Planificación

Planificación

Introducción

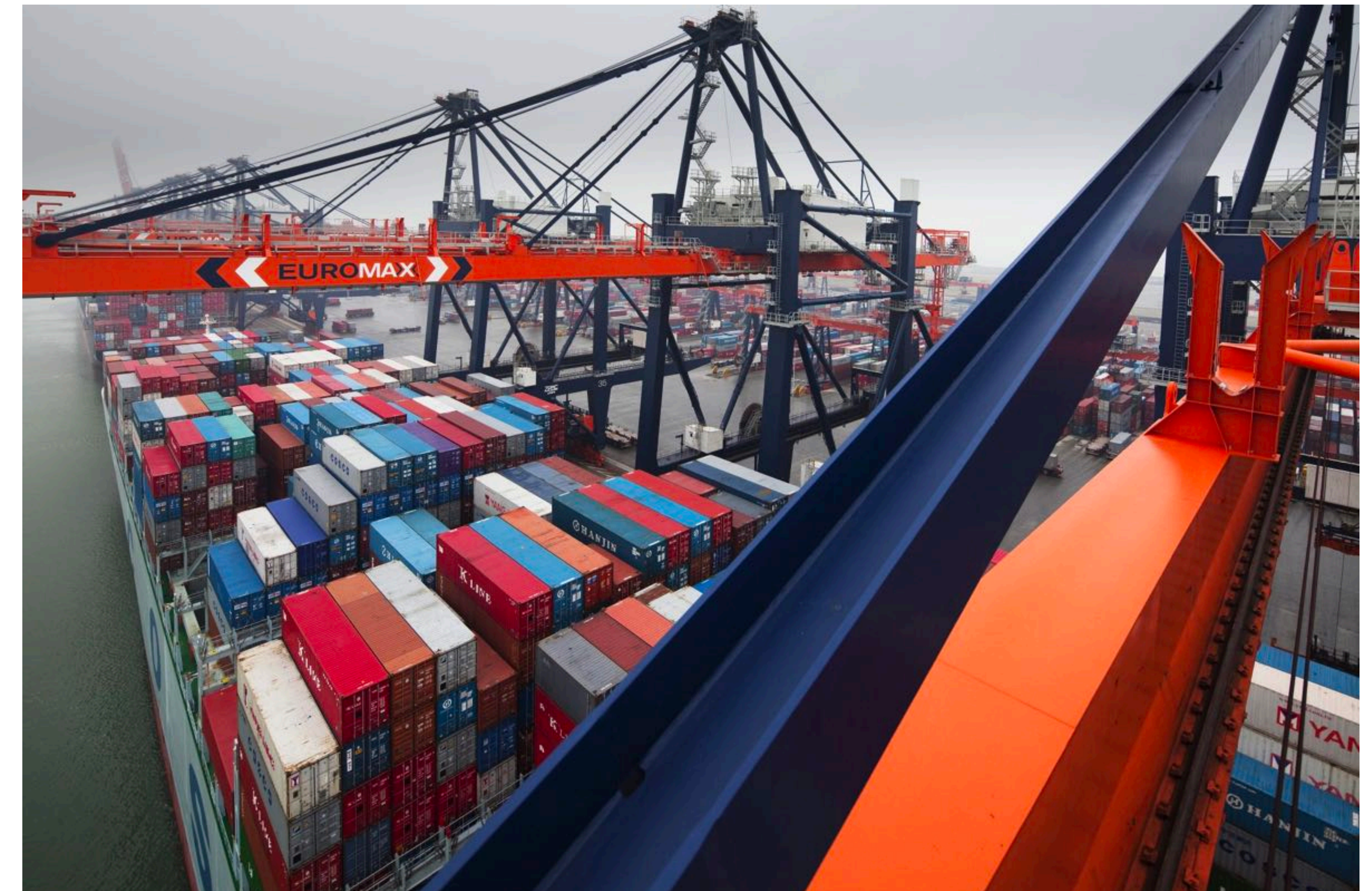
La planificación es el proceso de generar un plan de acción para lograr un objetivo específico en un entorno dado. La planificación es un componente importante en muchos sistemas de inteligencia artificial, incluyendo sistemas robóticos, sistemas de control de procesos industriales y sistemas de asistencia personal.

Planificación

Introducción

Aplicaciones:

- **Robótica** (robots móviles y vehículos autónomos)
- **Simulación** (entrenamiento y juegos)
- **Logística** “Workflows” (fábricas y cadenas de montaje)
- **Gestión de crisis** (evacuaciones, incendios…)



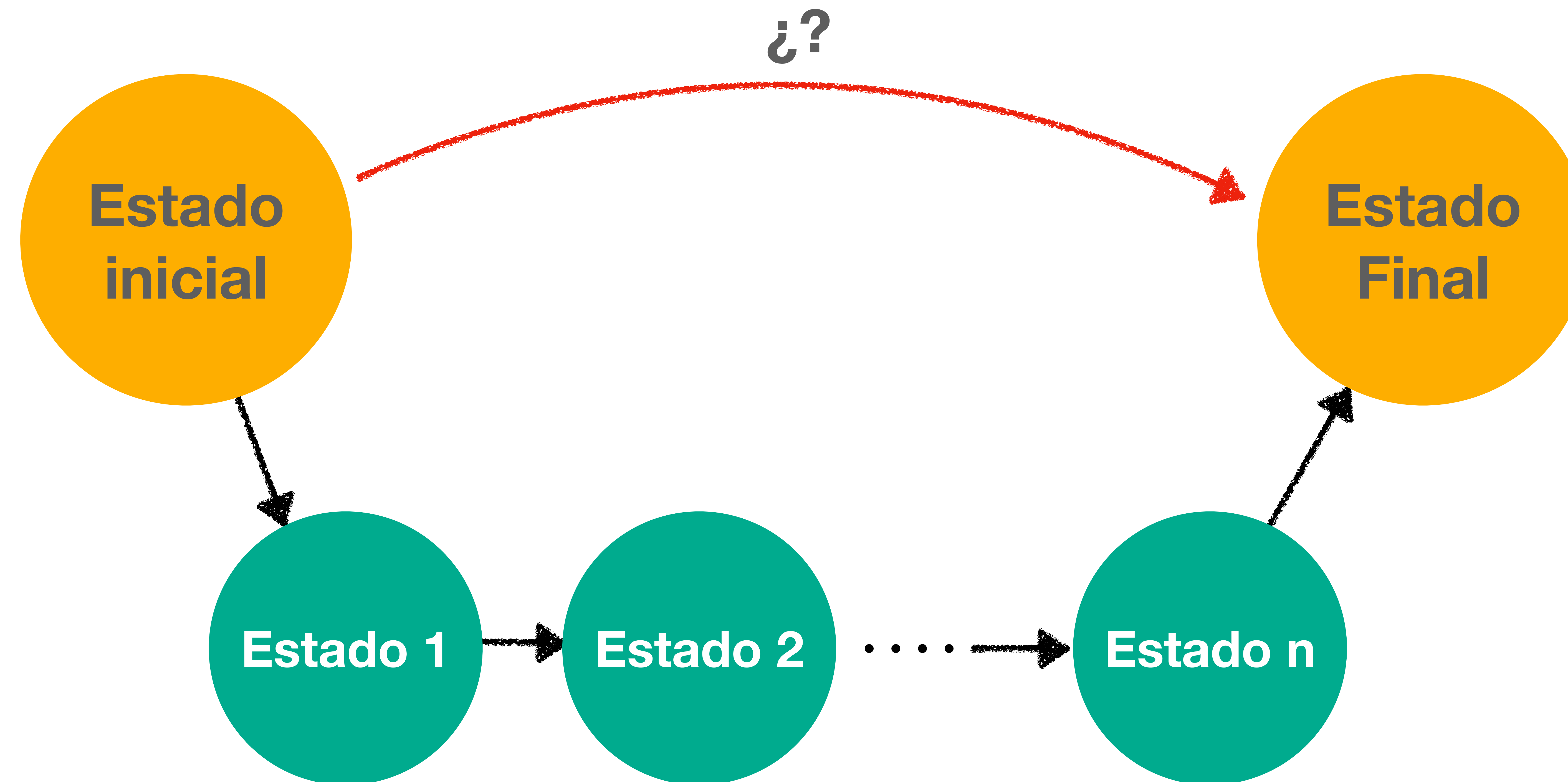
Planificación

Introducción

Problema de planificación:

Dados una descripción del “mundo” (un modelo), un **estado inicial**, una **descripción del objetivo** y un conjunto de **acciones** que pueden cambiar el mundo, se trata de encontrar una **secuencia de acciones** que, partiendo del estado inicial, sea posible llegar a un estado que satisfaga el objetivo.

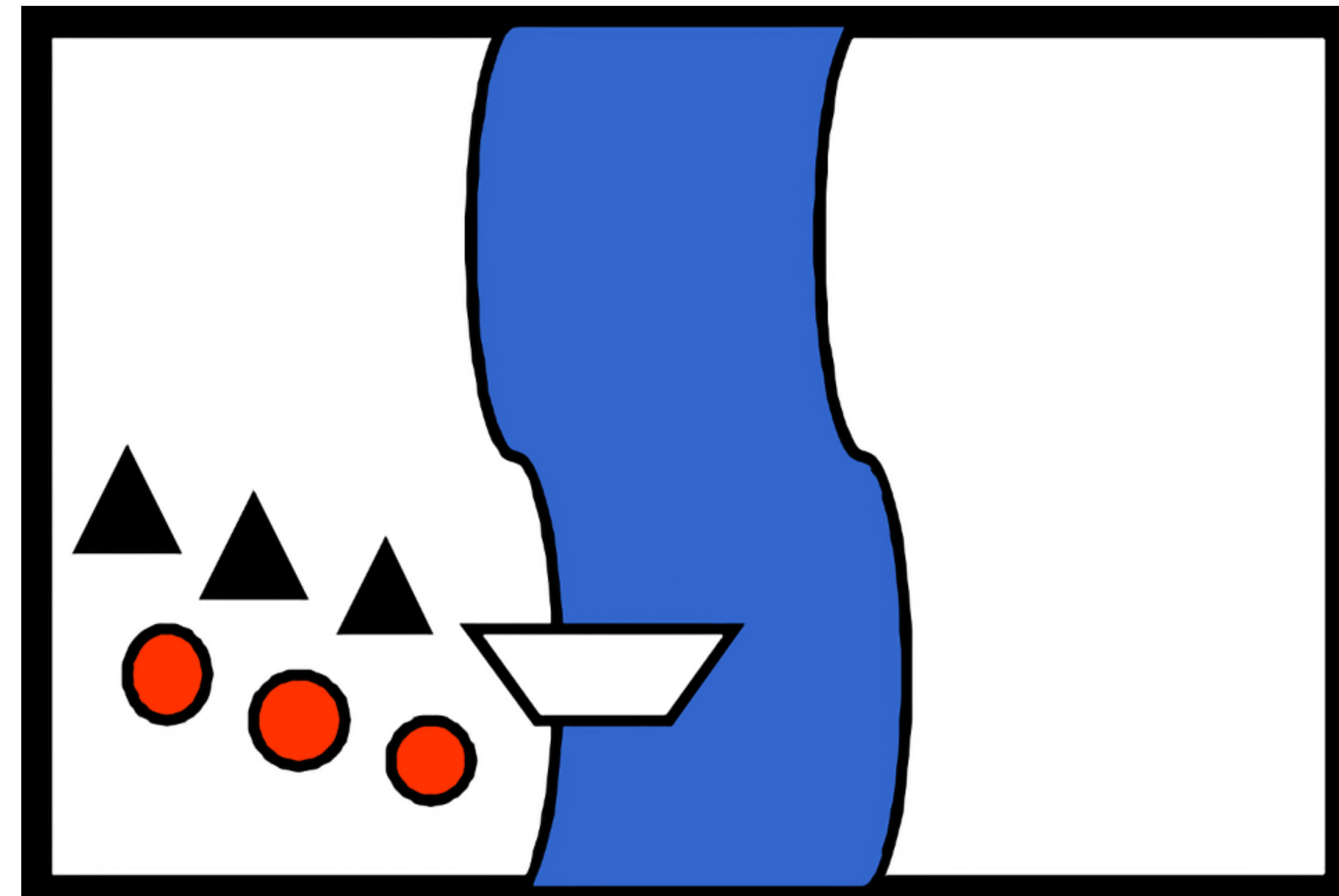
Planificación
Introducción



Planificación

Misioneros y caníbales

Tres misioneros y tres caníbales deben cruzar un río en un bote que sólo puede llevar hasta dos personas a la vez. Además, si en cualquier orilla hay más caníbales que misioneros, los caníbales se comerán a los misioneros.



Planificación

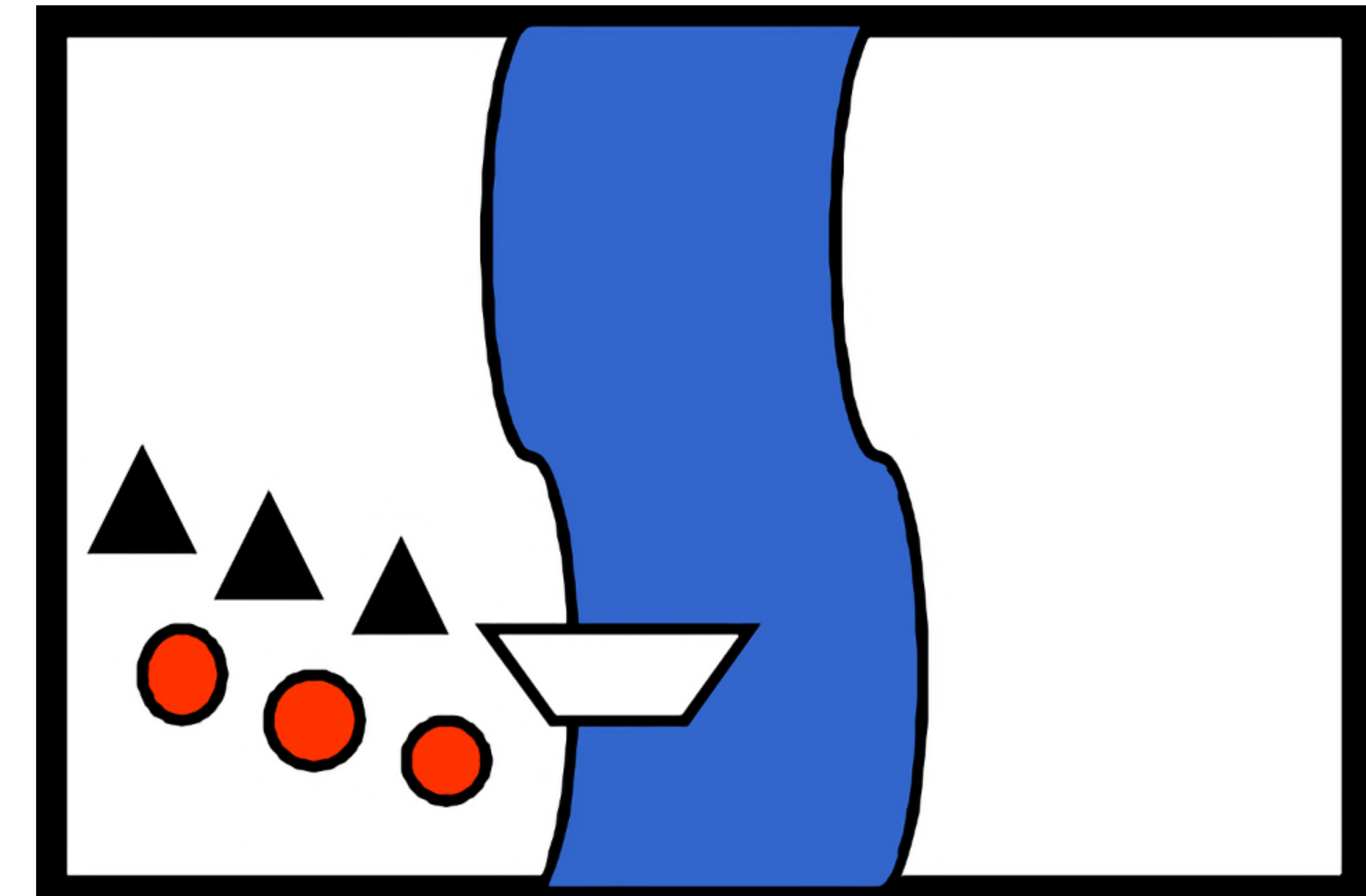
Misioneros y caníbales

Estado inicial: Todos los misioneros y caníbales en la orilla izquierda

Objetivo: Todos los misioneros y caníbales en la orilla derecha

Acciones posibles:

- 1 misionero cruza el río
- 1 caníbal cruza el río
- 2 misioneros cruzan el río
- 2 caníbales cruzan el río
- 1 misionero y 1 caníbal cruzan el río



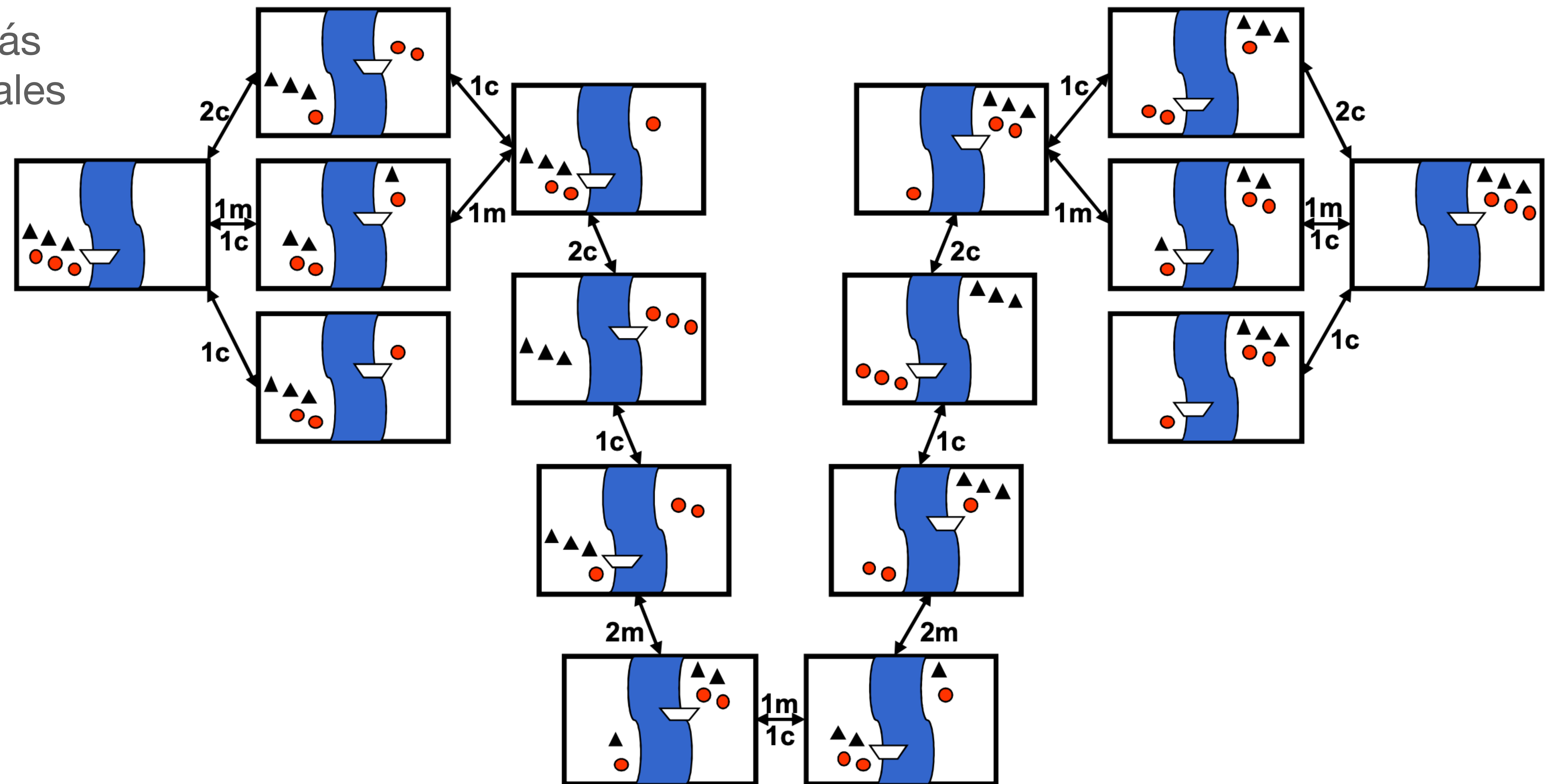
Coste de la solución: +1 por cada vez que el bote cruza el río

Solución óptima: 4 soluciones de coste 11

Planificación

Misioneros y caníbales

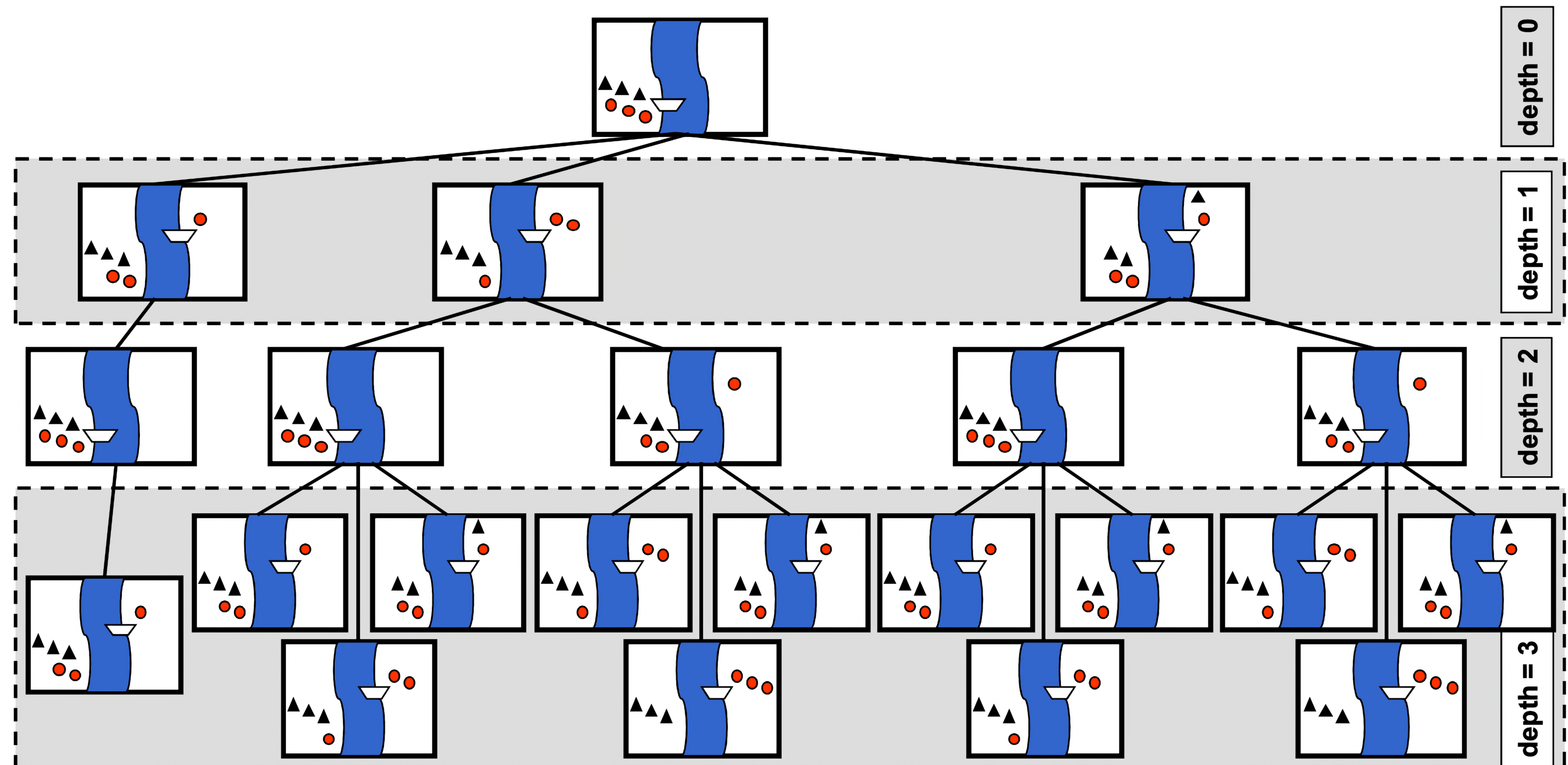
Tres misioneros y tres caníbales deben cruzar un río en un bote que sólo puede llevar hasta dos personas a la vez. Además, si en cualquier orilla hay más caníbales que misioneros, los caníbales se comerán a los misioneros.



Planificación

Misioneros y caníbales

Estrategias de búsqueda (búsqueda en anchura o IDS -Iterative Deepening Search-)



Planificación

Búsqueda

Estrategias de búsqueda (búsqueda en anchura o búsqueda iterativa en profundidad)

La búsqueda en anchura (**BFS -Breadth First Search-**) explora y recorre un grafo o árbol de búsqueda. El algoritmo comienza en un nodo raíz y explora todos los nodos vecinos a ese nodo en el mismo nivel antes de avanzar al siguiente nivel de nodos.

La búsqueda iterativa en profundidad (**IDS -Iterative Deepening Search-**) consiste en realizar una serie de búsquedas en profundidad limitadas, comenzando con un límite de profundidad de uno y aumentando gradualmente hasta encontrar una solución. En cada iteración, el algoritmo realiza una búsqueda en profundidad limitada en el árbol de búsqueda hasta la profundidad límite actual. Si no se encuentra una solución, el límite de profundidad se incrementa en uno y se vuelve a realizar la búsqueda.

```
función BuscarPlan(estadoInicial, meta, acciones):
    cola <- [estadoInicial]  # Inicializamos la cola de estados a explorar con el estado inicial
    visitados <- [estadoInicial]  # Inicializamos la lista de estados ya visitados con el estado inicial
    plan <- []  # Inicializamos el plan como una lista vacía

    mientras cola no esté vacía:
        estadoActual <- cola.pop()  # Sacamos el primer estado de la cola

        si estadoActual cumple la meta:
            # Si hemos alcanzado la meta, generamos el plan de regreso
            mientras estadoActual no sea el estado inicial:
                acción, estadoAnterior <- estadoActual.padre
                plan.append(acción)
                estadoActual <- estadoAnterior
            plan.reverse()  # Invertimos el plan para que esté en orden cronológico
            retornar plan

        # Generamos nuevos estados a partir del estado actual y las reglas
        nuevosEstados <- []
        para acción en acciones:
            si acción.es_aplicable(estadoActual):
                nuevosEstados.append(acción.aplicar(estadoActual))

        # Agregamos los nuevos estados a la cola y la lista de visitados
        para estado en nuevosEstados:
            si estado no está en visitados:
                estado.padre <- (acción, estadoActual)  # Guardamos la acción que llevó a este estado y su estado padre
                visitados.append(estado)
                cola.append(estado)

    retornar "No se encontró plan"  # Si llegamos aquí, no se encontró un plan válido
```

Planificación

Búsqueda hacia adelante (progresión)

Ventajas:

- La búsqueda hacia adelante es correcta (si devuelve un plan, este plan es una solución válida).
- La búsqueda hacia adelante es completa (si existe un plan, la búsqueda lo termina encontrando).

Desventajas:

- Número elevado de acciones aplicables en cada estado.
- Factor de ramificación demasiado grande.
- No resulta resulta viable para planes con muchos pasos.

Una posible alternativa: Buscar hacia atrás, partiendo del objetivo…

Planificación

Búsqueda hacia atrás (regresión)

Este método de razonamiento comienza en el objetivo final y retrocede a través de las reglas y el conocimiento disponible para determinar qué hechos o condiciones son necesarios para alcanzar esa meta. La búsqueda hacia atrás trabaja desde una conclusión para encontrar las premisas necesarias que la sustenten.

```
sub_objetivo ← objetivo final
plan ← []
mientras el estado inicial no satisfaga el sub_objetivo:
  aplicables ← acciones que nos llevan al sub_objetivo
  si aplicables está vacío retornar fallo
  acción ← seleccionar una acción de aplicables
  sub_objetivo ← resultado⁻¹(sub_objetivo, acción)
  plan ← [acción] + plan

retornar plan
```

Planificación

Búsqueda hacia atrás (regresión)

Ventajas:

- La búsqueda hacia atrás es correcta (si devuelve un plan, este plan es una solución válida).
- La búsqueda hacia atrás es completa (si existe un plan, la búsqueda lo termina encontrando).

Desventajas:

- Aunque el factor de ramificación sea, por lo general, menor que en la búsqueda hacia adelante, el espacio de búsqueda sigue siendo demasiado grande.

Planificación

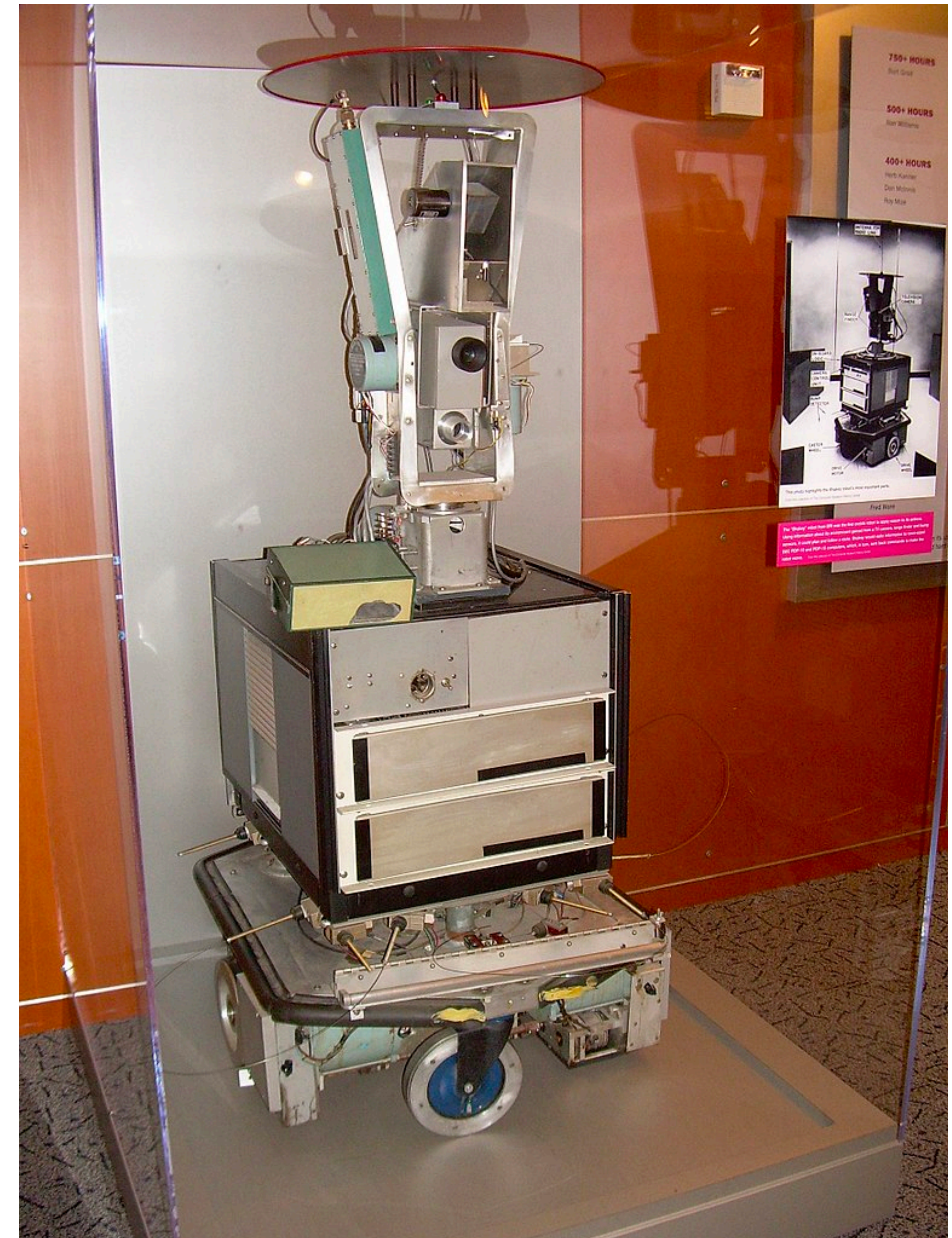
Búsquedas

Una búsqueda genérica no parece la estrategia más adecuada para resolver un problema de planificación ya que no se tienen en cuenta las diferencias entre el estado inicial y el objetivo para elegir las acciones adecuadas. Los planes podrían empezar con acciones obvias y después refinarse. Muchos objetivos son compuestos y se podrían obtener de forma independiente (o, al menos, de forma relativamente independiente).

Planificación

STRIPS

El **Stanford Research Institute Problem Solver (STRIPS)**, es un **planificador automatizado** desarrollado por Richard Fikes y Nils Nilsson en 1971 en SRI International. Posteriormente se utilizó el mismo nombre para referirse al **lenguaje formal** usado como entrada a este **planificador**. Este lenguaje es la base para la mayoría de los lenguajes que expresan instancias de problemas de planificación automatizada y que se usan en la actualidad. Estos lenguajes se conocen comúnmente como **"action languages"**.



Planificación

STRIPS

Representación de un problema de planificación clásico:

- Estado inicial.
- Descripción del objetivo que se desea conseguir.
- Conjunto de objetos, acciones, precondiciones y efectos.

STRIPS básico:

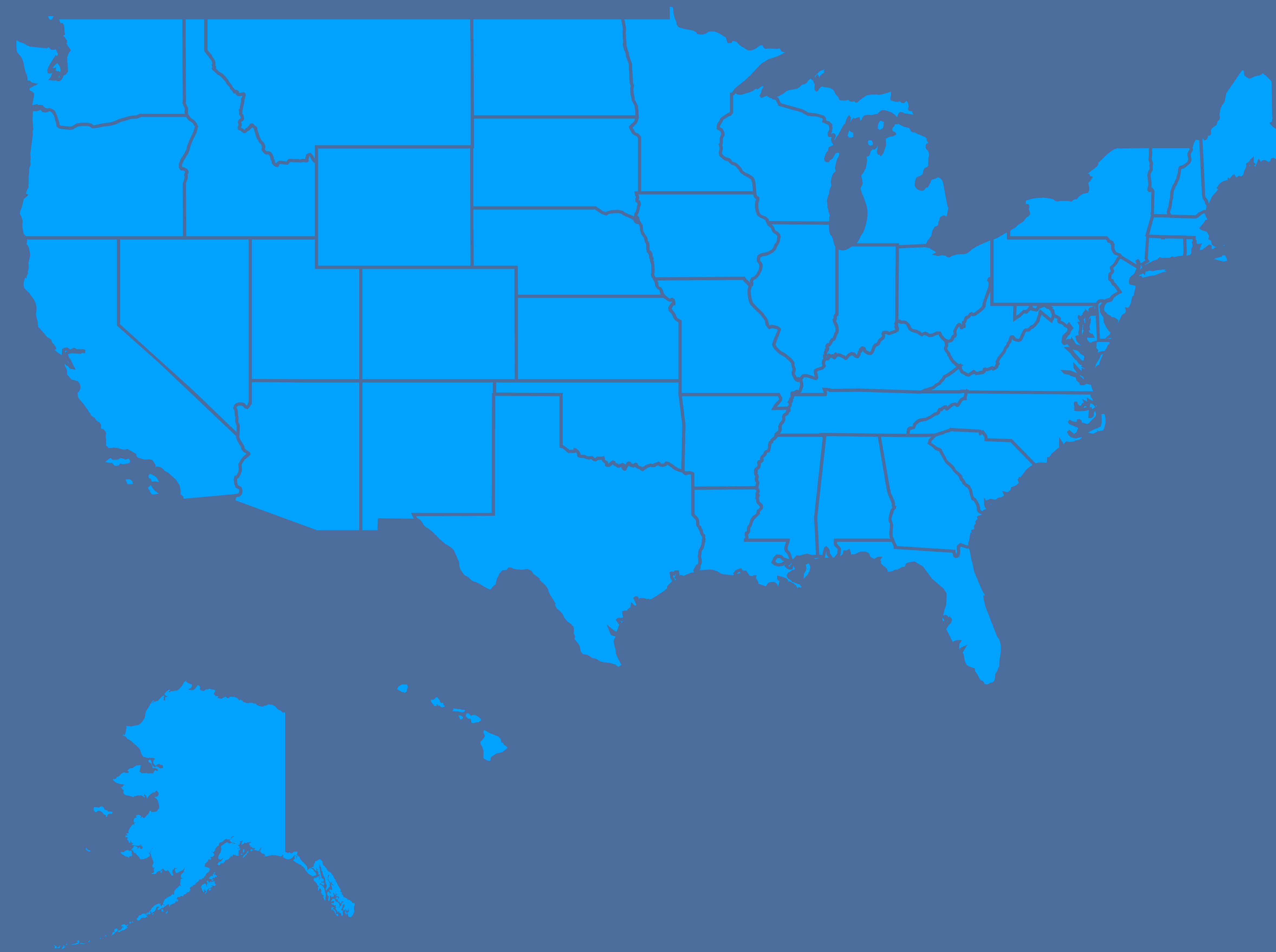
- Uso de predicados (lógica de primer orden).
- Sólo literales positivos.
- Sólo conjunciones de literales simples en el objetivo.
- Sólo se especifica aquello que cambia.
- Hipótesis de mundo cerrado.

Planificación

STRIPS

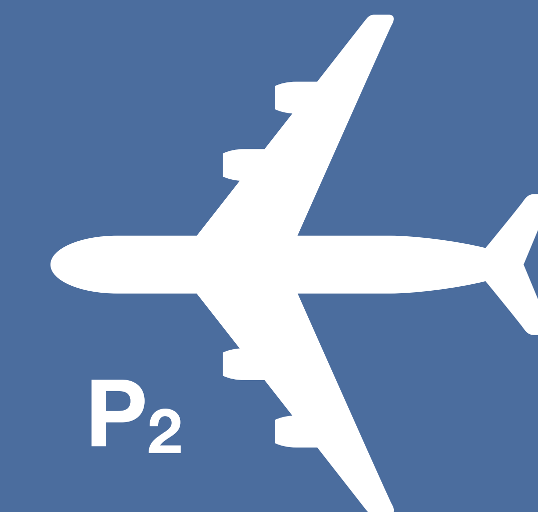
SFO

C_1



JFK

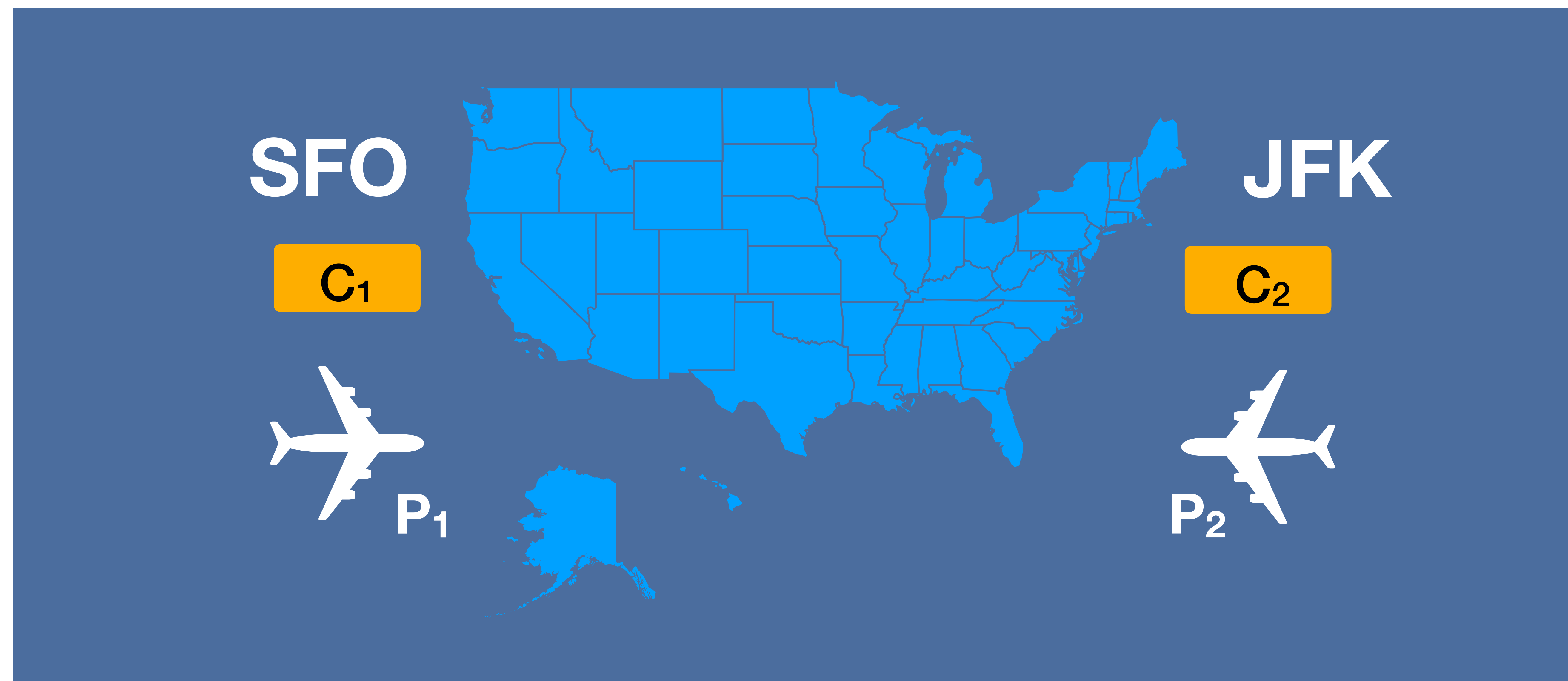
C_2



Planificación

STRIPS

La figura muestra cómo un problema de transporte de carga aéreo lleva asociado procesos de carga y descarga entre aviones que vuelan entre diferentes destinos. El problema puede ser definido con tres acciones: **Carga**, **Descarga**, y **Vuelo**. Las acciones afectan a dos predicados: **Dentro(c, p)** significa que la carga c está dentro del avión p, y **En(x, a)** significa que el objeto x (tanto avión como carga) está en el aeropuerto a. Notemos que la carga no está **En** cualquier sitio cuando se encuentra **Dentro** de un avión concreto, por tanto **En** realmente significa «disponible para su uso en una localización determinada».



Planificación

STRIPS

Iniciar (

Carga(C1) ∧ Carga(C2) ∧ Avión(P1) ∧ Avión(P2) ∧ Aeropuerto(JFK) ∧ Aeropuerto(SFO) ∧

En(C1, SFO) ∧ En(C2, JFK) ∧ En(P1, SFO) ∧ En(P2, JFK) ∧

Objetivo(En(C1, JFK) ∧ En(C2, SFO))

)

Acción (Cargar(c, p, a),

PRECOND: Carga(c) ∧ Avión(p) ∧ Aeropuerto(a) ∧ En(c, a) ∧ En(p, a)

EFECTO: ¬En(c, a) ∧ Dentro(c, p)

Acción (Descargar(c, p, a),

PRECOND: Carga(c) ∧ Avión(p) ∧ Aeropuerto(a) ∧ Dentro (c, p) ∧ En (p, a)

EFECTO: En(c, a) ∧ ¬Dentro(c, p))

Acción (Volar(p, desde, hasta),

PRECOND: Aeropuerto(desde) ∧ Aeropuerto(hasta) ∧ En(p, desde) ∧ Avión(p)

EFECTO: ¬En(p, desde) ∧ En(p, hasta))

Planificación

STRIPS

El siguiente plan es una solución al problema:

[
    Carga(C1, P1, SFO),
    Volar(P1, SFO, JFK),
    Carga(C2 , P2, JFK),
    Volar(P2, JFK, SFO)
]

También es una solución el siguiente plan:

[
    Carga(C1, P1, SFO),
    Carga(C2 , P2, JFK),
    Volar(P1, SFO, JFK),
    Volar(P2, JFK, SFO)
]

¿Qué diferencia hay entre ellos?

Planificación

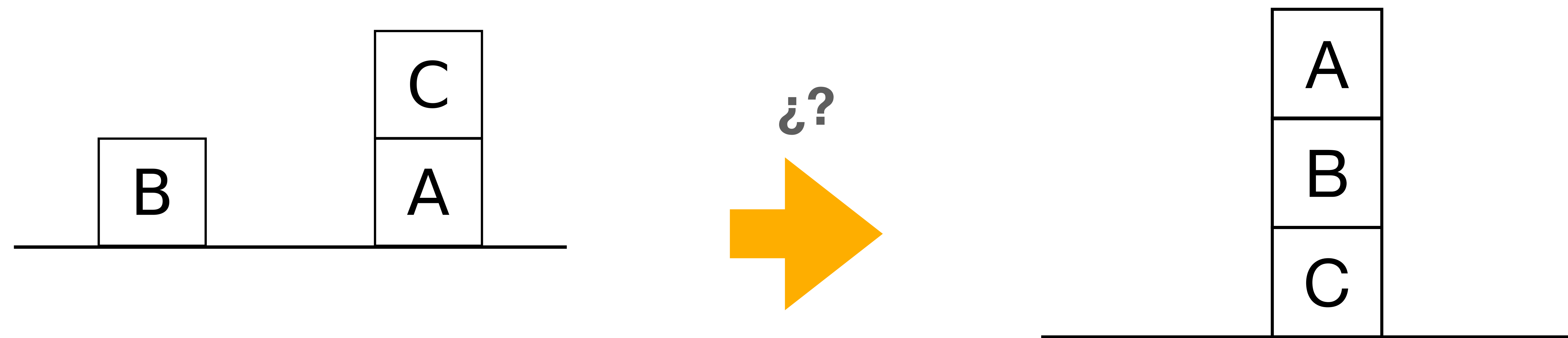
La anomalía de Sussman

La anomalía de **Sussman** es un problema de inteligencia artificial, descrito por primera vez por Gerald Sussman, que ilustra una debilidad de los algoritmos de **planificación no intercalados**, que fueron prominentes a principios de la década de 1970. La mayoría de los sistemas de planificación modernos no están restringidos a la **planificación no intercalada** y, por lo tanto, pueden manejar esta anomalía. Si bien el significado/valor del problema ahora es histórico, sigue siendo útil para explicar por qué la planificación no es trivial.

Planificación

La anomalía de Sussman: ejemplo

En el problema, tres bloques (etiquetados como A, B y C) descansan sobre una mesa. El agente debe apilar los bloques de manera que A esté encima de B, que a su vez está encima de C. Sin embargo, solo puede mover un bloque a la vez. El problema comienza con B sobre la mesa, C encima de A y A sobre la mesa:



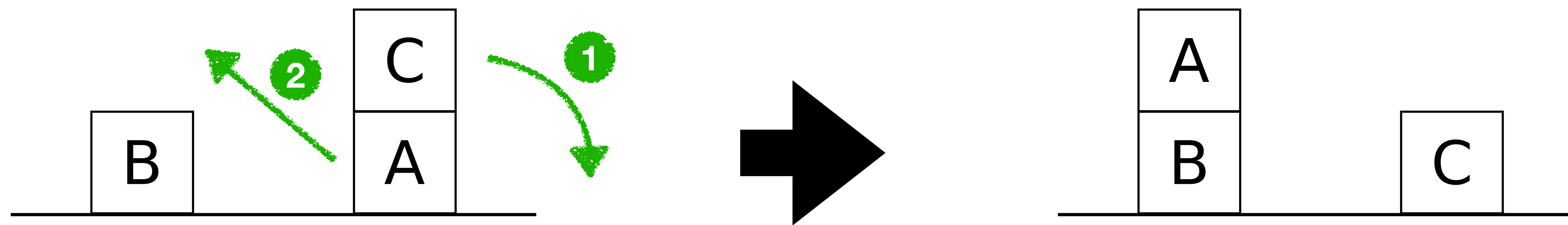
Sin embargo, los **planificadores no intercalados** normalmente separan el objetivo en subobjetivos, como:

- Poner A encima de B (subobjetivo 1)
- Poner B encima de C (subobjetivo 2)

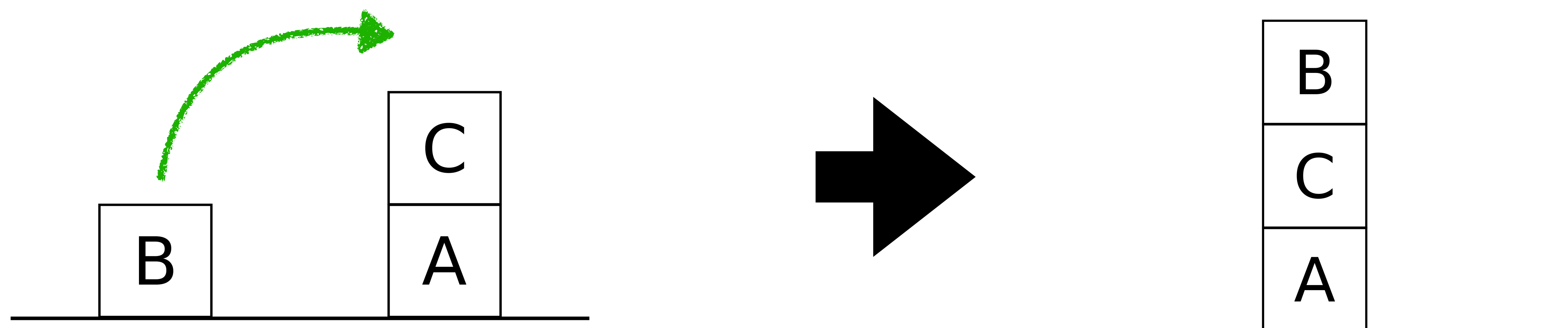
Planificación

La anomalía de Sussman: ejemplo

Supongamos que el planificador comienza persiguiendo el subjetivo 1. La solución directa es mover C fuera del camino y luego poner A encima de B. Pero mientras esta secuencia logra el subobjetivo 1, el agente ahora no puede realizar el segundo subobjetivo sin deshacer el primero.



Si, en cambio, el planificador comienza con el subobjetivo 2, la solución más eficiente es mover B sobre C. Pero nuevamente, el planificador no puede lograr el subobjetivo 1 sin deshacer el segundo.



Planificación

La anomalía de Sussman: ejemplo

Por tanto, la planificación es un proceso no tal sencillo como puede parecer a primera vista. La solución de un problema no siempre se puede resolver mediante la concatenación de sus subproblemas.

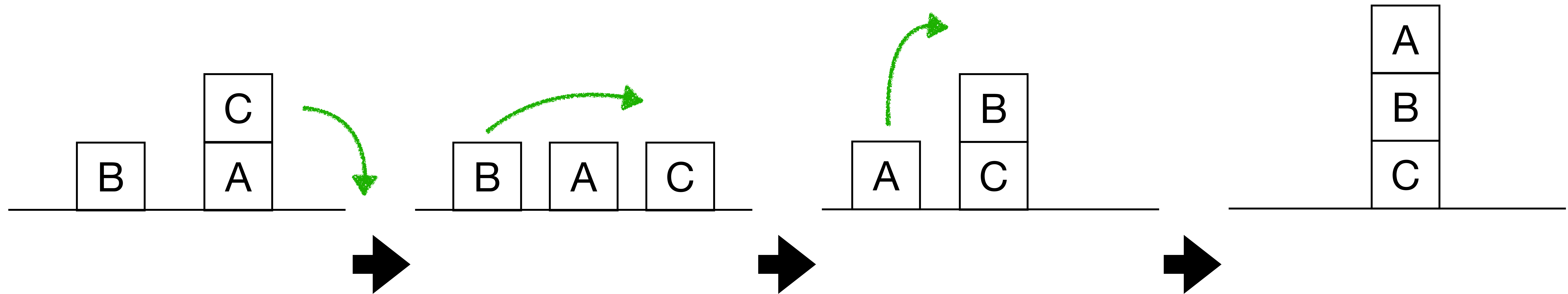
$$\text{solución} \neq \text{subobjetivo 1} + \text{subobjetivo 2}$$

La Anomalía de Sussman es un fenómeno que puede ocurrir en sistemas complejos de planificación de tareas, en el que un cambio en una tarea aparentemente trivial puede tener un impacto dramático en el tiempo y el orden de ejecución de otras tareas en el sistema debido a interdependencias complejas entre ellas. Esto destaca la importancia de tener en cuenta todas las interacciones y dependencias entre las diferentes tareas en la planificación de sistemas complejos.

Planificación

La anomalía de Sussman: ejemplo

Si no consideráramos la solución del problema como una simple concatenación de subjetivos, sino que entrelazáramos los pasos de ambos subobjetivos, podríamos lograr una solución sencilla:



Planificadores

PDDL

Planificación

PDDL

El estándar **PDDL** (Planning Domain Definition Language) es un lenguaje de modelado utilizado en la planificación automatizada de tareas. PDDL define la sintaxis y la semántica para especificar los elementos esenciales de un dominio de planificación, incluyendo los estados del mundo, las acciones posibles y los objetivos a alcanzar.

El lenguaje PDDL es utilizado por muchos planificadores automáticos y ha sido diseñado para ser independiente del problema y del dominio, lo que significa que los mismos planificadores pueden ser utilizados para resolver diferentes problemas en diferentes dominios.

Planificación

Planificadores

http://planning.domains/ - http://editor.planning.domains/

https://www.fast-downward.org/

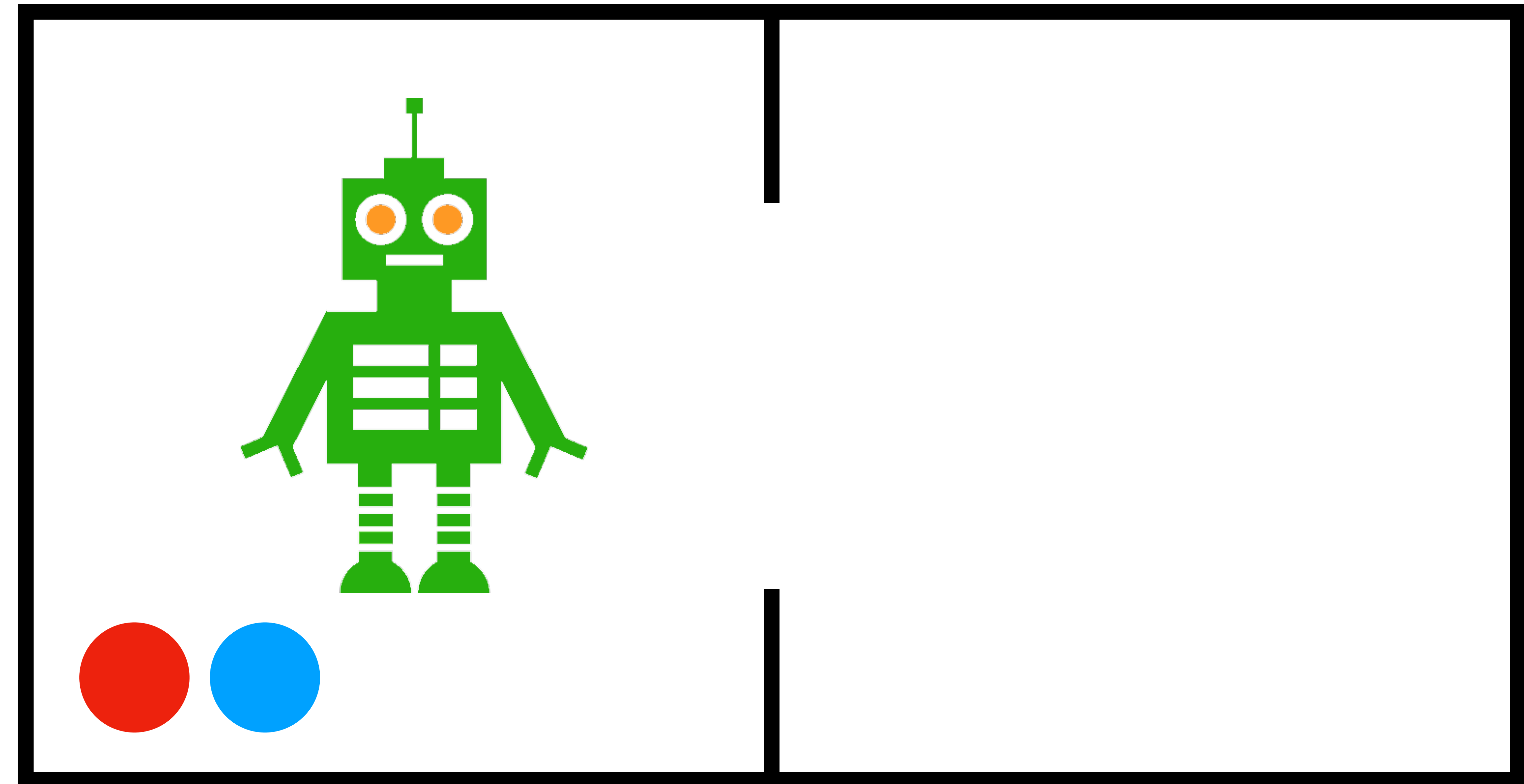
Referencia PDDL

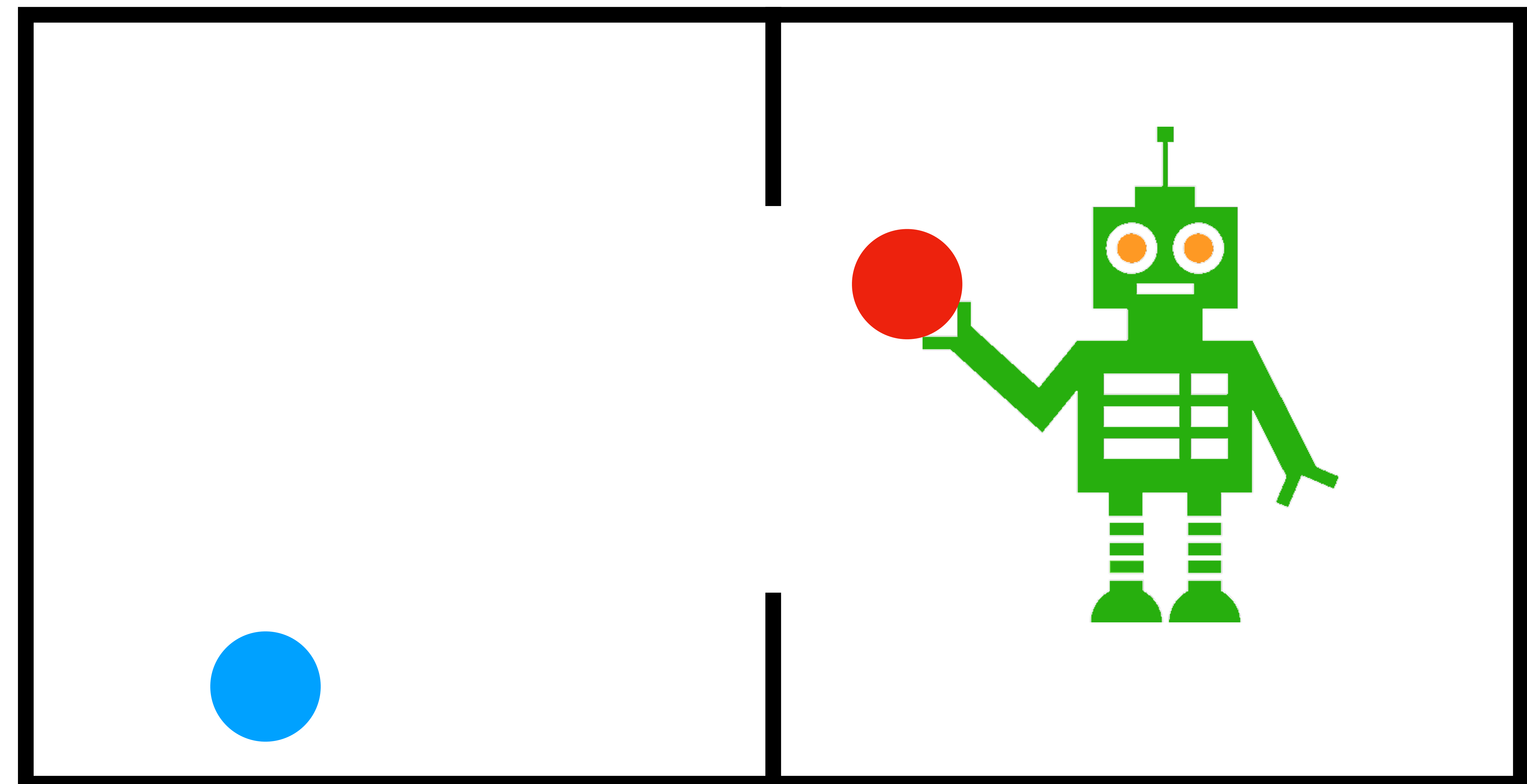
planning.wiki https://planning.wiki/ref/pddl#pddl-12

Planificación

PDDL: ejemplo

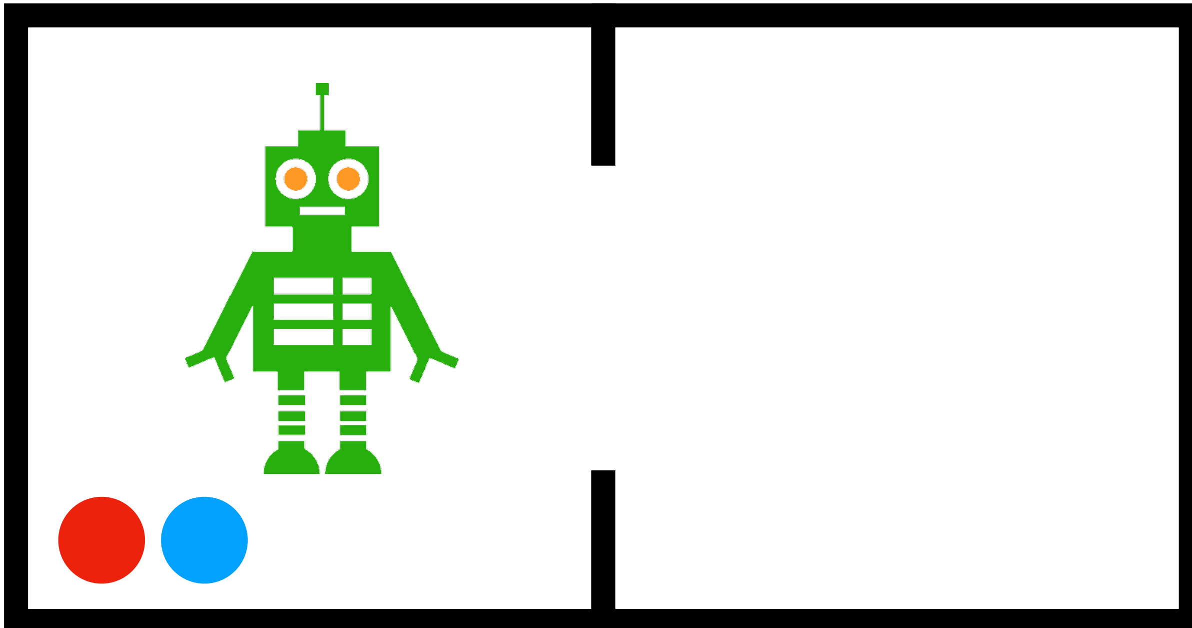
Supongamos que nuestro problema consiste en que un robot se encuentra en una habitación con dos bolas (1 y 2). Éste tiene que trazar un plan para coger una de las bolas y llevarla a la segunda habitación.





Planificación

PDDL: ejemplo

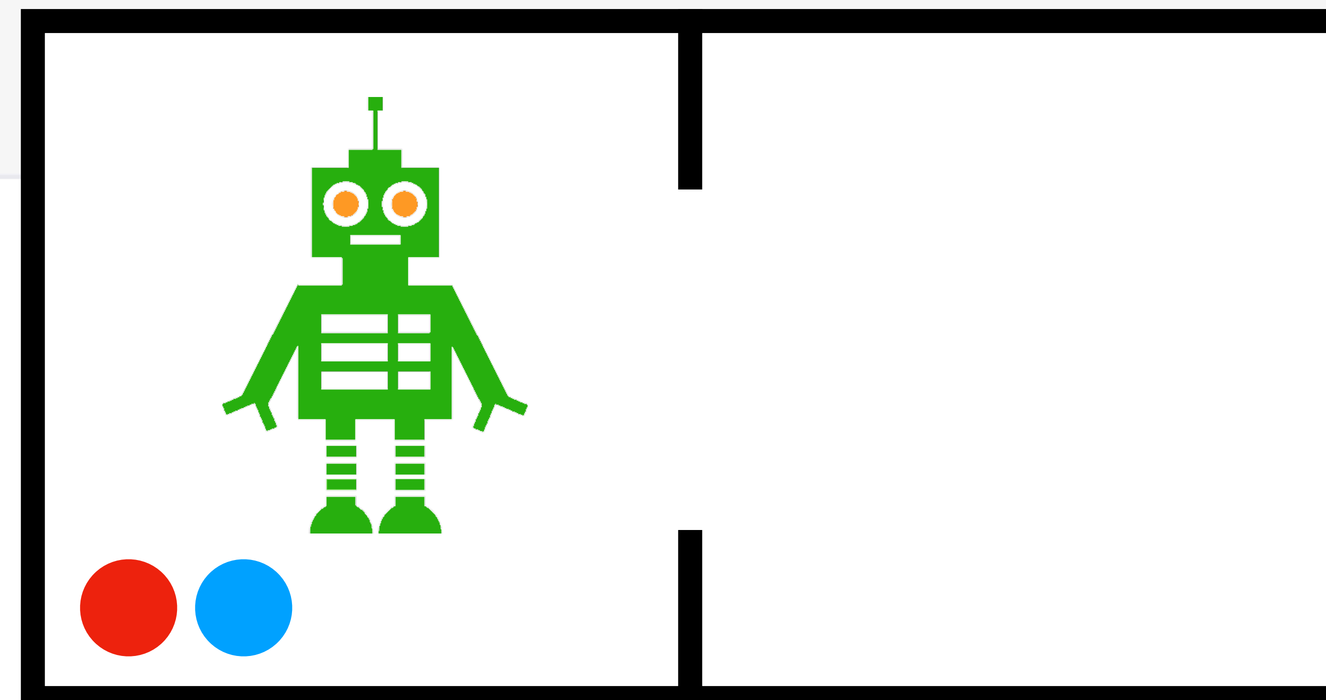


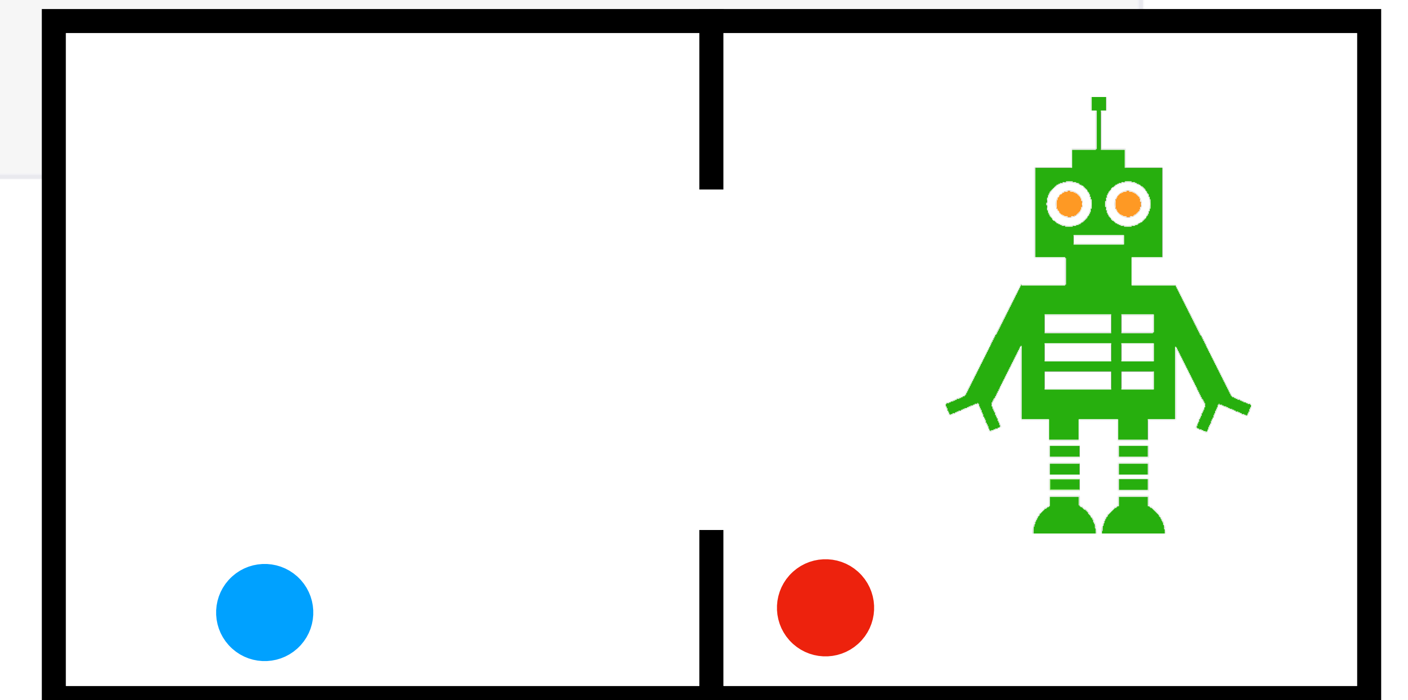
```
(define (domain gripper-strips)
  (:predicates (room ?r) (ball ?b) (gripper ?g) (at-robby ?r)
               (at ?b ?r) (free ?g) (carry ?o ?g))
  (:action move
   :parameters (?from ?to)
   :precondition (and (room ?from)
                      (room ?to)
                      (at-robby ?from))
   :effect (and (at-robby ?to)
                (not (at-robby ?from))))
  (:action pick
   :parameters (?obj ?room ?gripper)
   :precondition (and (ball ?obj)
                      (room ?room)
                      (gripper ?gripper)
                      (at ?obj ?room)
                      (at-robby ?room)
                      (free ?gripper))
   :effect (and (carry ?obj ?gripper)
                (not (at ?obj ?room))
                (not (free ?gripper))))
  (:action drop
   :parameters (?obj ?room ?gripper)
   :precondition (and (ball ?obj)
                      (room ?room)
                      (gripper ?gripper)
                      (carry ?obj ?gripper)
                      (at-robby ?room))
   :effect (and (at ?obj ?room)
                (free ?gripper)
                (not (carry ?obj ?gripper)))))
```

Planificación

PDDL: ejemplo

```
(define (problem strips-gripper2)
    (:domain gripper-strips)
    (:objects rooma roomb ball1 ball2 left right)
    (:init (room rooma)
           (room roomb)
           (ball ball1)
           (ball ball2)
           (gripper left)
           (gripper right)
           (at-robby rooma)
           (free left)
           (free right)
           (at ball1 rooma)
           (at ball2 rooma))
    (:goal (at ball1 roomb)))
```





Planificación

PDDL: ejemplo

Planificación

PDDL: ejemplo

PDDL Editor | File | Session | Import | Solve | Plugins | Help | planning.domains

domain.pddl
problem.pddl
Plan (1)

```
1  (define (problem strips-gripper2)
2      (:domain gripper-strips)
3      (:objects rooma roomb ball1 ball2 left right)
4      (:init (room rooma)
5             (room roomb)
6             (ball ball1)
7             (ball ball2)
8             (gripper left)
9             (gripper right)
10            (at-robby rooma)
11            (free left)
12            (free right)
13            (at ball1 rooma)
14            (at ball2 rooma))
15     (:goal (at ball1 roomb)))
```

Planificación

PDDL: ejemplo

PDDL Editor File Session Import Solve Plugins Help planning.domains

domain.pddl

problem.pddl

Plan (1)

Found Plan (output)

(pick ball1 rooma right)

(move rooma roomb)

(drop ball1 roomb right)

```
(:action pick
  :parameters (ball1 rooma right)
  :precondition
    (and
      (ball ball1)
      (room rooma)
      (gripper right)
      (at ball1 rooma)
      (at-robby rooma)
      (free right)
    )
  :effect
    (and
      (carry ball1 right)
      (not
        (at ball1 rooma)
      )
      (not
        (free right)
      )
    )
)
```

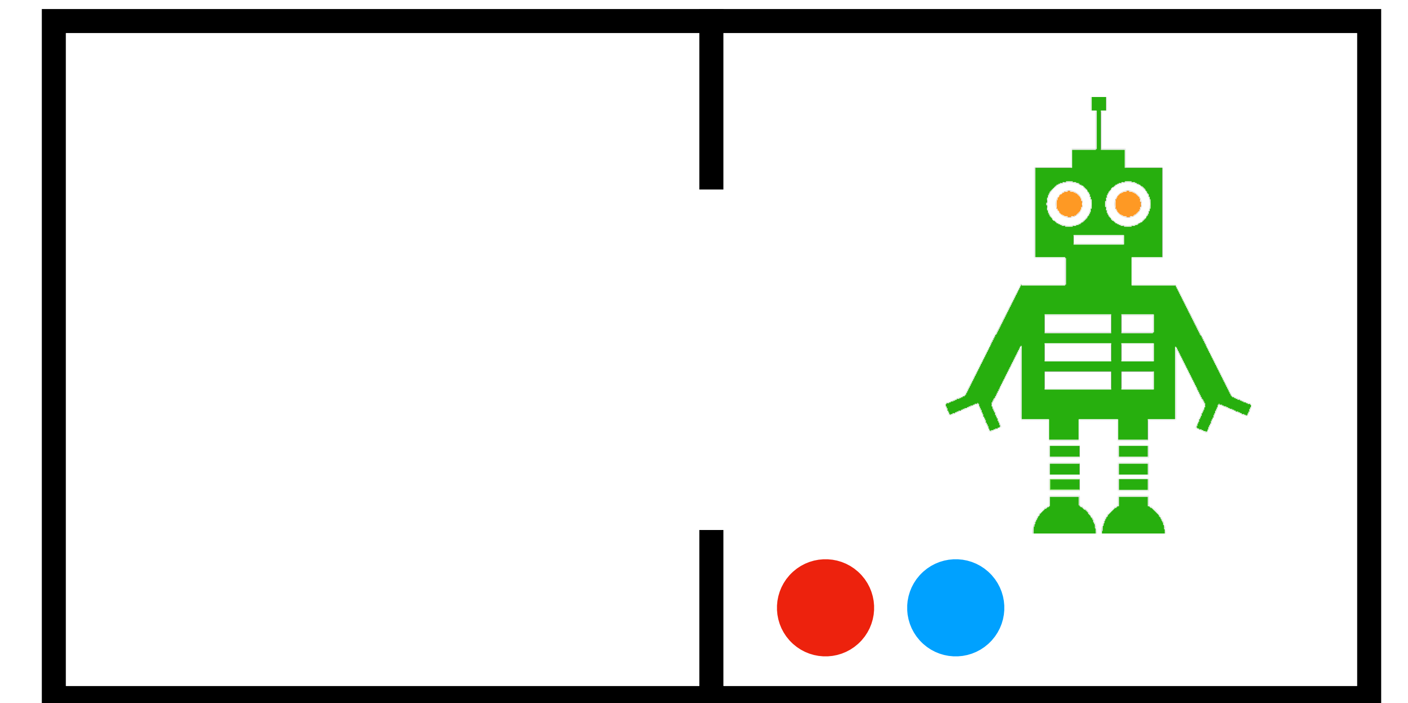
Planificación

PDDL: ejercicio 1

Modifica el ejemplo anterior para que el robot lleve las dos pelotas a la segunda habitación.

Tip: considera el conector “and” en la definición del objetivo.

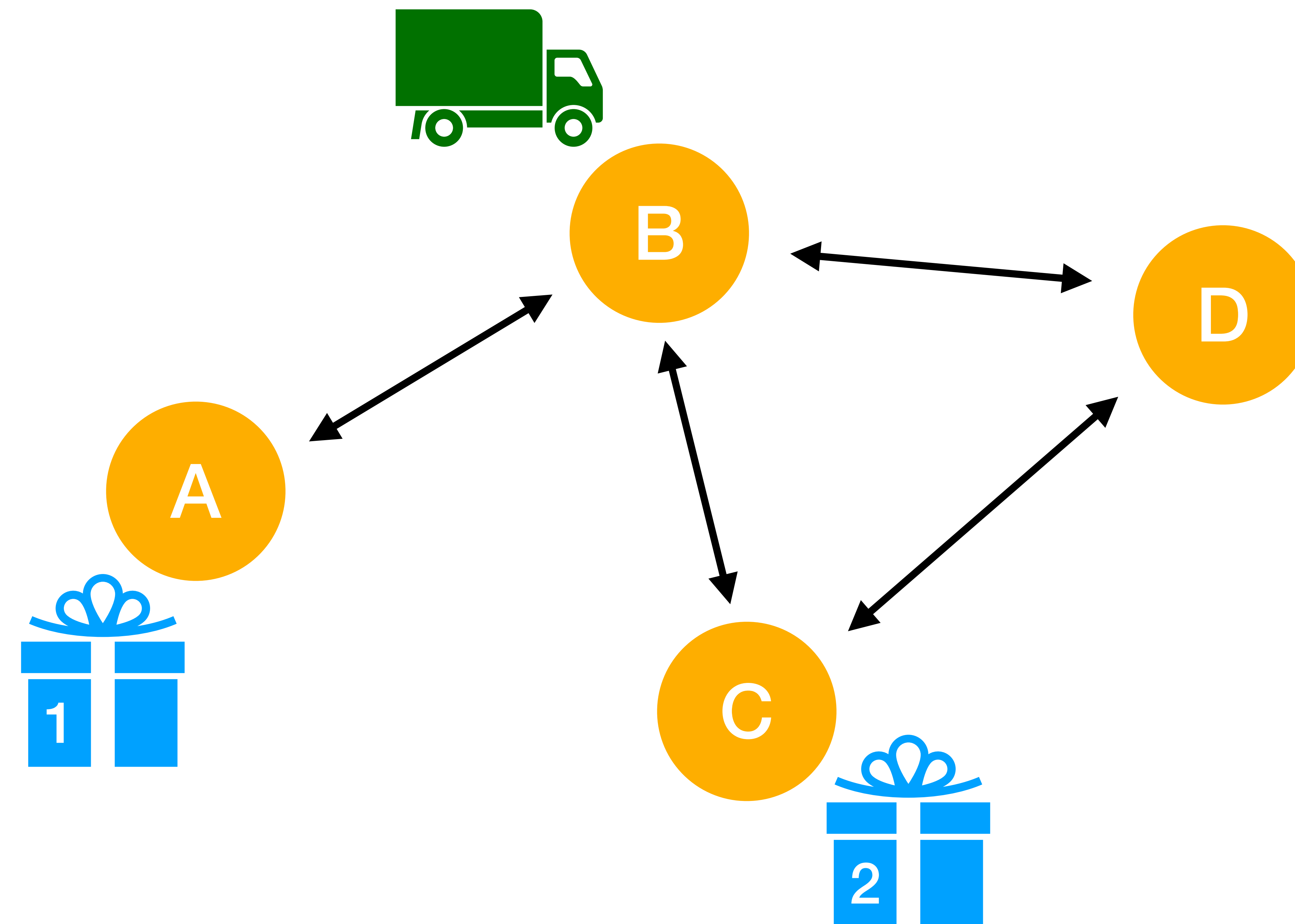
Una vez obtenido el plan, ¿ha cogido una pelota en cada pinza o las ha llevado una a una?



Planificación

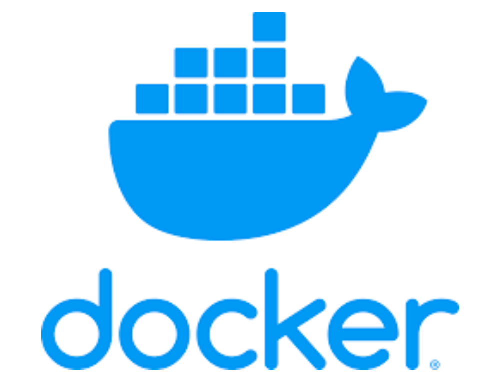
PDDL: ejercicio 2

Un camión debe recoger, transportar y dejar dos paquetes. El paquete 1 está situado en A y paquete 2 en C. El camión parte desde B y tiene que dejar el paquete 1 en C y el paquete 2 en D. Luego tiene que volver a B. Obtén un plan para esta tarea.



Planificación

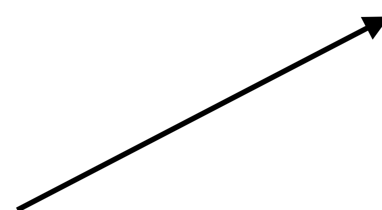
PDDL: Fast Downward

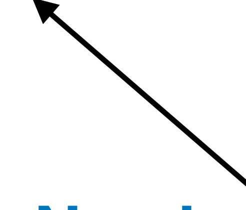


Para usar este planificador lo más sencillo es descargar la imagen docker.

Para invocarlo haremos:

```
docker run --rm -v /[mi_path]/PDDL:/PDDL aibasel/downward --alias lama-first /PDDL/Logistic/problem.pddl
```





Subdirectorio donde tenemos los ficheros “domain” y “problem”

Subdirectorio donde el planificador busca dentro del docker

Nombre de la imagen

Ruta hasta el fichero que queramos procesar