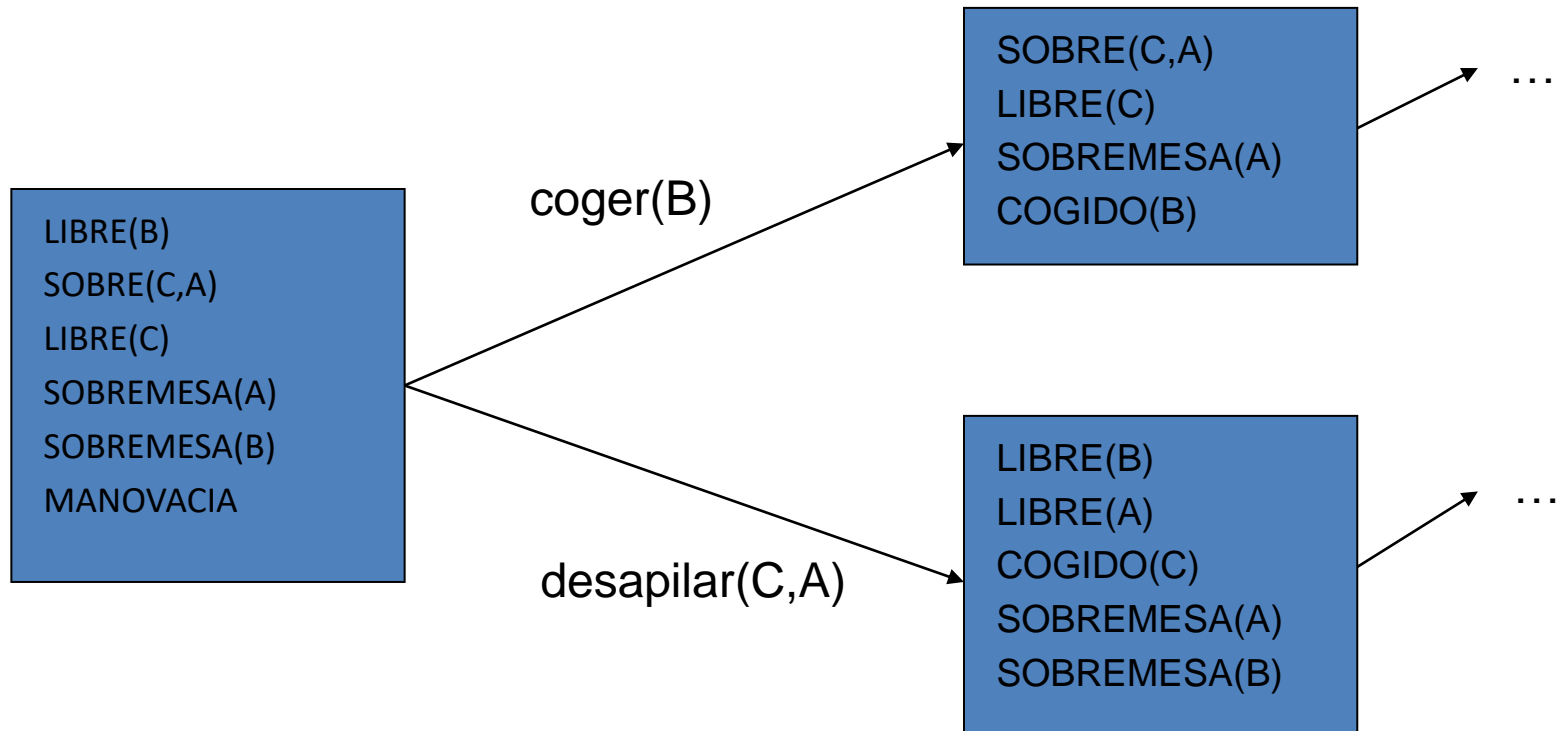
PRECOND:

EFFECT:  $\neg$  *At*(*Spare*, *Ground*)  $\wedge$   $\neg$  *At*(*Spare*, *Axle*)  $\wedge$   $\neg$  *At*(*Spare*, *Trunk*)  
 $\wedge$   $\neg$  *At*(*Flat*, *Ground*)  $\wedge$   $\neg$  *At*(*Flat*, *Axle*)  $\wedge$   $\neg$  *At*(*Flat*, *Trunk*))

# Planificación como búsqueda en un espacio de estados

- Progresión
- Regresión
- STRIPS

# Progresión



# Regresión

Objetivo

$$\{L \wedge G_1 \wedge \dots \wedge G_n\}$$

Operador D

Operador Inverso  
asociado a D

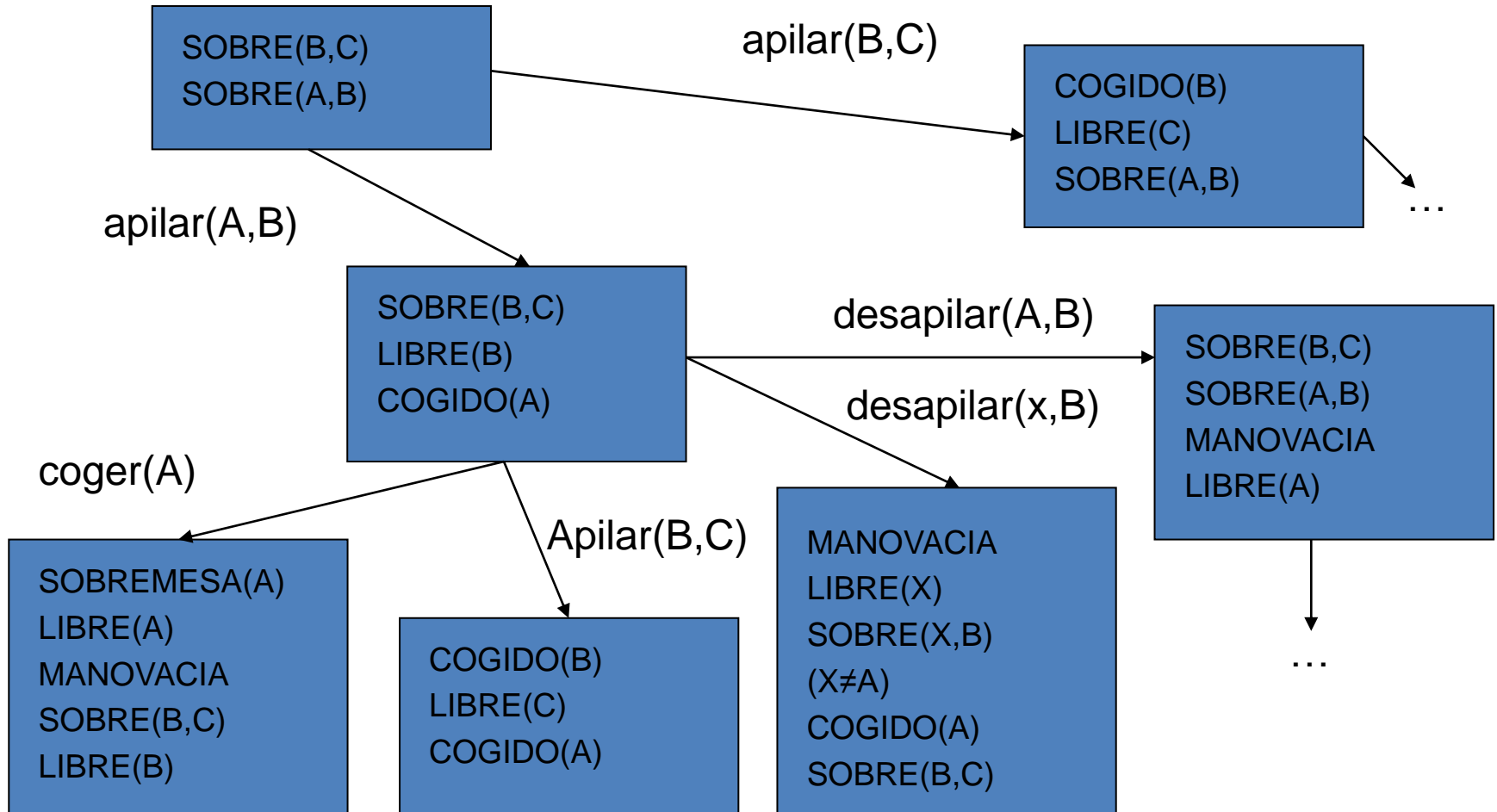
$$\{FPu \wedge G_1' \wedge \dots \wedge G_n'\}$$

$G_i'$  es la regresión de  $G_i$  a través del operador particularizado

# Regresión

- Si  $Q_u$  es un literal de la lista de adición del operador particularizado la regresión es  $V$
- Si  $Q_u$  es un literal de la lista de supresión del operador particularizado la regresión es  $F$
- En otro caso, la regresión es el  $Q_u$

# Ejemplo



# Heurísticas para planificación

- Ni los métodos hacía adelante ni hacia atrás podrían ser eficientes sin una buena función heurística.
- Uso de heurísticas basadas en métodos relajados:
  - Heurística basada en ignorar precondiciones.
  - Heurística basada en ignorar la lista de supresión.
- Descomposición de problemas.

# STRIPS

- Descomposición de los problemas.
- Incorporar operadores al plan relevantes, aunque no sean aplicables al estado actual.
- Representar simbólicamente el conocimiento sobre el dominio del problema.
- Razonamiento hacia atrás y hacia adelante.

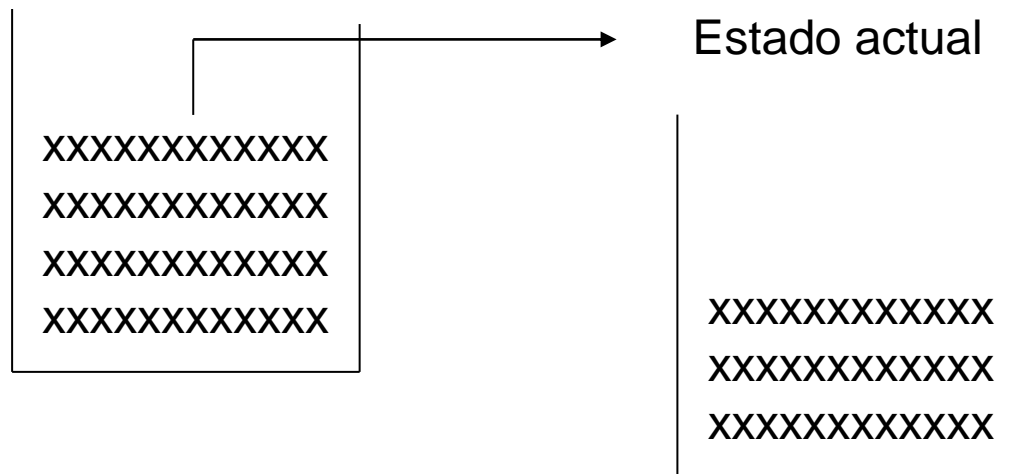


# STRIPS

- Representación del conocimiento mediante:
  - Una pila de objetivos
  - El estado actual del problema

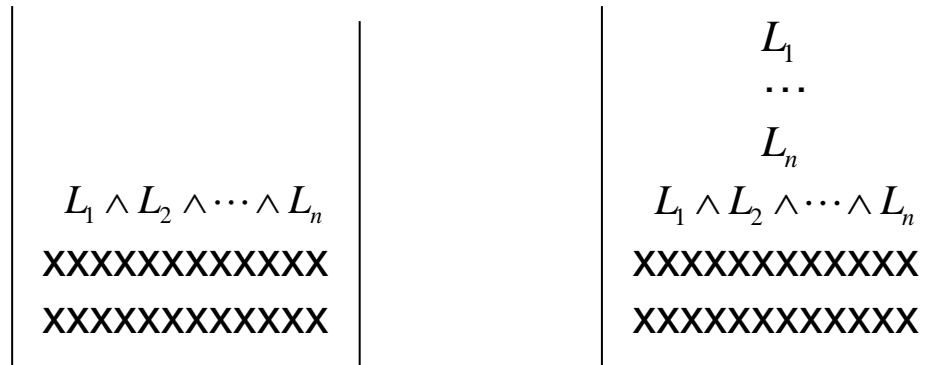
# STRIPS

**1. Emparejar:** si el objetivo de la parte superior de la pila empareja con el estado actual, se suprime este objetivo de la pila.



# STRIPS

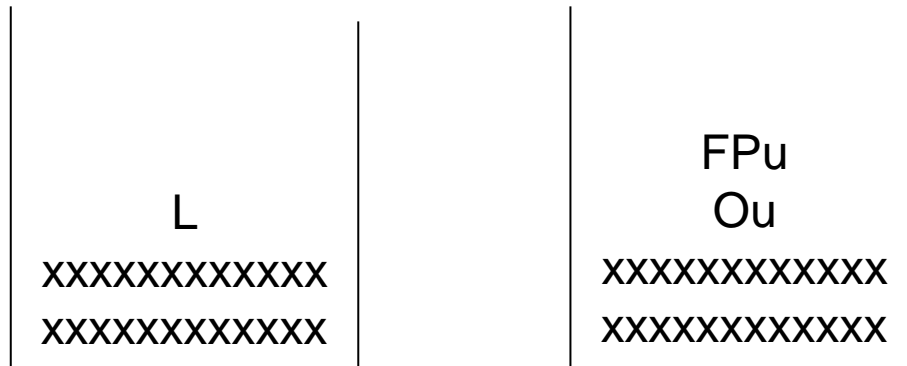
**2. Descomponer:** si el objetivo en la parte superior de la pila es compuesto, entonces añadir los literales componentes en la parte superior de la pila.



Dispositivo de Seguridad: interacciones

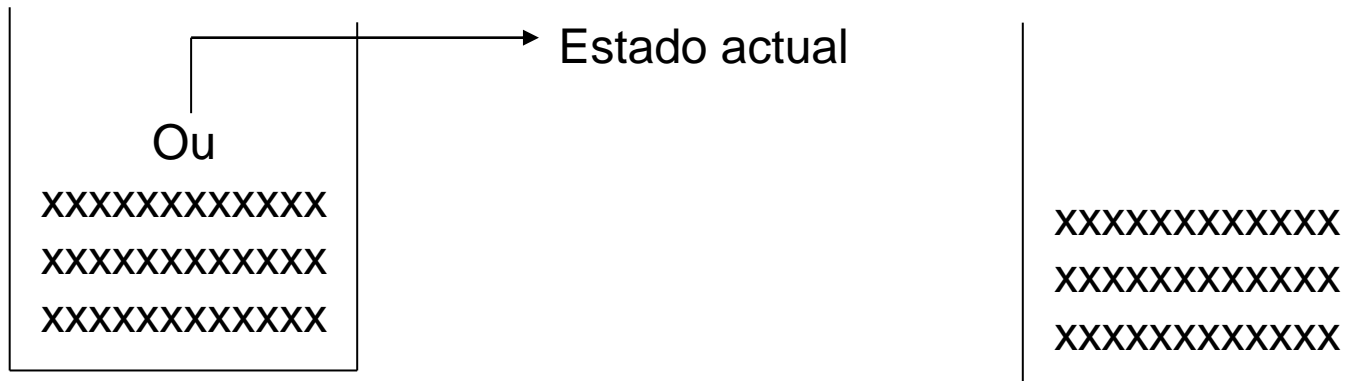
# STRIPS

**3. Resolver:** Cuando el objetivo que se encuentra en la parte superior de la pila es un único literal no resuelto, entonces se busca un operador cuya lista de adición contenga un literal que empareje con él.



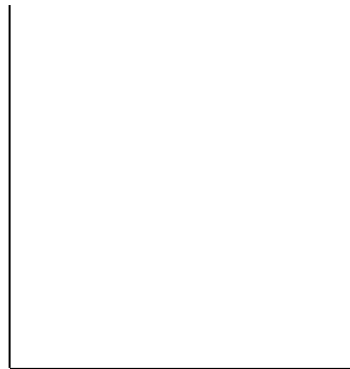
# STRIPS

**4. Aplicar:** Cuando el término en la parte superior de la pila es un operador, entonces se suprime de la pila, se aplica sobre el estado actual modificándolo y se anota en el plan.



# STRIPS

- El proceso se repite hasta que la pila se quede vacía



El estado debe de ser  
concordante con el objetivo

# Proceso de búsqueda asociado

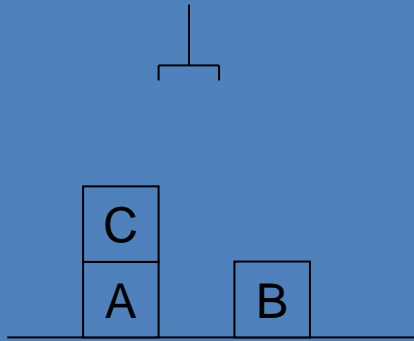
- Ordenación de las componentes de un objetivo compuesto.
- Elección entre las distintas particularizaciones posibles.
- Selección de la regla relevante, cuando hay más de una.

# Ejemplo 1





## Descripción de Estado

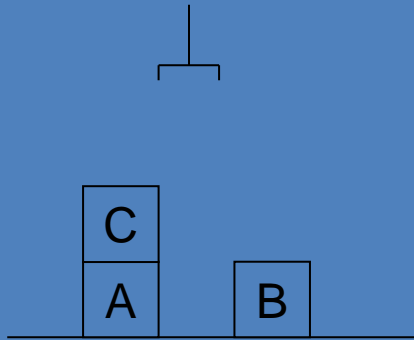


LIBRE(B)  
LIBRE(C)  
SOBRE(C,A)  
SOBREMESA(A)  
SOBREMESA(B)  
MANOVACIA

## Pila de Objetivos

SOBRE(C,B)  
SOBRE(A,C)  
SOBRE(C,B) $\wedge$ SOBRE(A,C)

## Descripción de Estado

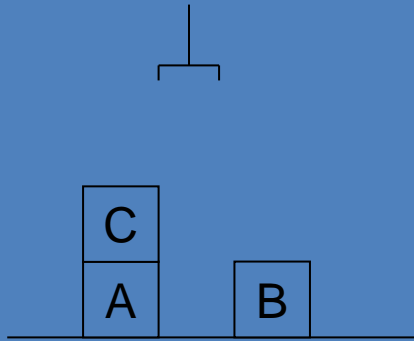


LIBRE(B)  
LIBRE(C)  
SOBRE(C,A)  
SOBREMESA(A)  
SOBREMESA(B)  
MANOVACIA

## Pila de Objetivos

LIBRE(B) $\wedge$ COGIDO(C)  
apilar(C,B)  
SOBRE(A,C)  
SOBRE(C,B) $\wedge$ SOBRE(A,C)

## Descripción de Estado



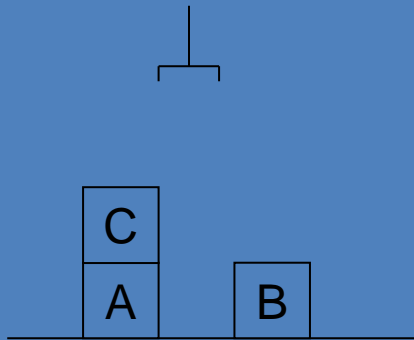
LIBRE(B)  
LIBRE(C)  
SOBRE(C,A)  
SOBREMESA(A)  
SOBREMESA(B)  
MANOVACIA

## Pila de Objetivos

COGIDO(C)  
LIBRE(B)  
LIBRE(B)  $\wedge$  COGIDO(C)  
apilar(C,B)  
SOBRE(A,C)  
SOBRE(C,B)  $\wedge$  SOBRE(A,C)



## Descripción de Estado



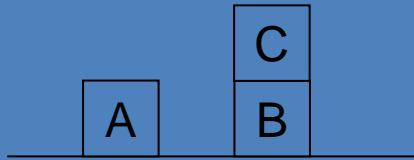
LIBRE(B)  
LIBRE(C)  
SOBRE(C,A)  
SOBREMESA(A)  
SOBREMESA(B)  
MANOVACIA

## Pila de Objetivos

MANOVACIA  $\wedge$  LIBRE(C)  $\wedge$  SOBRE(C,y)  
desapilar(C,y)  
LIBRE(B)  
LIBRE(B)  $\wedge$  COGIDO(C)  
apilar(C,B)  
SOBRE(A,C)  
SOBRE(C,B)  $\wedge$  SOBRE(A,C)



## Descripción de Estado



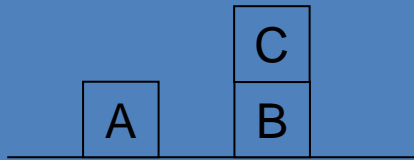
LIBRE(C)  
LIBRE(A)  
SOBRE(C,B)  
SOBREMESA(A)  
SOBREMESA(B)  
MANOVACIA

## Pila de Objetivos

SOBRE(A,C)  
SOBRE(C,B)  $\wedge$  SOBRE(A,C)



## Descripción de Estado

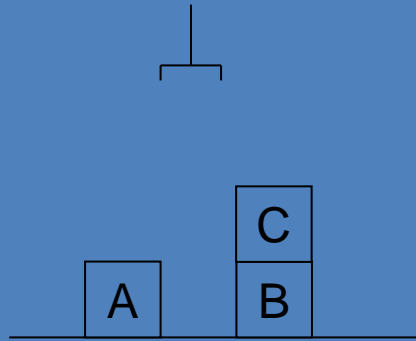


LIBRE(C)  
LIBRE(A)  
SOBRE(C,B)  
SOBREMESA(A)  
SOBREMESA(B)  
MANOVACIA

## Pila de Objetivos

LIBRE(C)  $\wedge$  COGIDO(A)  
apilar(A,C)  
SOBRE(C,B)  $\wedge$  SOBRE(A,C)

## Descripción de Estado



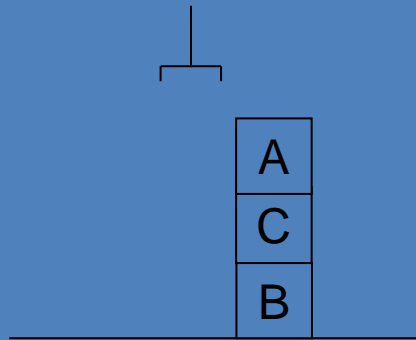
LIBRE(C)  
LIBRE(A)  
SOBRE(C,B)  
SOBREMESA(A)  
SOBREMESA(B)  
MANOVACIA

## Pila de Objetivos

SOBREMESA(A)  $\wedge$  LIBRE(A)  $\wedge$  MANOVACIA  
coger(A)  
LIBRE(C)  $\wedge$  COGIDO(A)  
apilar(A,C)  
SOBRE(C,B)  $\wedge$  SOBRE(A,C)



## Descripción de Estado



LIBRE(A)  
SOBRE(A,C)  
SOBRE(C,B)  
SOBREMESA(B)  
MANOVACIA

## Pila de Objetivos

NADA

# Interacción entre subobjetivos

Pila de Objetivos

$L1 \wedge L2$

Pila de Objetivos

L1  
L2  
 $L1 \wedge L2$

Pila de Objetivos

L2  
 $L1 \wedge L2$

L1 cierto en el estado

Resolviendo L2, L1 deja de ser cierto en el estado

# Ejemplo 2

Descripción de Estado	Pila de Objetivos
 <p data-bbox="568 562 938 891">LIBRE(B) LIBRE(C) SOBRE(C,A) SOBREMESA(A) SOBREMESA(B) MANOVACIA</p>	<p data-bbox="1161 576 1760 629">SOBRE(B,C)<math>\wedge</math>SOBRE(A,B)</p>

# Ejemplo 3

Estado inicial:

$\text{CONT}(X,A) \wedge \text{CONT}(Y,B)$

Objetivo:

$\text{CONT}(X,B) \wedge \text{CONT}(Y,A)$

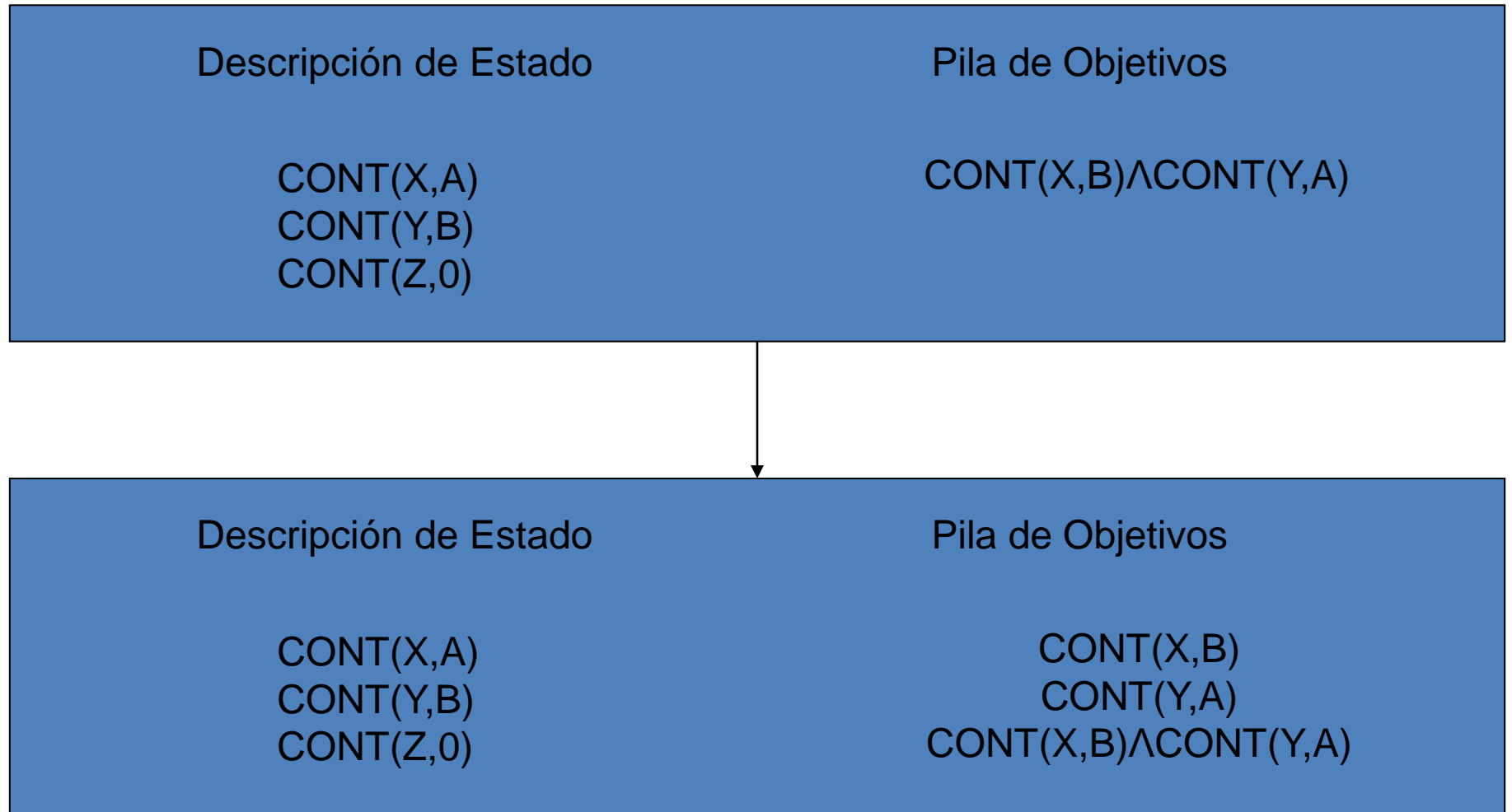
Operadores:

$\text{asigna}(u,r,t,s)$

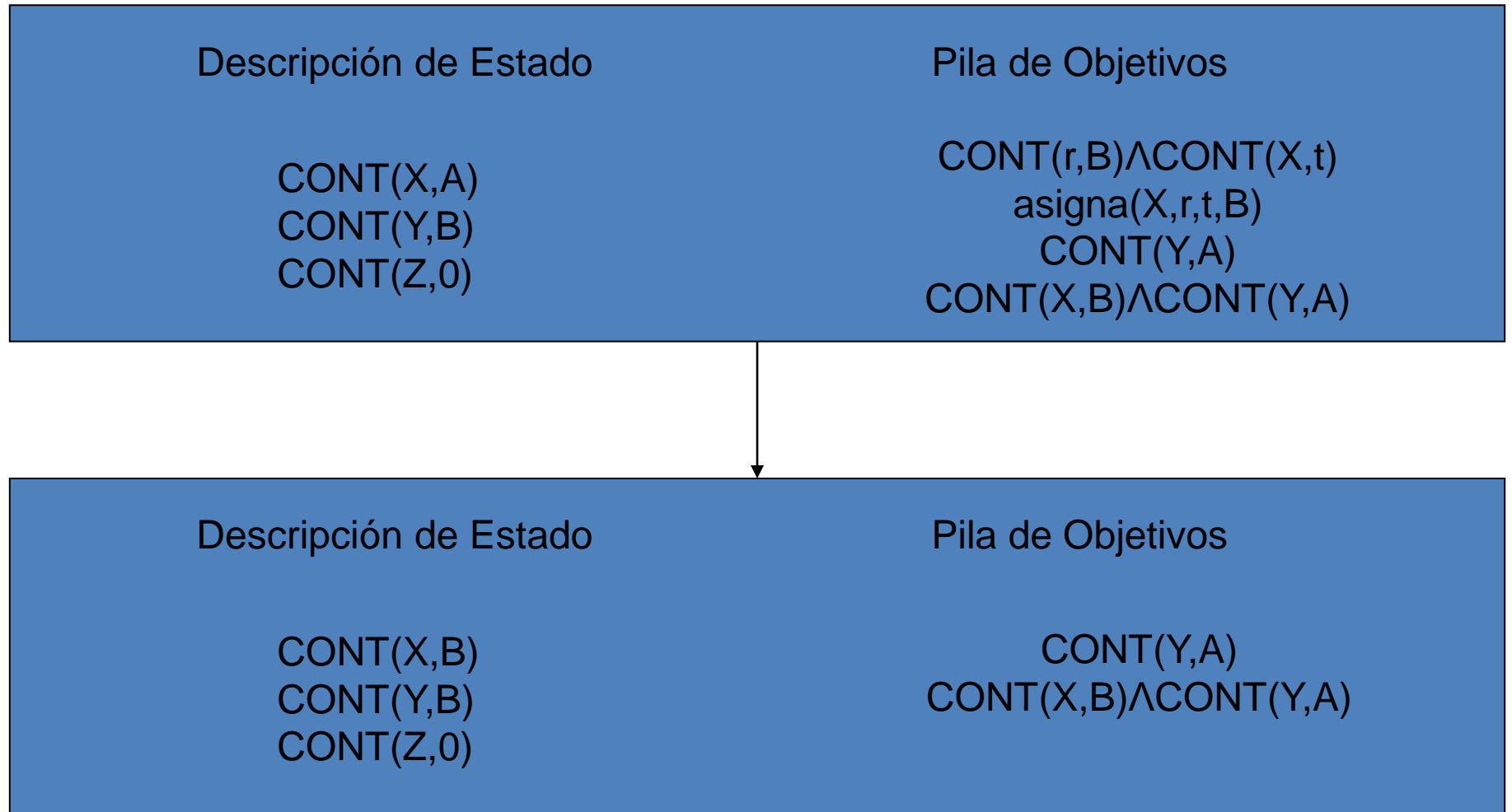
FP:  $\text{CONT}(r,s) \wedge \text{CONT}(u,t)$

LS:  $\text{CONT}(u,t)$

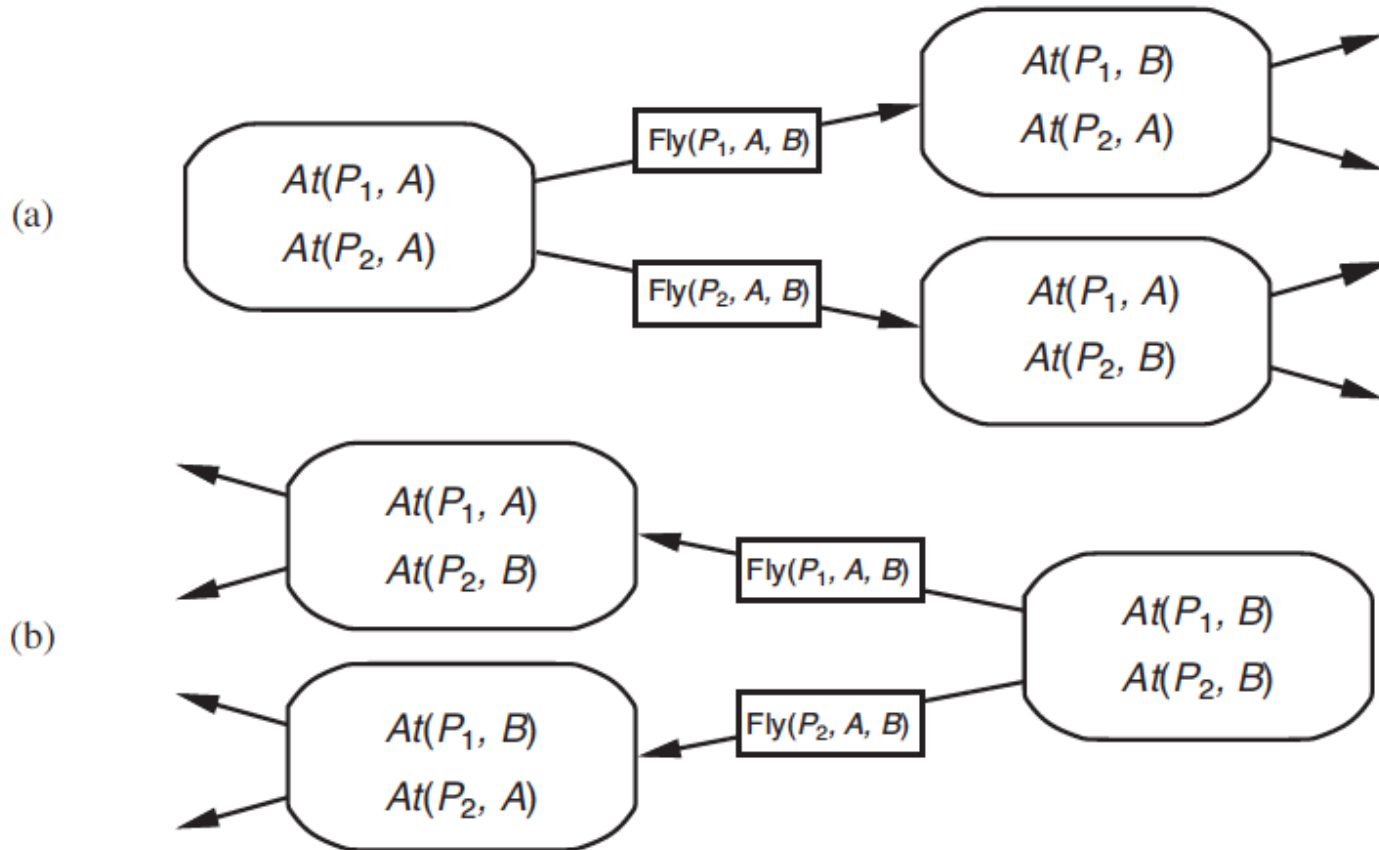
LA:  $\text{CONT}(u,s)$







# Planificación como búsqueda en un espacio de estados: HSP, FF



# Planificación como búsqueda en un espacio de estados: HSP, FF

- El intentar resolver el problema de planificación mediante técnicas de búsqueda heurística tiene como principal dificultad el obtener una buena función heurística.

# Planificación como búsqueda en un espacio de estados: HSP

- HSP (Heuristic Search Planner)
- La heurística se basa en el uso de modelos simplificados.
- La idea es aplicar operadores e ignorar la lista de supresión.
- Usa una estimación para el cálculo de la longitud de la solución simplificada.
- Como técnica de búsqueda usó en primer lugar un algoritmo de escalada por la máxima pendiente, en donde los empates se resuelven con un criterio aleatorio.
- Una segunda versión utilizó un algoritmo  $A^*$  con peso.

# Planificación como búsqueda en un espacio de estados: HSP

- Define  $\text{weight}(f)=0$  para todos los literales  $f$  del estado inicial, y del resto le asigna valor infinito.
- Aplica todas las acciones y actualiza pesos.
- Para cada acción con precondition  $\text{pre}(o)$  que añada el literal  $f$

$$\text{weight}(f) := \min(\text{weight}(f), \text{weight}(\text{pre}(o)) + 1)$$

# Planificación como búsqueda en un espacio de estados: HSP

- El peso de un conjunto de literales se define

$$weight(F) := \sum_{f \in F} weight(f)$$

- El proceso se repite hasta que los pesos no cambian en dos iteraciones sucesivas.
- La heurística se estima entonces:

$$h_{HSP}(S) := weight(\mathcal{G}) = \sum_{g \in \mathcal{G}} weight(g)$$

- En donde  $\mathcal{G}$  es el conjunto de objetivos.

# Planificación como búsqueda en un espacio de estados: HSP

- La heurística usa un criterio aditivo que asume la independencia de los objetivos

$$h_{HSP}(S) := weight(\mathcal{G}) = \sum_{g \in \mathcal{G}} weight(g)$$

- Que no tiene que ser cierta.
- Por esta razón la heurística no es admisible.

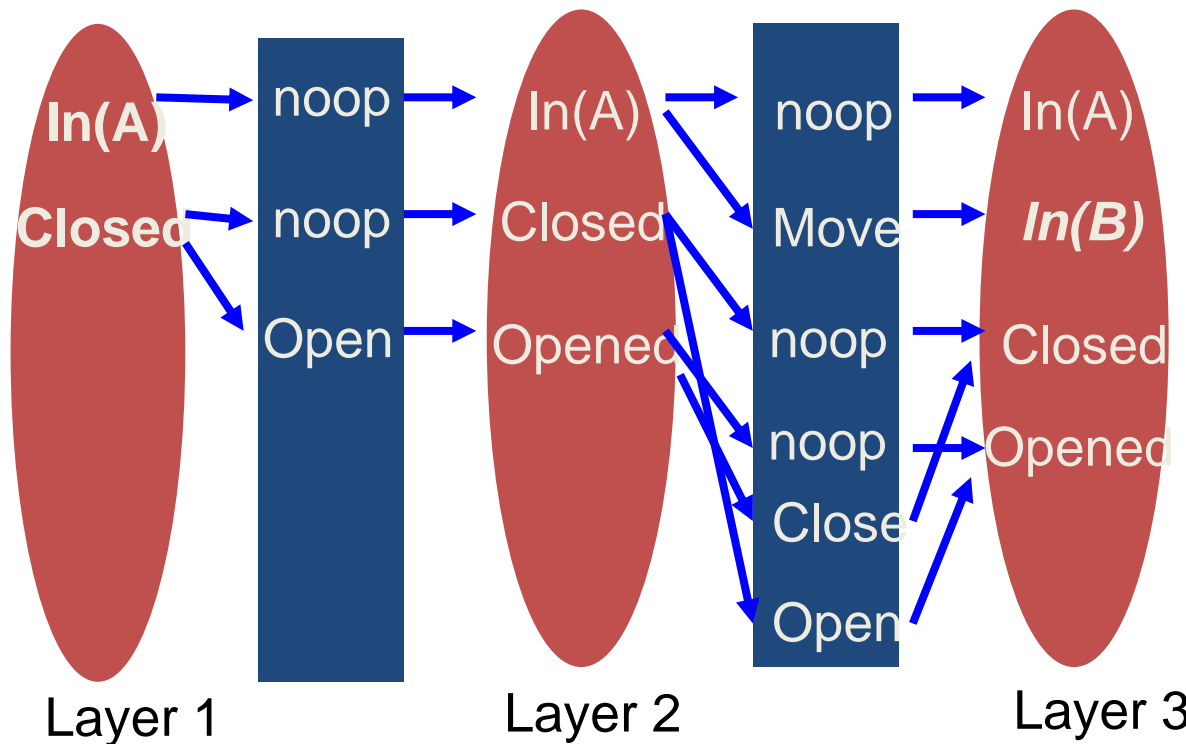
# Planificación como búsqueda en un espacio de estados: FF

- FF: The Fast Forward Planning System
- Sobre la base de HSP usa tres elementos nuevos:
  - Una nueva estimación heurística basada en la idea del planificador Graphplan
  - Un nuevo método de búsqueda: un método de escalada forzado
  - Un proceso para ordenar los descendientes.



# Planificación como búsqueda en un espacio de estados: FF

- Para la construcción de la heurística en primer lugar se construye una versión simplificada del grafo utilizado por Graphplan



# Planificación como búsqueda en un espacio de estados: FF

- Se termina cuando en la última capa se alcanzan todos los objetivos.
- Una vez alcanzados los objetivos se extrae un plan de la siguiente forma:
  - Nos situamos en la última capa, y para cada objetivo hacemos lo siguiente en la capa  $i$ 
    - Si el objetivo está presente en la capa  $i-1$ , entonces lo insertamos como objetivo a alcanzar en la capa  $i-1$
    - En otro caso, seleccionamos una acción en la capa  $i-1$  que añada el objetivo e insertamos cada precondition de la acción como objetivo de la capa  $i-1$ .
    - Cuando ya hemos trabajado con todos los objetivos de la capa  $i$ , continuamos con los objetivos de la capa  $i-1$ .
    - Paramos al llegar a la primera capa.

# Planificación como búsqueda en un espacio de estados: FF

- El plan simplificado es una secuencia
- $\{O_0, O_1, \dots, O_{m-1}\}$
- En donde cada  $O_i$  es el conjunto de acciones seleccionadas en cada capa.
- La longitud del plan se estima de la siguiente forma:

$$h_{FF}(S) := \sum_{i=0, \dots, m-1} |O_i|$$

# Planificación como búsqueda en un espacio de estados: FF

- Método de escalada forzado FF evalúa todos los sucesores, cambio irrevocablemente al mejor de todos si mejora al actual, si no es el caso realiza una búsqueda en anchura con el objetivo de buscar un nodo mejor que el de partida.
- La búsqueda continua desde ese nodo.
- Guarda la traza de los nodos seleccionados para generar el plan.

# Planificación como búsqueda en un espacio de estados: FF

- Considera en el proceso de búsqueda un conjunto de acciones útiles
- Para un estado  $S$ , el conjunto  $H(S)$  de acciones útiles se define como

$$H(S) := \{o | pre(o) \subseteq S, add(o) \cap G_1 \neq \emptyset\}$$

- En donde  $G_1$  representa el conjunto de objetivos marcados de la siguiente capa.

# Planificación como búsqueda en un espacio de estados: HSP, FF

- “Planning as Heuristic Search,” by Blai Bonet and Hector Geffner, *Artificial Intelligence Journal*, 2001.
- “The FF Planning System: Fast Plan Generation Through Heuristic Search,” by Jorg Hoffmann and Bernhard Nebel, *Journal of Artificial Intelligence Research*, 2001.

# Planificación como búsqueda en un espacio de planes

- **Planificación de orden parcial (POP)**
- **Hipótesis del menor compromiso:** es necesario ocuparse de las decisiones que actualmente interesen, dejando cualquier otra decisión para más tarde

# Planificación como búsqueda en un espacio de planes

- Representación de estados y objetivos

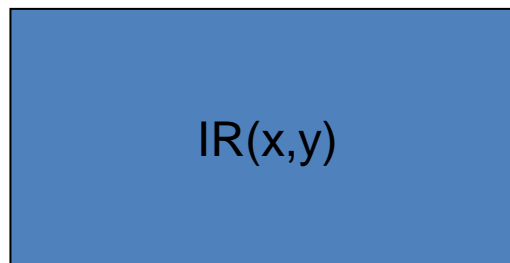
$En(Casa) \wedge Vende(Supermercado, Leche) \wedge Vende(Supermercado, Pan)$

$En(x) \wedge Vende(x, Leche)$

- Representación de acciones

$Op(ACCION : IR(x, y), PRECONDICION : EN(x) \wedge RUTA(x, y), EFECTO : EN(y) \wedge \neg EN(x))$

$EN(x) \wedge RUTA(x, y)$



$EN(y) \wedge \neg EN(x)$



# Ejemplo

Objetivo: ZapatoDerechoPuesto  $\wedge$  ZapatoIzquierdoPuesto

Estado inicial: sin literales

Operadores:

Op(ACCION: ZapatoDerecho, PRECONDICION: CalcentínDerechoPuesto,  
EFECTO: ZapatoDerechoPuesto

Op(ACCION: CalcentínDerecho, PRECONDICION: no tiene, EFECTO:  
CalcentínDerechoPuesto

Op(ACCION: ZapatoIzquierdo, PRECONDICION: CalcentínIzquierdoPuesto,  
EFECTO: ZapatoIzquierdoPuesto

Op(ACCION: CalcentínIzquierdo, PRECONDICION: no tiene,  
EFECTO: CalcentínIzquierdoPuesto

# Definición de plan

- Un conjunto de nodos (operadores)
- Un conjunto de relaciones de orden:  $s_i < s_j$  que significan que  $s_i$  debe de producirse en algún momento antes que  $s_j$
- Un conjunto de restricciones de variables, del tipo  $x=A$
- Un conjunto de **vínculos causales**:  $s_i \xrightarrow{c} s_j$  que significa que  $s_i$  logra  $c$  para  $s_j$

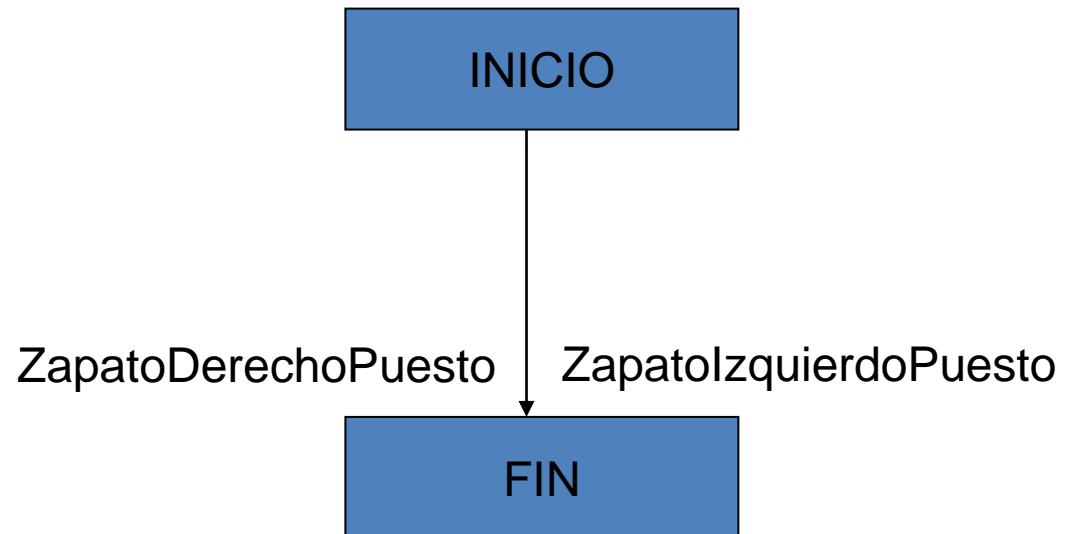
# Plan inicial

- Incluye dos operadores ficticios INICIO y FIN
  - INICIO no tiene precondiciones y como efecto el estado inicial
  - FIN tiene como precondiciones el objetivo y no tiene efectos

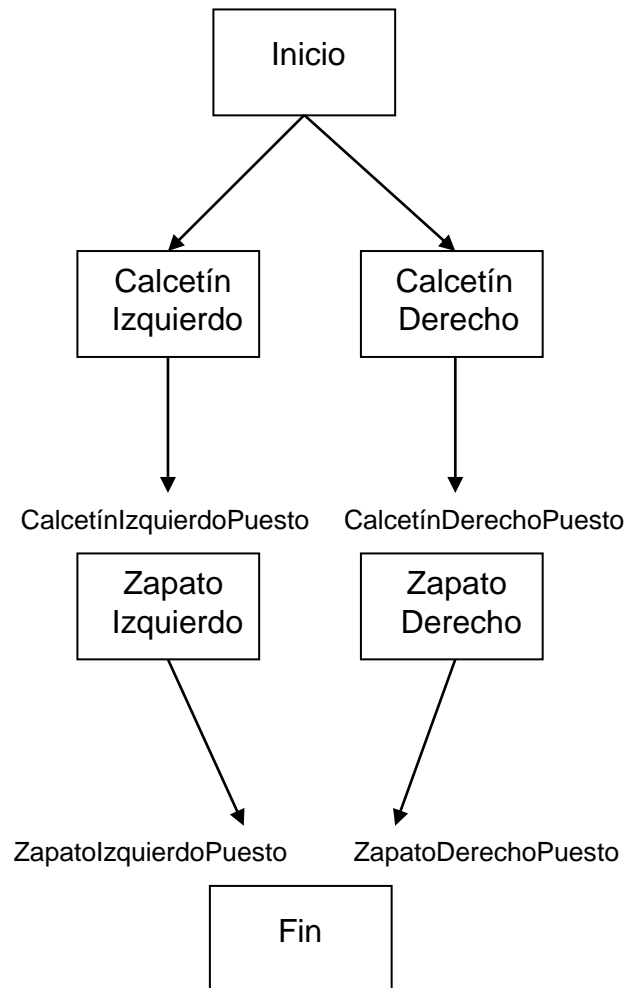
# EJEMPLO

PLAN(NODOS: {  $s_1$ : OP(ACCION: INICIO, PRECONDICION: no tiene, EFECTO: no tiene),  
                   $s_2$ : OP(ACCION: FIN, PRECONDICION:  
ZapatoDerechoPuesto ZapatoIzquierdoPuesto), EFECTO: no  
tiene},  
      ORDEN: { $s_1 < s_2$ },  
      RESTRICCIONES: { },  
      VINCULOS: { })

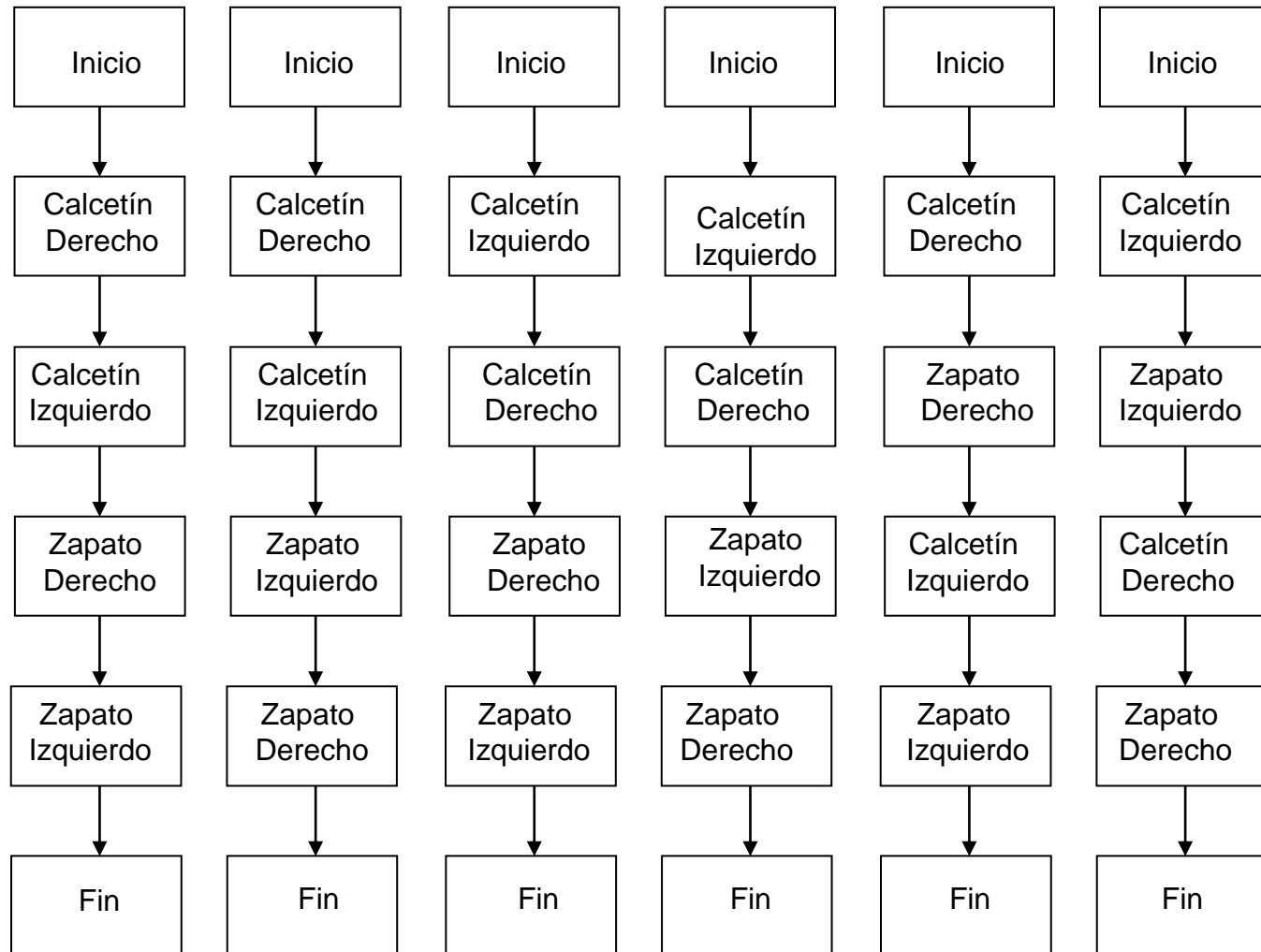
# Plan inicial



# Plan de orden parcial



# Planes de orden total



# Plan completo y consistente

- PLAN COMPLETO

- Cada precondition de cada operador se debe de satisfacer mediante otro operador
- $s_i$  logra una precondition  $c$  de  $s_j$  si
  - $s_i < s_j$  y  $c \in EFECTOS(s_i)$
  - $\neg \exists s_k$  con  $\neg c \in EFECTOS(s_k)$  y  $s_i < s_k < s_j$  en alguna linealización del plan

- PLAN CONSISTENTE

- No hay contradicciones en las restricciones de orden ni de variables
  - $x=A$  y  $x=B$  sería inconsistente
  - $s_i < s_j$ ,  $s_j < s_k$  y  $s_k < s_i$  sería inconsistente



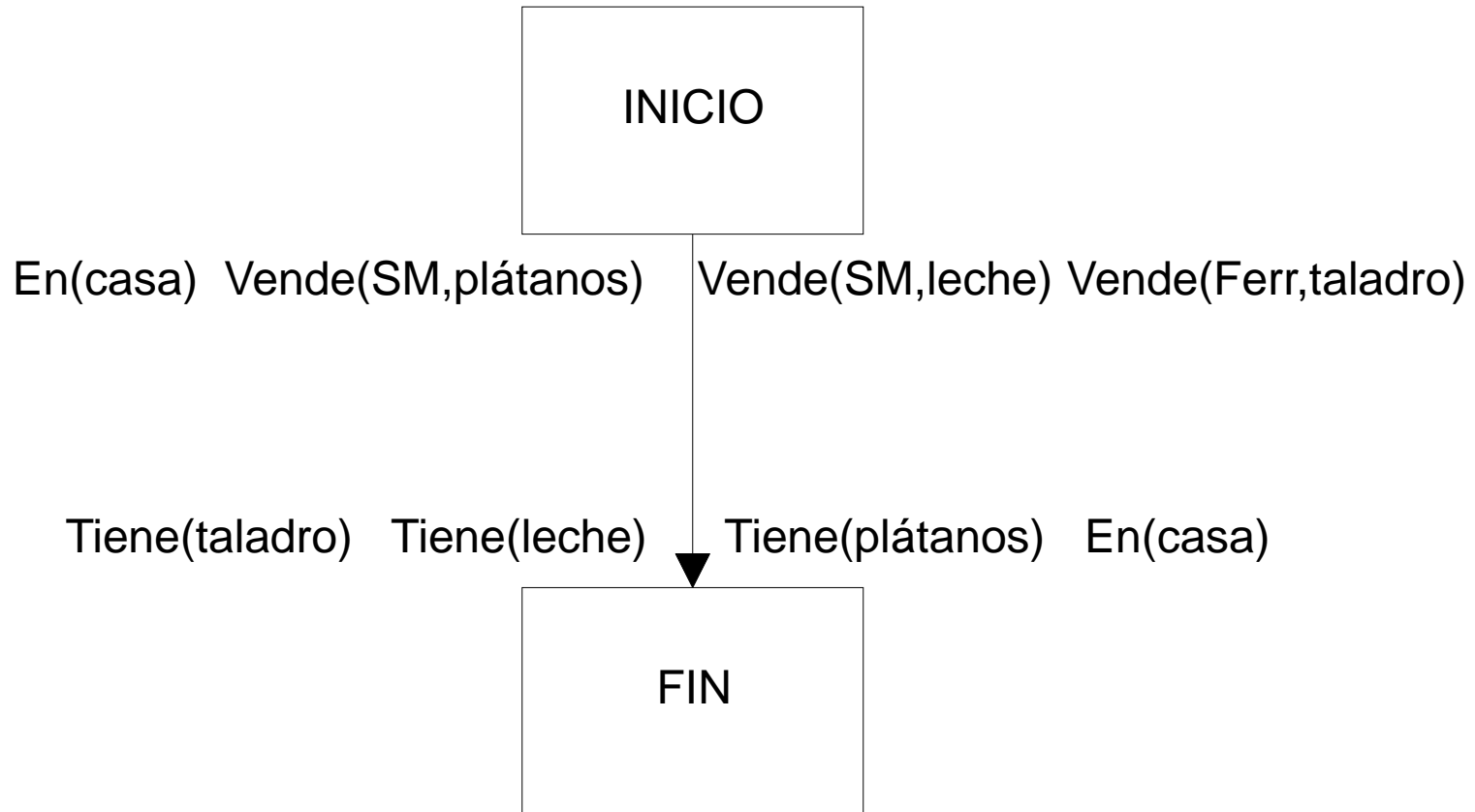
1. **Terminación:** Si **agenda** esta vacía, devuelve  $\langle A, O, L \rangle$ .
2. **Selección del objetivo:** Sea  $\langle Q, A_e \rangle$  un par de la **agenda** (por definición  $A_e \in A$  y  $Q$  es un elemento de la conjunción de la precondition de  $A_e$ ).
3. **Selección de la acción:** Sea  $A_{nuevo} = \text{elección}$  de una acción que añada  $Q$  (ya sea mediante una nueva acción de  $R$  o mediante una acción existente en  $A$  que pueda ordenarse consistentemente antes de  $A_e$ ). Si no existiese tal acción entonces devolver fallo. En otro caso, sea  $L' = L \cup \{A_{nuevo} \xrightarrow{Q} A_e\}$ , y sea  $O' = O \cup \{A_{nuevo} < A_e\}$ . Si  $A_{nuevo}$  es una acción nueva, entonces  $A' = A \cup \{A_{nuevo}\}$  y  $\{A_0 < A_{nuevo} < A_\infty\}$  (en otro caso  $A' = A$ ).
4. **Actualizar el conjunto de objetivos:** Sea **agendanueva** = **agenda** -  $\{\langle Q, A_e \rangle\}$ . Si  $A_{nuevo}$  no estaba antes, entonces para cada elemento  $Q_i$  de la conjunción de su precondition añadir  $\langle Q_i, A_{nuevo} \rangle$  a **agendanueva**.
5. **Protección de enlaces causales:** Para acción  $A_t$  que pudiese amenazar a un enlace causal  $A_p \xrightarrow{H} A_c \in L'$  elegir una restricción de orden consistente, o bien
  - a) **ascenso:** añadir  $A_t < A_p$  a  $O'$ , o
  - b) **degradación:** añadir  $A_c < A_t$  a  $O'$ .

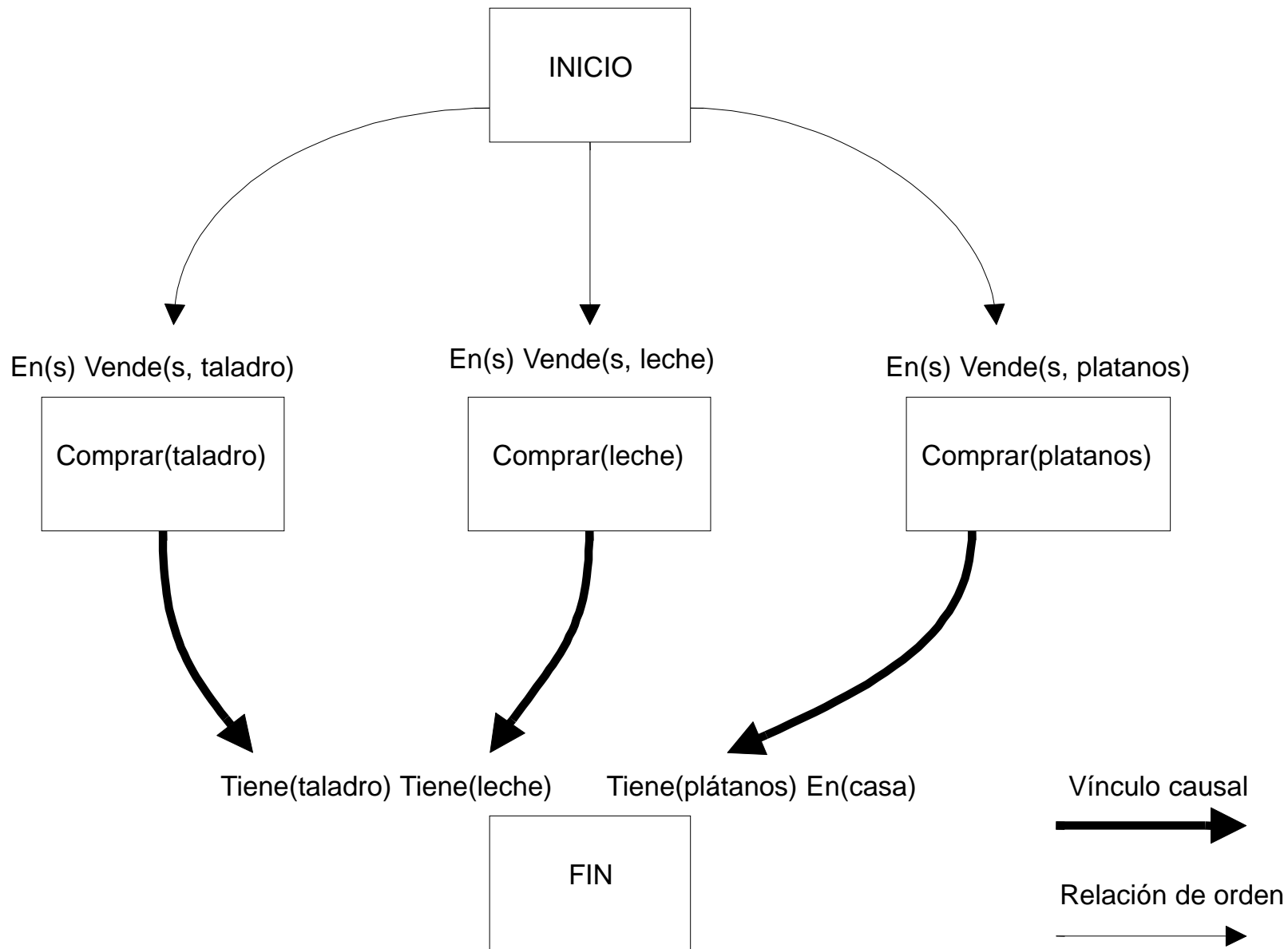
Si ninguna restricción es consistente, entonces devolver fallo.
6. **Recursión:** Llamar a POP( $\langle A', O', L' \rangle$ , agendanueva, R).

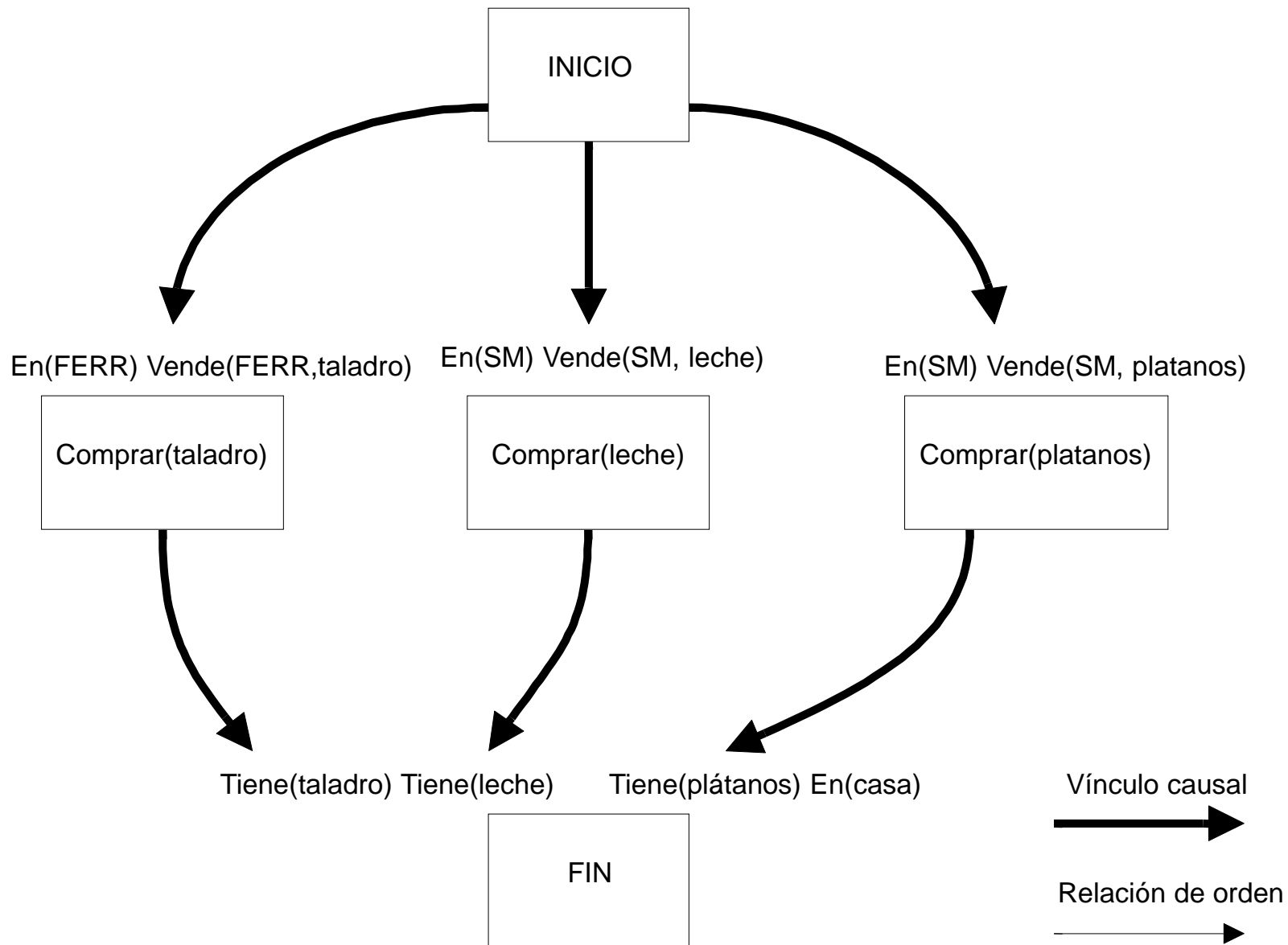
# Ejemplo

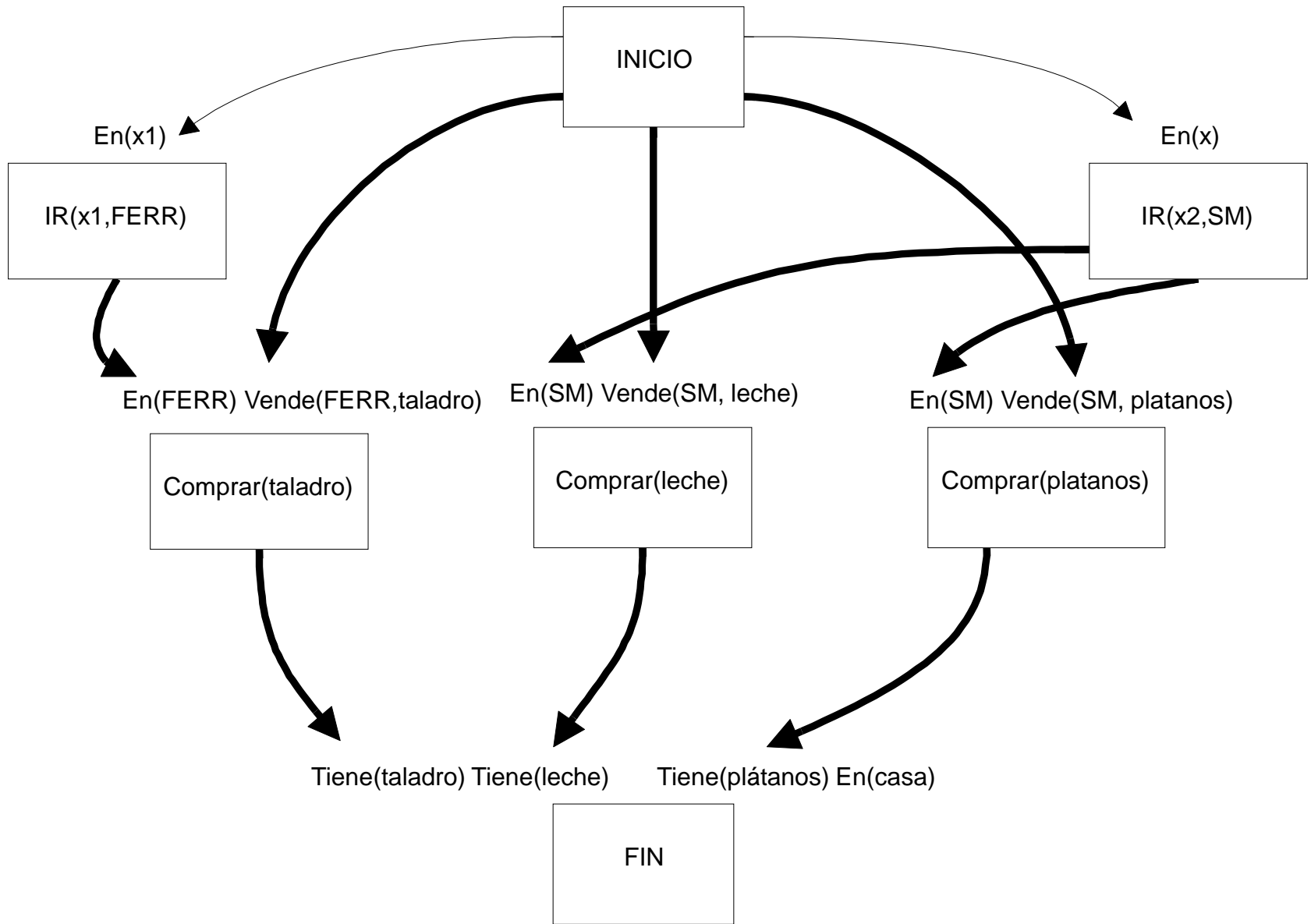
- $Op(ACCION: INICIO, PRECONDICION: \text{no tiene}, EFECTO: EN(CASA) \wedge VENDE(FERR, TALADRO) \wedge VENDE(SM, LECHE) \wedge VENDE(SM, PLATANOS))$
- $Op(ACCION: FIN, PRECONDICION: TIENE(TALADRO) \wedge TIENE(LECHE) \wedge TIENE(PLATANOS) \wedge EN(CASA), EFECTO: \text{no tiene})$
- $Op(ACCION: IR(x,y), PRECONDICION: EN(x), EFECTO: EN(y) \wedge \neg EN(x))$
- $Op(ACCION: COMPRAR(x), PRECONDICION: EN(TIENDA) \wedge VENDE(TIENDA,x), EFECTO: TIENE(x))$

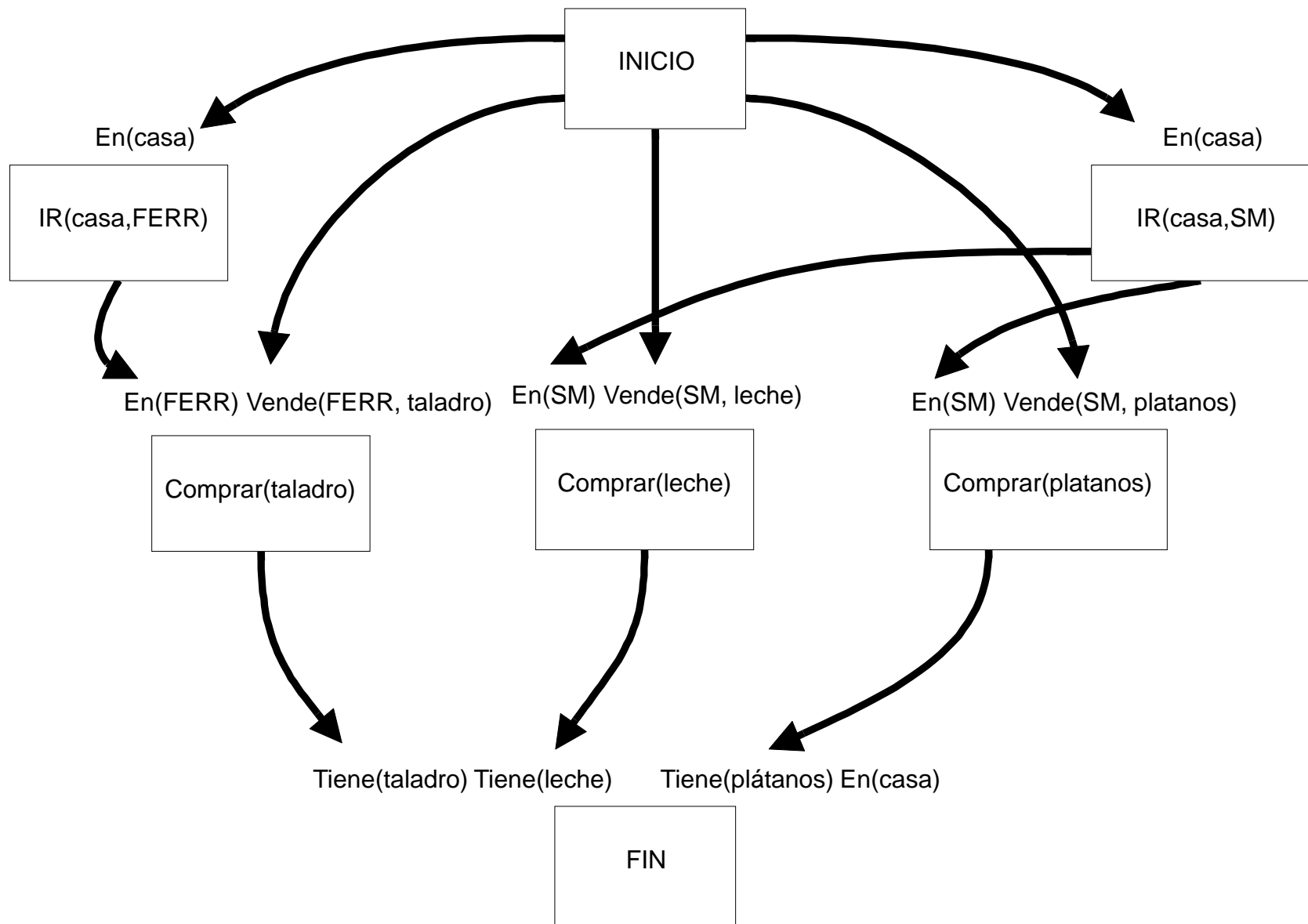
# Plan inicial:



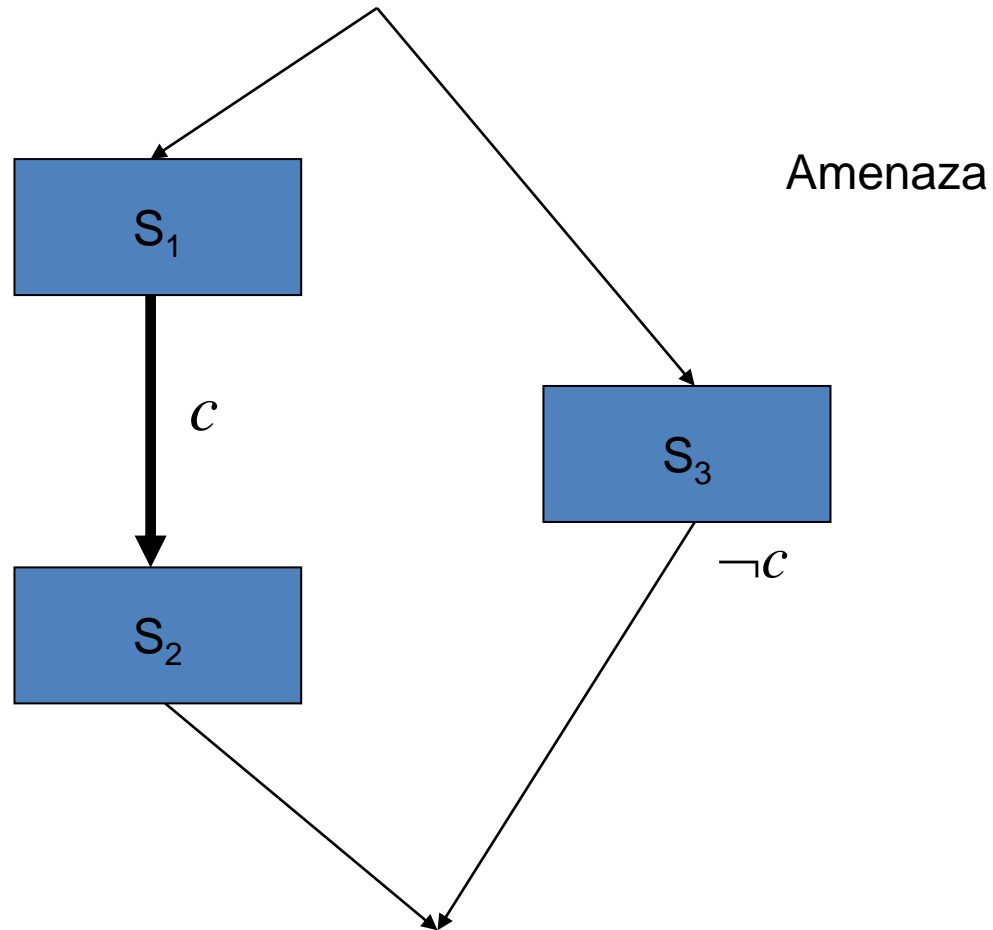






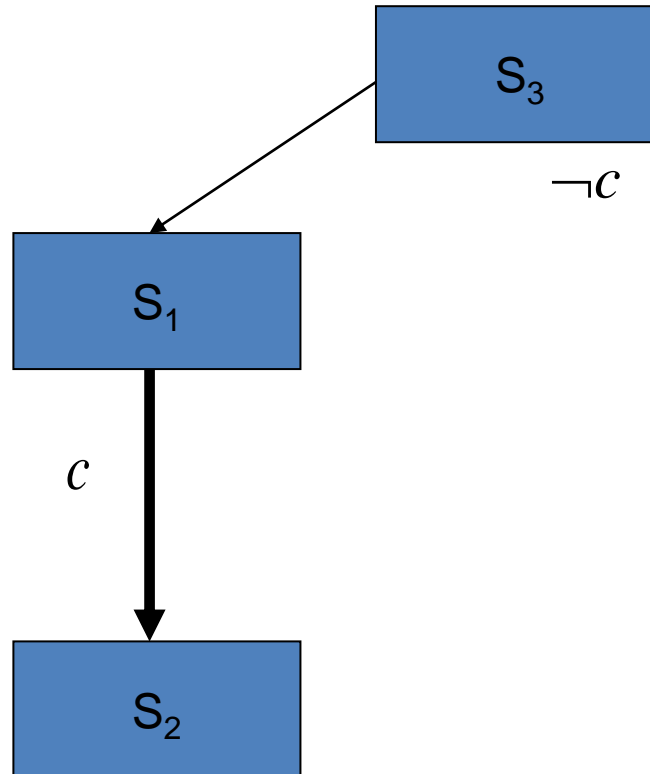


# Amenazas

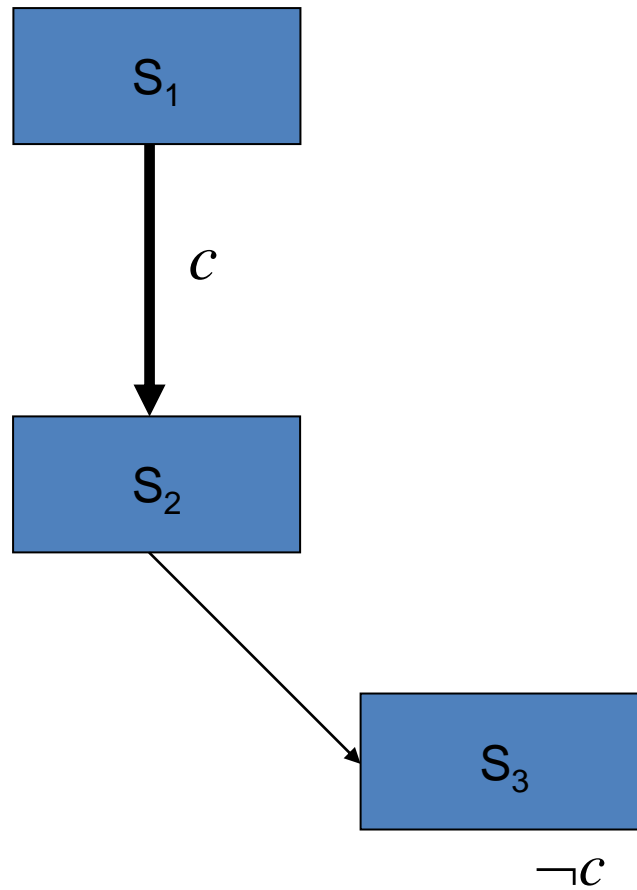




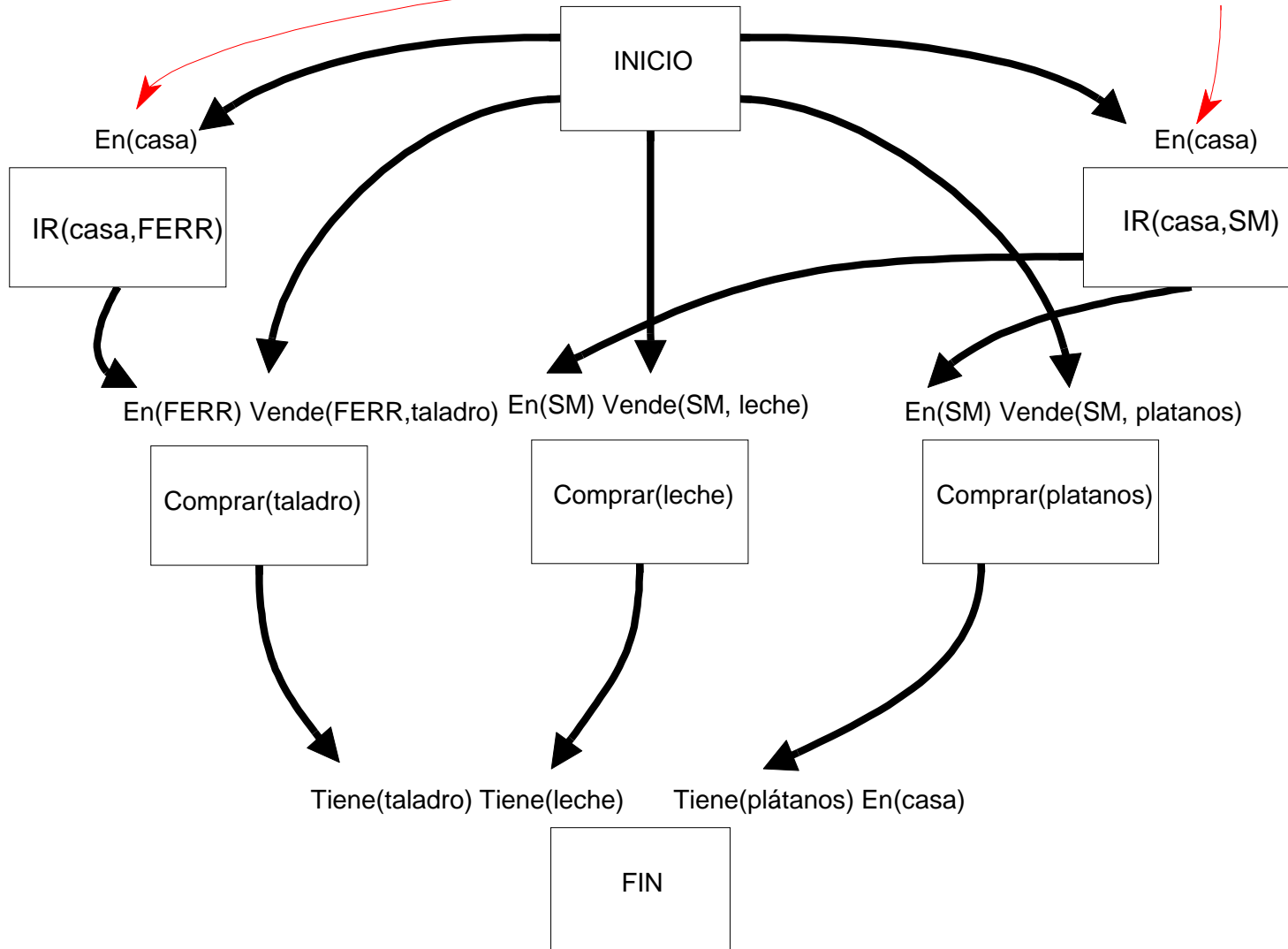
# Ascenso

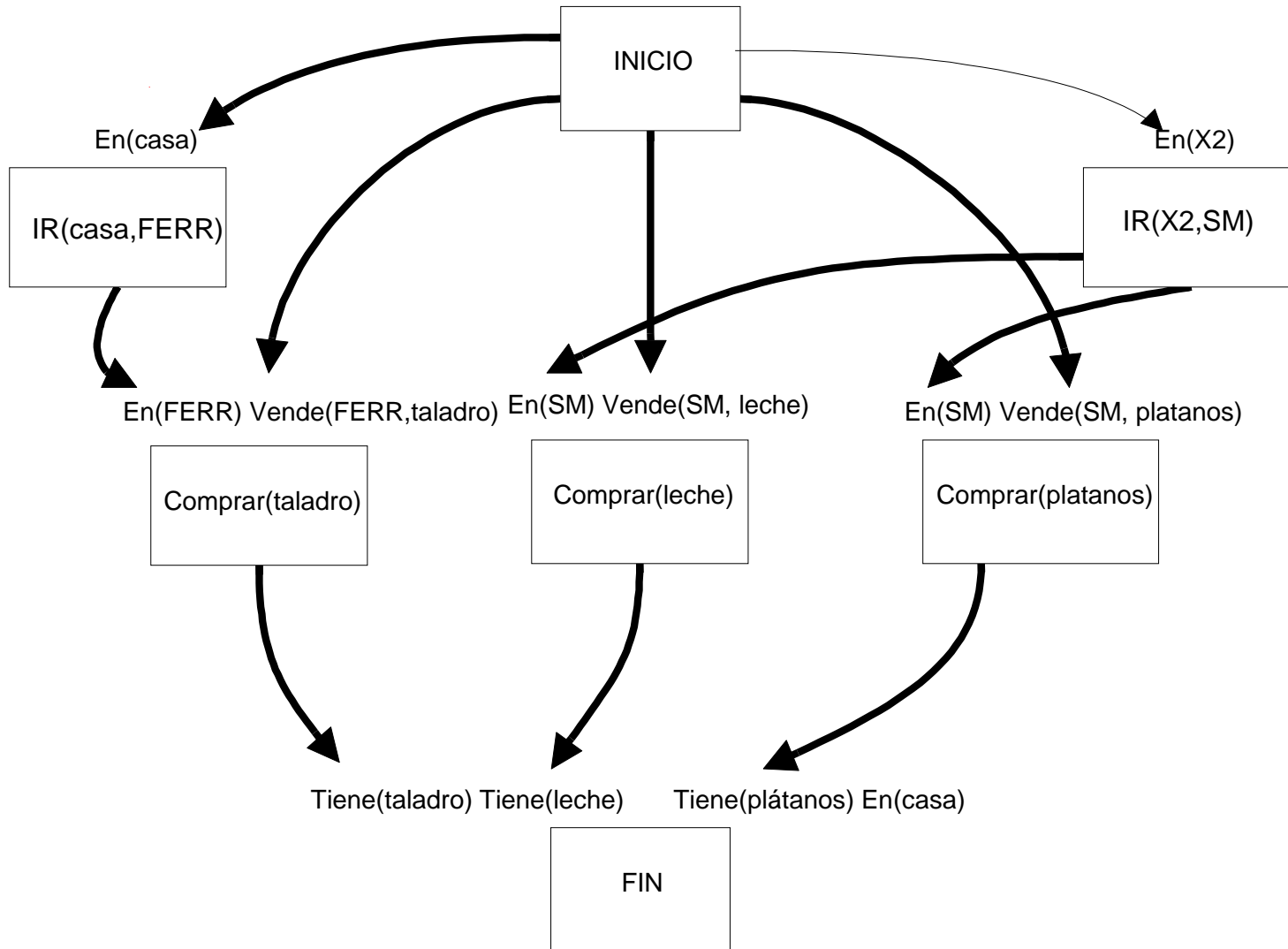


# Degradación

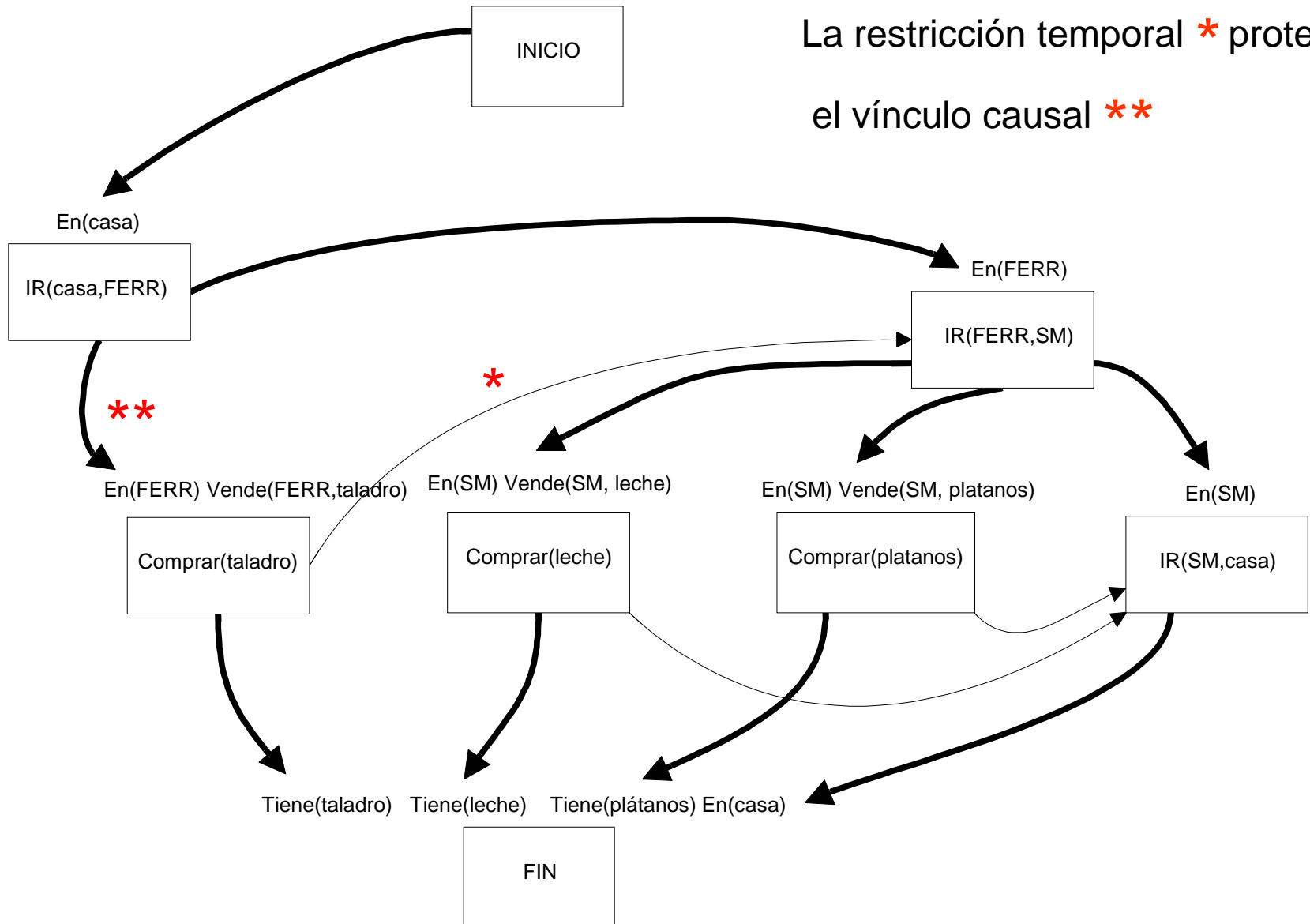


Amenaza!!!

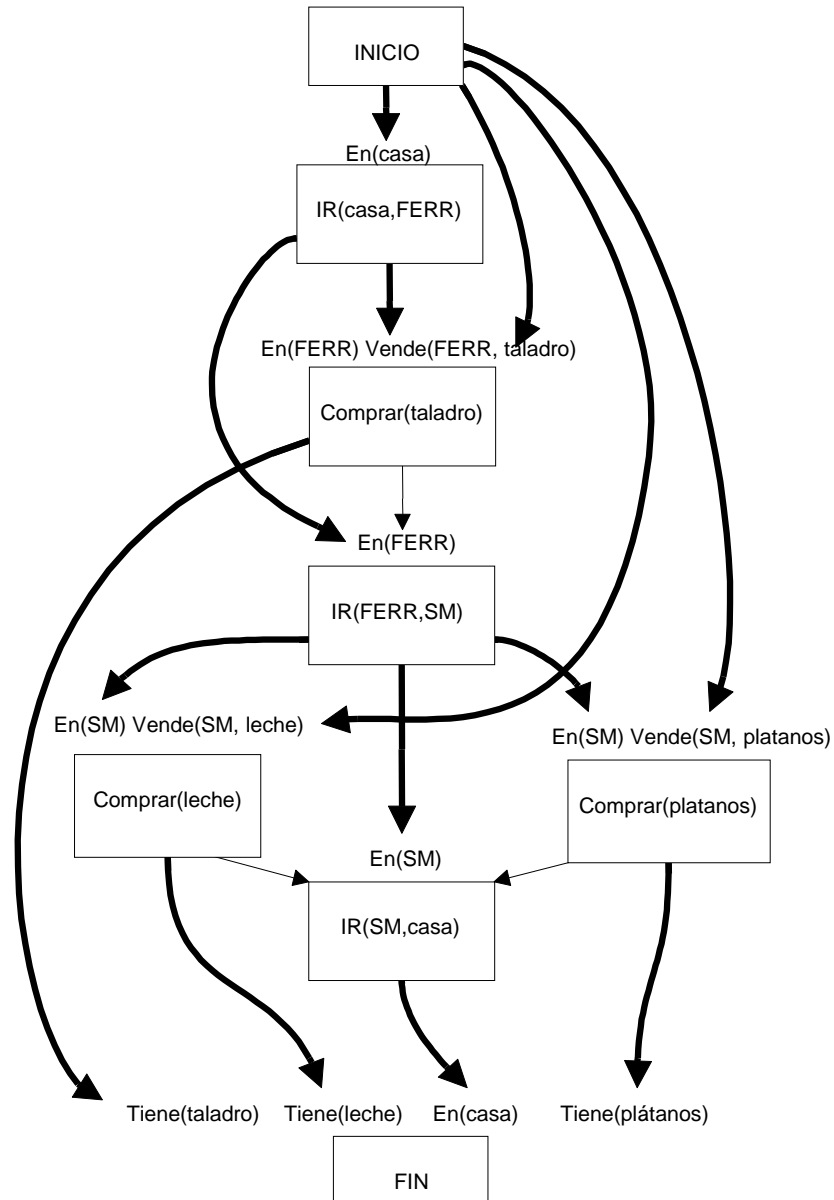




La restricción temporal \* protege  
el vínculo causal \*\*



# Plan total:



# Heurísticas para planificación de orden parcial

- Heurísticas para selección de planes:
  - Número de pasos  $N$
  - Precondiciones sin resolver  $OP$
- Frecuentemente se utiliza un algoritmo  $A^*$  con heurística:

$$f(P) = N(P) + OP(P)$$

# Planificación jerárquica

- A menudo es interesante generar un plan inicial compuesto por tareas de alto nivel para luego ir desglosándolo en acciones más simples.
- Esto es especialmente útil en problemas complejos en donde planificar desde cero todas las acciones de bajo nivel puede resultar una tarea muy costosa.



# ABSTRIPS

Coger(x)            2            2            1  
FP: SOBREMESA(x), LIBRE(x), MANOVACIA  
LA: COGIDO(x)  
LS: SOBREMESA(x), LIBRE(x), MANOVACIA

Dejar(x)            2  
FP: COGIDO(x)  
LA: SOBREMESA(x), LIBRE(x), MANOVACIA  
LS: COGIDO(x)

Apilar(x,y)        2            2  
FP: COGIDO(x), LIBRE(y)  
LA: MANOVACIA, SOBRE(x,y), LIBRE(x)  
LS: COGIDO(x), LIBRE(y)

Desapilar(x,y)    1            2            3  
FP: MANOVACIA, LIBRE(x), SOBRE(x,y)  
LA: COGIDO(x), LIBRE(y)  
LS: MANOVACIA, LIBRE(x), SOBRE(x,y)

# Redes de planificación jerárquica de tareas

- Existen dos tipos de acciones:
  - Métodos (alto nivel).
  - Acciones primitivas (bajo nivel).
- Los métodos se pueden descomponer en subtareas que pueden ser otros métodos o acciones primitivas.
- Esta descomposición se define mediante una red de tareas, que establece la ordenación total o parcial de las subtareas de los métodos.

# Redes de planificación jerárquica de tareas

- El funcionamiento de un planificador HTN es muy similar al de un planificador clásico.
- El objetivo del problema consiste en aplicar una serie de tareas mediante la descomposición y planificación de una secuencia de métodos que, finalmente, se desglosará en las acciones primitivas.

# Redes de planificación jerárquica de tareas

- Este tipo de planificación presenta dos ventajas:
  - El dominio del problema se describe en términos de acciones estructuradas jerárquicamente, resultando más intuitivo para el experto que modela el problema.
  - La función de planificador consiste en refinar estas estructuras generando las expansiones necesarias y simplifica la resolución del problema original.

# Ejemplo de descripción jerárquica tipo HTN

- **Para colocar X encima de Y**
  - dejar libre X
  - dejar libre Y
  - poner X sobre Y
- **Para dejar libre X**
  - si X es la mesa NADA
  - si X está libre NADA
  - Si hay Y sobre X entonces dejar libre Y y poner Y sobre la mesa

(define (domain bloques)	(:derived
(:requirements	(igual ?x ?x) ()))
:typing	
:fluents	(:task sobre
:derived-predicates	:parameters (?x ?y)
:negative-preconditions	(:method poner_encima
:htn-expansion)	:precondition () ; vacio
	:tasks (
(:types	(limpiar ?x)
bloque superficie - object)	(limpiar ?y)
	(colocar ?x ?y)
(:constants mesa - superficie)	)
	))
(:predicates	(:task limpiar
(manovacia)	:parameters (?x)
(libre ?x - bloque)	(:method limpiar_mesa
(cogido ?x - bloque)	:precondition (igual ?x mesa)
(sobremesa ?x - bloque)	:tasks())
(sobre ?x ?y - bloque)	(:method limpiar_libre
(igual ?x ?y)	:precondition (libre ?x)
(distinto ?x ?y))	:tasks ()))
(:derived	(:method limpiar_ocupado
(distinto ?x ?y) (not (igual ?x ?y)))	:precondition (sobre ?y ?x)
	:tasks ((limpiar ?y)(colocar ?y mesa))))

```
(:task colocar
:parameters (?x ?y)
(:method colocar
:precondition ()
:tasks ((primero-coge ?x)(despues-deja
?x ?y))))
```

```
(:task primero-coge
:parameters (?x - bloque)
(:method cogelo_de_la_mesa
:precondition (sobremesa ?x)
:tasks (coger ?x))
(:method cogelo_de_la_pila
:precondition (sobre ?x ?y)
:tasks (desapilar ?x ?y)))
```

```
(:task despues-deja
:parameters (?x - bloque ?y - object)
(:method dejalo_en_la_mesa
:precondition (igual ?y mesa)
:tasks (dejar ?x))
(:method dejalo_en_la_pila
:precondition (distinto ?y mesa)
:tasks (apilar ?x ?y)))
```

```
(:action coger
:parameters (?x - bloque)
:precondition (and (sobremesa ?x)(libre
?x)(manovacia))
:effect (and (not (sobremesa ?x)) (not (libre ?x))(not
(manovacia))
(cogido ?x)))

(:action dejar
:parameters (?x - bloque)
:precondition (cogido ?x)
:effect (and (sobremesa ?x) (libre ?x) (manovacia)
(not (cogido ?x))))

(:action apilar
:parameters (?x ?y - bloque)
:precondition (and (cogido ?x)(libre ?y))
:effect (and (not (cogido ?x)) (not (libre ?y)) (libre ?x)
(sobre ?x ?y) (manovacia)))

(:action desapilar
:parameters (?x ?y - bloque)
:precondition (and (manovacia) (libre ?x) (sobre ?x
?y))
:effect (and (cogido ?x) (libre ?y) (not (libre ?x)) (not
(sobre ?x ?y)) (not (manovacia))))
)
```

```
(define (problem bloques-1) (:domain bloques))
```

```
(:objects  
  A B C - bloque  
)
```

```
(:init  
  (sobremesa A)  
  (sobre B A)  
  (sobre C B)  
  (libre C)  
  (manovacia)  
)
```

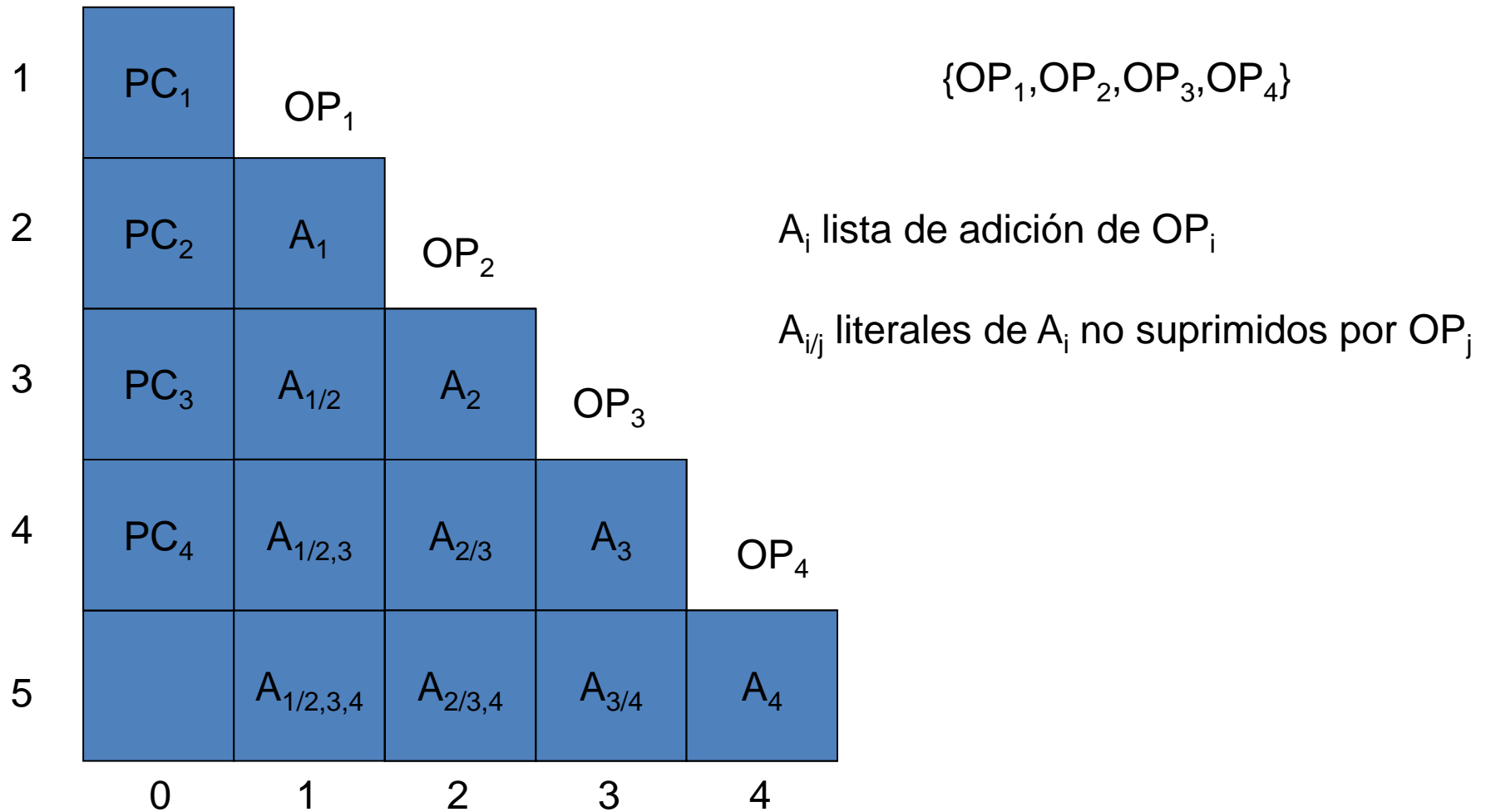
```
(:tasks-goal  
  :tasks(  
    (sobre A C)  
  )  
)
```



# Representación para planes

- El objetivo es conseguir una representación para planes que permita:
  - Usar un plan previamente generado para la resolución de problemas posteriores.
  - Controlar inteligentemente la ejecución de un plan concreto.

# Tablas triangulares



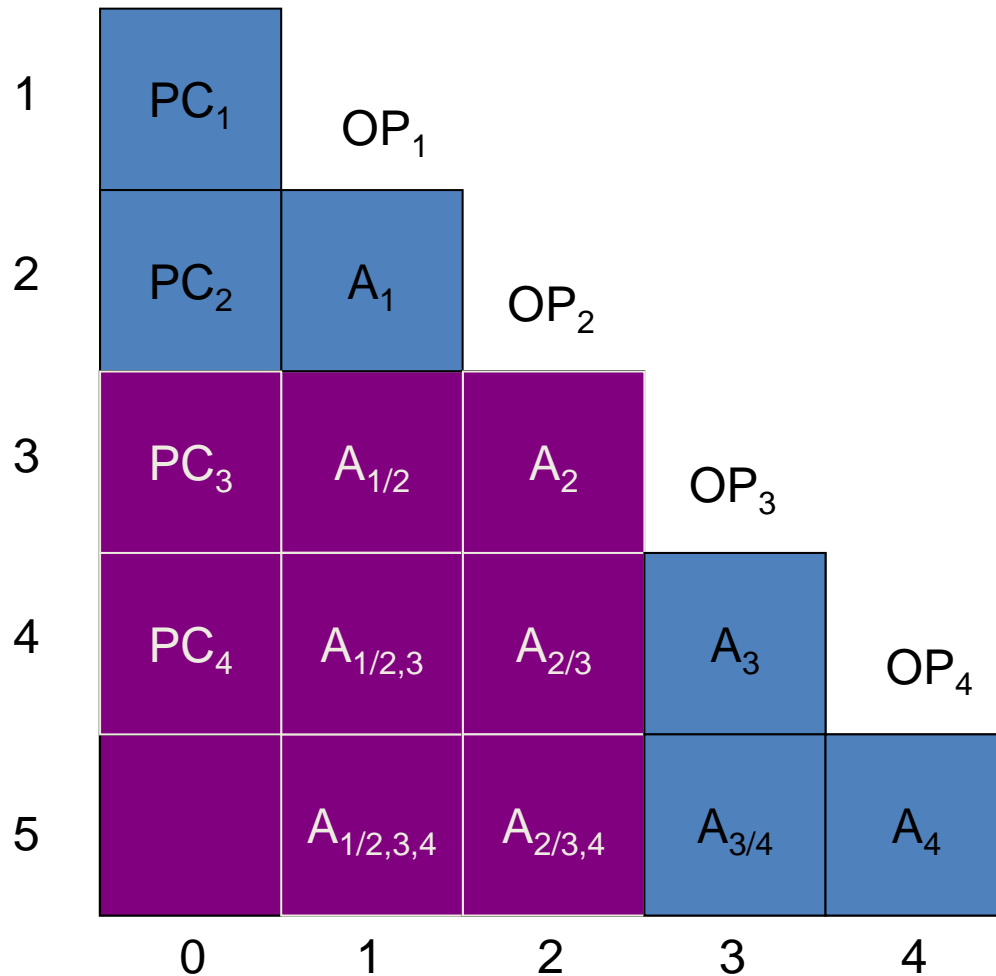
1	PC <sub>1</sub>	OP <sub>1</sub>			
2	PC <sub>2</sub>	A <sub>1</sub>	OP <sub>2</sub>		
3	PC <sub>3</sub>	A <sub>1/2</sub>	A <sub>2</sub>	OP <sub>3</sub>	
4	PC <sub>4</sub>	A <sub>1/2,3</sub>	A <sub>2/3</sub>	A <sub>3</sub>	OP <sub>4</sub>
5		A <sub>1/2,3,4</sub>	A <sub>2/3,4</sub>	A <sub>3/4</sub>	A <sub>4</sub>
	0	1	2	3	4

Unión de los literales de la i-ésima fila:  
lista de adición de la secuencia

$\{OP_1, OP_2, \dots, OP_{i-1}\}$

Columna cero: relacionada con  
las precondiciones de los  
operadores del plan.

Núcleo i-ésimo: literales marcados en la intersección de las filas que quedan por  
Debajo de la i-ésima, esta incluida, con las columnas que quedan a la izquierda  
la i-ésima.



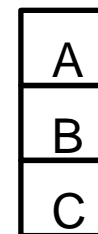
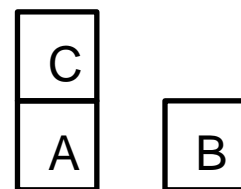
El núcleo representa la  
precondición de la  
secuencia parcial

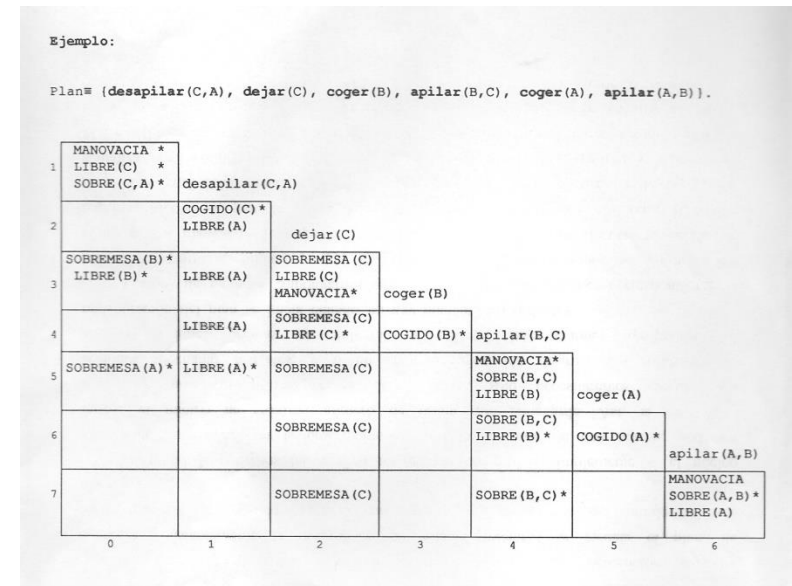
$\{OP_i, OP_{i+1}, \dots, OP_n\}$

Ejemplo:

Plan  $\equiv$  {desapilar(C,A), dejar(C), coger(B), apilar(B,C), coger(A), apilar(A,B)}.

1	MANOVACIA *						
	LIBRE (C) *						
	SOBRE (C,A) *	desapilar (C,A)					
2		COGIDO (C) *					
		LIBRE (A)	dejar (C)				
3	SOBREMESA (B) *		SOBREMESA (C)				
	LIBRE (B) *	LIBRE (A)	LIBRE (C)				
			MANOVACIA*	coger (B)			
4		LIBRE (A)	SOBREMESA (C)				
			LIBRE (C) *	COGIDO (B) *	apilar (B,C)		
5	SOBREMESA (A) *	LIBRE (A) *	SOBREMESA (C)		MANOVACIA*		
					SOBRE (B,C)		
					LIBRE (B)	coger (A)	
6			SOBREMESA (C)		SOBRE (B,C)		
					LIBRE (B) *	COGIDO (A) *	
						apilar (A,B)	
7			SOBREMESA (C)		SOBRE (B,C) *		
						MANOVACIA	
						SOBRE (A,B) *	
						LIBRE (A)	
	0	1	2	3	4	5	6





126