# Técnicas Avanzadas de Inteligencia Artificial

Curso 2016-2017

German Rigau y Maite Urretavizcaya

{german.rigau, maite.urretavizcaya}@ehu.eus

Grado en Ingeniería en Informática

# Temario

1. Agentes Inteligentes
2. Sistemas Multiagentes
3. Planificación

# 3 Planificación

1. Introducción
2. Planificación clásica
3. Planificación mundo real

# Introducción

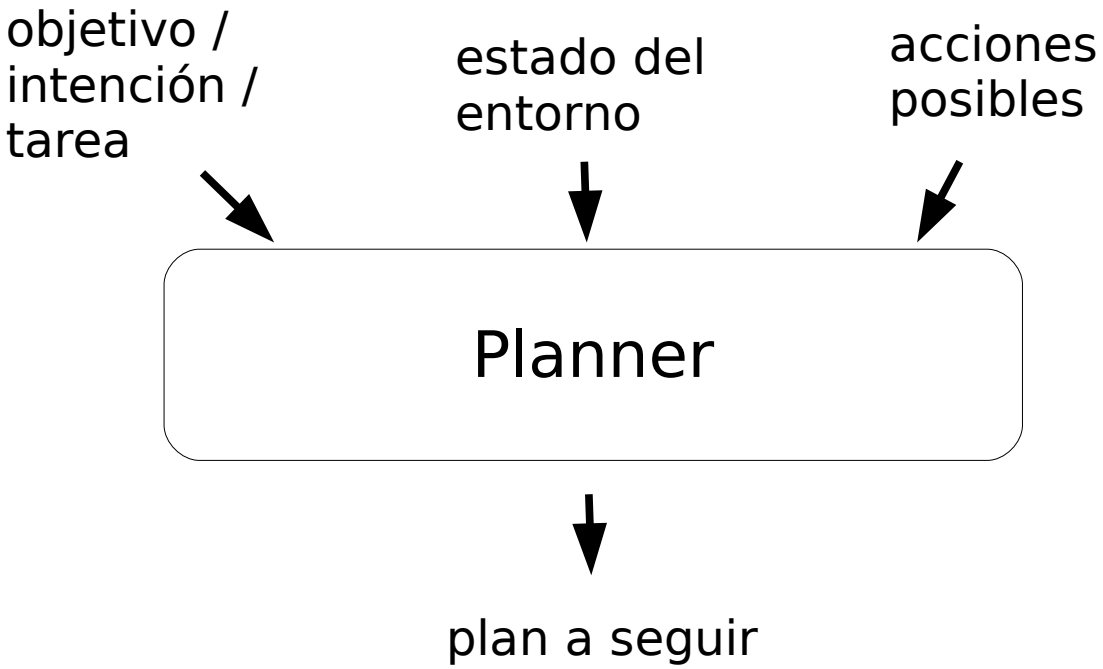
- RAE
  - plan.  (De plano).

    3. m. Modelo sistemático de una actuación pública o privada, que se elabora anticipadamente para dirigirla y encauzarla.
  - planificación.

    2. f. Plan general, metódicamente organizado y frecuentemente de gran amplitud, para obtener un objetivo determinado, tal como el desarrollo armónico de una ciudad, el desarrollo económico, la investigación científica, el funcionamiento de una industria, etc.

# Introducción

- Desde principios de los ‘70, la comunidad de IA especializada en planificación se ha preocupado del problema de diseño de agentes artificiales capaces de actuar en un entorno.
- La planificación se puede ver como una forma de programación automática: el diseño de un curso de acción que satisfará un cierto objetivo
- Dentro de la comunidad de la IA simbolica, se ha asumido desde hace tiempo que algun tipo de sistema planificador debe formar parte de los componentes centrales de cualquier agente artificial
- La idea básica es dotar al agente planificador:
  - representación del **objetivo** a alcanzar
  - representación de las **acciones** que puede realizar
  - representación del **entorno**
  - Capacidad de generar un **plan** para alcanzar el objetivo

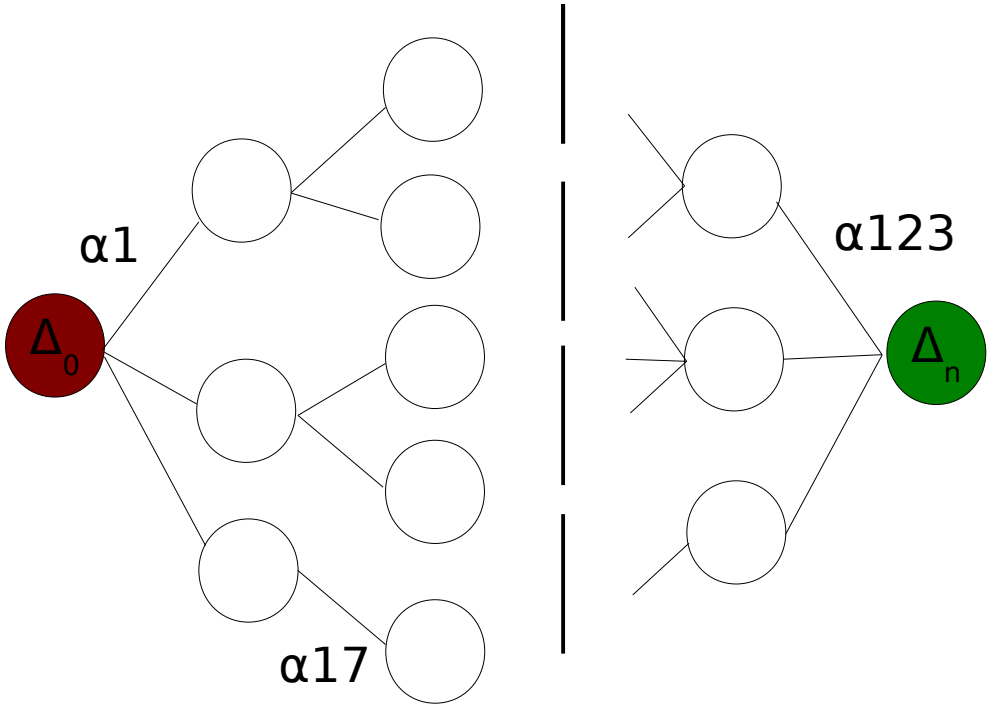



Cómo representar ...

- Objetivo a alcanzar
- Estado del entorno
- Acciones disponibles para el agente
- El propio plan

# Introducción

- Un plan es una secuencia (lista) de acciones, que llevan de un estado inicial a un estado final.
- La planificación se puede ver como un problema de búsqueda en un espacio de estados.



# Introducción

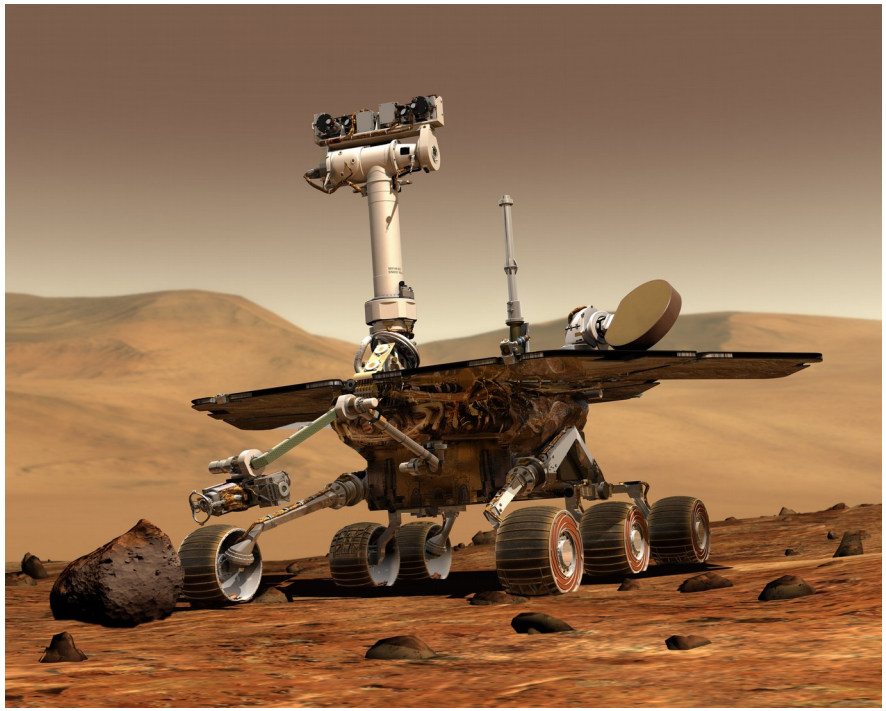
- Los algoritmos de búsqueda clásicos se interesan sólo en devolver el estado final o estado-solución.
- Los algoritmos de planificación no solo se interesan por encontrar el estado solución, sino en mantener todos los estados intermedios que llevan desde el estado inicial al final.
- Los algoritmos de planificación suelen usar no solo el conocimiento dentro del heurístico, sino también las descripciones de los efectos de las acciones para guiar su búsqueda (utilizan la estructura lógica del problema).
- Muchos algoritmos de planificación reducen la complejidad del problema descomponiendolo en sub-objetivos
  - Esto solo se puede realizar en problemas reales que sean descomponibles o quasi-descomponibles (el planificador descompone el problema y luego resuelve pequeños conflictos al recomponer la solución)

# Introducción

- Nuestra visión del mundo es incompleta: racionalidad limitada
- El mundo cambia constantemente: dinamismo
- Las acciones tardan en ejecutarse: razonamiento temporal
- Nuestras metas son contradictorias: dependencia entre metas
- No todos los planes son buenos: calidad
- Los planes no siempre son válidos: ejecución y replanificación
- El mundo no es determinista: incertidumbre
- Nos adaptamos al mundo: aprendizaje
- La planificación y la filosofía: creencias, intenciones y deseos

# Introducción

- Mars Exploration Rovers [NASA]
  - La planificación de las tareas a realizar durante un día marciano se realiza automáticamente por un programa a partir de los objetivos de exploración que fija el personal de misión en la Tierra.



# Introducción

- Tipos de Planificadores:
  - Planificadores específicos del dominio:
    - Específicamente diseñados para un dominio y difícilmente pueden utilizarse en otros dominios: muchas aplicaciones prácticas.
  - Planificadores independientes del dominio:
    - El mecanismo de planificación es lo suficientemente general como para utilizarse en dominios que cumplan determinadas restricciones: No eficientes y No aplicaciones prácticas
  - Planificadores configurables al dominio:
    - El mecanismo de planificación es independiente del dominio, pero la entrada al planificador incluye conocimiento del dominio para restringir la búsqueda del planificador.

# Planificación clásica

- Restricciones del dominio (1):
  - Restricción 0: **finito**. Conjunto finito de estados.
  - Restricción 1: **totalmente observable**. Se tiene un conocimiento completo del entorno. el planificador percibe perfectamente el estado del entorno y el efecto de sus acciones en el entorno
  - Restricción 2: **determinista**. Se pueden predecir y predefinir los efectos de todas las acciones).
  - Restricción 3: **estático**. Los cambios suceden sólo cuando actúa el agente planificador.
  - Restricción 4: **discreto**. El entorno se puede describir de forma discreta:
    - Tiempo, acciones, objetos, efectos, ...

# Planificación clásica

- Restricciones del dominio (2):
  - Restricción 5: **tiempo implícito.** Las acciones no tienen duración, los estados de transición son instantáneos. No representan el tiempo explícitamente.
  - Restricción 6: **planes secuenciales.** Una solución es una secuencia finita linealmente ordenada de acciones.
  - Restricción 7: **planificación off.line.** El planificador no tiene en cuenta cualquier cambio que se pueda producir en el entorno mientras está planificando. Planifica según el estado inicial y objetivos dados sin observar los cambios, si lo hay.
  - Restricción 8: **objetivo alcanzable (!)**

# Planificación clásica

- Restricciones del dominio (2):
  - Restricción 5: **tiempo implícito.** Las acciones no tienen duración, los estados de transición son instantáneos. No representan el tiempo explícitamente.
  - Restricción 6: **planes secuenciales.** Una solución es una secuencia finita linealmente ordenada de acciones.
  - Restricción 7: **planificación off.line.** El planificador no tiene en cuenta cualquier cambio que se pueda producir en el entorno mientras está planificando. Planifica según el estado inicial y objetivos dados sin observar los cambios, si lo hay.
  - Restricción 8: **objetivo alcanzable (!)**

# Planificación clásica

- $Ac = \{\alpha_1, \ldots, \alpha_n\}$: un conjunto fijo de acciones.
- $< P_\alpha, D_\alpha, A_\alpha >$ es un descriptor para una acción $\alpha \in Ac$
- $P_\alpha$ es un conjunto de formulas que caracterizan la precondición de la acción $\alpha$
- $D_\alpha$ es un conjunto de fórmulas que caracterizan aquellos hechos que se vuelven falsos por la ejecución de $\alpha$ (‘delete list’)
- $A_\alpha$ es un conjunto de fórmulas que caracterizan aquellos hechos que se vuelven ciertos por la ejecución de $\alpha$ (‘add list’)
- $\pi = (\alpha_1, \ldots, \alpha_n)$ es un plan con respecto al problema de planificación

# Planificación clásica

- Representación abierta del conocimiento sobre estados, objetivos y acciones.
  - Lenguaje formal
    - Ej: lógica de predicados, lógica de primer orden, ...
  - Estados y metas (objetivos) se representan por conjuntos de sentencias lógicas
    - Ej: *en(p1, bcn), avion(p1), ...*
    - Es habitual en algunos planificadores que lo que no aparece explícitamente representado en un estado es falso: suposición del mundo cerrado.
  - Las acciones se representan por descripciones lógicas de precondiciones y efectos
    - Ej: *acción(volar(A, O, D),*

      *PRECOND:*
      *en(A,O)∧avion(A)∧aeropuerto(O)∧aeropuerto(D),*
      *EFECTO: ¬en(A,O)∧en(A,D))*

# Planificación clásica

- STRIPS (STanford Research Institute Problem Solver) (Fikes & Nilsson 1971) fue el primero de los grandes sistemas de planificación.
- Para ciertos problemas reales se ha demostrado que STRIPS no posee suficiente expresividad.
- El lenguaje ADL (Action Description Language) amplia ideas de STRIPS
- Las notaciones STRIPS y ADL son adecuadas para muchos dominio reales.

# Planificación clásica

- Desde 1998 la comunidad de investigadores en planificación ha desarrollado un lenguaje standard de descripción de planes: Planning Domain Description Language (PDDL)
- Objetivo inicial: lenguaje común para competición mundial de planificadores.
- En la actualidad se ha convertido en un estándar de facto
- WARNINGS:
  - existen varias versiones de PDDL, desde la 1.0 a la 3.1, cada una de ellas con diferentes niveles de expresividad
  - No existe ningún planificador que soporte la especificación 3.1 completa, sino subconjuntos de ella.

# Planificación clásica: STRIPS

- Representación de estados: los planificadores descomponen el mundo en condiciones lógicas, representando un estado como una conjunción de literales positivos:
  - Proposiciones: *pobre* ∧ *aburrido*
  - Literales de 1er orden: *en(avion1, bcn)* ∧ *en(avion2, jfk)*
- Representación de objetivos: un objetivo es un estado parcialmente especificado
  - Un estado s satisface un objetivo o si s contiene todos los atomos de o (y posiblemente algunos más)
    - Ej: el estado *rico* ∧ *famoso* ∧ *guapo* satisface el objetivo *rico* ∧ *famoso*

# Planificación clásica: STRIPS

- Representación de acciones: Las acciones se especifican en terminos de las precondiciones que se han de cumplir antes de que se puedan ejecutar y de los efectos que producen una vez se han ejecutado
  - Ej: *acción(volar(A, O, D),*

    *PRECOND:*
    $en(A,O) \wedge avion(A) \wedge aeropuerto(O) \wedge aeropuerto(D)$,

    *EFECTO:* $\neg en(A,O) \wedge en(A,D))$
- La precondición es una conjunción de literales positivos que especifica que debe de ser verdadero en un estado antes de que la accion se ejecute. Todas las variables en la precondición han de aparecer en la lista de parámetros de la acción.
- El efecto es una conjunción de literales describiendo como cambia el estado cuando la acción se ejecuta. Todas las variables han de aparecer también en la lista de parámetros de la acción.

# Planificación clásica: STRIPS

- Una acción es aplicable en cualquier estado que satisfaga la precondición
  - En 1er orden: existe una substitución para las variables en la precondición. Por ejemplo, el estado:

    *en(a1, jfk) ∧ avion(a1) ∧ en(a2, bcn) ∧ avion(a2) ∧ aeropuerto(jfk) ∧ aeropuerto(bcn)*
  - satisface la precondición de la acción volar:

    *en(A, O) ∧ avion(A) ∧ aeropuerto(O) ∧ aeropuerto(D)*
- El resultado de ejecutar la acción en un estado s es un estado s' al que se añaden los literales positivos del efecto y se eliminan los literales negativos
  - Por ejemplo, el efecto de la acción volar sobre el estado anterior:
    - *en(a1, bcn) ∧ avion(a1) ∧ en(a2, bcn) ∧ avion(a2) ∧ aeropuerto(jfk) ∧ aeropuerto(bcn)*
  - Se ha eliminado: *en(a1,jfk)*

# Ejemplo STRIPS: Transporte aereo de carga

- Dos cargas (c1 y c2) estan en 2 aeropuertos (bcn, jfk)
- Tenemos dos aviones (a1 y a2) para transportar las cargas, uno en cada aeropuerto
- Describimos el estado inicial así:
  - *inicio( en(c1, bcn) ∧ en(c2, jfk) ∧ en(a1, bcn) ∧ en(a2, jfk) ∧ carga(c1) ∧ carga(c2) ∧ avion(a1) ∧ avion(a2) ∧ aeropuerto(bcn) ∧ aeropuerto(jfk) )*
- El objetivo es que c1 acabe en jfk y c2 en bcn
- Describimos el objetivo así:
  - *objetivo( en(c1, jfk) ∧ en(c2, bcn) )*

# Ejemplo STRIPS: Transporte aereo de carga

- Describimos las acciones de cargar, descargar y volar:
  - *acción(cargar(C, A, AE),*

    *PRECOND: en(C, AE) ∧ en(A, AE) ∧ carga(C) ∧ avion(A) ∧ aeropuerto(AE)*

    *EFECTO: ¬en(C, AE) ∧ dentro(C, A) )*
  - *acción(descargar(C, A, AE),*

    *PRECOND: dentro(C, A) ∧ en(A, AE) ∧ carga(C) ∧ avion(A) ∧ aeropuerto(AE)*

    *EFECTO: en(c, AE) ∧ ¬dentro(C, A) )*
  - *acción(volar(A, O, D),*

    *PRECOND: en(A, O) ∧ avion(A) ∧ aeropuerto(O) ∧ aeropuerto(D)*

    *EFECTO: ¬en(A, O) ∧ en(A, D) )*

# Ejemplo STRIPS: Transporte aereo de carga

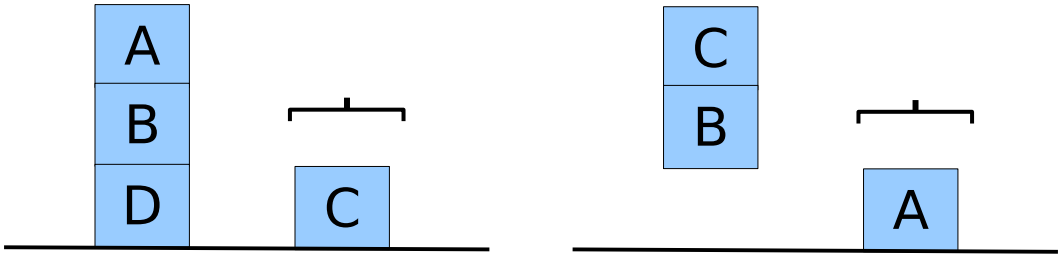
- Solución: el plan lo compone una secuencia de acciones.
- En este caso hay varias soluciones:
  - Ej. Solucion 1: usamos los dos aviones para hacer el traslado

    *[cargar(c1, a1, bcn), volar(a1, bcn, jfk), descargar(c1, a1, jfk), cargar(c2, a2, jfk), volar(a2, jfk, bcn), descargar(c2, a2, bcn)]*
  - Ej. Solucion 2: usamos solo un avión

    *[cargar(c1, a1, bcn), volar(a1, bcn, jfk), descargar(c1, a1, jfk), cargar(c2, a1, jfk), volar(a1, jfk, bcn), descargar(c2, a1, bcn)]*

# Ejemplo STRIPS: el mundo de bloques

- Un conjunto de bloques, una mesa y un brazo de un robot.
- Todos los bloques son iguales de tamaño, forma y color. Se diferencian en el nombre.
- La mesa tiene extensión ilimitada
- Cada bloque puede estar encima de la mesa, encima de un solo bloque o sujeto por el brazo del robot.
- La mano del robot sólo puede sujetar un bloque cada vez.
- Resolver problemas supone pasar de una configuración (estado) inicial a un estado en el que sean ciertas unas metas.




Estado inicial

Estado final

# Ejemplo STRIPS: el mundo de bloques

- Se podrían utilizar los siguientes predicados:
  - sobre(X, Y) : el bloque x está encima del bloque y.
  - en_mesa(X) : el bloque x está encima de la mesa.
  - libre(X) : el bloque x no tiene ningún bloque encima.
  - en_mano(X) : el robot tiene en su mano el bloque x
  - mano_libre : el robot no tiene ningún bloque en su mano
- Estado inicial
  - sobre(a, b), sobre(b, d), en_mesa(d), en_mesa(c), libre(a), libre(c), mano_libre
- Metas:
  - en_mesa(a), sobre(c, b)

# Ejemplo STRIPS: el mundo de bloques

- DESAPILAR(X, Y)
  *precondición:* sobre(X,Y), libre(X), mano_libre
  *añadido:* en_mano(X), libre(Y)
  *borrado:* sobre(X, Y), libre(X), mano_libre
- COGER(X)
  *precondición:* en_mesa(X), libre(X), mano_libre
  *añadido:* en_mano(X)
  *borrado:* en_mesa(X), libre(X) , mano_libre
- APILAR(X, Y)
  *precondición:* en_mano(X), libre(Y)
  *añadido:* sobre(X, Y), libre(X), mano_libre
  *borrado:* en_mano(X), libre(Y)
- DEJAR(X)
  *precondición:* en_mano(X)
  *añadido:* en_mesa(X), libre(X), mano_libre
  *borrado:* en_mano(X)

# Ejemplo: The Dock Worker Robots (DWR) domain

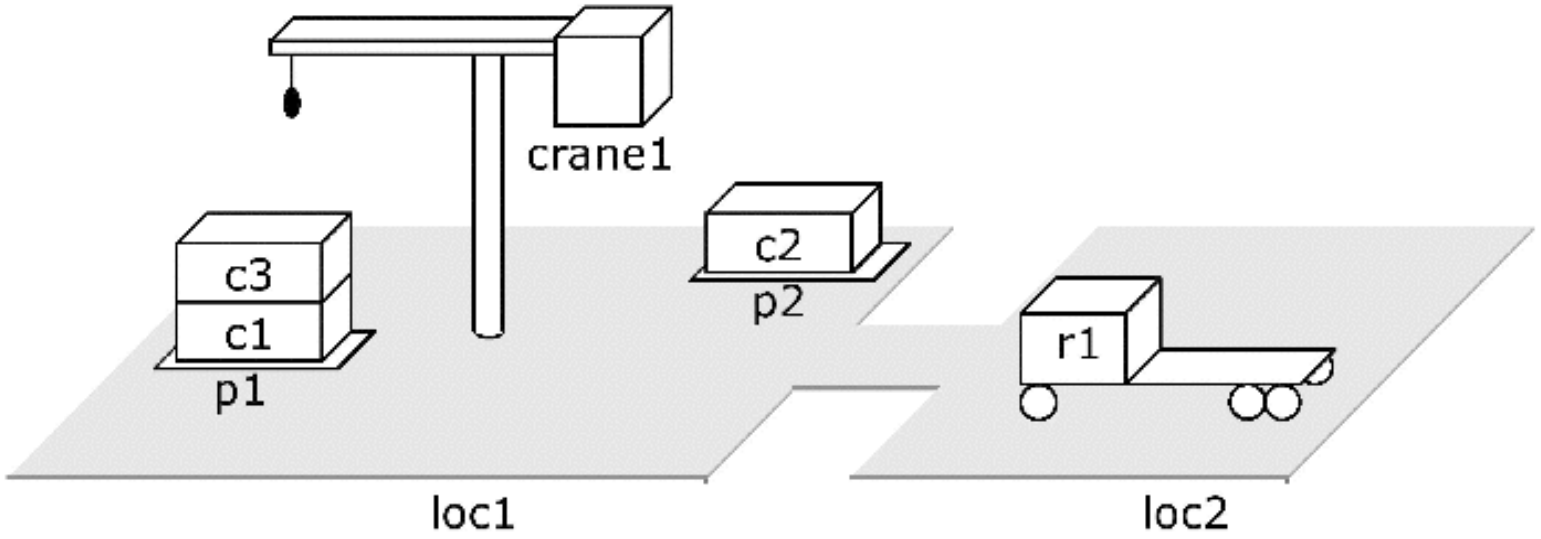
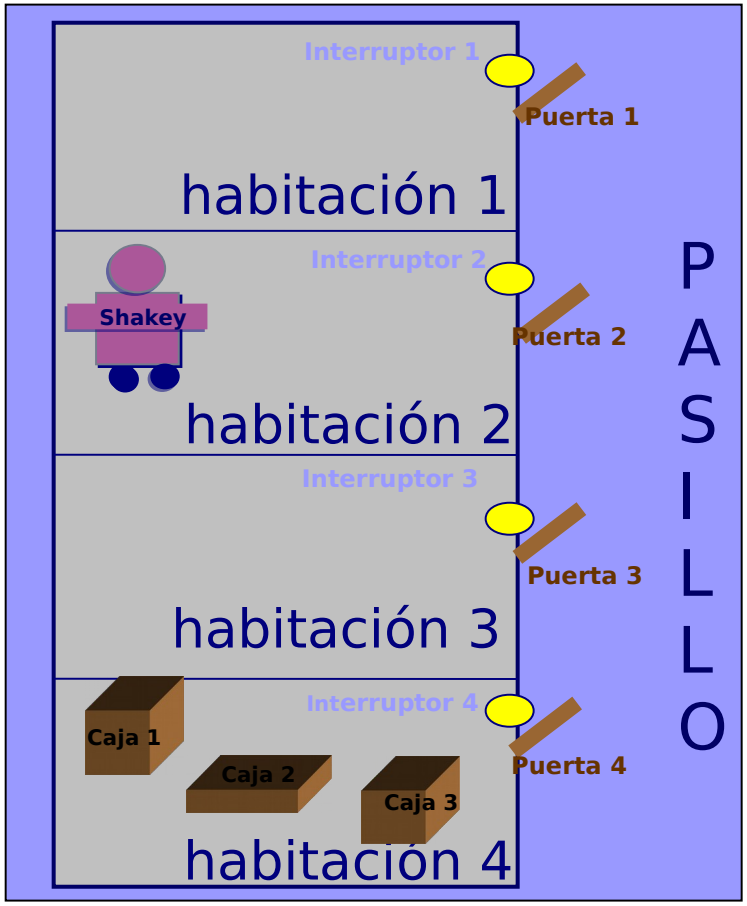



Figure 2.2: The DWR state $s_1$={attached(p1,loc1), in(c1,p1), in(c3,p1), top(c3,p1), on(c3,c1), on(c1,pallet), attached(p2,loc1), in(c2,p2), top(c2,p2), on(c2,pallet), belong(crane1,loc1), empty(crane1), adjacent(loc1,loc2), adjacent(loc2,loc1), at(r1,loc2), occupied(loc2), unloaded(r1)}.

# Ejemplo: el mundo de Shakey




Estado Inicial

Shakey es un robot que se puede mover entre varias habitaciones, empujar objetos, trepar a objetos rígidos, encender y apagar las luces.

# Lenguaje PDDL

## Fichero de descripción 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
  ...)

...)
```

Como hay diferentes niveles de expresividad posibles, cada descripción en PDDL dice los requisitos necesarios para procesarla. Los más comunes son:
:strips expresividad como en STRIPS
:equality el dominio usa el predicado =
:typing el dominio define tipos de vars.
:adl expresividad extendida:
1) disyunciones y cuantificadores en precondiciones y objetivos,
2) Efectos cuantificados y condicionales

# Lenguaje PDDL

Fichero de descripción del problema

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

# Lenguaje PDDL

## Fichero de descripción del dominio

```
(define (domain driverlog)
  (:requirements :strips :typing)
  (:types location locatable - object
          driver truck obj - locatable
  )
  (:predicates
    (at ?obj - locatable ?loc - location)
    (in ?obj1 - obj ?obj - truck)
    (driving ?d - driver ?v - truck)
    (link ?x ?y - location) (path ?x ?y - location)
    (empty ?v - truck)
  )
  (:action LOAD-TRUCK
    :parameters
      (?obj - obj
       ?truck - truck
       ?loc - location)
    :precondition
      (and (at ?truck ?loc) (at ?obj ?loc))
    :effect
      (and (not (at ?obj ?loc)) (in ?obj ?truck)))
```

## Fichero de descripción del problema

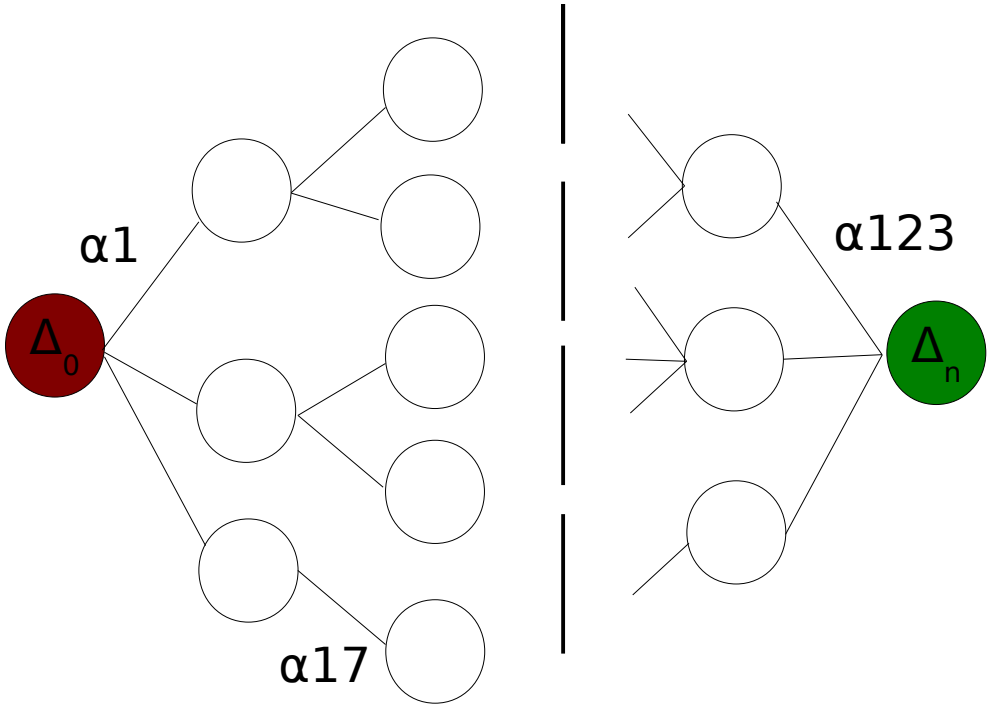
```
(define (problem DLOG-2-2-2)
  (:domain driverlog)
  (:objects
    driver1 - driver
    truck1 - truck
    package1 - obj
    s0 - location
    s1 - location ...)
  (:init
    (at driver1 s12)
    (at truck1 s0)
    (empty truck1)
    (at package1 s0)
    (path s1 p1-0)
    (path p1-0 s1)
    ...
    (link s0 s1)
    (link s1 s0)
    ... )
  (:goal (and (at driver1 s1)
              (at truck1 s1)
              (at package1 s0)
  )))
```

# Planificación automática clásica

- Aproximaciones más relevantes:
  - Planificación en el espacio de estados (State-space planning)
  - Planificación en el espacio de planes (Plan-space planning ó PSP)
  - Planificación jerárquica (Hierarchical Task Network Planning ó HTN)
- Otros resultados interesantes
  - Reutilización de Planes
  - Planificación específica de un dominio (Domain-specific planning)

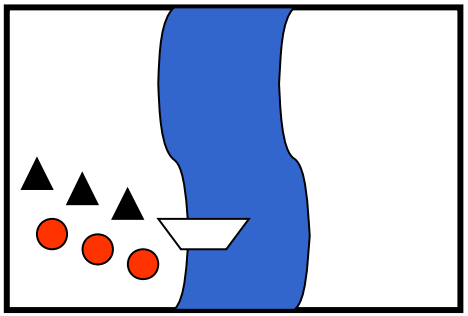
# Espacio de estados (State-space)

- Cada nodo representa un estado del mundo
- Los estados se define mediante un conjunto de predicados y variables
- Un plan es un camino dentro del espacio de estados

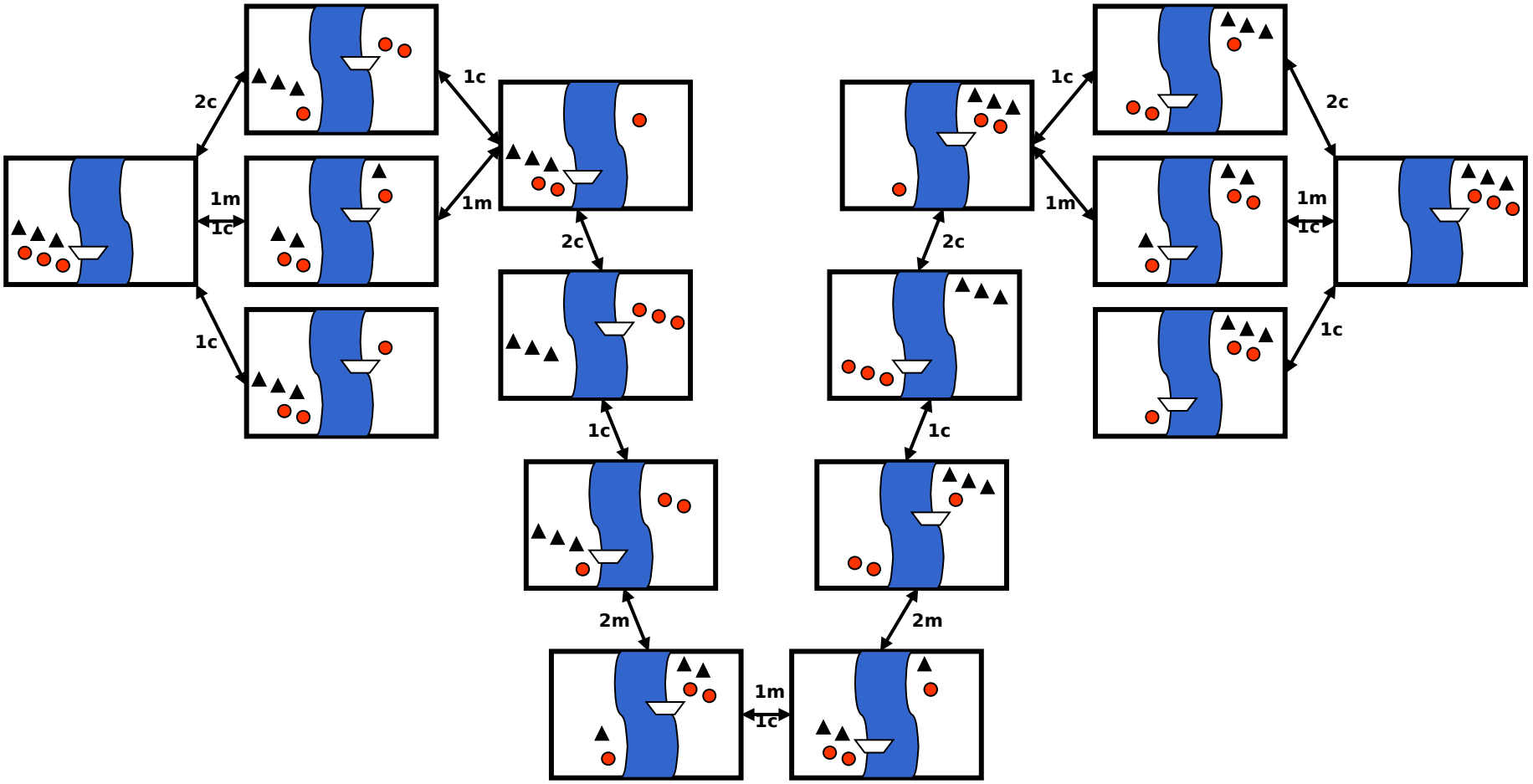


# Espacio de estados: Misioneros y caníbales

- Estado inicial:
  - los misioneros, los caníbales, y el barco se encuentran en la margen izquierda
- 5 acciones posibles:
  - Cruza un misionero
  - Cruza un caníbal
  - Cruzan dos misioneros
  - Cruzan dos canívales
  - Cruza un misionero y un caníval

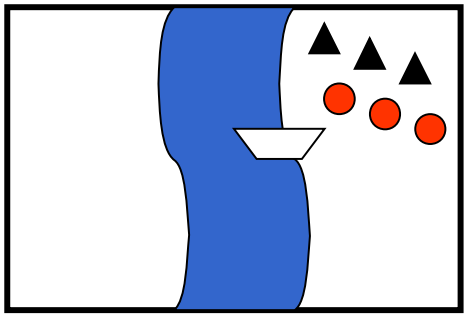


# Espacio de estados: Misioneros y caníbales



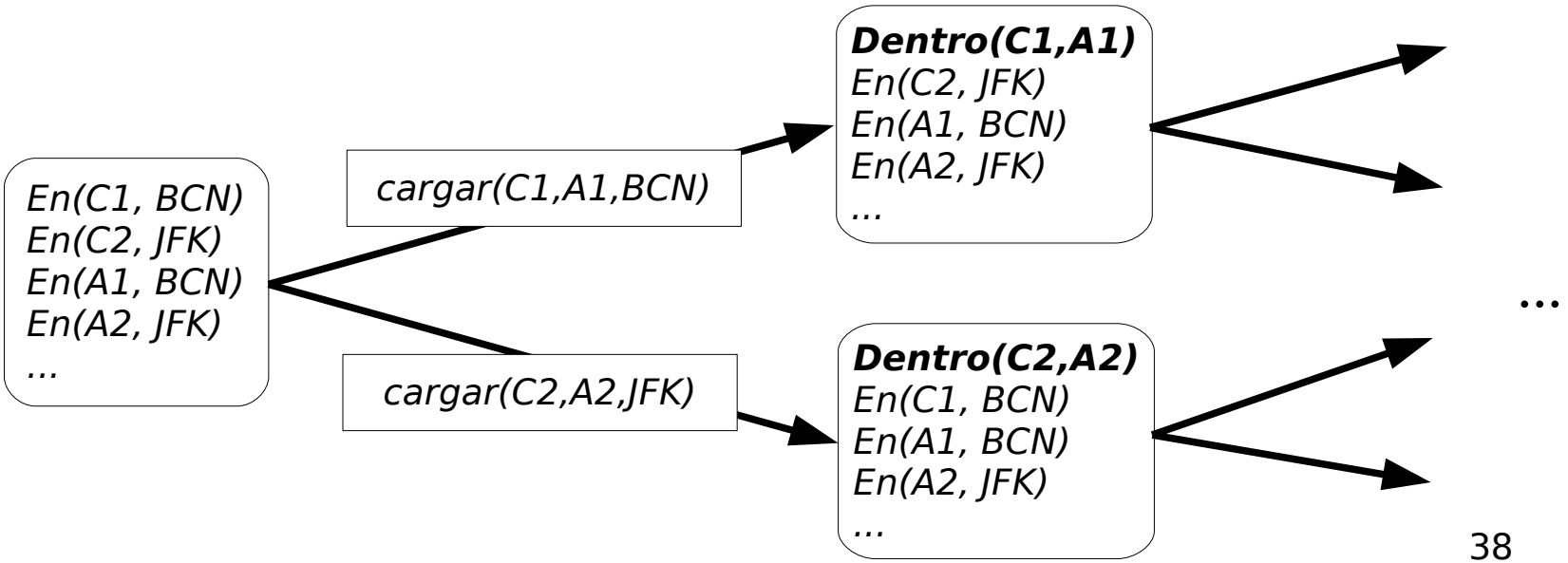
# Espacio de estados: Misioneros y caníbales

- Estado objetivo:
  - los misioneros, los caníbales, y el barco se encuentran en la margen derecha
- Coste de la solución:
  - Coste por arco: 1 por cada cruce
  - Coste del camino: número de cruces = longitud del camino
- Camino solución:
  - Cuatro soluciones óptimas
  - Coste = 11



# Estrategias en espacio de estados

- Busqueda hacia delante (planificación progresiva)
  - El estado inicial de la búsqueda es el estado inicial del problema
  - En cada momento se intenta unificar con las precondiciones de las acciones
  - Se cambia la descripción del estado añadiendo o eliminando literales de los efectos de las acciones



# Planificación progresiva determinista

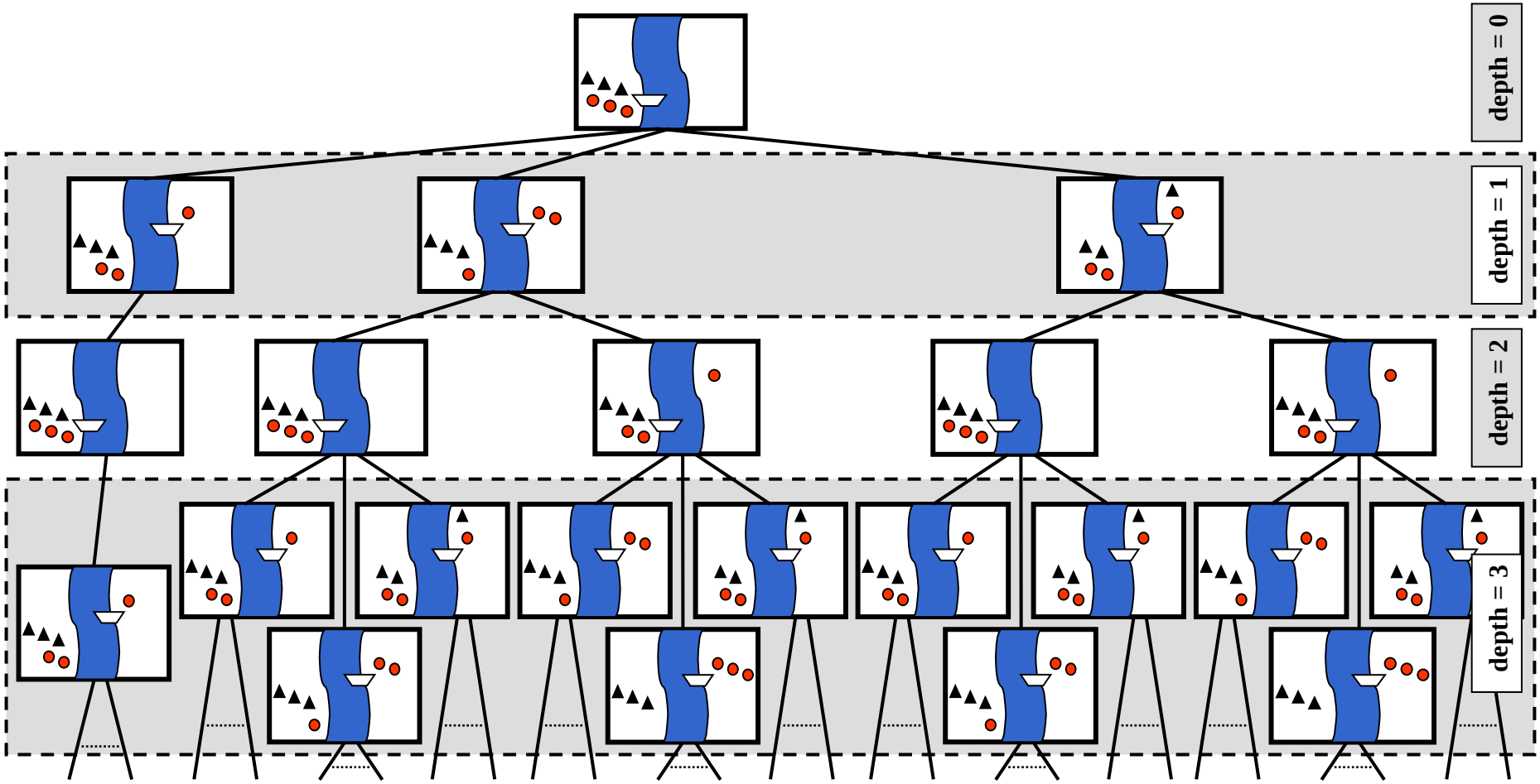
- Implementaciones deterministas de búsqueda hacia adelante:
  - Anchura prioritaria (*breadth-first search*)
  - Profundidad prioritaria (*depth-first search*)
  - best-first search (ej.: A*)
  - greedy best first
- Anchura prioritaria y best first son completas...
  - ... pero no suelen ser prácticas al necesitar demasiada memoria (exponencial en la longitud de la solución)
- En la práctica se suelen usar profundidad prioritaria o greedy
  - Problema: no son completas
  - Pero la planificación clásica tiene un conjunto finito de estados
  - Profundidad prioritária se puede hacer completa controlando los ciclos

# Planificación progresiva determinista

- Problema: Cuando el factor de ramificación es muy elevado:
  - Existen muchas acciones aplicables que no nos llevan al objetivo
  - Las implementaciones deterministas pueden perder mucho tiempo probando múltiples acciones irrelevantes

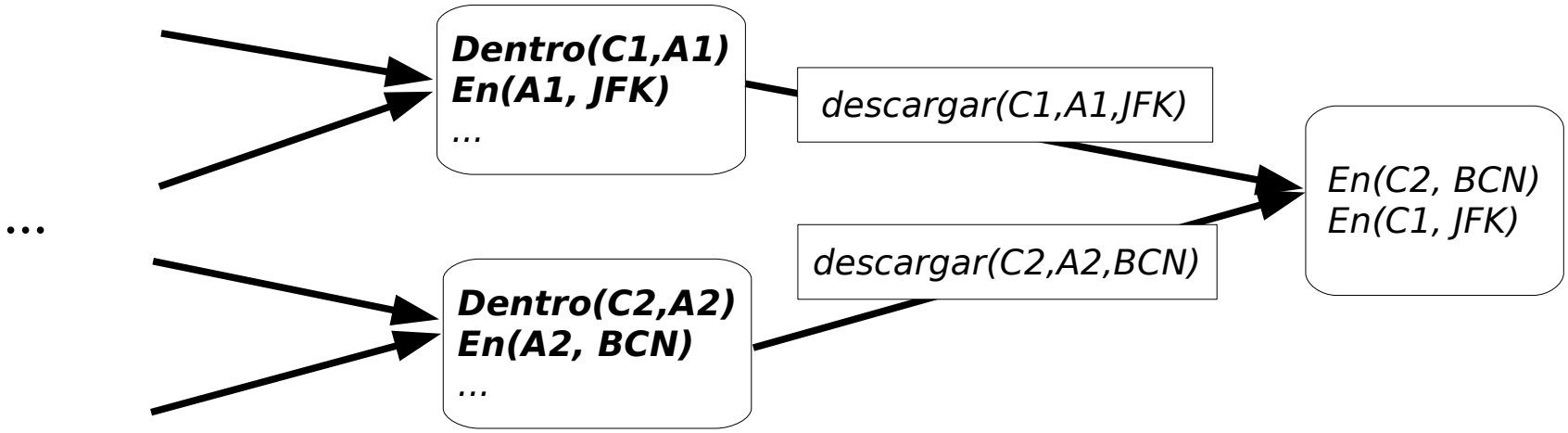
- Una posible solución: añadir heurísticos específicos del dominio

# Espacio de estados: Misioneros y caníbales



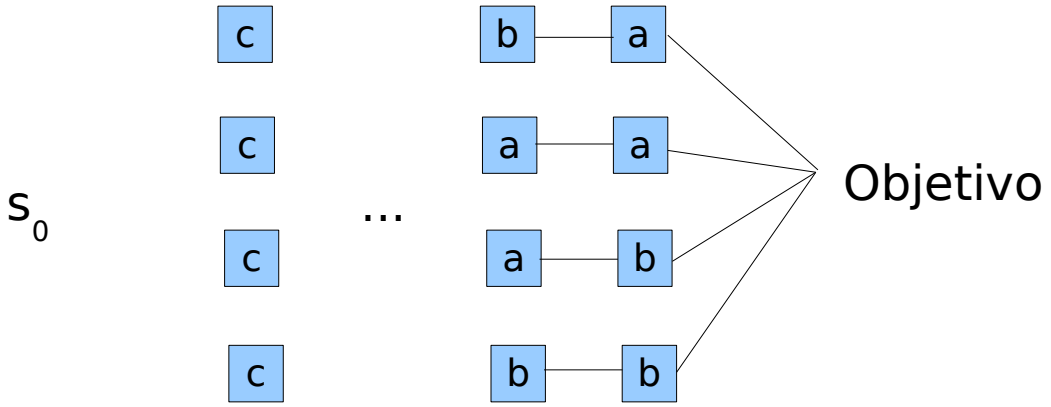
# Estrategias en espacio de estados

- Busqueda hacia atrás (planificación regresiva)
  - El estado inicial de la búsqueda es el estado final del problema
  - En cada momento se intenta unificar con los efectos de las acciones. Los efectos positivos se eliminan de la descripción.
  - Se añaden los literales de las precondiciones excepto si ya aparecen en la descripción actual
  - La búsqueda acaba cuando todas las precondiciones son satisfechas por el estado inicial del problema



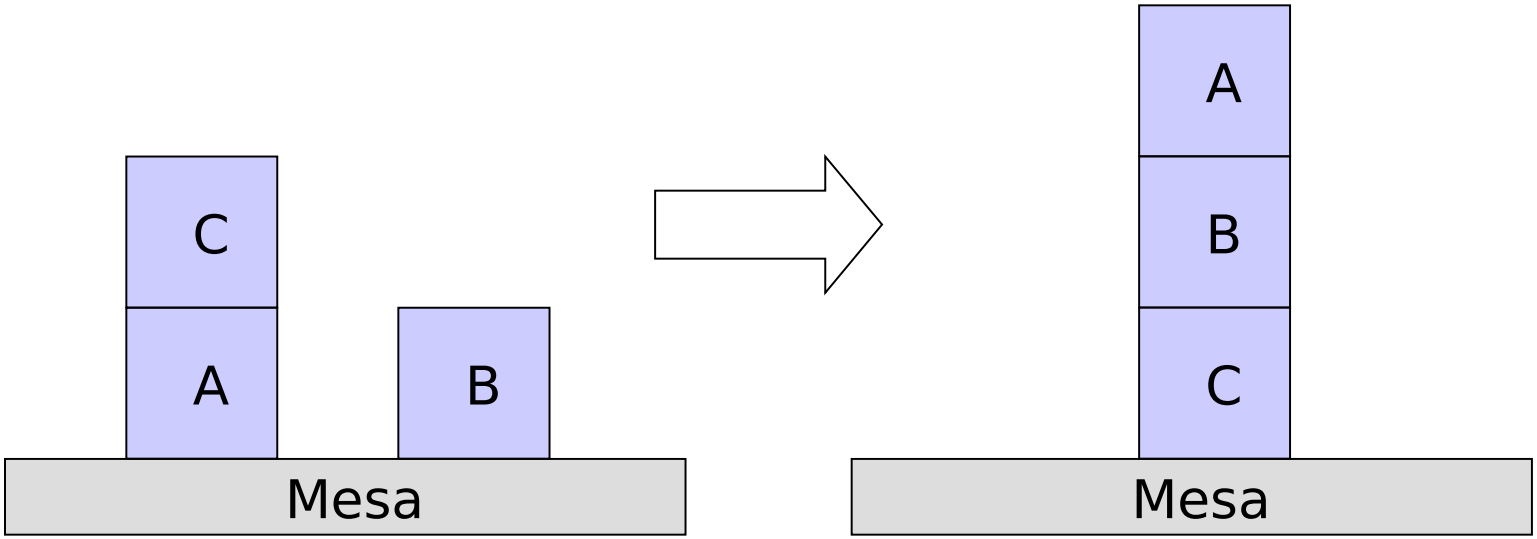
# Estrategias en espacio de estados

- Problema de la planificación regresiva
  - Aunque genera un espacio de busqueda algo más pequeño, aún puede ser muy grande y algo ineficiente
- Ejemplo:
  - En el caso de tres acciones *a y b* independientes, una acción *c* que las ha de preceder siempre, y que no hay ningún camino desde $s_0$ al estado necesario como input de *c*
  - El algoritmo intenta todas las ordenaciones posibles de *a y b* antes de darse cuenta que no hay solución.



# Problemas con STRIPS

- STRIPS no es completo:
  - no puede encontrar una solución para algunos problemas, por ejemplo, intercambiando los valores de dos variables
  - no puede encontrar la solución óptima en otros, por ejemplo, anomalía Sussman (1975):




{sobre(A,B), sobre(B,C)}

# Problemas con STRIPS

- Entrelazado planes para una solución óptima:
  - Solución más corta para lograr sobre(A, B):
    - poner C de la A a la mesa
    - poner A en B
  - Solución más corta para lograr sobre(B, C):
    - poner B en C
  - Solución más corta para lograr sobre(A, B) y sobre(B, C):
    - poner C de la A a la mesa
    - poner B en C
    - poner A en B
  - La solución óptima no se puede encontrar por el algoritmo STRIPS porque:
    - no puede cambiar el sub-objetivo durante la búsqueda y para tan pronto como encuentra una ruta de acceso al estado inicial

# Espacio de planes (Plan-space)

- Idea:
  - Búsqueda hacia atrás desde el objetivo
  - Cada nodo del espacio de búsqueda es un plan parcial que incluye:
    - un conjunto de operadores parcialmente instanciados
    - un conjunto de restricciones sobre los operadores
- Proceso:
  - Descomponer conjuntos de objetivos en objetivos individuales
  - Planificar para cada uno de ellos por separado
  - Se van detectando y resolviendo los ‘fallos’ que hacen que aun no sea un plan, imponiendo más y más restricciones hasta que se tiene un plan parcialmente ordenado.
- Una extensión de planificación temporal en el espacio de planes se ha usado en los Mars Rovers de NASA.

# Planes totales vs. parciales

- Principio del menor compromiso:
  - Sólo hay que hacer elecciones de lo que interesa en cada momento, dejando las otras elecciones para más tarde.

- Planificador de orden total:
  - genera una lista de pasos, conocido como “linealización” de un plan P al añadir restricciones de orden a P.

- Planificador de orden parcial (POP):
  - algoritmo que puede representar algunas acciones ordenadas entre sí, y otras quedan sin ordenar

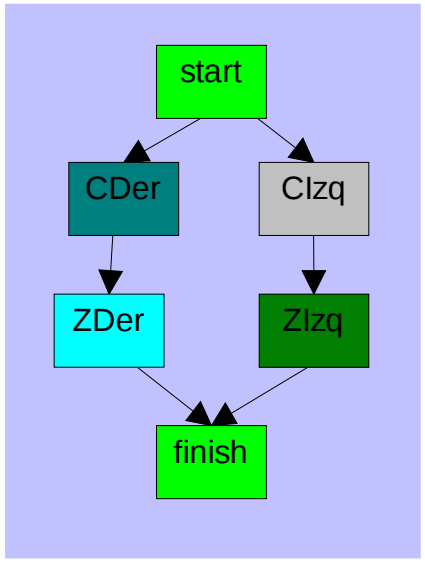
# Principio del menor compromiso

- Problema del calzado de los pies:
  - Op(ACTION: ZapatoDerecho,
    PRECOND:CalcetinDerechoPuesto,
    EFECTO: ZapatoDerechoPuesto)
  - Op(ACTION: CalcetinDerecho,
    EFECTO: CalcetinDerechoPuesto)
  - Op(ACTION: ZapatoIzquierdo,
    PRECOND:CalcetinIzquierdoPuesto,
    EFECTO: ZapatoIzquierdo Puesto)
  - Op(ACTION: CalcetinIzquierdo,
    EFECTO: CalcetinIzquierdoPuesto)

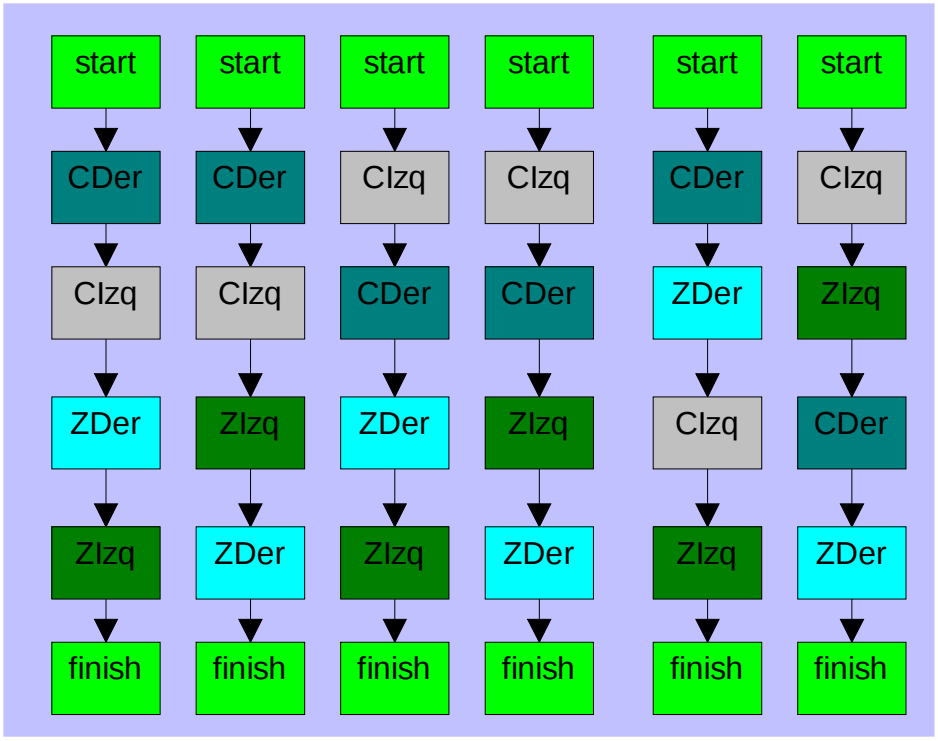
# Principio del menor compromiso

- Solución al problema del calzado de los pies:

Plan de orden parcial




Planes de orden totales



# Espacio de planes (Plan-space)

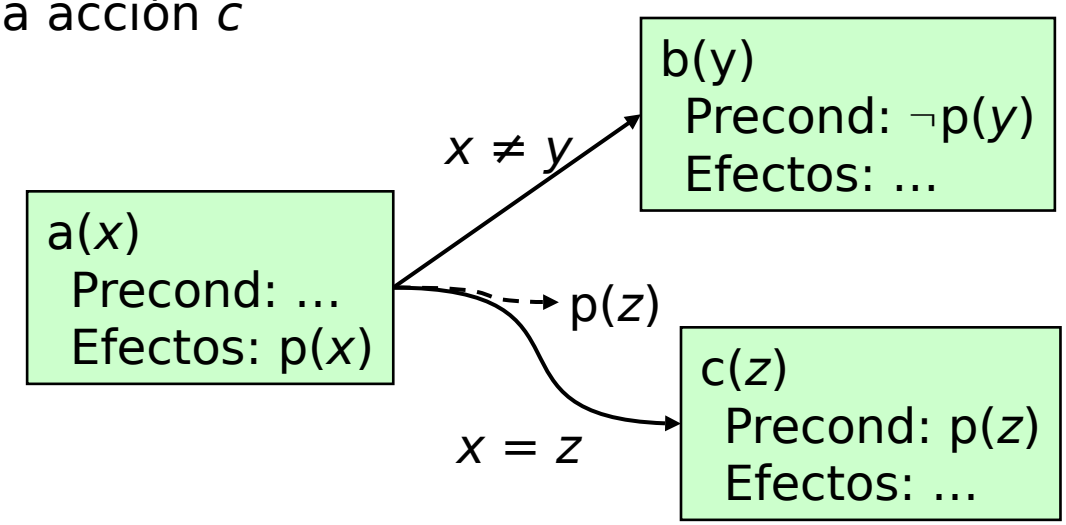
- Búsqueda en el espacio de planes: buscar a través de grafo de planes parciales
  - nodos: planes parcialmente especificados
  - arcos: operaciones de refinamiento del plan
    - Principio del menos de compromiso: no añadir restricciones al plan que no están estrictamente necesarios
  - soluciones: planes de orden parcial
    - plan de orden parcial: conjunto de acciones + conjunto de ordenamientos, y no necesariamente ordenados totalmente
  - Los algoritmos de búsqueda en espacios de estado también mantienen un plan parcial - pero siempre en orden total

# Espacio de planes (Plan-space)

- Un plan parcial contiene acciones
  - estado inicial
  - condiciones objetivo
  - conjunto de los operadores con diferentes variables
    - se puede representar como dos acciones con sólo efectos o precondiciones
- Incorporación de nuevas acciones
  - Para cumplir condiciones previas insatisfechas
  - Para cumplir condiciones objetivo insatisfechas
- Las nuevas acciones siempre se pueden añadir en cualquier lugar del plan parcial vigente

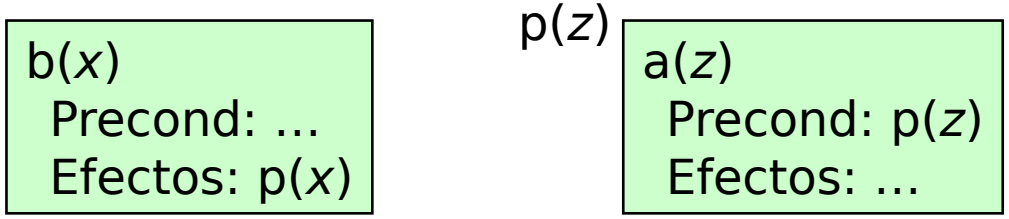
# Espacio de planes (Plan-space)

- 3 tipos de restricciones:
  - restricciones de precedencia: *a* debe preceder a *b*
  - restricciones de asignación:
    - restricciones de desigualdad: x ≠ y
    - restricciones de igualdad: x = z
  - enlaces causales (causal links):
    - usar la acción *a* para obtener la precondición *p* necesaria para la acción *c*

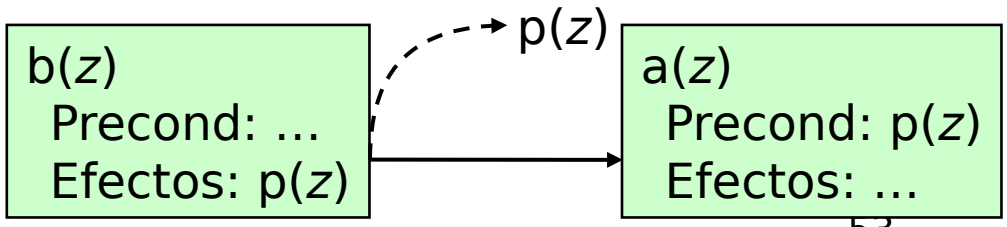


# Plan-Space Planning: Fallos - 1. Objetivos abiertos

- Objetivo abierto:
  - Una acción *a* tiene una precondición *p* que no hemos decidido como obtener

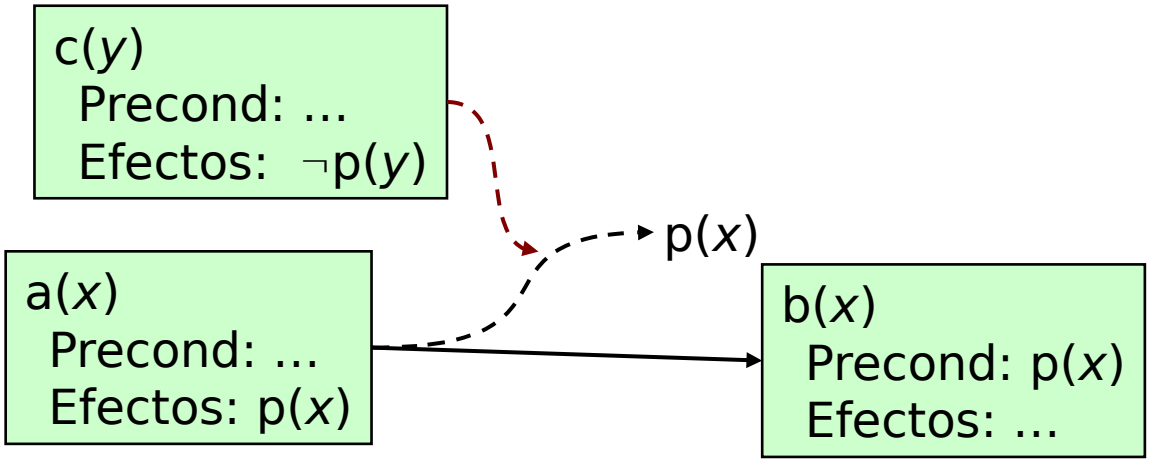



- Resolviendo el fallo:
  - Encontrar una acción b (ya en el plan o añadirla) que pueda usarse para obtener p
    - Puede preceder a y producir p
  - Instanciar variables y/o restringir las asignaciones de variables
  - Crear un enlace causal



# Plan-Space Planning: Fallos - 2. Ataques

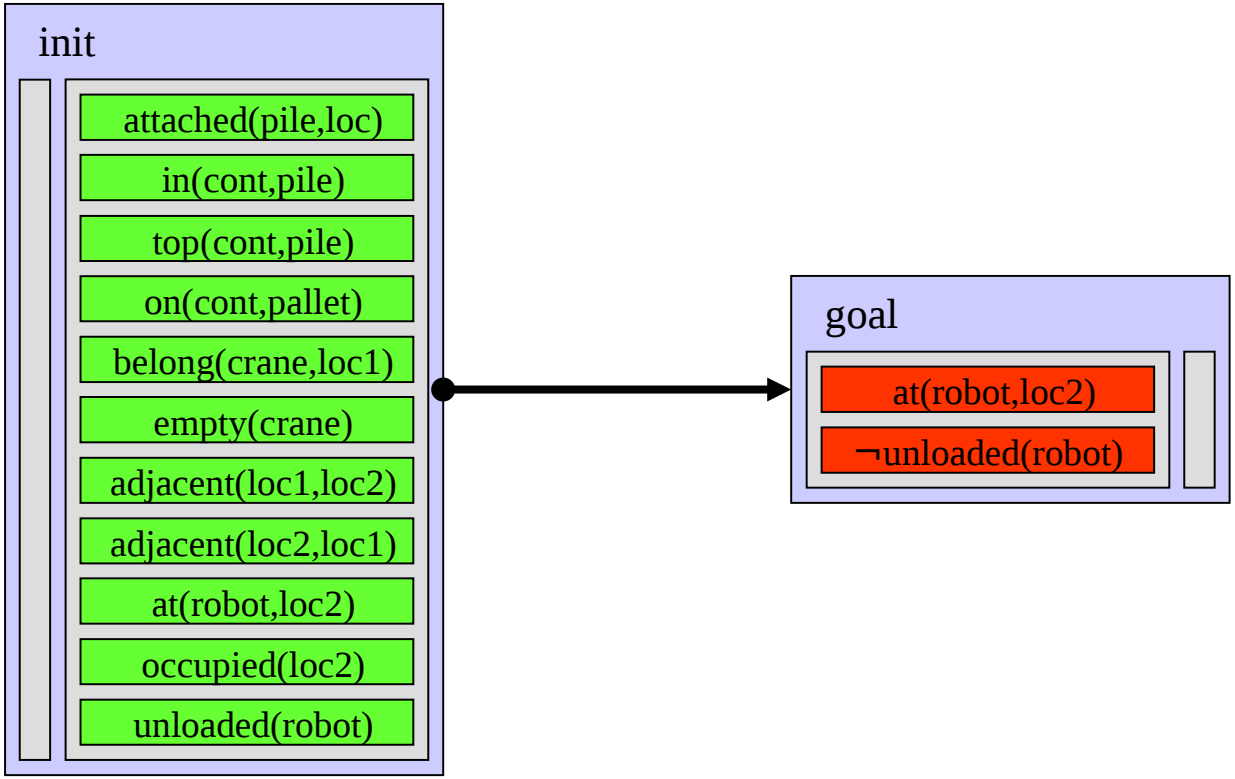
- Ataque: una interacción que elimina condiciones
  - la acción *a* genera la precondition (e.g., p(x)) de una acción *b*
  - otra acción *c* es capaz de eliminar *p*
- Resolviendo el fallo:
  - Imponer una restricción para evitar que *c* elimine *p*
    - Haciendo que *b* preceda a *c*
    - Haciendo que *c* preceda a *a*
    - Restringir variables para prevenir que *c* elimine a *p*



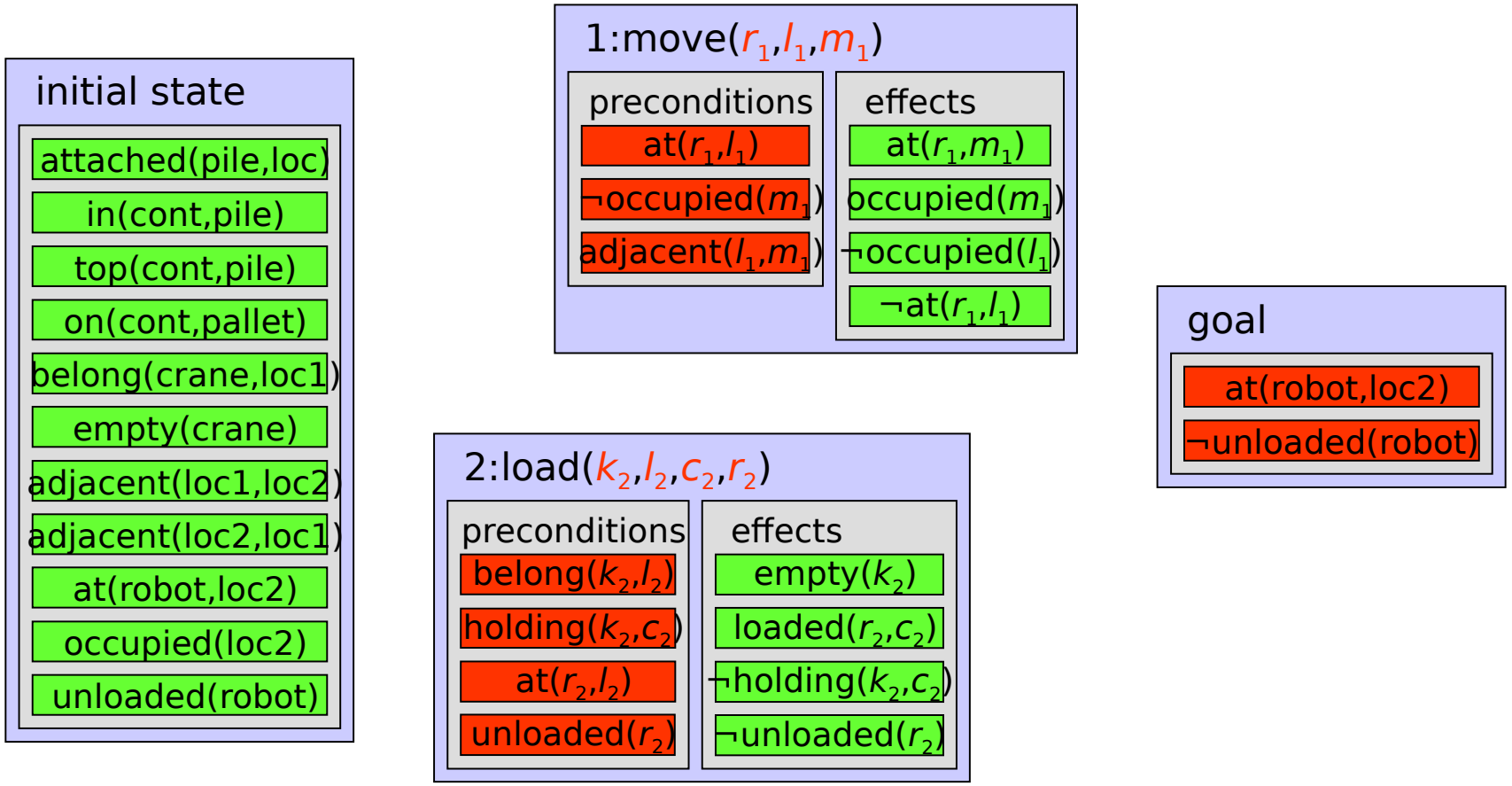
# Plan Space Planner (PSP)

- PSP es completo
- Devuelve un plan ordenado parcialmente
- Principio fundamental:
  - afinar el plan parcial π hasta que π no tenga más fallos
- Operaciones básicas:
  - encontrar los fallos de π, es decir, sus sub-metas y sus amenazas
  - seleccionar uno de los fallos
  - encontrar formas de resolver el fallo elegido
  - elegir uno de los resolutores de la falla
  - refinar π según la resolución elegida

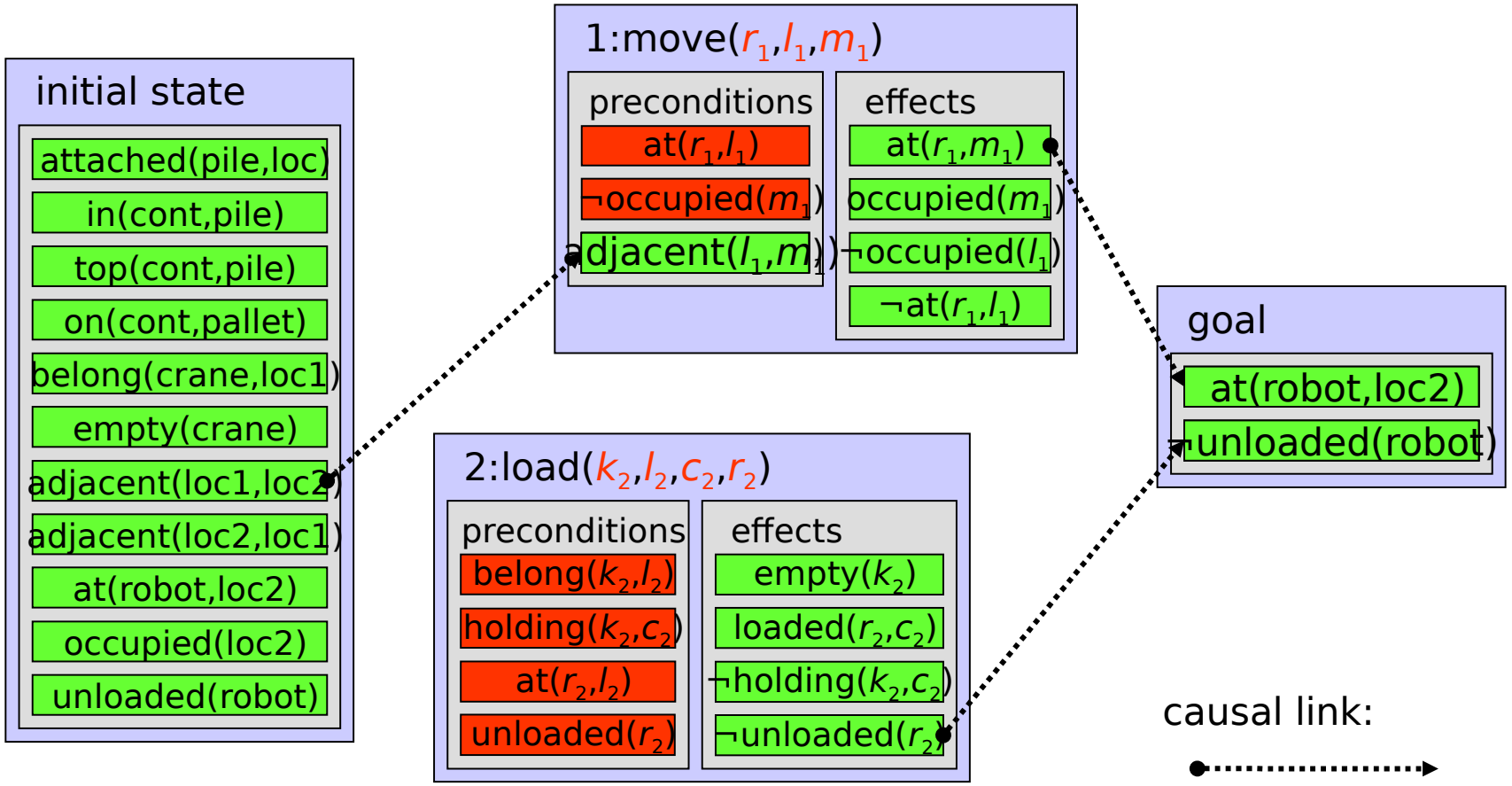
# Plan Space Planner (PSP)



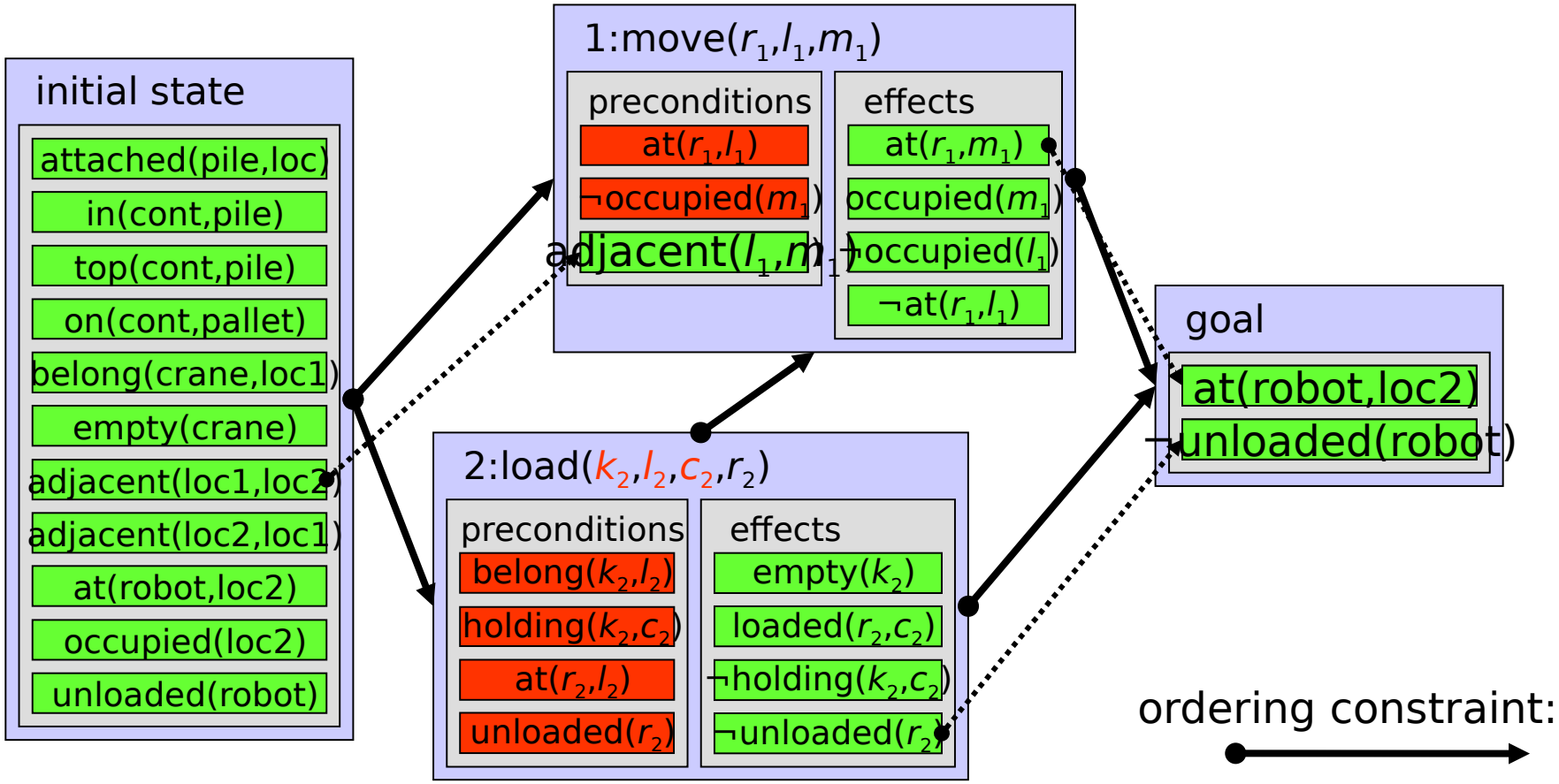
# Plan Space Planner (PSP)



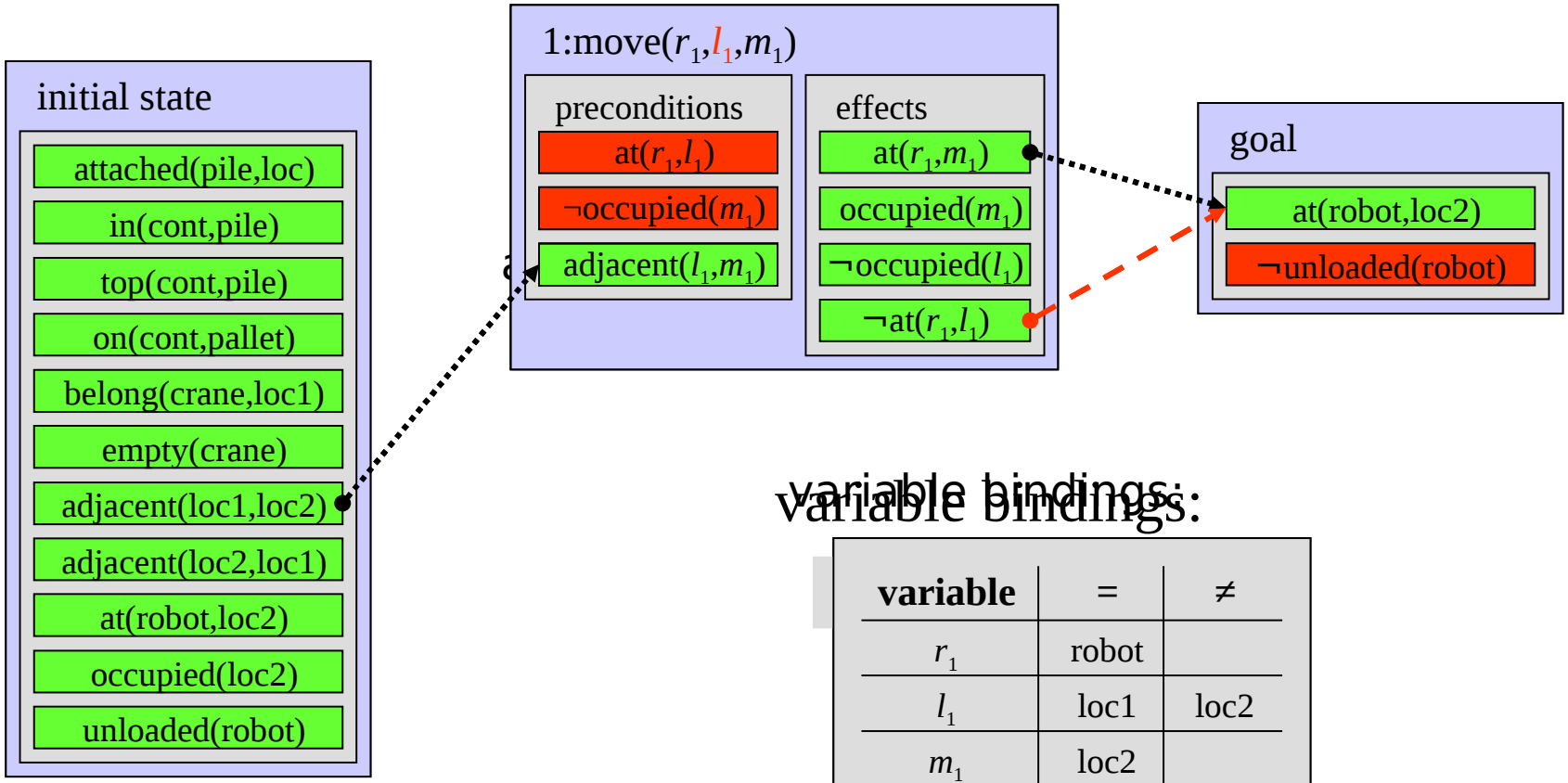
# Plan Space Planner (PSP)



# Plan Space Planner (PSP)



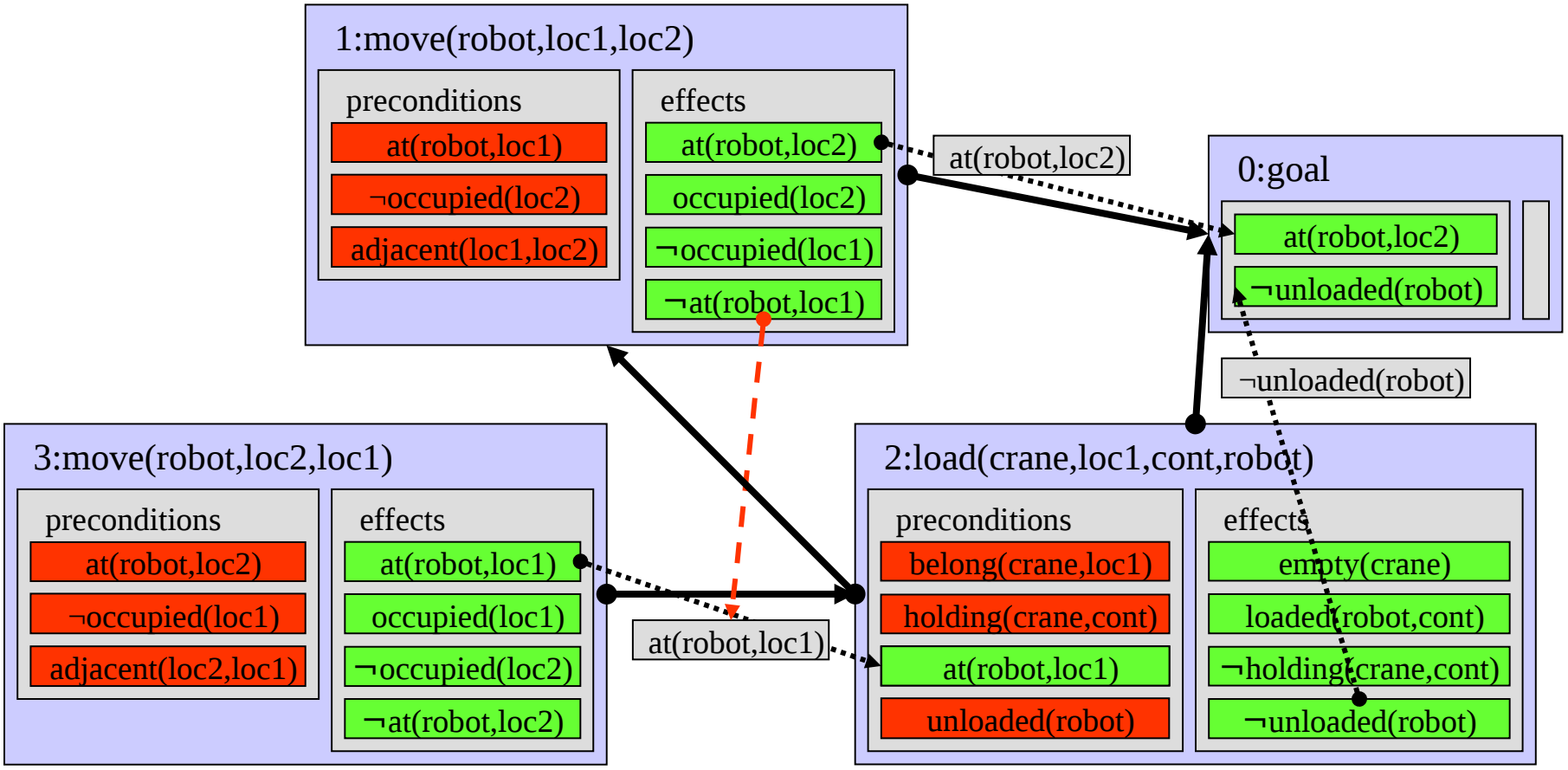
# Plan Space Planner (PSP)




variable bindings:

| variable | = | ≠ |
|---|---|---|
| $r_1$ | robot | |
| $l_1$ | loc1 | loc2 |
| $m_1$ | loc2 | |

# Plan Space Planner (PSP)



# Técnicas Avanzadas de Inteligencia Artificial

Curso 2016-2017

German Rigau y Maite Urretavizcaya

{german.rigau, maite.urretavizcaya}@ehu.eus

Grado en Ingeniería en Informática