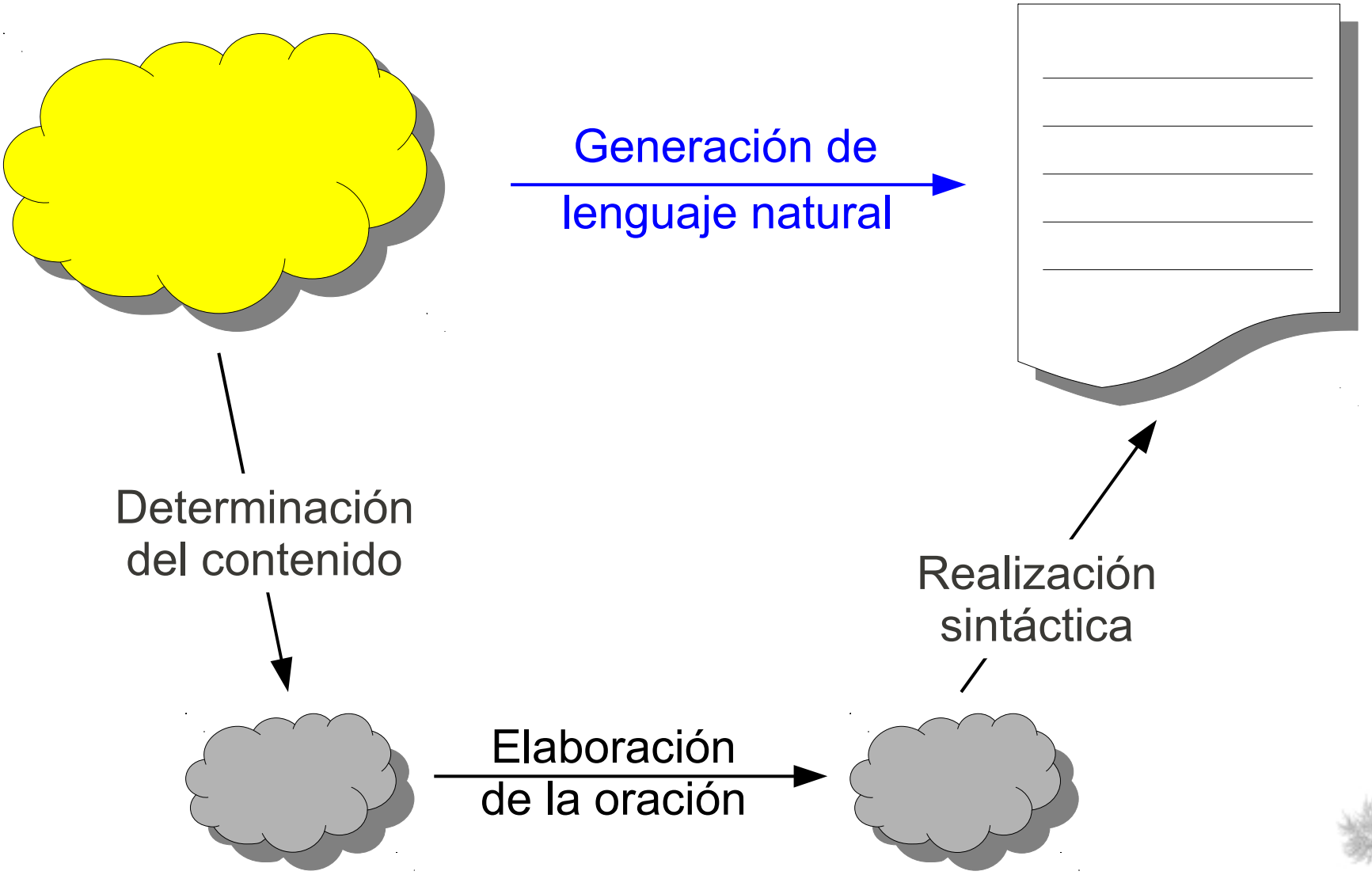
Generación de lenguaje natural

PARTE 2: Generación de instrucciones en un entorno virtual

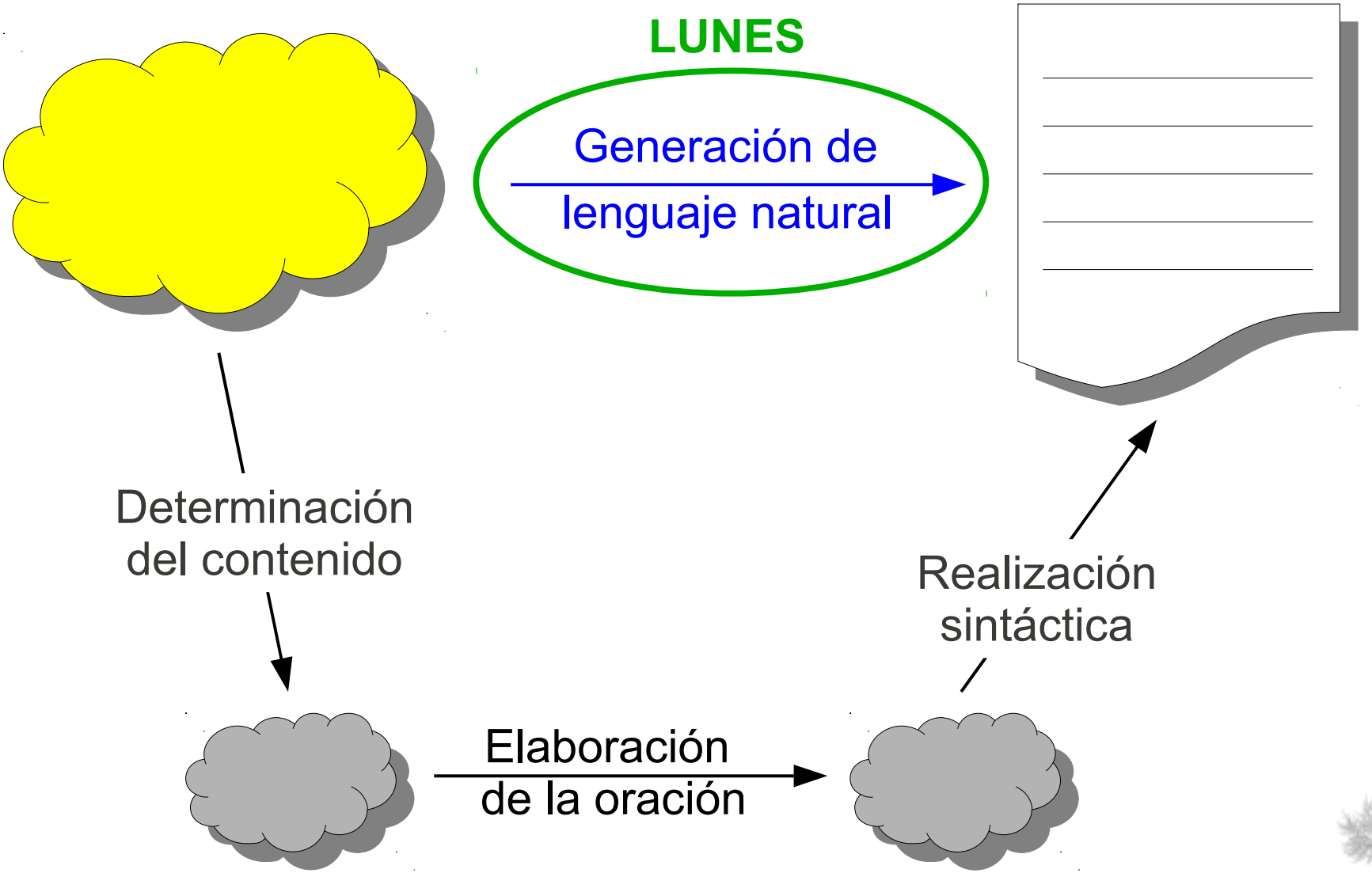
JUEVES

Luciana Benotti & Carlos Areces
luciana.benotti@gmail.com
Equipo PLN – Universidad Nacional de Cordoba
INRIA Nancy Grand Est

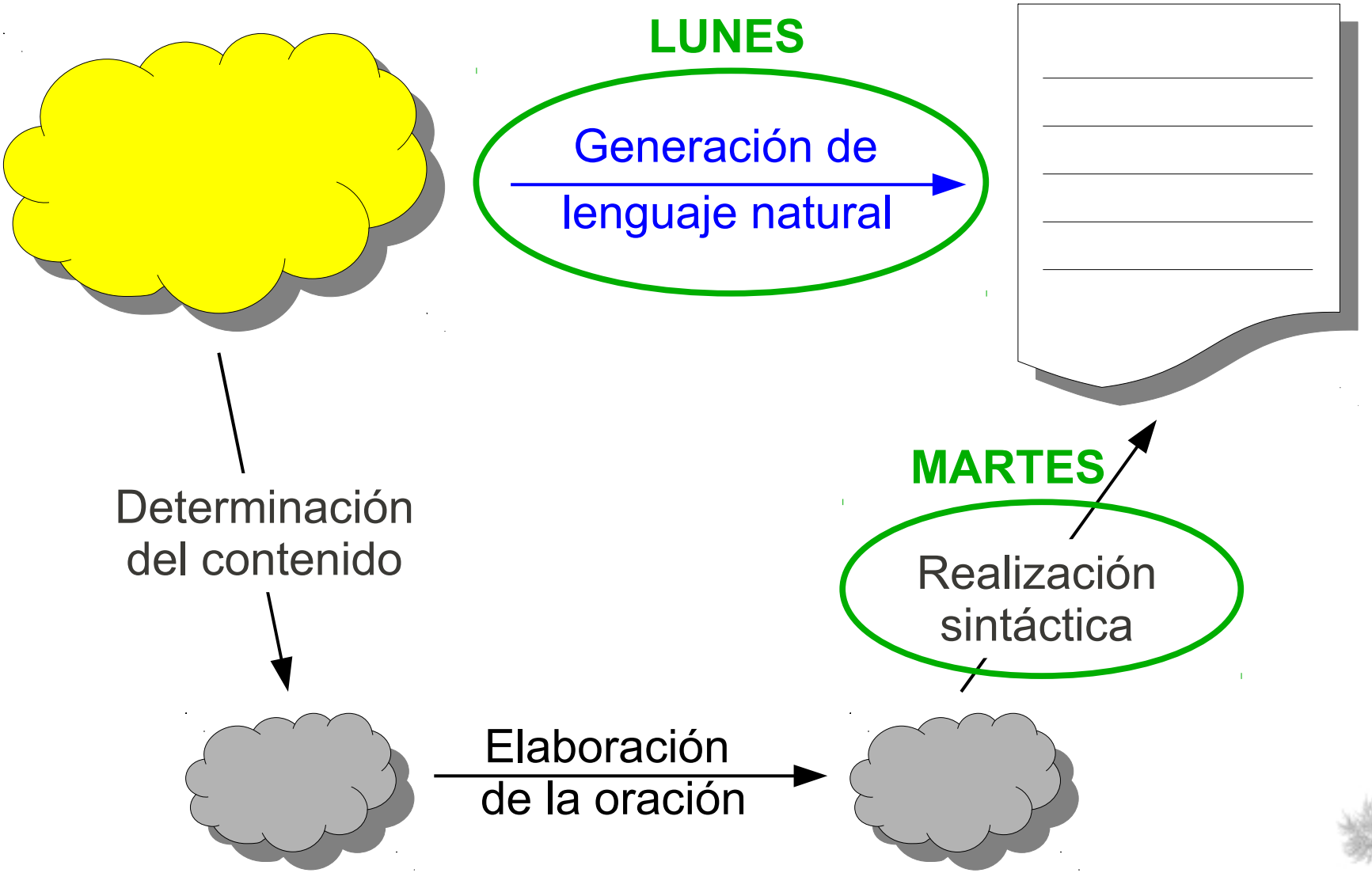
La historia hasta ahora ...



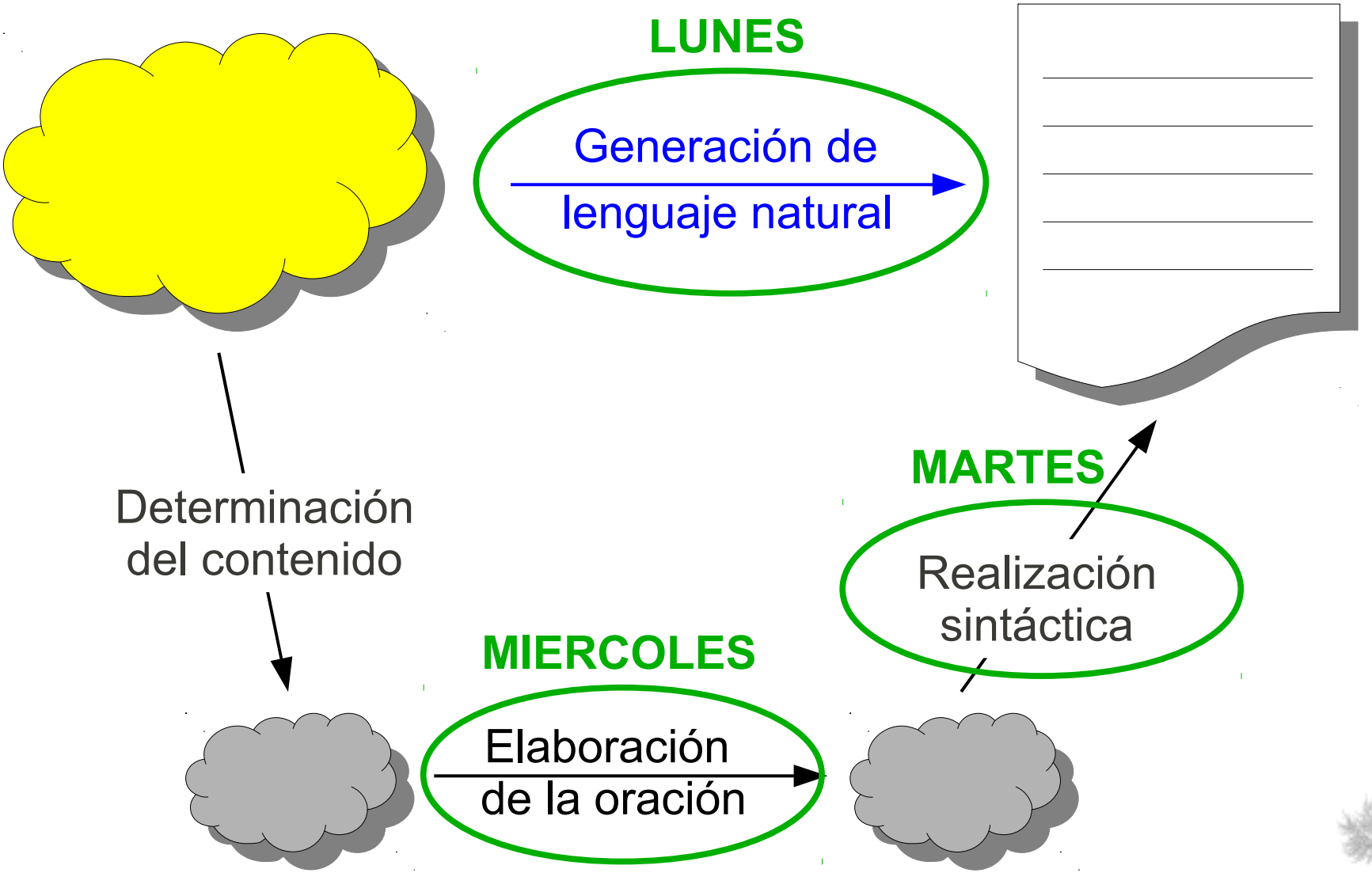
La historia hasta ahora ...



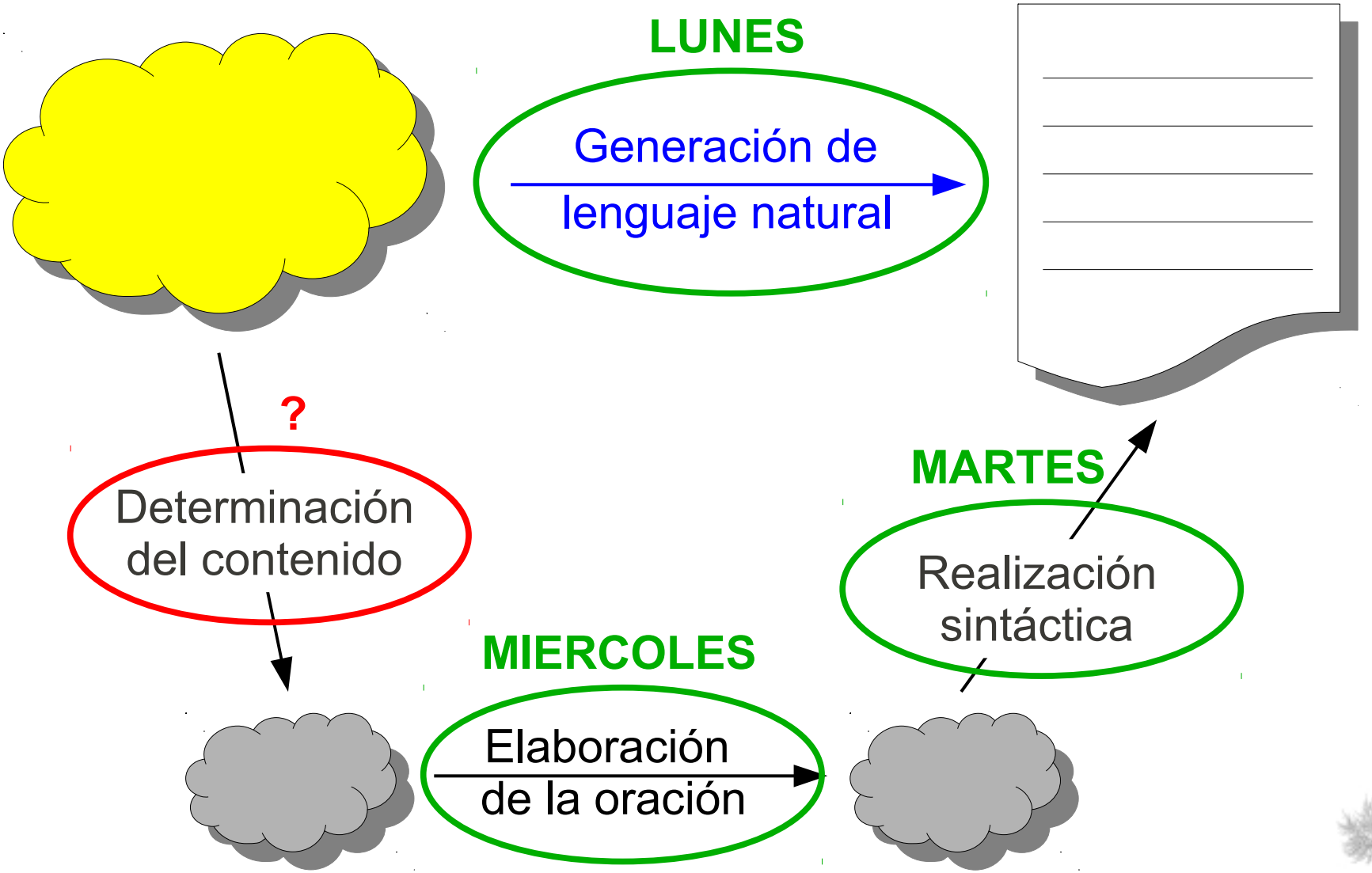
La historia hasta ahora ...



La historia hasta ahora ...



La historia hasta ahora ...



La segunda parte

Generación de lenguaje natural

Marissa, enter the Student Center

Determinación del contenido

Realización sintáctica

Elaboración de la oración

La segunda parte

Generación de lenguaje natural

Marissa, enter the Student Center

HOY

Determinación del contenido

Realización sintáctica

Elaboración de la oración

La segunda parte

MAÑANA

Generación de lenguaje natural

Marissa, enter the Student Center

HOY

Determinación del contenido

MAÑANA

Elaboración de la oración

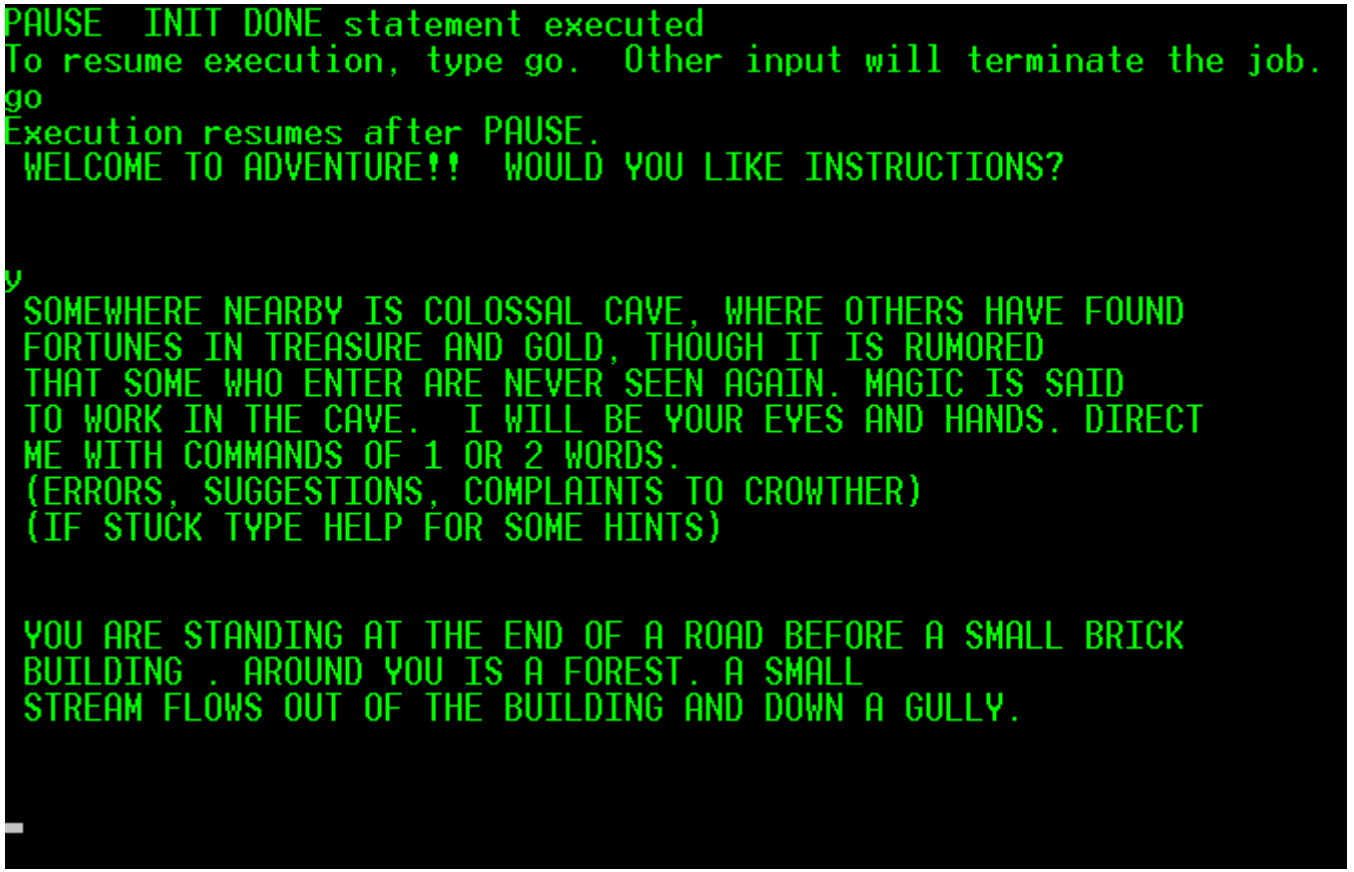
Realización sintáctica

Qué vamos a hacer hoy?

- Introducción a los entornos virtuales
- Herramientas para determinación del contenido
  - Introducción a planning, planning vs búsqueda
  - Representación de un problema de planning
  - El algoritmo de graphplan
  - Búsqueda en el espacio de estados y heurísticas
- GIVE World: Un mundo virtual simple
  - Generación de instrucciones en entornos virtuales

Entornos virtuales: Una rápida historia

- '70: Interacción basada en texto



Entornos virtuales: Una rápida historia

- '80: Empiezan los primeros gráficos

Entornos virtuales: Una rápida historia

- '90: Empiezan los juegos online multijugador

Entornos virtuales: Una rápida historia

- 2000: Llegan las comunidades online con contenido creado por los usuarios

2010': Llega la educación virtual?

- Mas de 100 universidades de todo el mundo tienen un 'campus virtual'

Aprender ciencia en un entorno virtual

Qué necesito para dar instrucciones?

- Saber qué es lo que quiero lograr con las instrucciones:
  - Representar y razonar sobre la meta
- Saber cuál es la situación en la que el mundo se encuentra en este momento:
  - Representar y razonar sobre el estado inicial
- Saber cómo puede cambiar el mundo:
  - Representar y razonar sobre las acciones posibles

Qué vamos a hacer hoy?

- Introducción a los entornos virtuales
- Herramientas para determinación del contenido
  - Introducción a planning, planning vs búsqueda
  - Representación de un problema de planning
  - El algoritmo de graphplan
  - Búsqueda en el espacio de estados y heurísticas
- GIVE World: Un mundo virtual simple
  - Generación de instrucciones en entornos virtuales

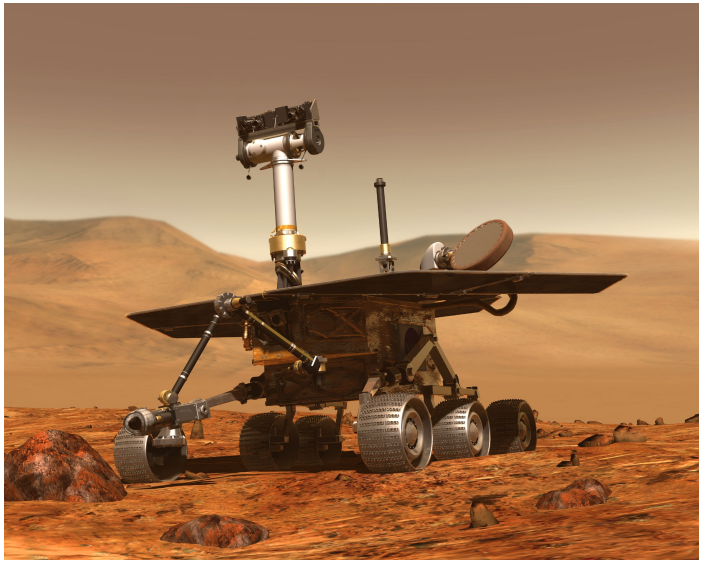
Qué es planning?

Planning es la tarea de inferir una secuencia de acciones que, al ser ejercutada en un determinado estado del mundo, llegará a una meta

- Planning es una tarea de inferencia
- Nosotros la usaremos para determinación del contenido de GLN en entornos virtuales
- Pero tiene otras aplicaciones ...

Aplicaciones de planning

- Decidir las acciones de una linea de producción
- Decidir los movimientos de un ascensor
- Decidir el movimiento de papel dentro de una fotocopiadora
- Decidir las acciones de un robot (como el Mars Rovers)



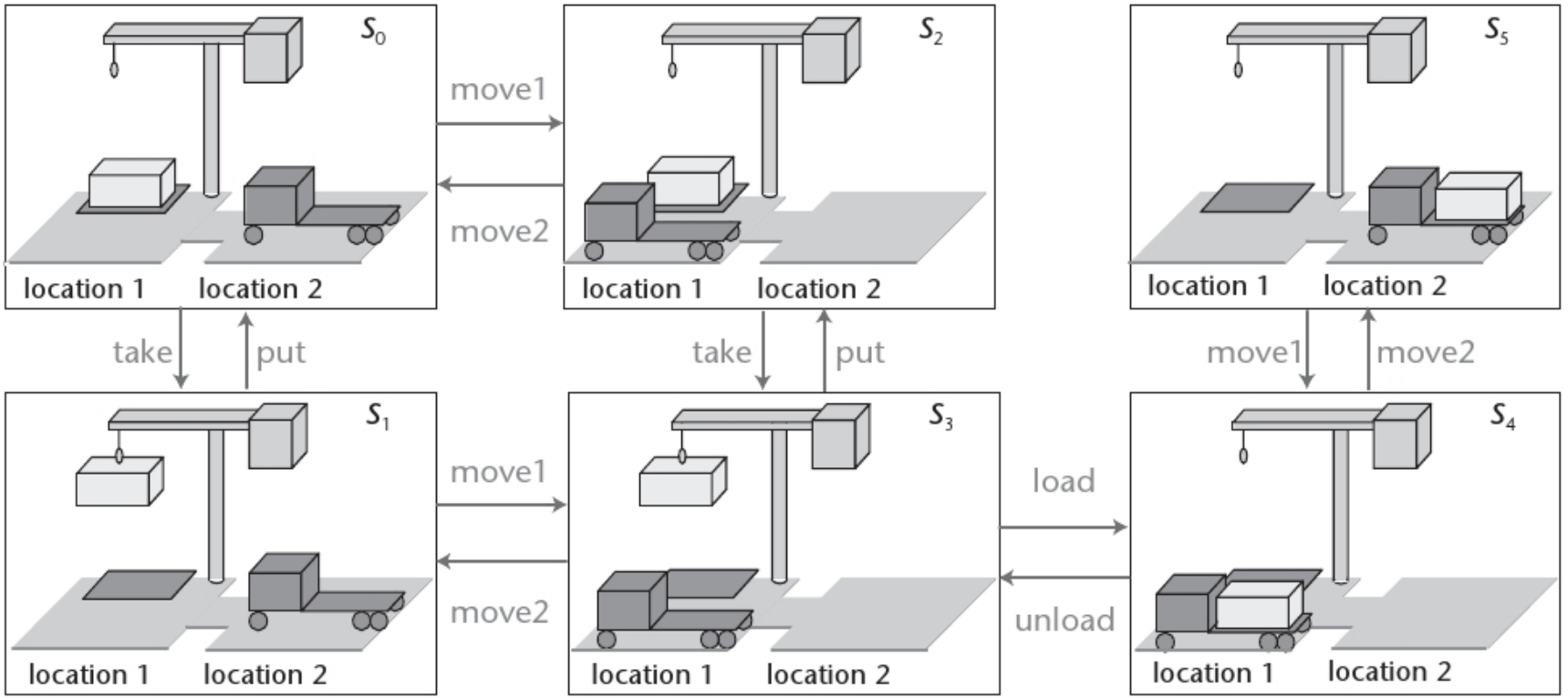
Definiendo un problema de planning

- El primer ingrediente es un sistema de transición de estados (STE)
- Un STE es un modelo formal del entorno virtual (o real) para el que queremos inferir planes
- Un STE se define como una tupla $\Sigma = (S, A, g)$
  - S es un conjunto de estados
  - A es un conjunto de acciones
  - $g: S \times A \rightarrow 2^S$ es una función de transición de estados

Definiendo un problema de planning

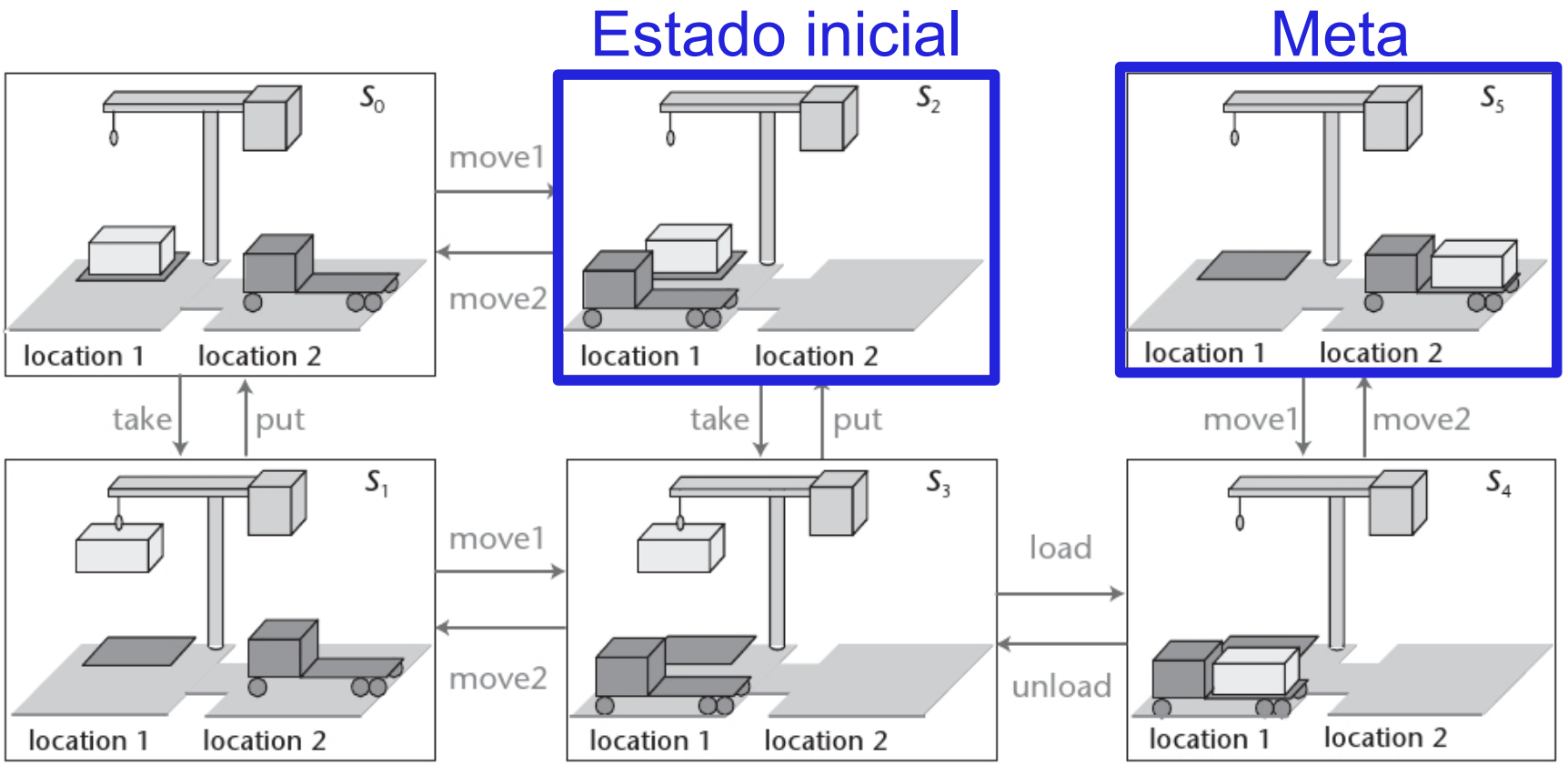
- Un problema de planning es una tupla $P = (\Sigma, s_i, S_g)$ donde:
  - $\Sigma=(S, A, g)$ es un STE
  - $s_i \in S$ es un estado inicial
  - S_g C S es la meta
- Un plan es una secuencia de acciones $a_1, a_2, \ldots, a_k$ tal que:
  - La secuencia de estados $s_i, s_1, \ldots, s_k$ satisfies $s_1 \in g(s_i,a_1)$, $s_2 \in g(s_1,a_2), \ldots, s_k \in g(s_{k-1},a_k)$,
  - $s_k \in S_g$

Ejemplo de un problema de planning



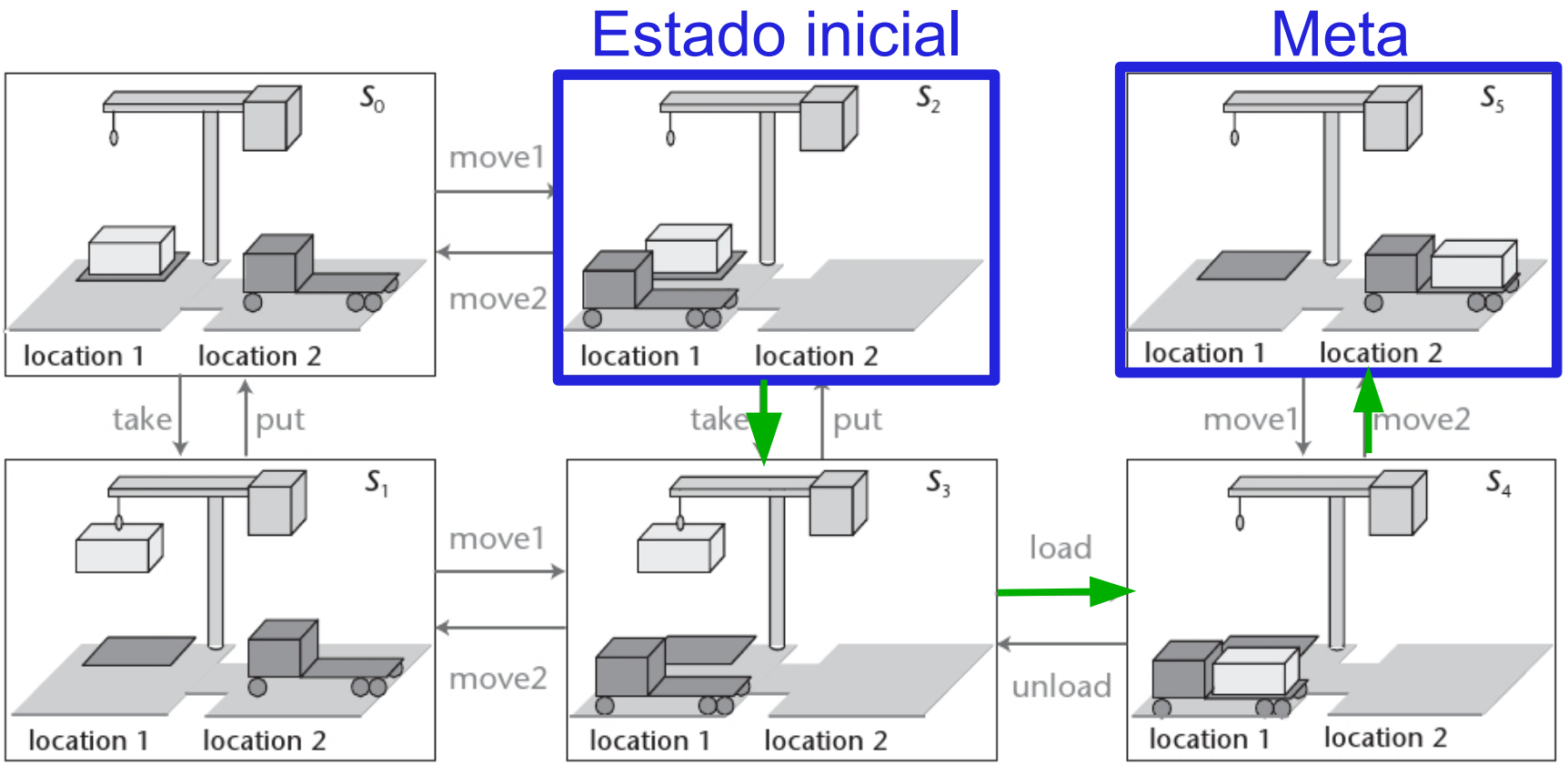
Un sistema de transición de estados puede representarse como un grafo dirigido

Ejemplo de un problema de planning



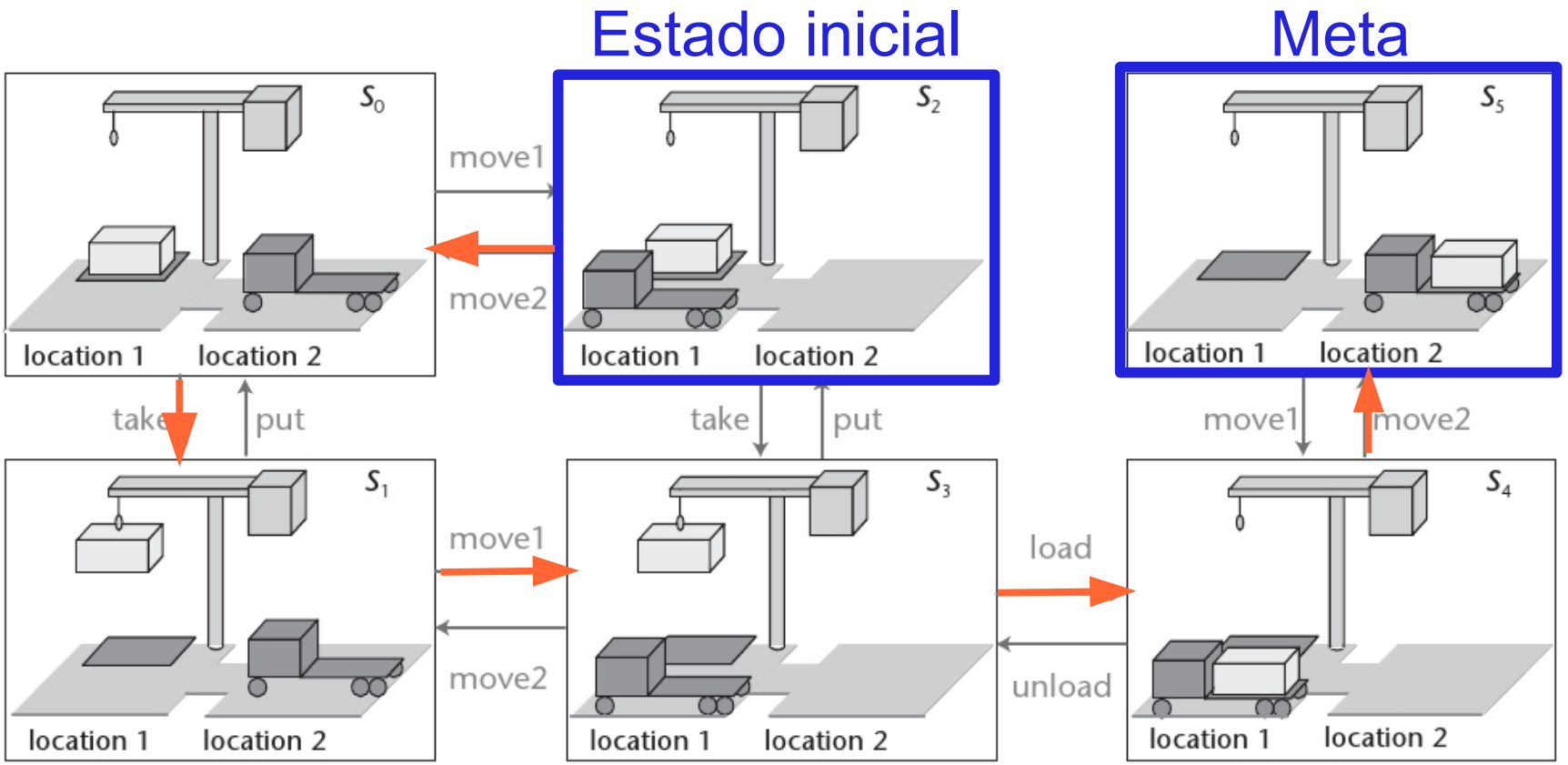
El estado inicial y la meta son nodos en el grafo

Ejemplo de un problema de planning



Un plan es un camino del estado inicial a la meta

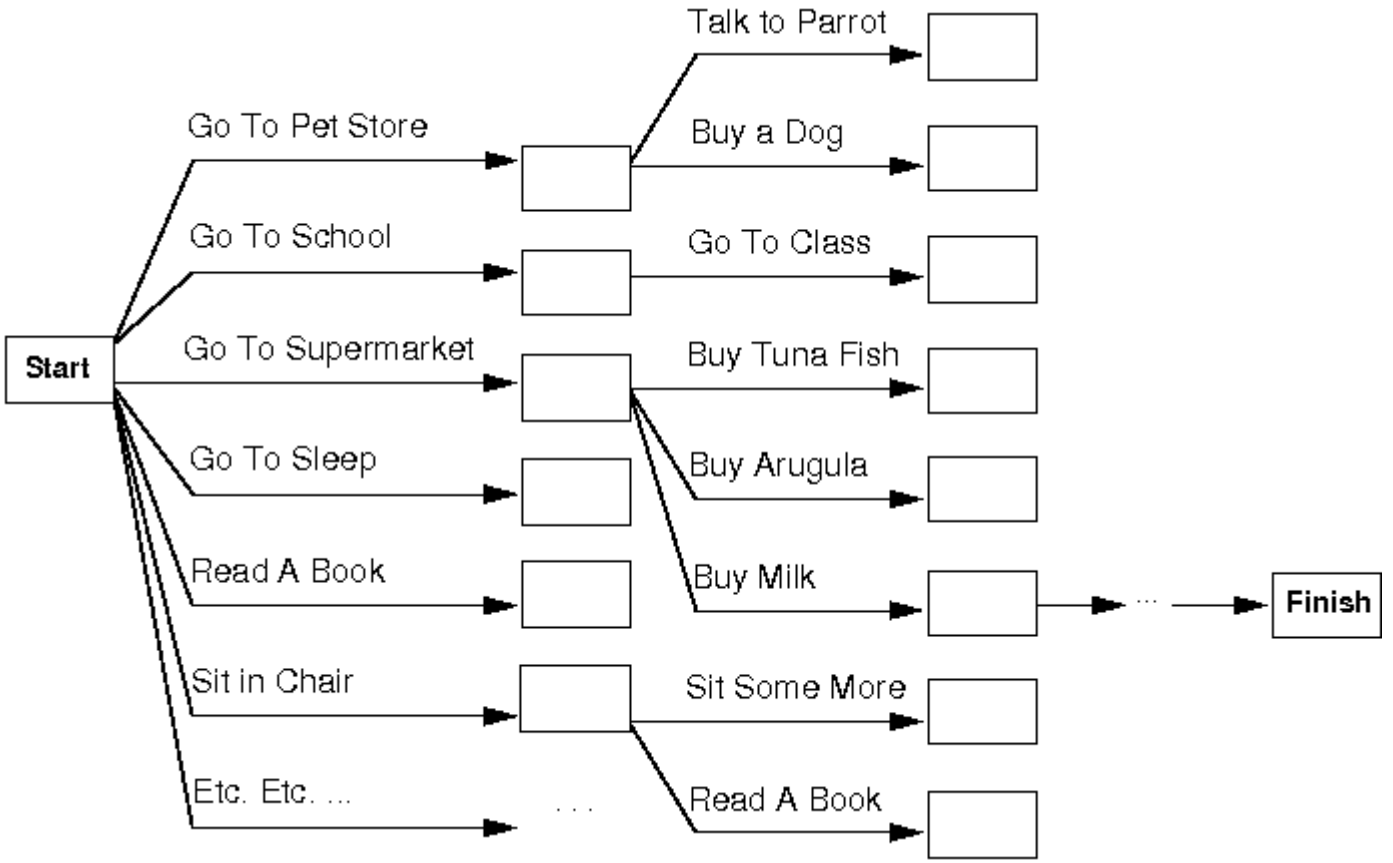
Ejemplo de un problema de planning



Un plan es un camino del estado inicial a la meta

Pero planning es *búsqueda*!

- Si, los problemas simples de planning se pueden resolver como búsqueda pero …



Búsqueda ciega

- La meta es vista como una caja negra por los metodos ciegos de búsqueda:
  - Depth-first search
  - Breath-first search
  - Etc.

- En búsqueda ciega no sabemos:
  - cuán lejos estamos de la meta ni
  - cómo distinguir acciones relevantes

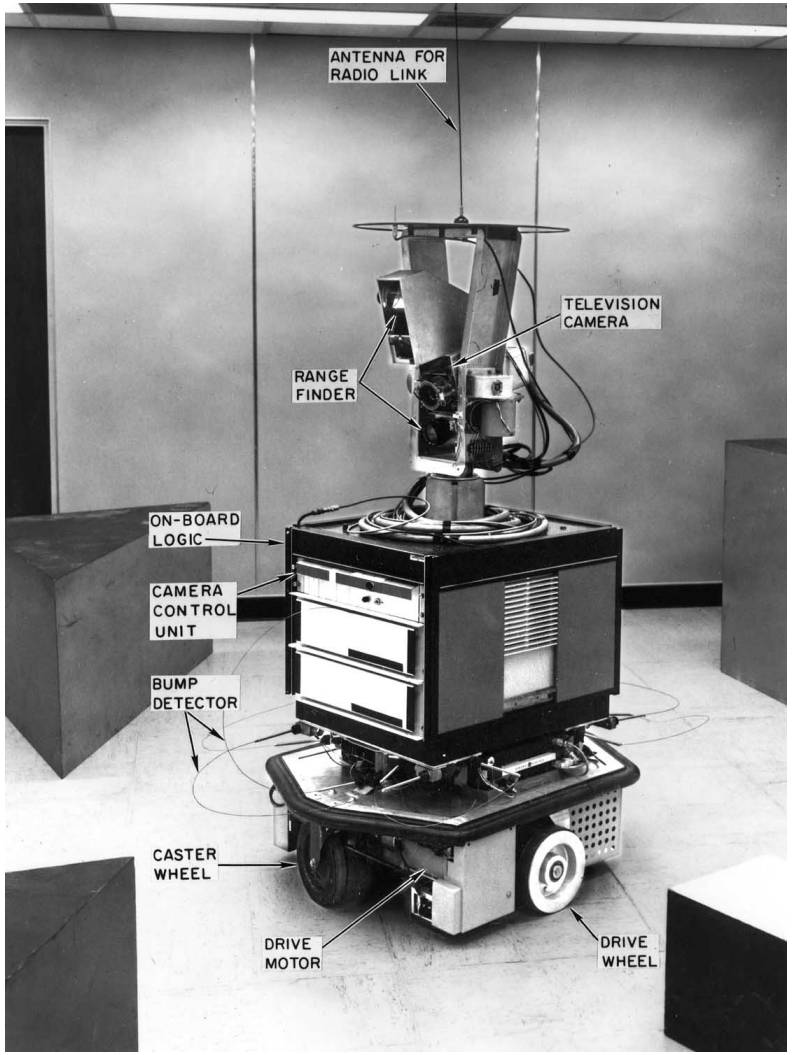
Qué vamos a hacer hoy?

- Introducción a los entornos virtuales
- Herramientas para determinación del contenido
  - Introducción a planning, planning vs búsqueda
  - Representación de un problema de planning
  - El algoritmo de graphplan
  - Búsqueda en el espacio de estados y heurísticas
- GIVE World: Un mundo virtual simple
  - Generación de instrucciones en entornos virtuales

Deseos para una representación

- Necesitamos una representación tal que los algoritmos puedan aprovechar la estructura del problema examinando:
  - Las transiciones posibles
  - La estructura de la meta
- Necesitamos un lenguaje que sea
  - Lo suficientemente expresivo como para describir una variedad de problemas
  - Pero restringido tal que existan algoritmos eficientes

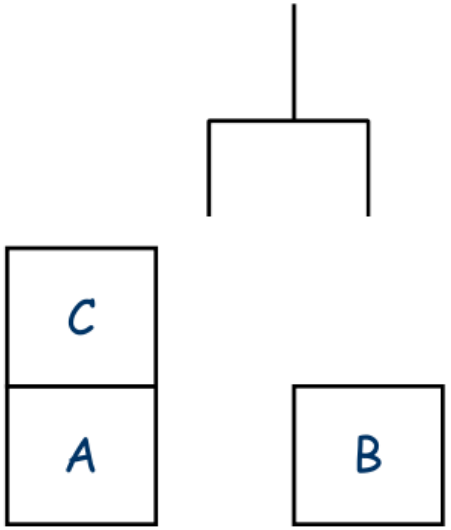
STRIPS: Introducción



- Stanford Research Institute Problem Solver
- Lenguaje más usado para planning
- Las acciones son modeladas como formas estructuradas de cambiar el mundo
- Simple pero expresivo

STRIPS: Sintaxis de estados

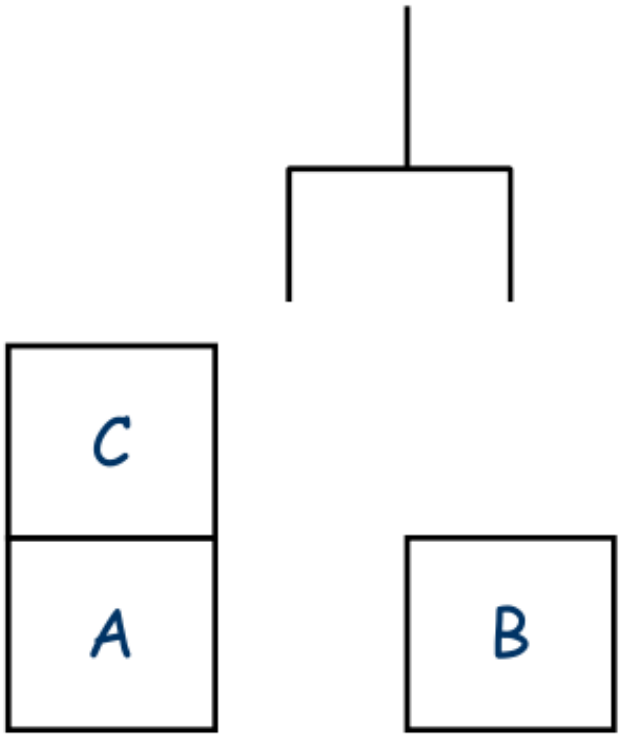
KB = {handempty
clear(c),
clear(b),
on(c,a),
ontable(a),
ontable(b)}



- El estado es como un conjunto de literales positivos de lógica de primer orden
- No hay literales negativos porque se usa CWA
- Los literales no pueden tener funciones ni variables:
  - ~~clear(f(c))~~
  - ~~on(X,Y)~~

STRIPS: Semántica de los estados

- Pueden pensar en un estado STRIPS como una base de datos o un modelo de primer orden.



KB = {handempty
clear(c),
clear(b),
on(c,a),
ontable(a),
ontable(b)}

1. clear(c) ∧ clear(b)?
2. ¬on(b,c)?
3. on(a,c) ∨ on(b,c)?
4. ∃X.on(X,c)? (D = {a,b,c})
5. ∀X.ontable(X) → X = a ∨ X = b?

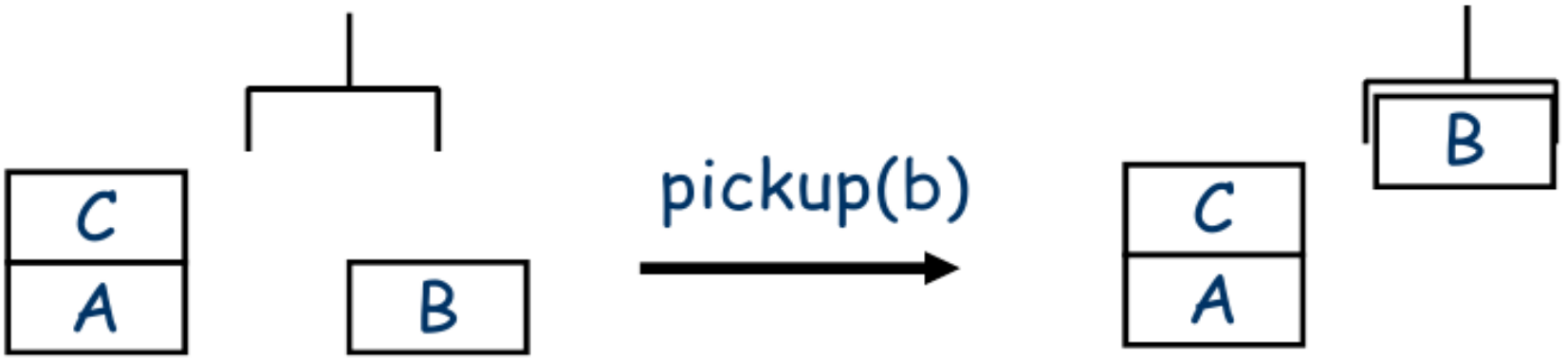
STRIPS: Sintaxis de las acciones

- STRIPS representa acciones usando un nombre y 3 listas:
  1. Precondiciones
  2. Efectos a agregar
  3. Efectos a borrar

pickup(X)
Pre: {clear(X), ontable(X), handempty}
Add: {holding(X)}
Del: {clear(X), ontable(X), handempty}

- pickup(X) es llamado un operador STRIPS, un operador contiene variables
- Cada instanciación de un operador, e.g., pickup(a) resulta en una acción específica

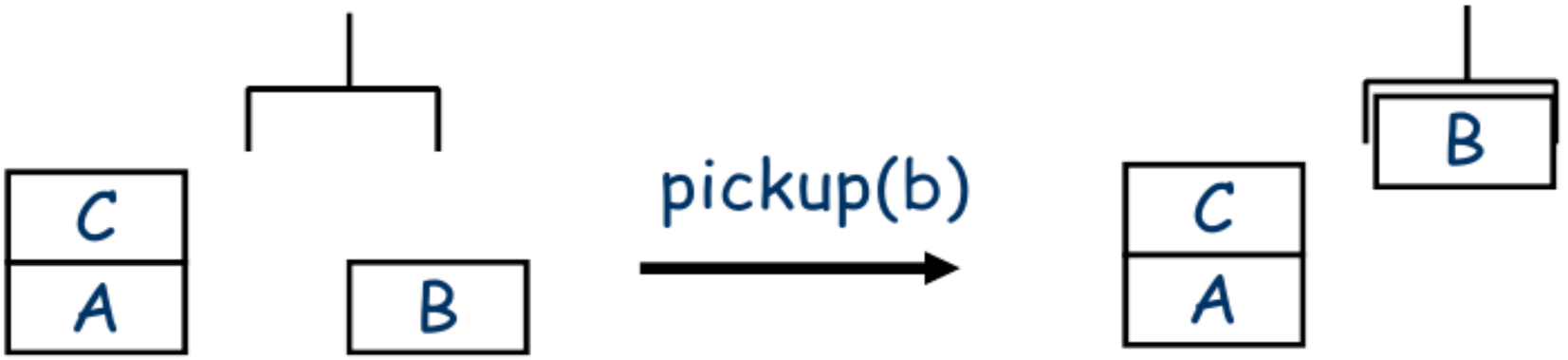
STRIPS: Semántica de la acciones



- pre = {handmpty, clear(b), ontable(b)}
- del = {handmpty, clear(b), ontable(b)}
- add = {holding(b)}

{handmpty
clear(c),
clear(b),
on(c,a),
ontable(a),
ontable(b)}

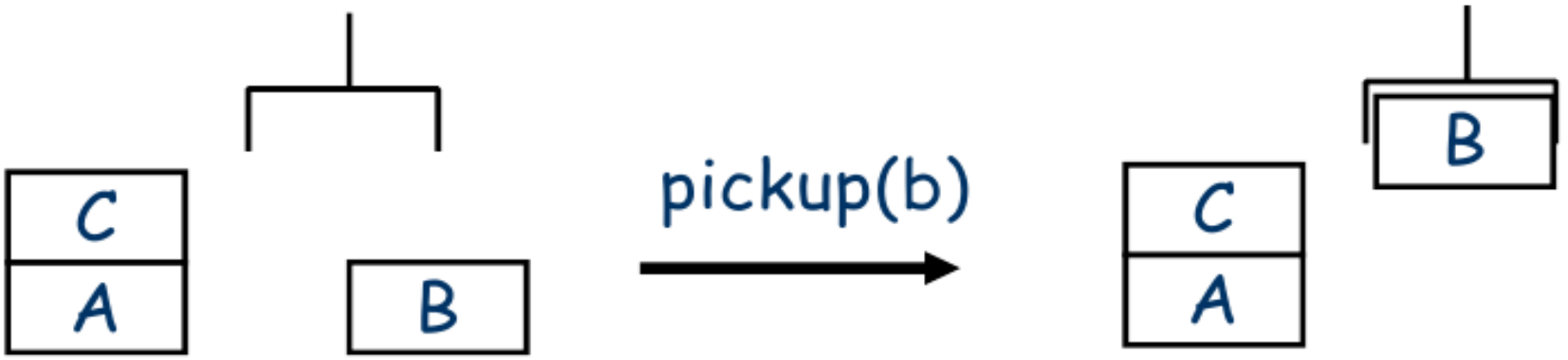
STRIPS: Semántica de la acciones



- pre = {handmpty, clear(b), ontable(b)}
- del = {handmpty, clear(b), ontable(b)}
- add = {holding(b)}

{handmpty
clear(c),
clear(b),
on(c,a),
ontable(a),
ontable(b)}

STRIPS: Semántica de la acciones



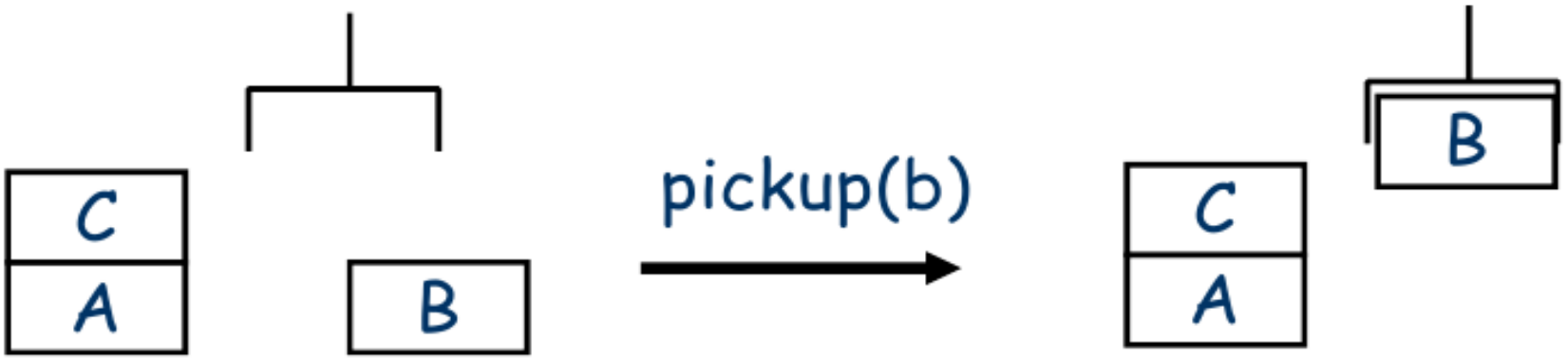
- pre = {handmpty, clear(b), ontable(b)}
- del = {handmpty, clear(b), ontable(b)}
- add = {holding(b)}

{handmpty
clear(c),
clear(b),
on(c,a),
ontable(a),
ontable(b)}

{
clear(c),

on(c,a),
ontable(a),
}

STRIPS: Semántica de la acciones



- pre = {handmpty, clear(b), ontable(b)}
- del = {handmpty, clear(b), ontable(b)}
- add = {holding(b)}

{handmpty
clear(c),
clear(b),
on(c,a),
ontable(a),
ontable(b)}

{
clear(c),

on(c,a),
ontable(a),
}

{
clear(c),

on(c,a),
ontable(a),

holding(b)}

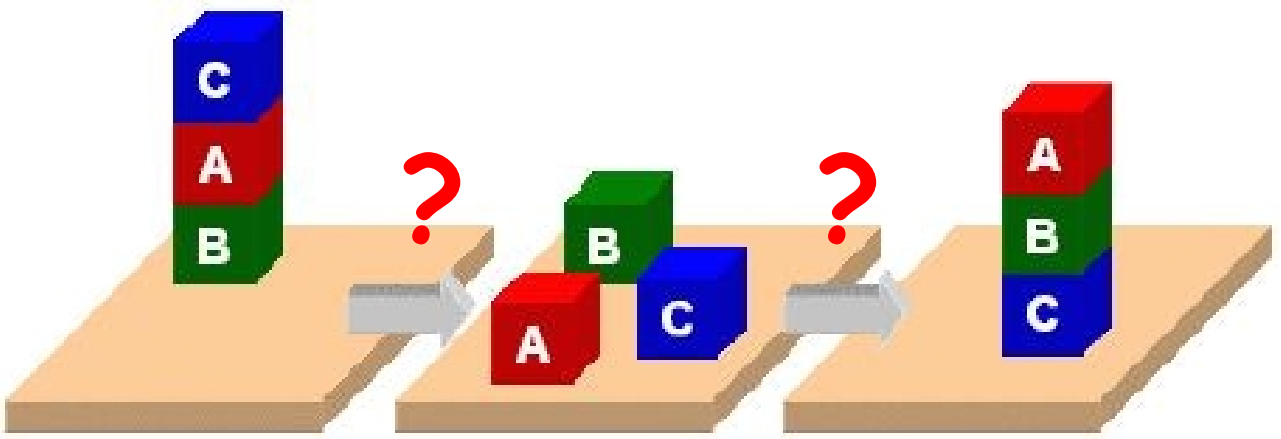
STRIPS: Semántica de las acciones

Para que una acción STRIPS sea aplicable a un estado todas sus precondiciones deben ser verdaderas en el estado

- Esto se puede hacer simplemente evaluando pertenencia ya que sólo tenemos átomos en la lista de precondiciones.
- Si la acción es aplicable, el nuevo estado es generado:
  - eliminando todos los literales en la lista Del
  - agregando todos los literales en la lista Add

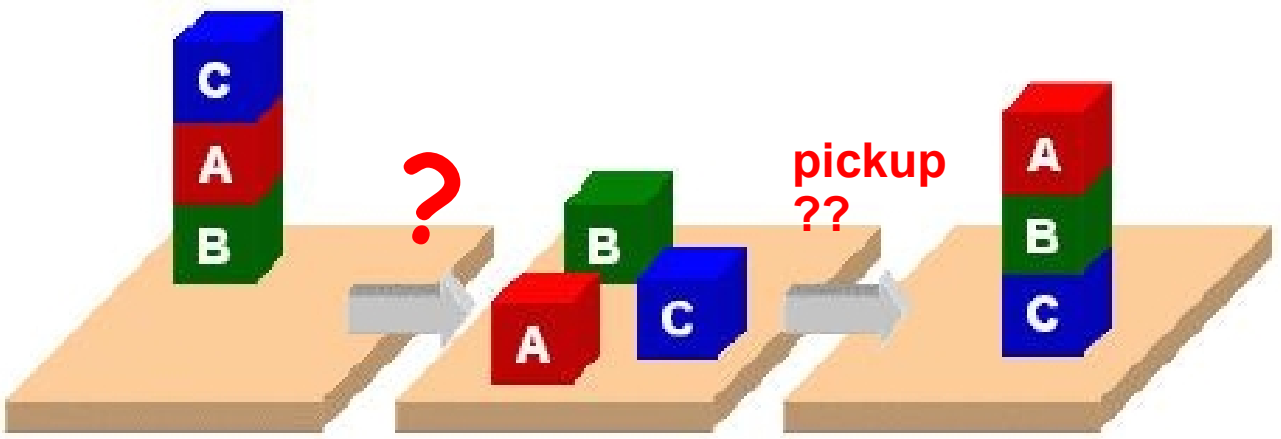
STRIPS: El mundo de bloques

- pickup(X)
  Pre: {clear(X), ontable(X), handempty}
  Add: {holding(X)}
  Del: {clear(X), ontable(X), handempty}



STRIPS: El mundo de bloques

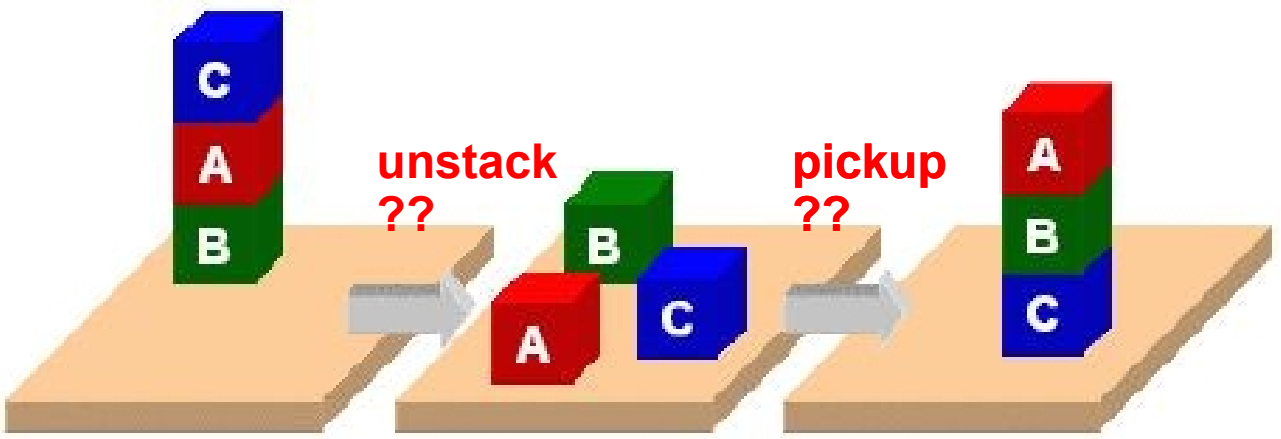
- pickup(X)
  Pre: {clear(X), ontable(X), handempty}
  Add: {holding(X)}
  Del: {clear(X), ontable(X), handempty}



STRIPS: El mundo de bloques

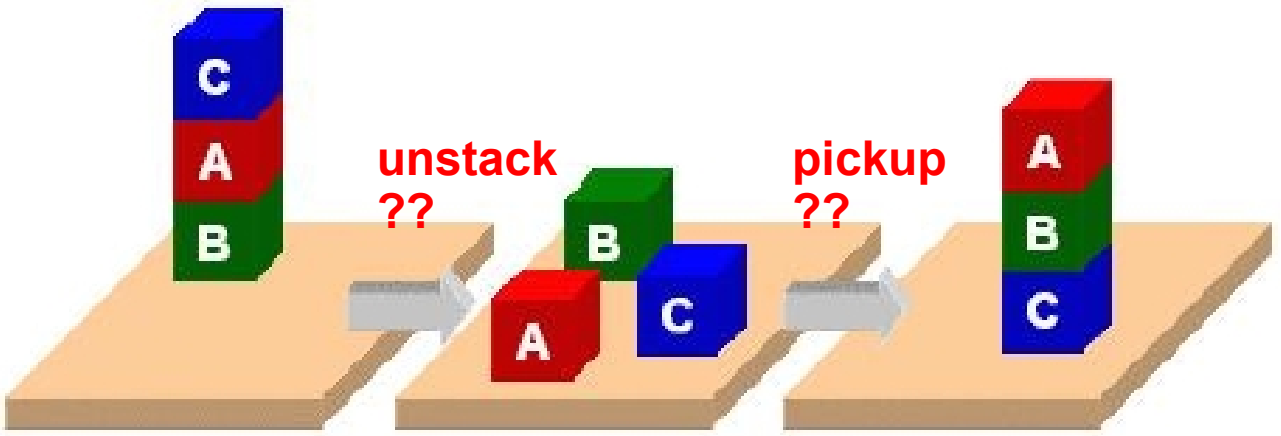
- pickup(X)
  Pre: {clear(X), ontable(X), handempty}
  Add: {holding(X)}
  Del: {clear(X), ontable(X), handempty}

- unstack(X,Y)
  Pre: {clear(X), on(X,Y), handempty}
  Add: {holding(X), clear(Y)}
  Del: {clear(X), on(X,Y), handempty}



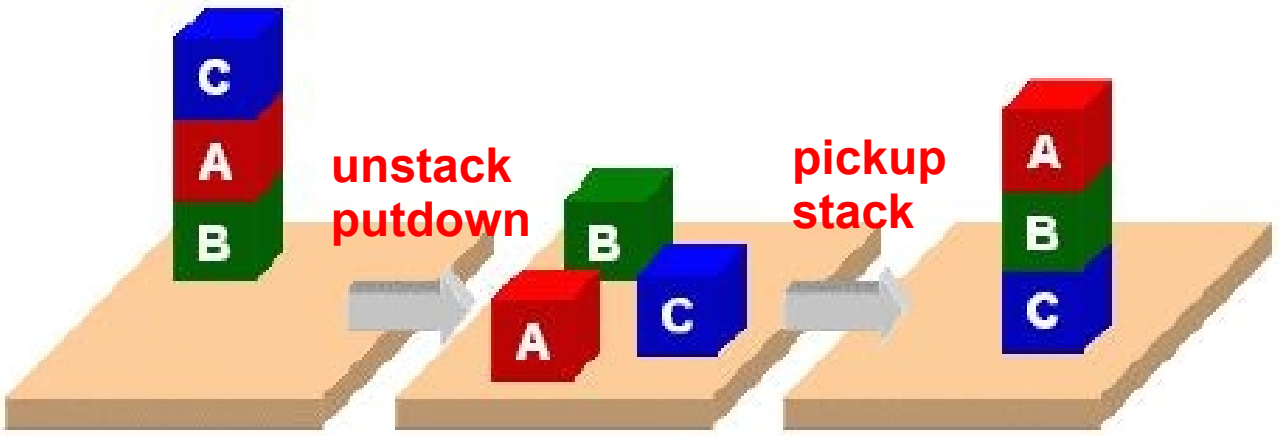
STRIPS: El mundo de bloques

- pickup(X)
  Pre: {clear(X), ontable(X), handempty}
  Add: {holding(X)}
  Del: {clear(X), ontable(X), handempty}
- putdown(X)
  Pre: {holding(X)}
  Add: {clear(X), ontable(X), handempty}
  Del: {holding(X)}
- unstack(X,Y)
  Pre: {clear(X), on(X,Y), handempty}
  Add: {holding(X), clear(Y)}
  Del: {clear(X), on(X,Y), handempty}
- stack(X,Y)
  Pre: {holding(X),clear(Y)}
  Add: {on(X,Y), handempty, clear(X)}
  Del: {holding(X),clear(Y)}



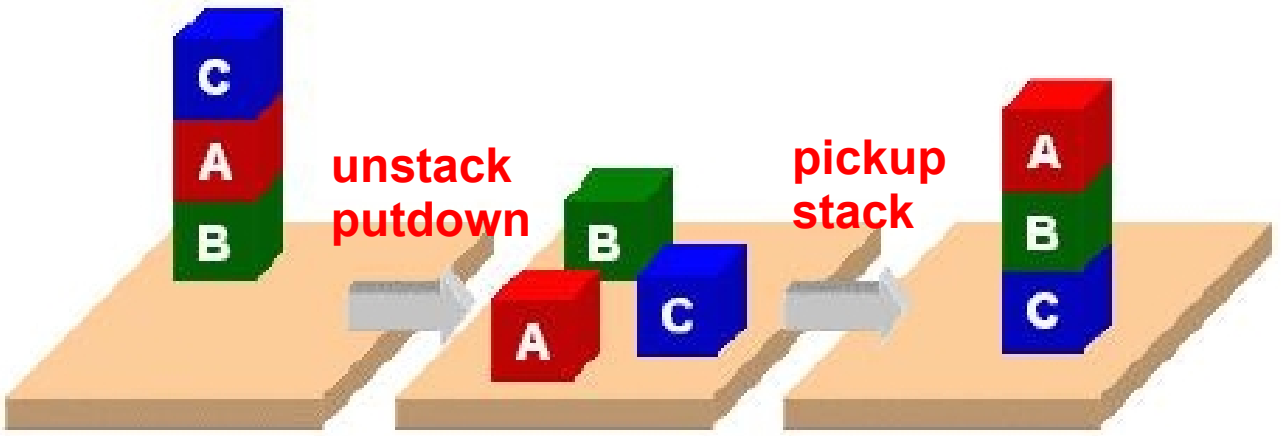
STRIPS: El mundo de bloques

- pickup(X)
  Pre: {clear(X), ontable(X), handempty}
  Add: {holding(X)}
  Del: {clear(X), ontable(X), handempty}
- putdown(X)
  Pre: {holding(X)}
  Add: {clear(X), ontable(X), handempty}
  Del: {holding(X)}
- unstack(X,Y)
  Pre: {clear(X), on(X,Y), handempty}
  Add: {holding(X), clear(Y)}
  Del: {clear(X), on(X,Y), handempty}
- stack(X,Y)
  Pre: {holding(X),clear(Y)}
  Add: {on(X,Y), handempty, clear(X)}
  Del: {holding(X),clear(Y)}



STRIPS: El mundo de bloques

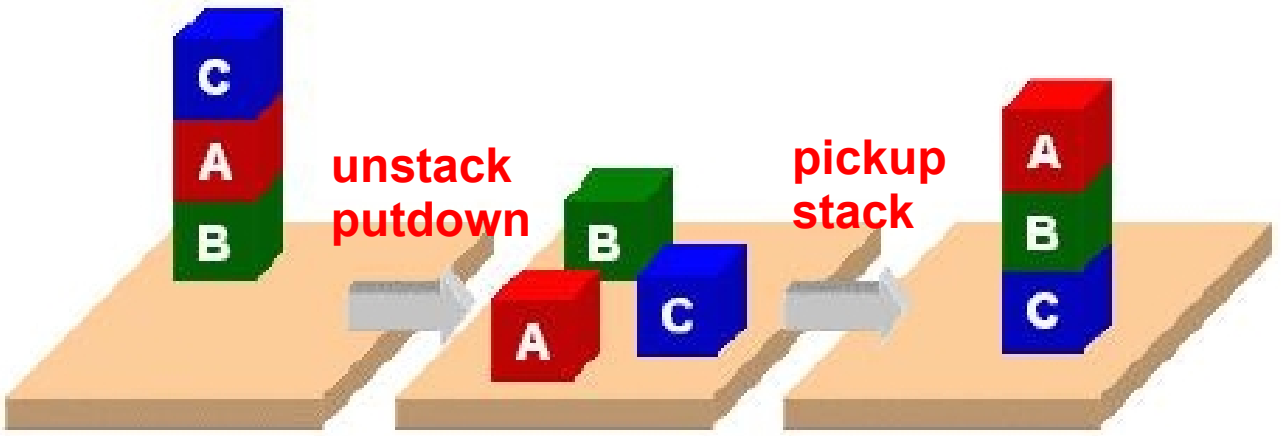
- pickup(X)
  Pre: {clear(X), ontable(X), handempty}
  Add: {holding(X)}
  Del: {clear(X), ontable(X), handempty}
- putdown(X)
  Pre: {holding(X)}
  Add: {clear(X), ontable(X), handempty}
  Del: {holding(X)}
- unstack(X,Y)
  Pre: {clear(X), on(X,Y), handempty}
  Add: {holding(X), clear(Y)}
  Del: {clear(X), on(X,Y), handempty}
- stack(X,Y)
  Pre: {holding(X),clear(Y)}
  Add: {on(X,Y), handempty, clear(X)}
  Del: {holding(X),clear(Y)}



Porqué tantas acciones?

STRIPS: El mundo de bloques

- pickup(X)
  Pre: {clear(X), ontable(X), handempty}
  Add: {holding(X)}
  Del: {clear(X), ontable(X), handempty}
- putdown(X)
  Pre: {holding(X)}
  Add: {clear(X), ontable(X), handempty}
  Del: {holding(X)}
- unstack(X,Y)
  Pre: {clear(X), on(X,Y), handempty}
  Add: {holding(X), clear(Y)}
  Del: {clear(X), on(X,Y), handempty}
- stack(X,Y)
  Pre: {holding(X),clear(Y)}
  Add: {on(X,Y), handempty, clear(X)}
  Del: {holding(X),clear(Y)}

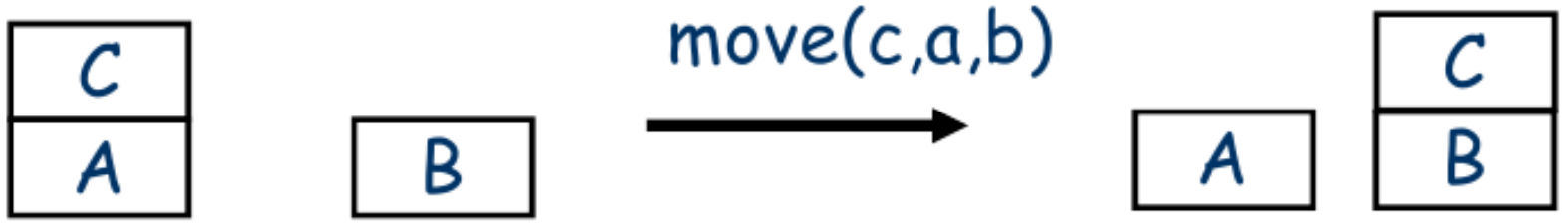


Porqué tantas acciones?

Extendiendo STRIPS: ADL

- Como STRIPS no tiene efectos condicionales, se debe especificar una acción para cada condición
- ADL extiende STRIPS con:
  - Efectos condicionales
  - Soporte sintáctico para igualdad/desigualdad
  - Soporte de tipos (X: Red)
  - Fórmulas arbitrarias en las precondiciones
- Pocos planners implementan ADL completo

ADL: Semántica de las acciones



move(c,a,b)

Pre: $on(c,a) \wedge clear(b) \wedge clear(c)$

Effs: ADD[on(c,b)]

DEL[on(c,a)]

$b \neq table \rightarrow$ DEL[clear(b)]

$a \neq table \rightarrow$ ADD[clear(a)]

KB = { clear(c), clear(b),
on(c,a),
on(a,table),
on(b,table)}

KB = { on(c,b)
clear(c), clear(a)
on(a,table),
on(b,table)}

Qué vamos a hacer hoy?

- Introducción a los entornos virtuales
- Herramientas para determinación del contenido
  - Introducción a planning, planning vs búsqueda
  - Representación de un problema de planning
  - El algoritmo de graphplan
  - Búsqueda en el espacio de estados y heurísticas
- GIVE World: Un mundo virtual simple
  - Generación de instrucciones en entornos virtuales

GraphPlan: Introducción

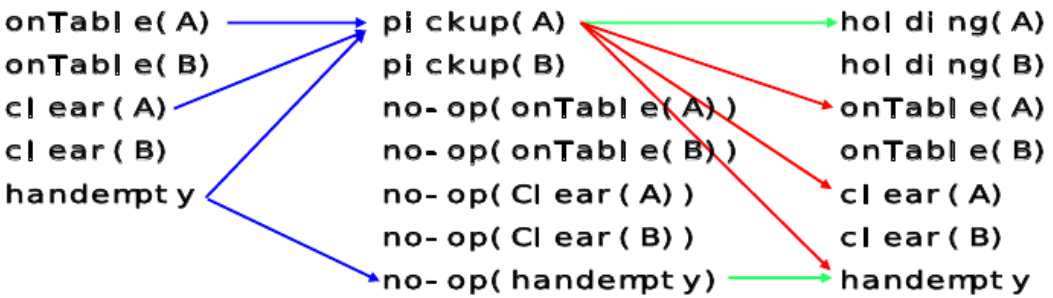
El algoritmo opera en 2 fases:

1. Construir un grafo de planning que codifique las interacciones entre acciones
2. Usar el grafo de planning para restringir la búsqueda de un plan válido
   - Si un plan existe, es un subgrafo del grafo de planning

- Un grafo de planning puede construirse en tiempo polinomial
- El algoritmo de búsqueda es completo y termina, pero es PSPACE-Complete

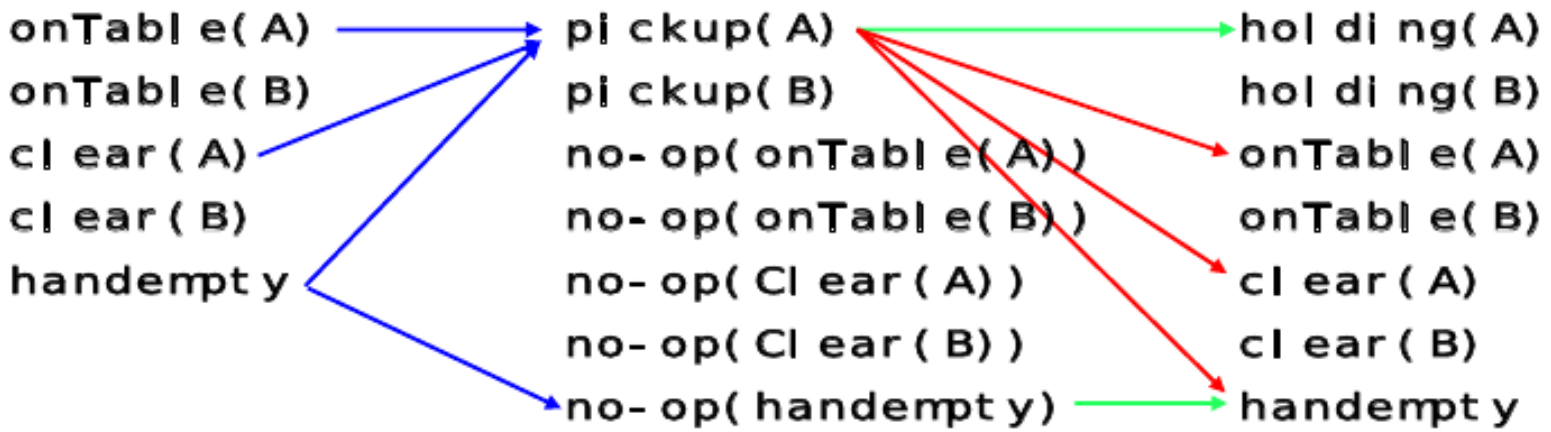
Graphplan: Construir el grafo

- 2 tipos de nodos (organizados en niveles):
  - Literales: P
  - Acciones: A
- 3 tipos de arcos:
  - Precondition: P -> A
  - Add: A -> P
  - Delete: A -> P



- Los niveles de literales y acciones se alternan
- Los niveles de acción incluyen acciones cuyas precondiciones se satisfacen en los niveles previous mas acciones no-op

Graphplan: construir el grafo



Estado inicial:

Sólo las propocisiones verdaderas en el estado inicial

Acciones posibles:

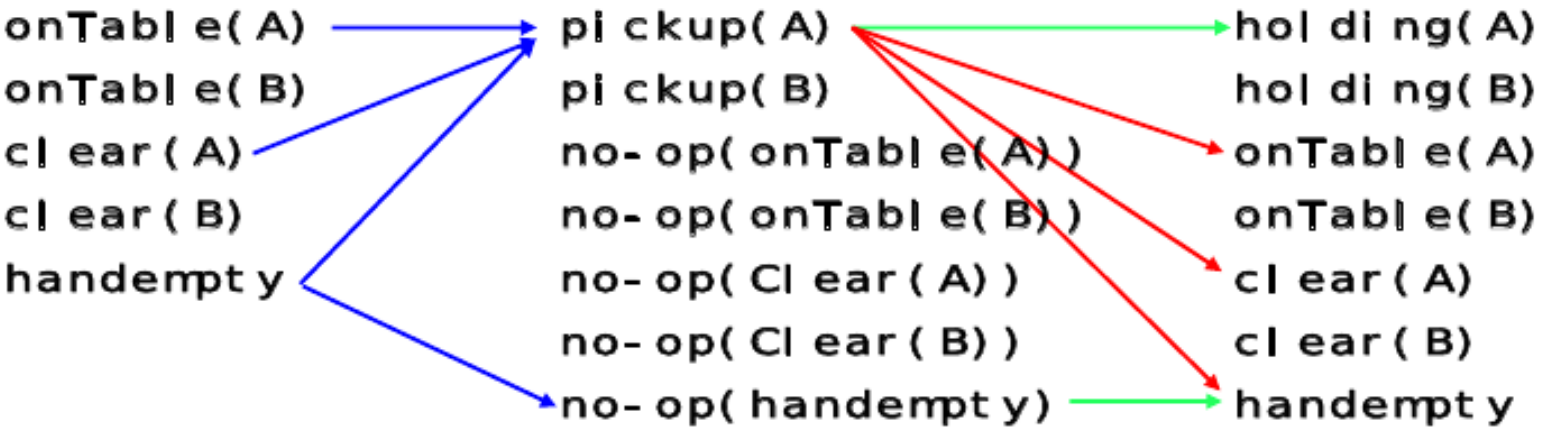
Sólo las acciones cuyas precondiciones están en el nivel anterior

También tenemos noops para capturar lo que no cambia

Todas las proposiciones agregadas por las acciones del nivel anterior

- Precondition
- Delete
- Add

Graphplan: construir el grafo



- El nivel P0 contiene hechos que son verdaderos en el estado inicial
- El nivel A0 contiene todas las acciones cuyas precondiciones son verdaderas en el estado inicial; mas una acción no-op para cada literal.
- La precondición y el efecto de una acción no-op es el mismo literal
- …
- El nivel Pi contiene todos los literales despues de aplicadas las acciones del nivel Ai-1 al nivel Pi-1
- El nivel Ai contiene todas las acciones cuyas precondiciones son verdaderas en Pi

Graphplan: Construir el grafo

- Además de hechos y acciones, graphplan calcula y agrega mutexes al grafo
- Los mutexes son arcos entre dos nodos en el grafo que indican que esos dos nodos no pueden ocurrir en ese nivel

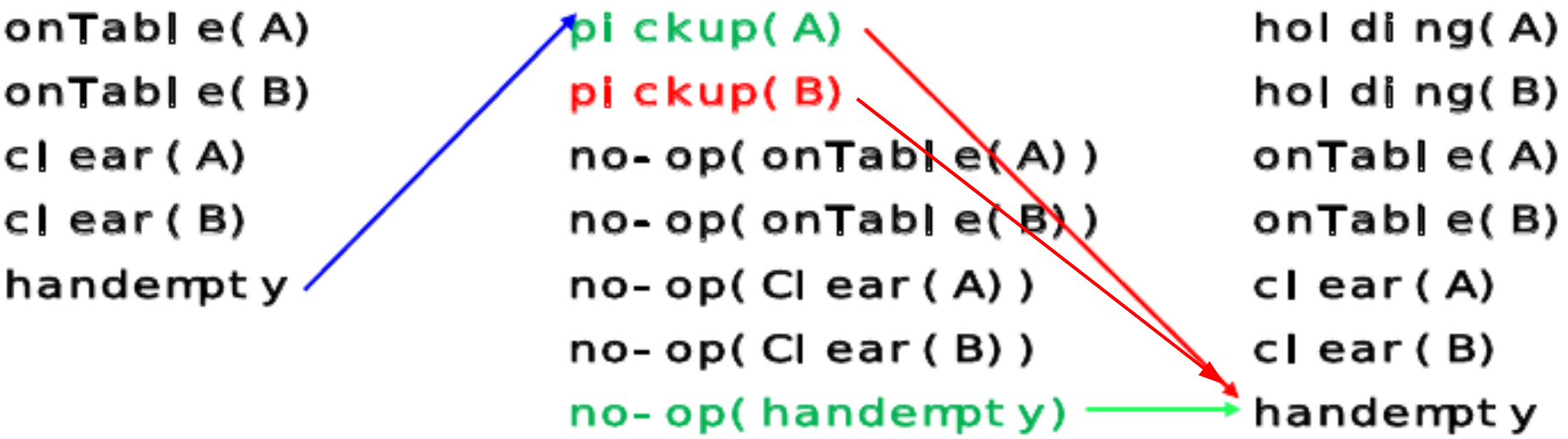
Graphplan: Construir el grafo

- Además de hechos y acciones, graphplan calcula y agrega mutexes al grafo
- Los mutexes son arcos entre dos nodos en el grafo que indican que esos dos nodos no pueden ocurrir en ese nivel
- Un mutex entre dos acciones a1 y a2 en la misma capa Ai significa que a1y a2 no podrán ser ejecutadas simultáneamente en el paso i-ésimo de ningún plan

Graphplan: Construir el grafo

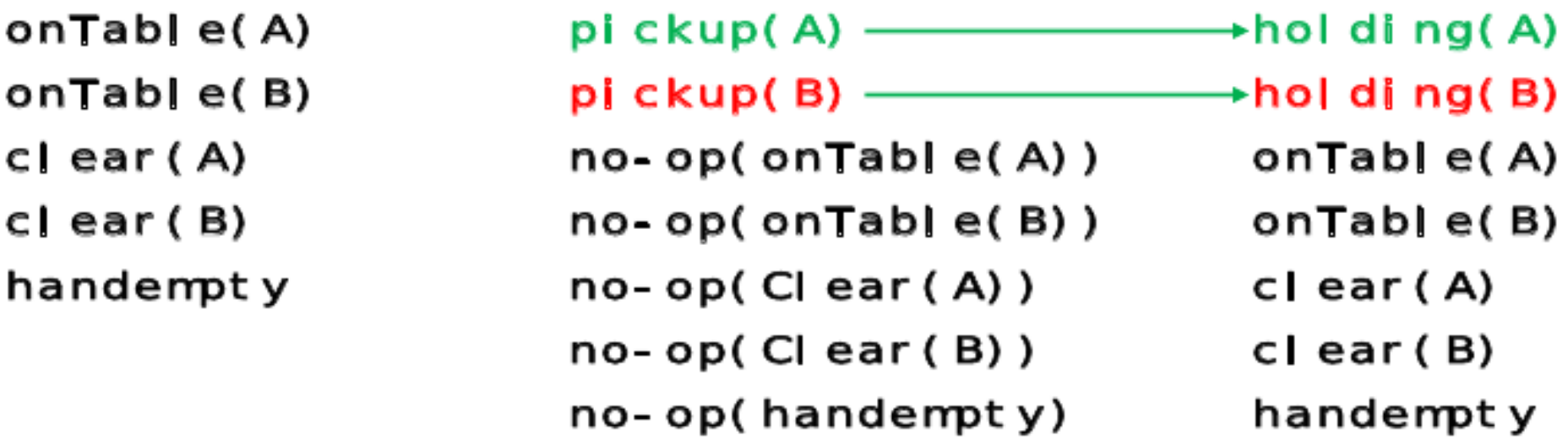
- Además de hechos y acciones, graphplan calcula y agrega mutexes al grafo
- Los mutexes son arcos entre dos nodos en el grafo que indican que esos dos nodos no pueden ocurrir en ese nivel
- Un mutex entre dos acciones a1 y a2 en la misma capa Ai significa que a1y a2 no podrán ser ejecutadas simultáneamente en el paso i-ésimo de ningún plan
- Un mutex entre dos literales F1 y F2 en la misma capa Pi significa que F1 y F2 no pueden ser verdaderos simultáneamente después de I pasos de ejecución

Graphplan: construir el grafo



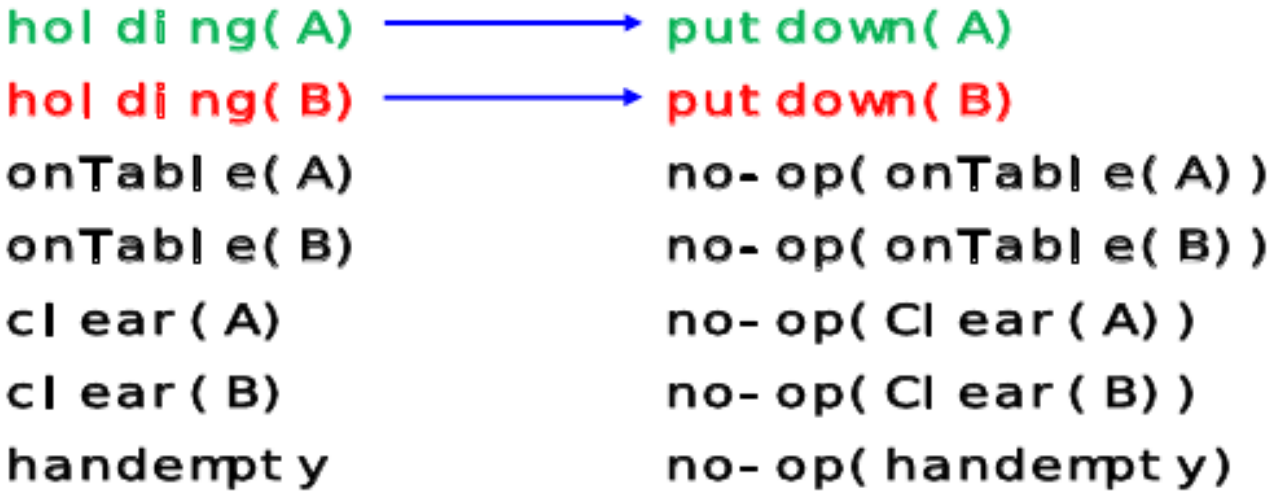
- Dos acciones son mutex si una de ellas borra una precondición o un efecto add de la otra.
- Los no-ops participan en los mutexes.

Graphplan: construir el grafo



- Dos proposiciones p y q son mutex si todas las acciones que agregan p son mutex con todas las acciones que agregan q.

Graphplan: construir el grafo



- Dos acciones son mutex si dos de sus precondiciones son mutex.

Graphplan: Construir mutex

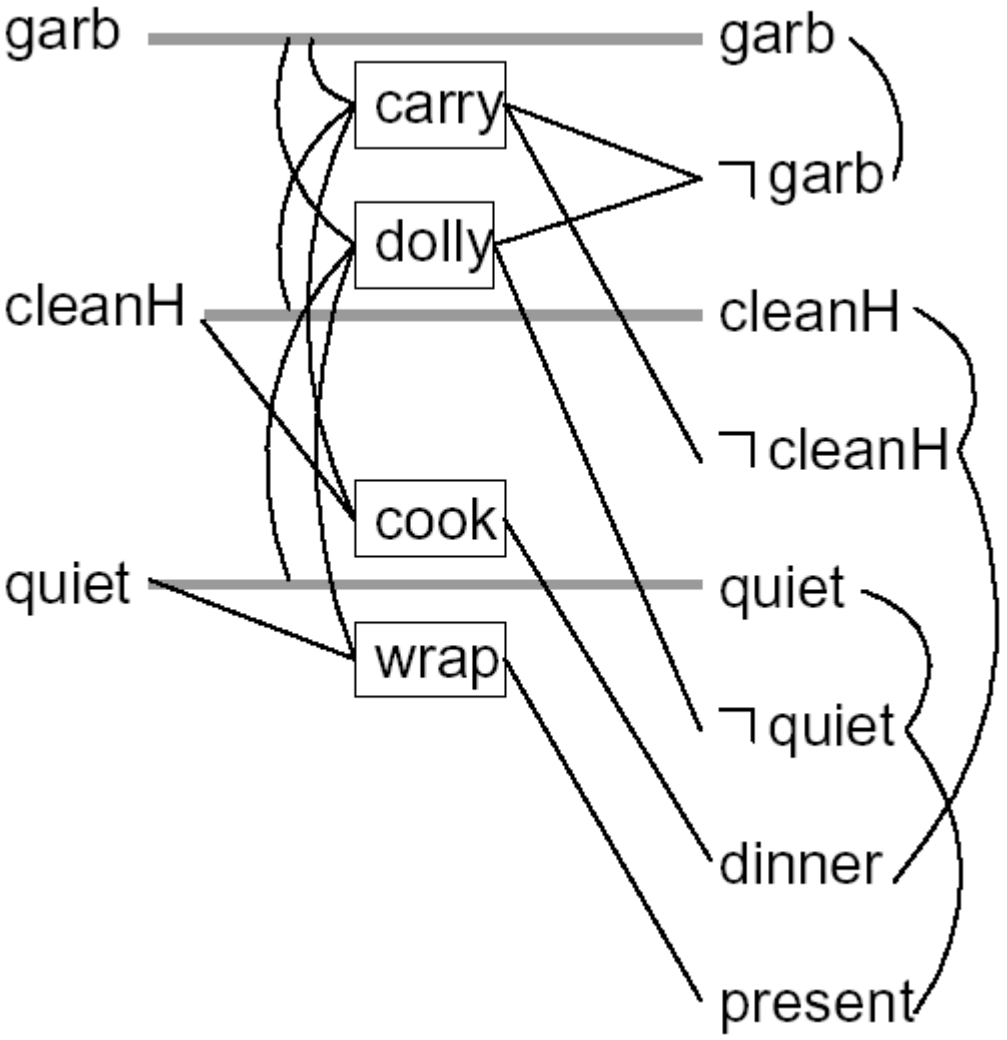
- Dos acciones o literales son mutex (mutualmente excluyentes) en algun nivel si no pueden suceder juntas en ese nivel
- Dos acciones son mutex si:
  1) Una borra las precondiciones o los efectos add de la otra

  2) Dos de sus precondiciones son mutex
- Dos proposiciones son mutex si:
  3) Todas las acciones que agregan una son mutex con todas las acciones que agregan la otra

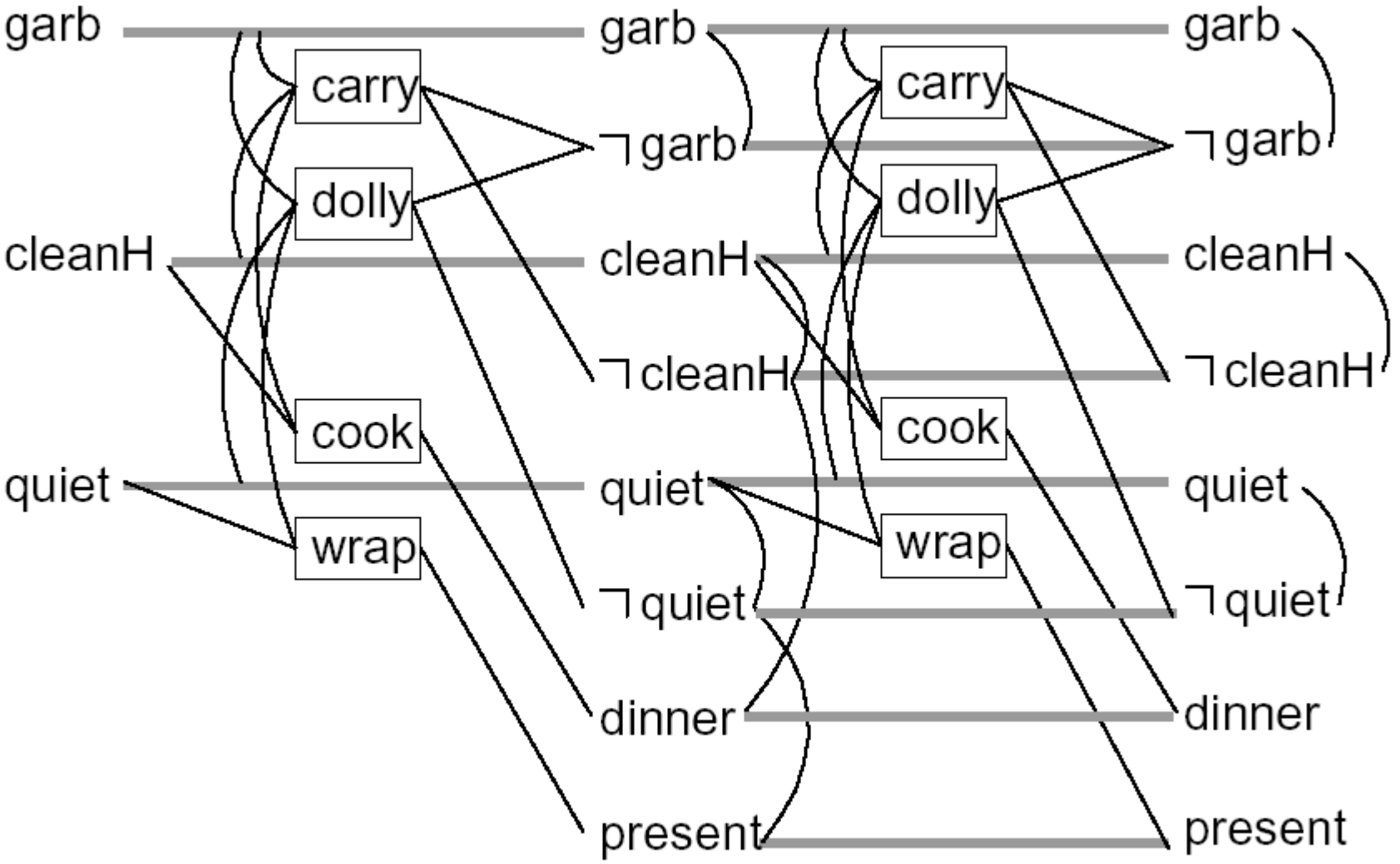
Graphplan: ejemplo simple

- Estado Inicial: (and (garbage) (cleanHands) (quiet))
- Meta: (and (dinner) (present) (not (garbage))
- Actiones:
  - Cook :precondition (cleanHands)
    :effect (dinner)
  - Wrap :precondition (quiet)
    :effect (present)
  - Carry :precondition
    :effect (and (not (garbage)) (not (cleanHands))
  - Dolly :precondition
    :effect (and (not (garbage)) (not (quiet)))

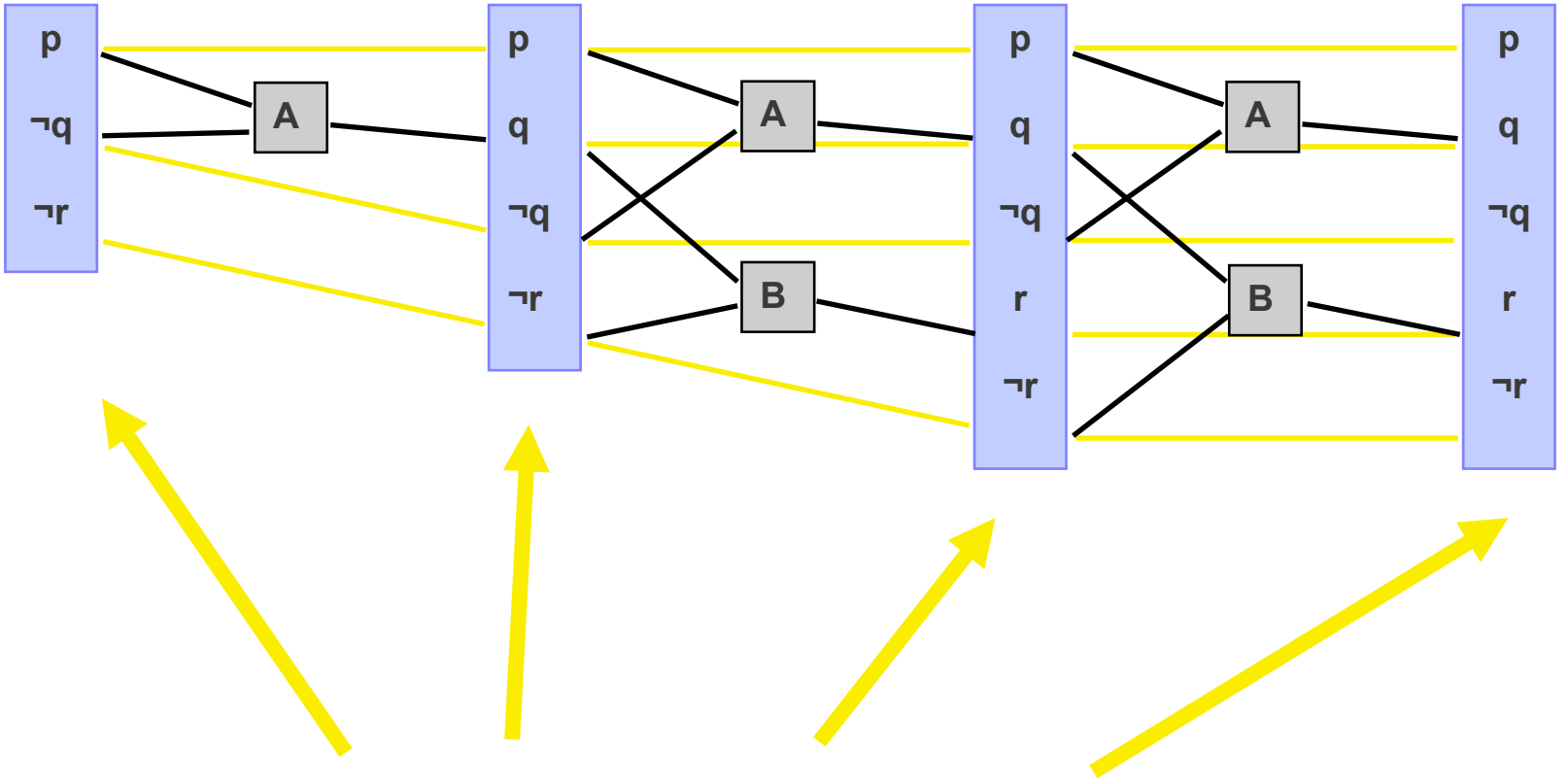
Graphplan: ejemplo simple



Graphplan: ejemplo simple

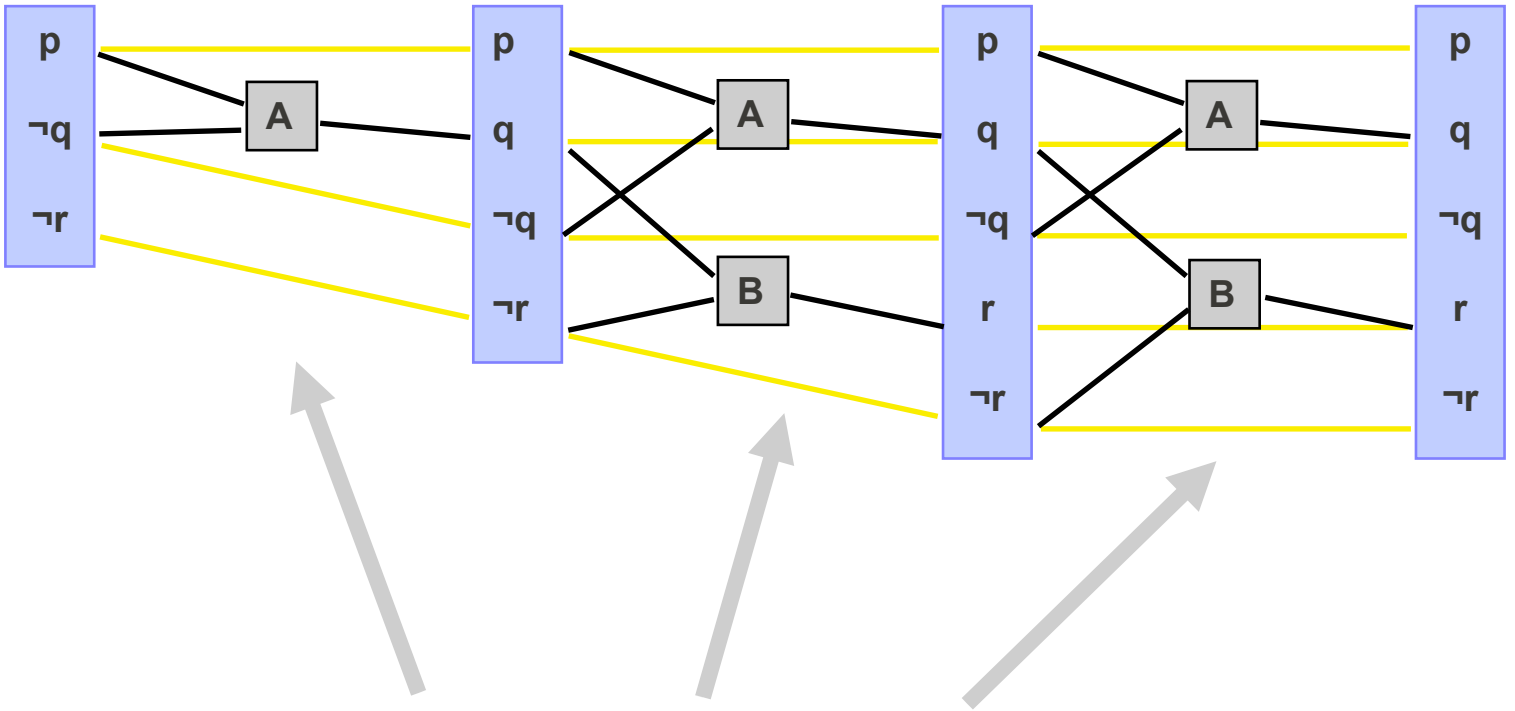


Graphplan: Observación 1



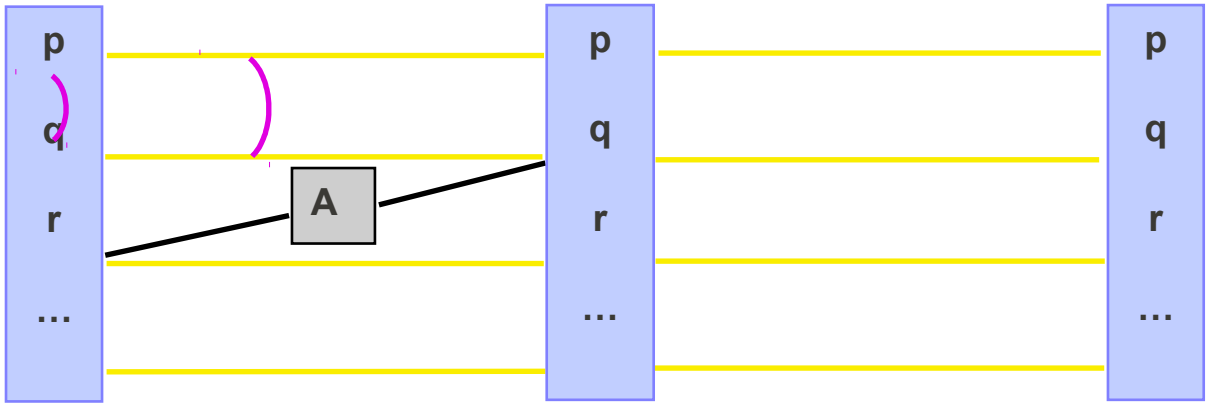
Las proposiciones crecen monotónicamente (siempre llevadas hacia adelante por los no-ops)

Graphplan: Observación 2



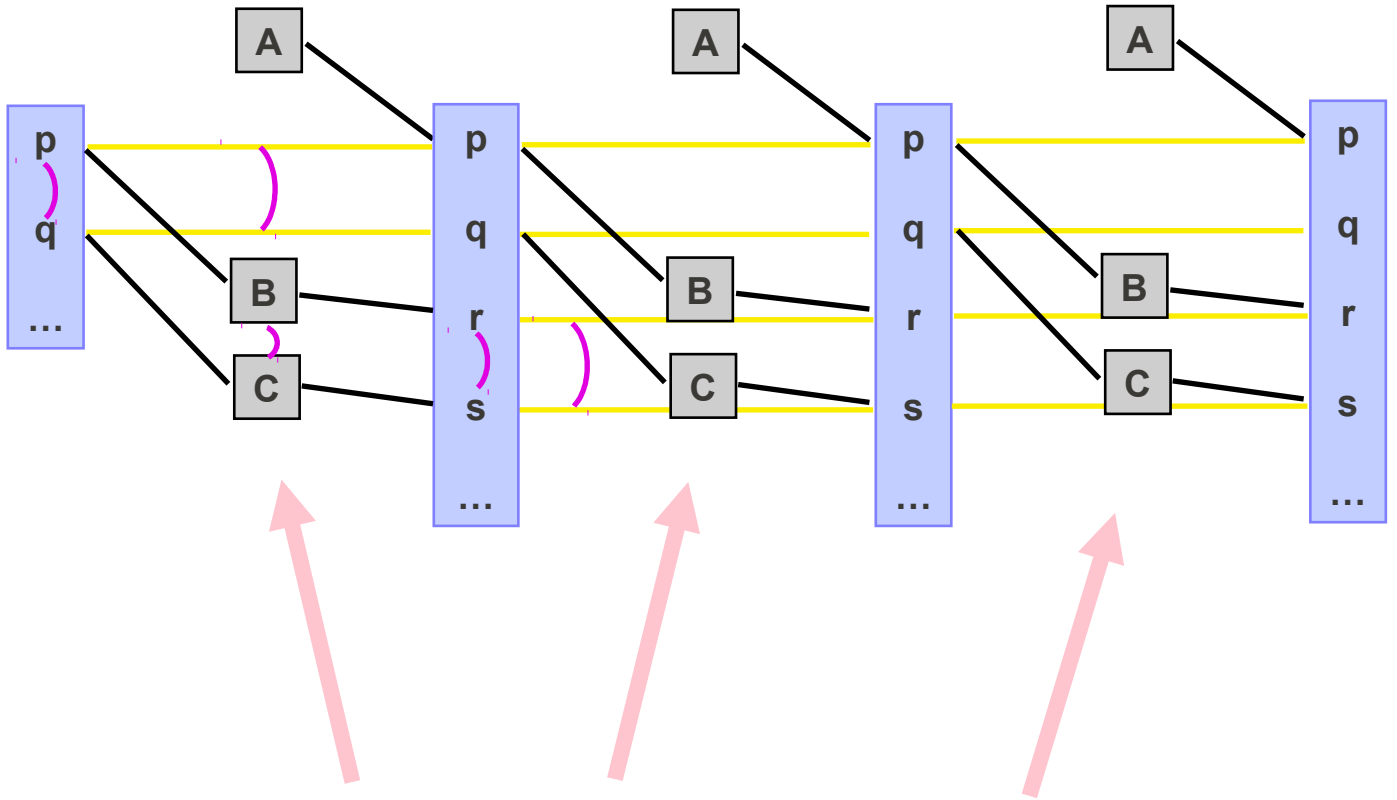
Las acciones crecen monotónicamente (dado que hay más proposiciones)

Graphplan: Observación 3



Los mutex entre literales decrecen monotónicamente (dado que hay más acciones)

Graphplan: Observación 4



Los mutex entre acciones decrecen monotónicamente (dado que hay menos mutex entre proposiciones)

Graphplan: Observación 5

El grafo de planning ‘levels off’:

- Se puede demostrar usando las 4 observaciones anteriores y el hecho de que los literales y las acciones posibles son finitios
- Después de un nivel k, todos los niveles subsiguientes serán idénticos
- Como el espacio es finito, el conjunto de literales nunca disminuye y los mutexes no reaparecen el algoritmo termina.

Graphplan: Extrayendo el plan

Un plan válido es un subgrafo del grafo de planning en el que:

- Las acciones de un mismo nivel no son mutex
- Todas las precondiciones de las acciones son hechas verdaderas por el plan
- Las metas se satisfacen

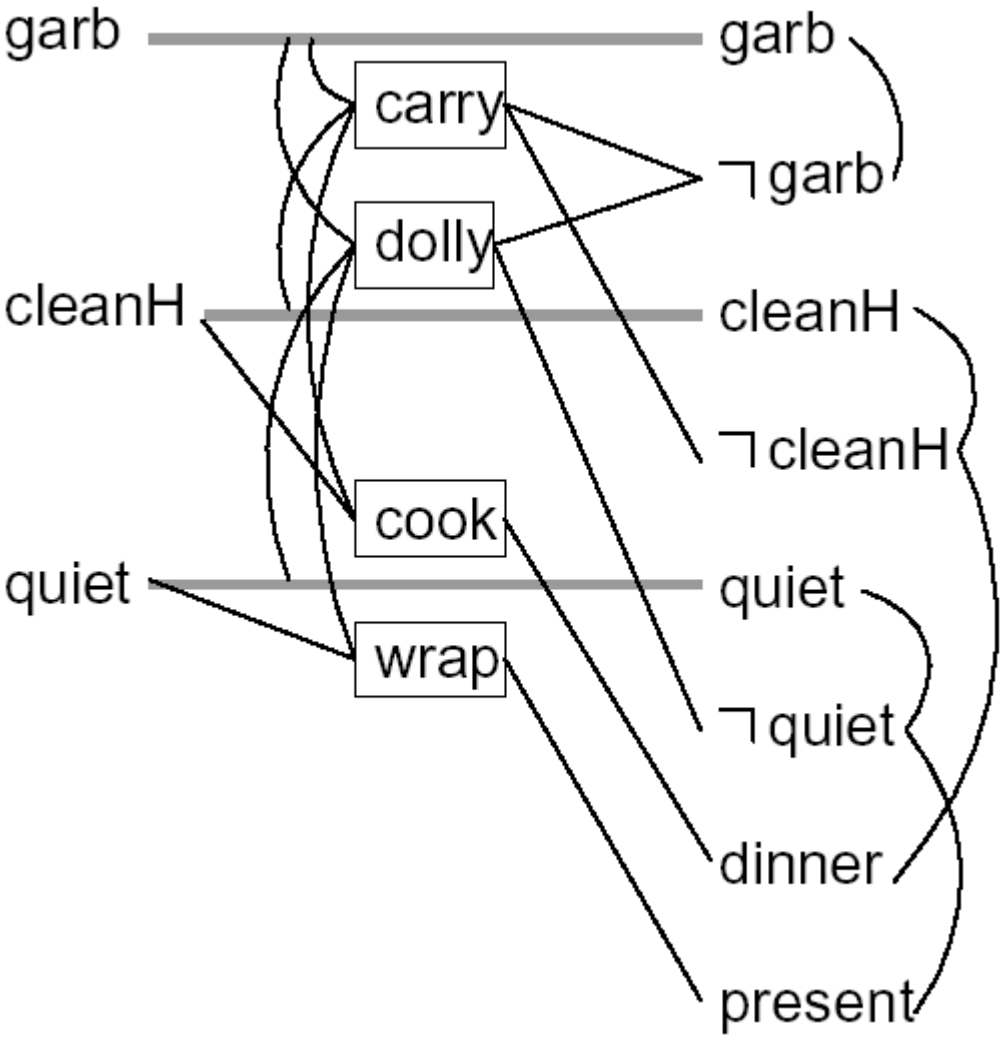
GraphPlan: idea del algoritmo

- Hacer crecer el grafo de planning hasta que se agreguen todas las metas a un nivel y estas sean no mutex entre si
- Si el grafo “levels off” antes de alcanzar las metas, fallar
- Buscar un plan válido en el grafo
- Si no se encuentra, agregar un nivel al grafo e intentar de nuevo

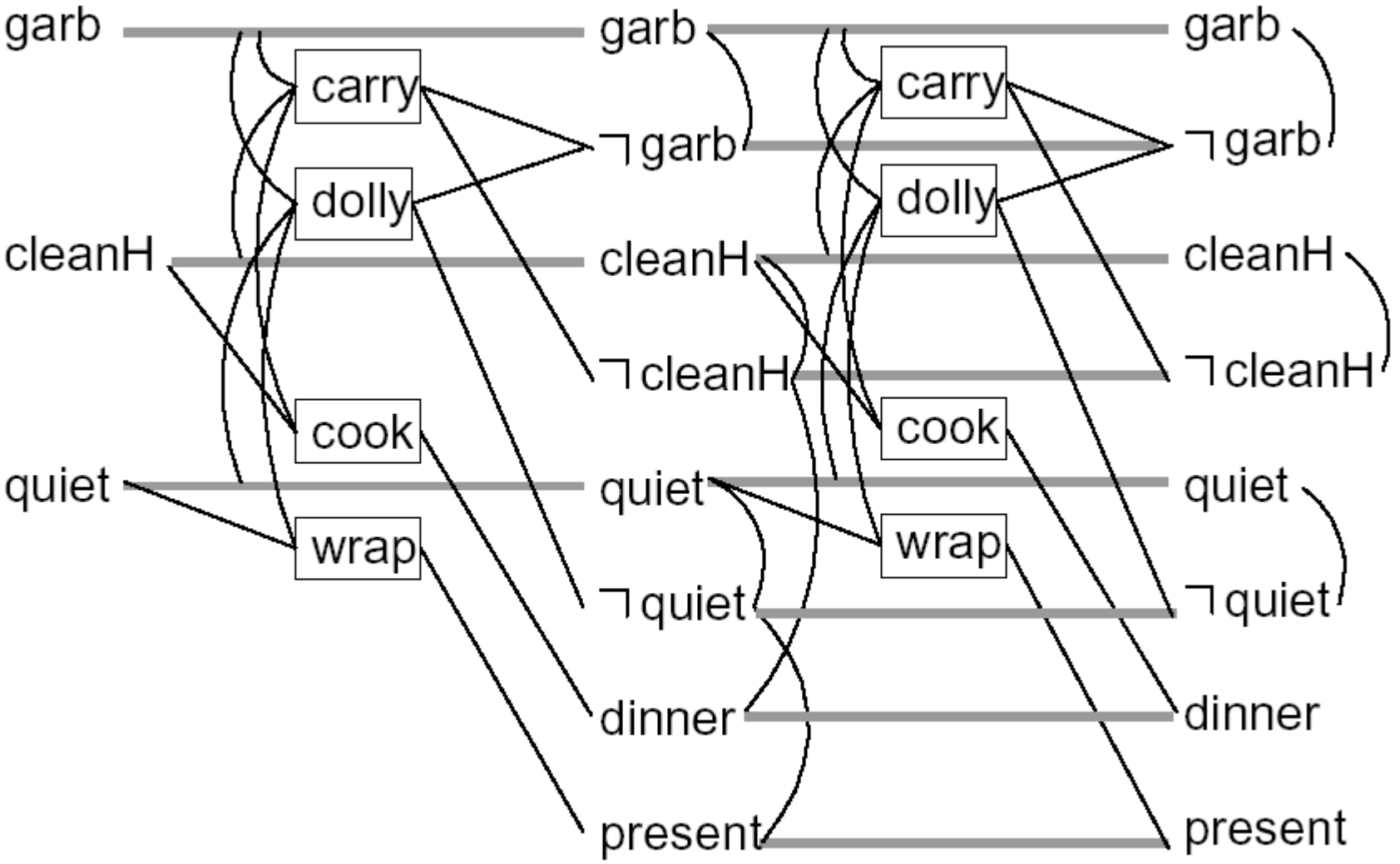
Graphplan: ejemplo simple

- Estado Inicial: (and (garbage) (cleanHands) (quiet))
- Meta: (and (dinner) (present) (not (garbage))
- Acciones:
  - Cook :precondition (cleanHands)
    :effect (dinner)
  - Wrap :precondition (quiet)
    :effect (present)
  - Carry :precondition
    :effect (and (not (garbage)) (not (cleanHands))
  - Dolly :precondition
    :effect (and (not (garbage)) (not (quiet)))

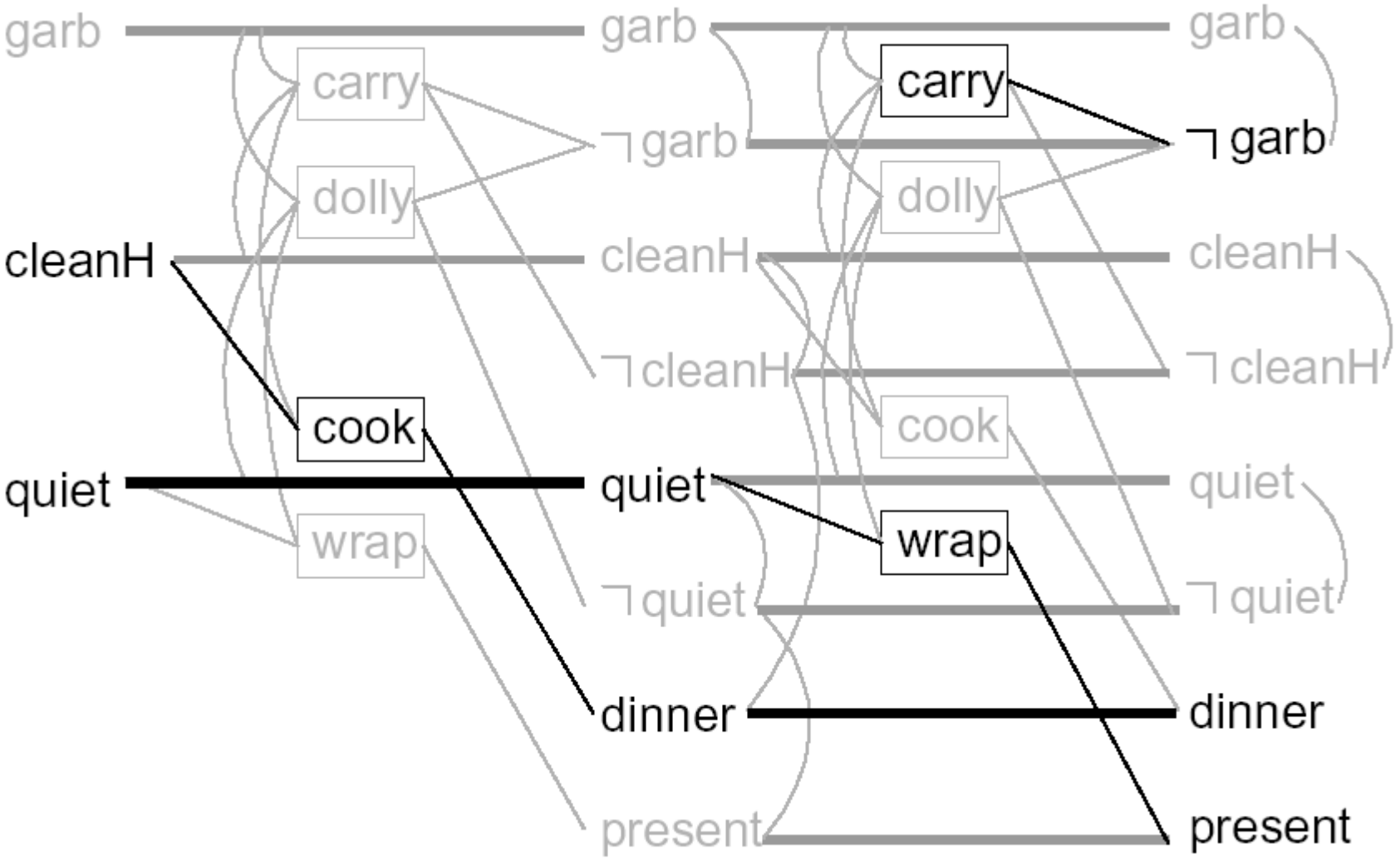
Graphplan: ejemplo simple



Graphplan: ejemplo simple



Graphplan: ejemplo simple

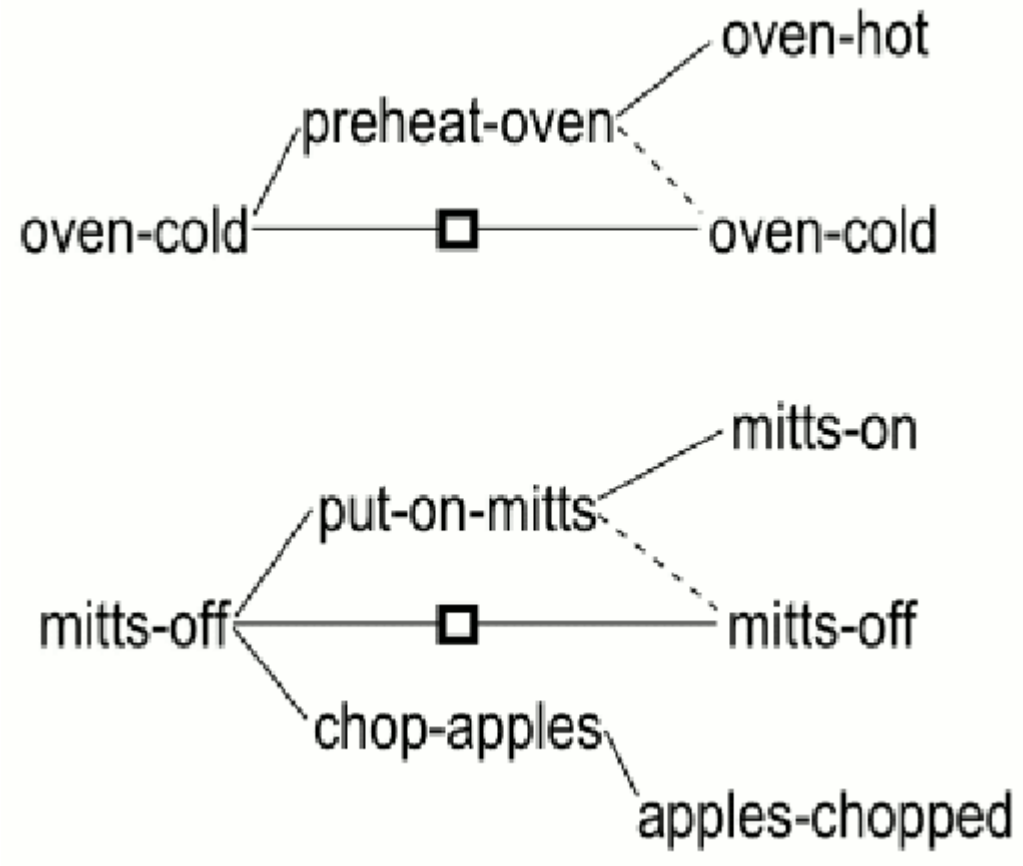


Graphplan: Buscando el plan

- “Backward chain” en el grafo de planning
- En el nivel k, elegir un subconjunto de acciones no mutex que alcance las metas actuales. Sus precondiciones se transforman en las metas del nivel k-1
  - Tomar cada meta del conjunto de metas y elegir una acción para agregar. Usar una acción ya seleccionada de ser posible.
- Si no se puede encontrar un subconjunto de acciones no mutex que alcance todas las metas de ese nivel, hacer backtracking y probar con un conjunto no probado en otro punto de decisión

Graphplan: ejemplo complejo

- Algunas acciones



- La meta

Graphplan: ejemplo complejo

Graphplan: complejidad

- El algoritmo de construcción del grafo es polinomial en:
  - t = número de niveles
  - n = número de objetos
  - m = número de operadores
  - p = proposiciones en el estado inicial
- Sabemos que planning es PSPACE-hard asi que la búsqueda del plan en el grafo tiene que ser PSPACE-hard

Graphplan: Resumen

- Graphplan fue un algoritmo muy influencial que lideró el área de AI planning por varios años
- Sin embargo, en los últimos años fue superado por una búsqueda heurística
- Esta búsqueda usa la construcción de un grafo de planning para calcular su heurística

Qué vamos a hacer hoy?

- Introducción a los entornos virtuales
- Herramientas para determinación del contenido
  - Introducción a planning, planning vs búsqueda
  - Representación de un problema de planning
  - El algoritmo de graphplan
  - Búsqueda en el espacio de estados y heurísticas
- GIVE World: Un mundo virtual simple
  - Generación de instrucciones en entornos virtuales

Planning como búsqueda 2

- Como ya vimos, planning puede ser resulto como un problema de búsqueda
- La KB inicial es el estado inicial
- Las acciones son operaciones que mapean un estado en un nuevo estado
- La meta se satisface cuando encontramos un estado que la incluye

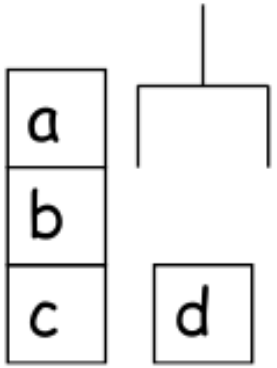
Problemas

- El árbol de búsqueda es muy grande
- Reconfigurar pocos bloques con búsqueda ciega puede llevar minutos u horas
- La búsqueda debería aprovechar la estructura del problema de planning para computar heurísticas de búsqueda.
- Cada acción sólo afecta un pequeño conjunto de hechos, y las acciones dependen entre ellas por sus precondiciones

Graphplan como heurística

- La parte más compleja de graphplan es buscar por un plan porque los mutex causan backtraking y extension del grafo
- El planner Fast Forward (que ganó todas las competencias de planning desde el 2002) usa graphplan como heurística
- Cuenta la cantidad de acciones en un plan del grafo
- Para contar las acciones de un plan necesitamos encontrarlo
- Calcular la heurística sería tan difícil como encontrar un plan
- La segunda idea detras de FF es usar graphplan sobre problemas en los cuales se ignora la lista de del.

Ignorando la lista de deletes

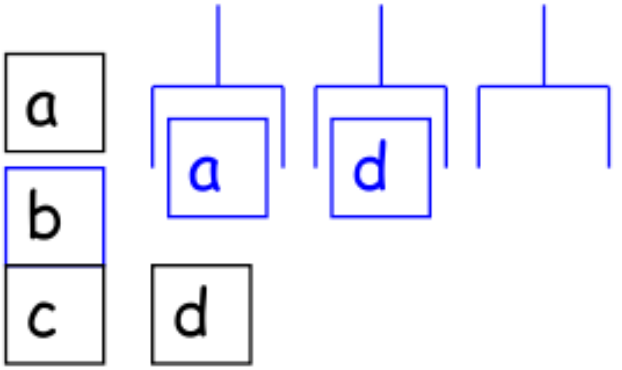


on(a,b),
on(b,c),
ontable(c),
ontable(d),
clear(a),
clear(d),
handempty

S_0

unstack(a,b)
pickup(d) s

A_0



on(a,b),
on(b,c),
ontable(c),
ontable(d),
clear(a),
handempty,
clear(d),
holding(a),
clear(b),
holding(d)

S_1

this is not a state!

Graphplan en problemas relajados

- En un problema relajado, las acciones agregan nuevos hechos pero nunca los borran
- Por lo tanto un grafo relajado no contiene mutex, no hay backtracking ni expansiones del grafo al buscar un plan
- Una vez encontrado un plan usamos el número de acciones del plan como heurística de la distancia del estado actual a la meta
- Encontrar un plan es mucho más eficiente pero el método no es completo (e.g. no es una heurística admisible para metas con literales negativos)

Determinación del contenido

- Planning es una forma eficiente de encontrar una secuencia de “mensajes a transmitir”
- La secuencia de mensajes mantiene una coherencia causal
- Las acciones (del mundo modelado) codifican información causal del mundo
- Esta estructura causal tiene un efecto determinante en la estructura de las instrucciones
  - No puedo pedirte que saques la basura antes de cocinar si no hay forma de lavarse las manos

Qué vamos a hacer hoy?

- Introducción a los entornos virtuales
- Herramientas para determinación del contenido
  - Introducción a planning, planning vs búsqueda
  - Representación de un problema de planning
  - El algoritmo de graphplan
  - Búsqueda en el espacio de estados y heurísticas
- GIVE World: Un mundo virtual simple
  - Generación de instrucciones

El mundo GIVE: Introducción

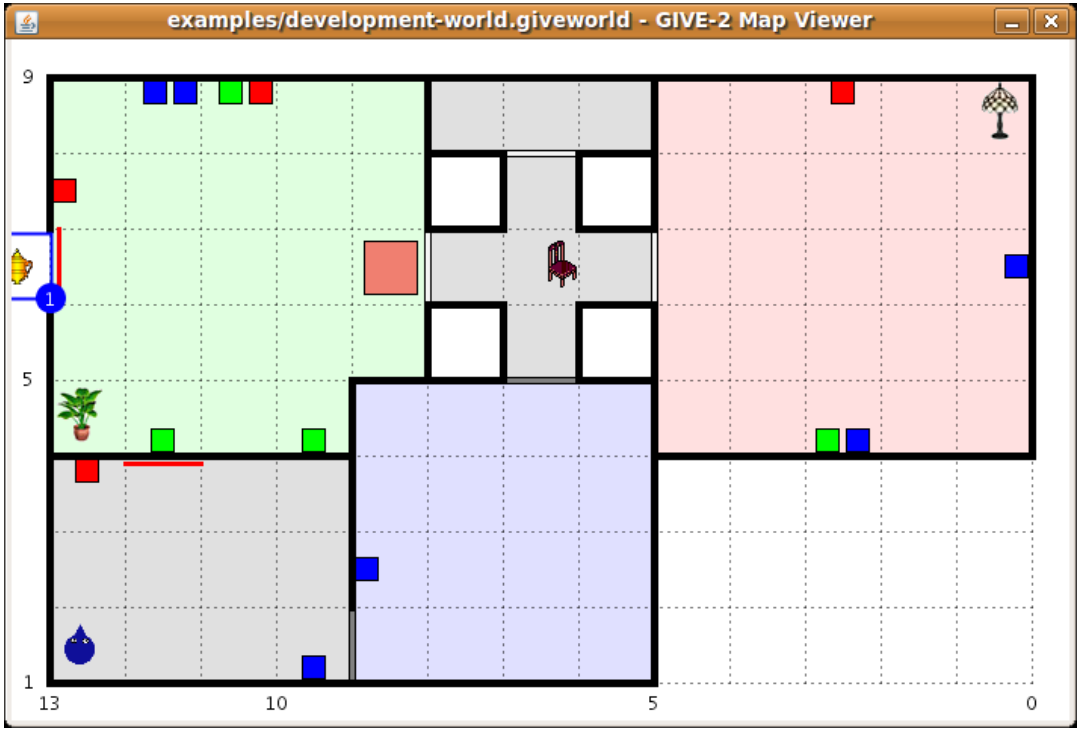
- La meta es variable, ej, encontrar un trofeo
- El usuario puede:
  - apretar botones,
  - abrir puertas,
  - desactivar alarmas, etc.
- El mundo representa propiedades estáticas (ej, color) y dinámicas de objetos (ej, visibilidad).

El mundo GIVE

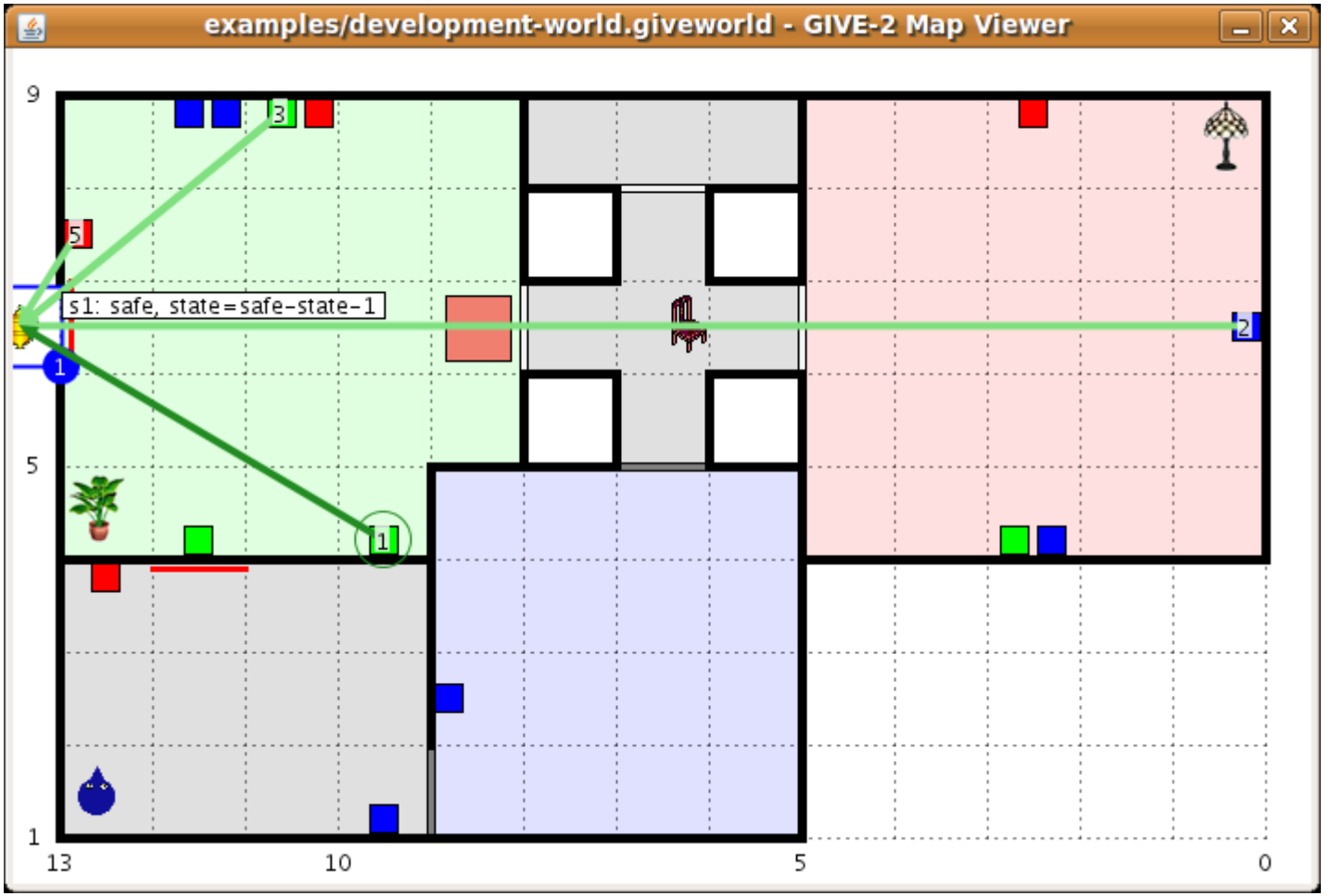
El mundo GIVE: Demo 3D

El mundo GIVE: Diseñando mapas

- Los mundos GIVE pueden diseñarse con distintas caracteristicas
- El mapa puede visualizarse
- Y las propiedades de los objetos

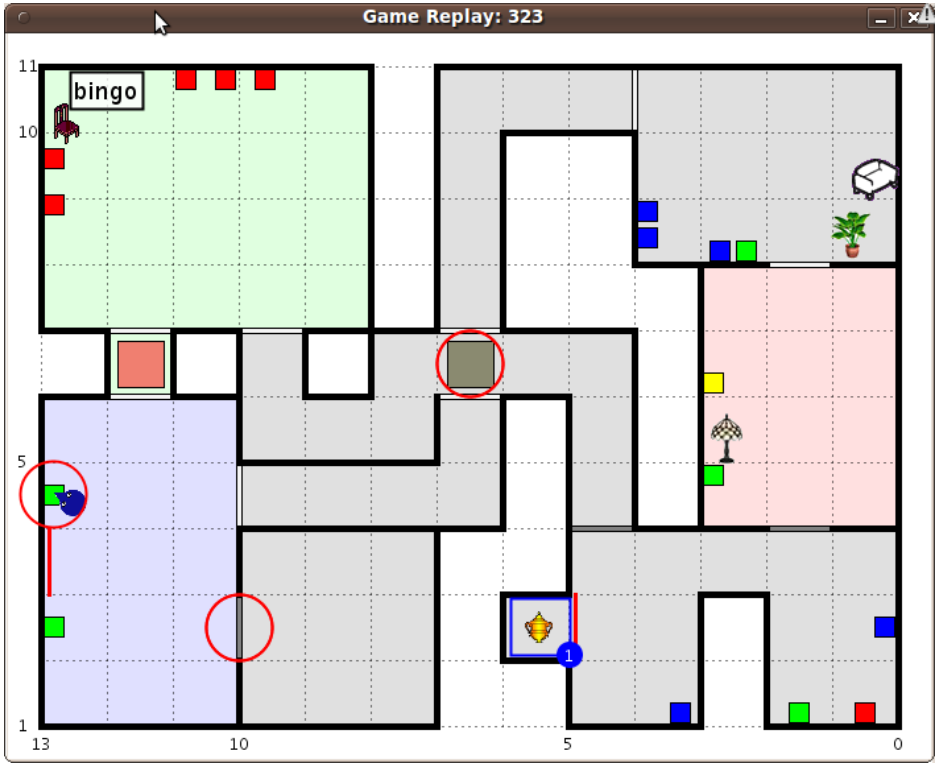


El mundo GIVE: Demo mapas



El mundo GIVE: Acciones

- El usuario puede apretar botones
- Y como efecto colateral:
  - Desactivar alarmas
  - Abrir puertas
- Si el usuario pisa una alarma activada pierde

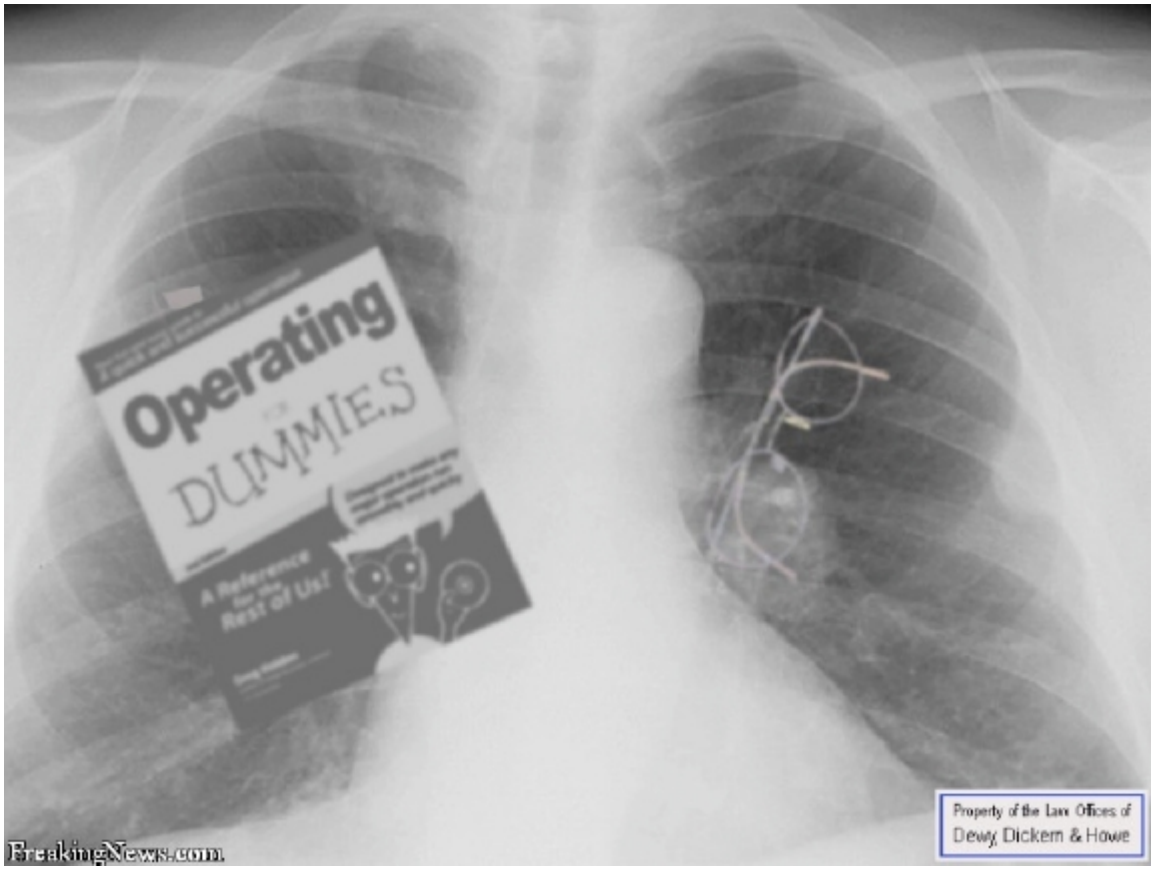


Qué vamos a hacer hoy?

- Introducción a los entornos virtuales
- Herramientas para determinación del contenido
  - Introducción a planning, planning vs búsqueda
  - Representación de un problema de planning
  - El algoritmo de graphplan
  - Búsqueda en el espacio de estados y heurísticas
- GIVE World: Un mundo virtual simple
  - Generación de instrucciones

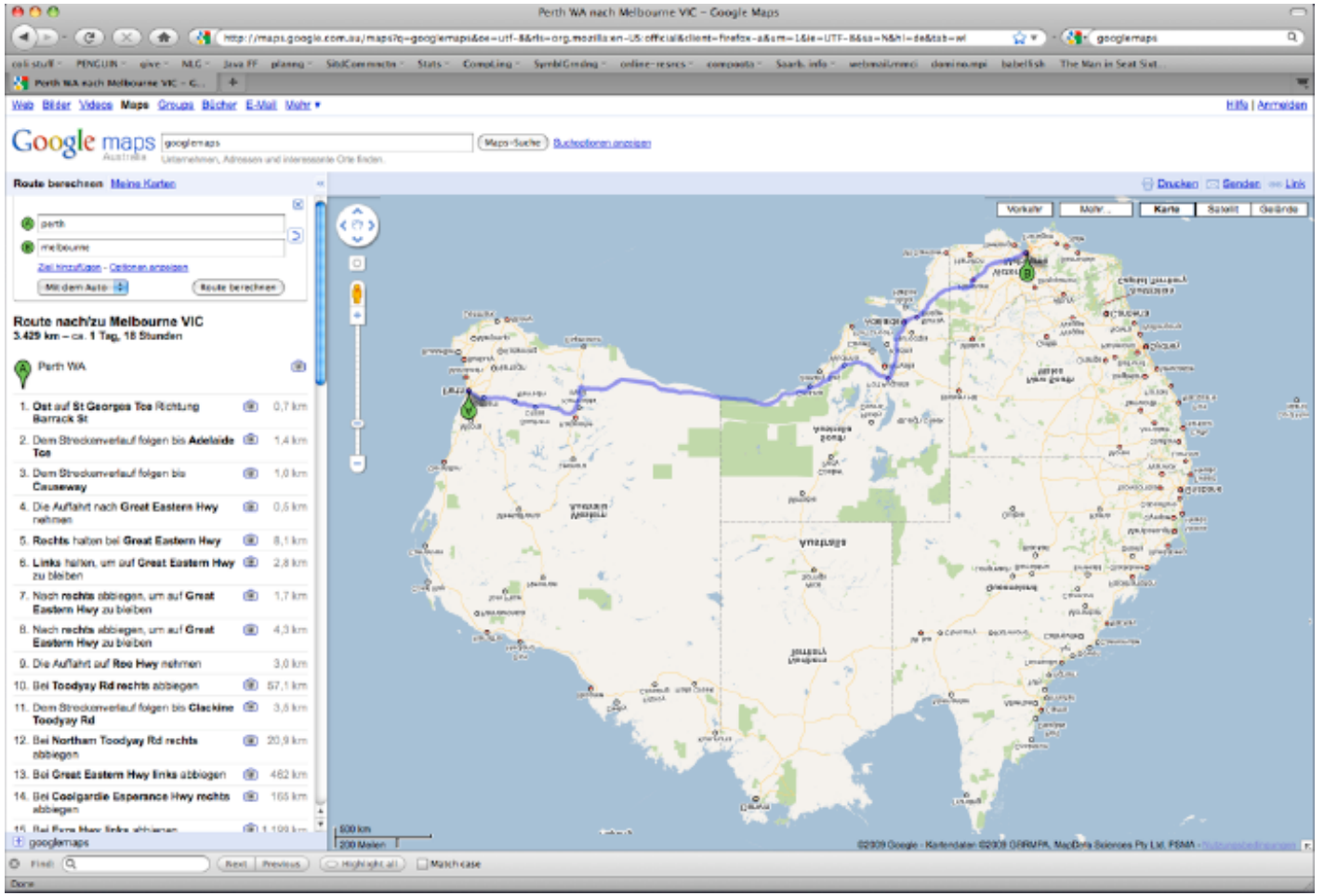
Formas de dar instrucciones

- Manuales de instrucciones



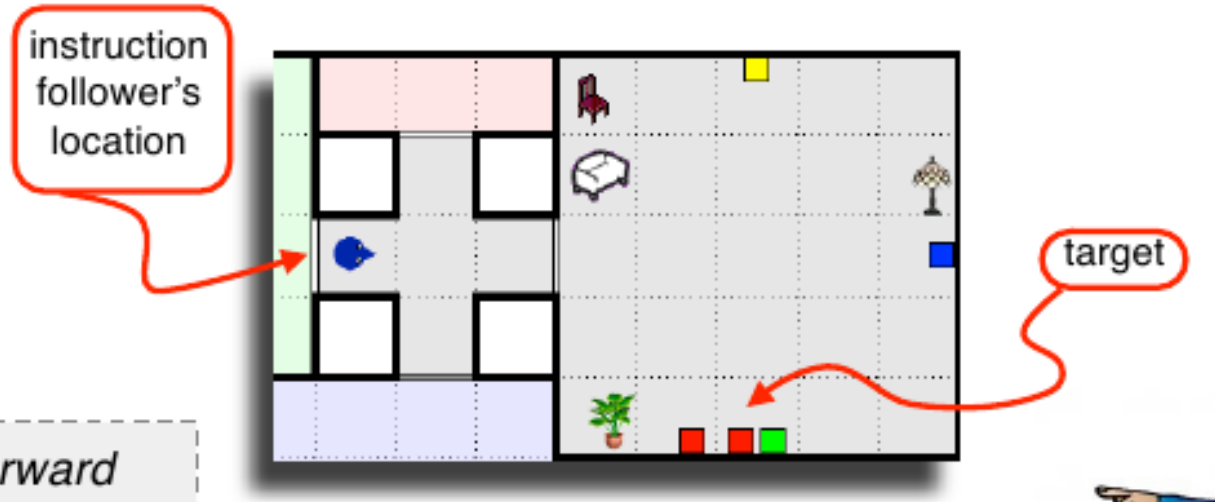
Formas de dar instrucciones

- Mapas de direcciones

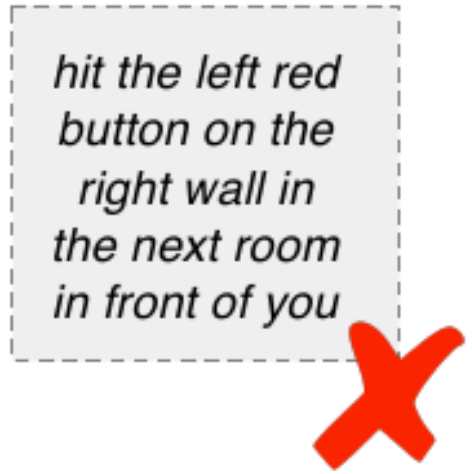


Formas de dar instrucciones

Dando instrucciones colaborativas



1. *walk forward and go through the door*
2. *(...) ok and then turn to your right*
3. *and then hit the left red button*



hit the left red button on the right wall in the next room in front of you