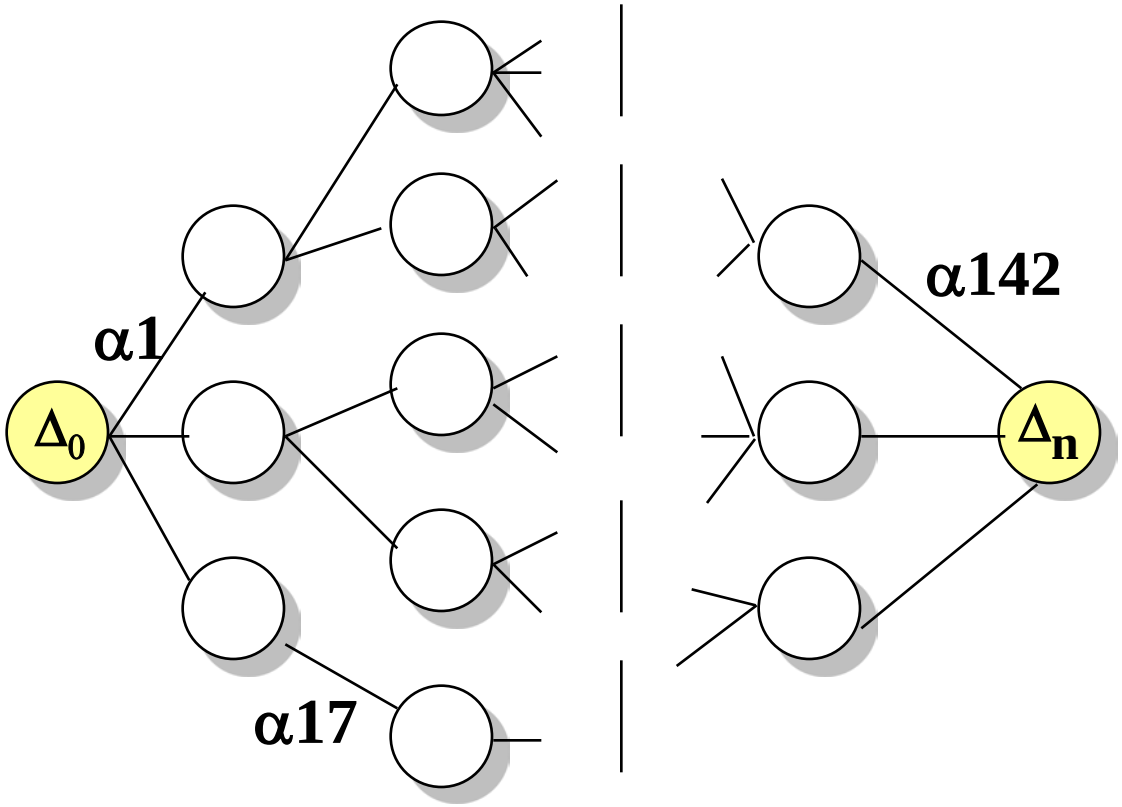
Aproximaciones a la planificación automática

- 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)
- La competición internacional de planificación (ICAPS)

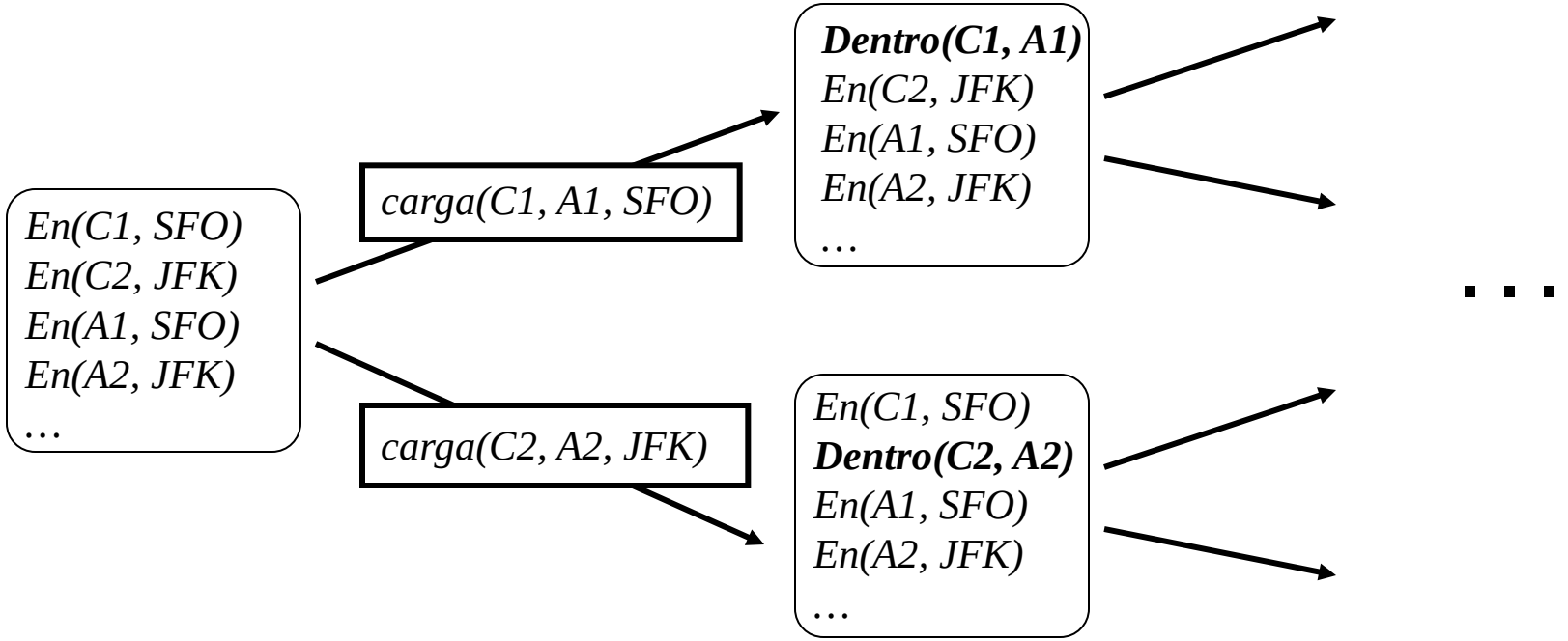
Planificación en el espacio de estados (State-space)

- Idea:
  - cada nodo representa un estado del mundo
  - El estado del mundo se define mediante un conjunto de predicados y variables
  - Un plan es un camino dentro del espacio de estados



Estrategias de planificación en espacio de estados(1)

- **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) → Memory Bound A*
- 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

Heurísticos en planificación progresiva determinista

- Durante muchos años los investigadores en planificación han buscado algoritmos generales, totalmente independientes del dominio
  - Las implementaciones heurísticas (Greedy, A*) usaban heurísticos que se calculaban automáticamente a partir de, por ejemplo, un grafo de los operadores y sus dependiencias (GRAPHPLAN).
- Problema: ¿Cómo hacer una búsqueda heurística estilo A* sin incluir conocimiento del dominio?
  - Durante varios años nadie consiguió encontrar una buena función *h*
  - Solución: heurísticos que se calculaban automáticamente a partir de un grafo de los operadores y sus dependiencias (GRAPHPLAN).
- Ej: FastForward [Hoffmann]

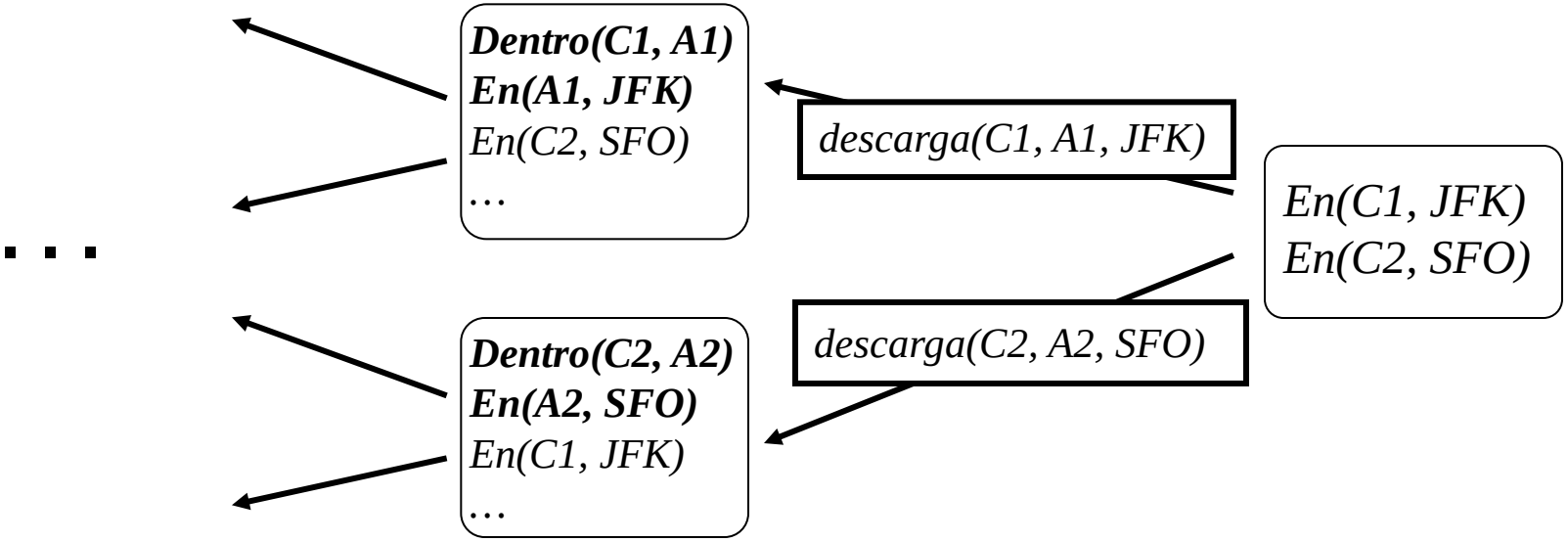
Problemas en la planificación progresiva determinista

- 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
  - Lo veremos más adelante.

Estrategias de planificación en espacio de estados (2)

- **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



Planificación regresiva: teoria subyacente (I)

- En la búsqueda hacia adelante, se empezaba en el estado inicial y se computaban transiciones de estados
  - nuevo estado $s' = \gamma(s,a)$
- En la búsqueda hacia atrás, empezamos en el objetivo y se computan transiciones inversas de estados
  - Nuevo conjunto de subobjetivos $g' = \gamma^{-1}(g,a)$

Planificación regresiva: teoria subyacente (II)

- Para definir $\gamma^{-1}(g,a)$, hemos de definir primero el concepto de *relevancia*:
  - Una acción a es relevante para un objetivo g si
    - a hace al menos uno de los literales de g cierto
      - $g \cap \text{efectos}(a) \neq \varnothing$
    - a no hace falso ninguno de los literales de g
      - $g^+ \cap \text{efectos}^-(a) = \varnothing$ y $g^- \cap \text{efectos}^+(a) = \varnothing$

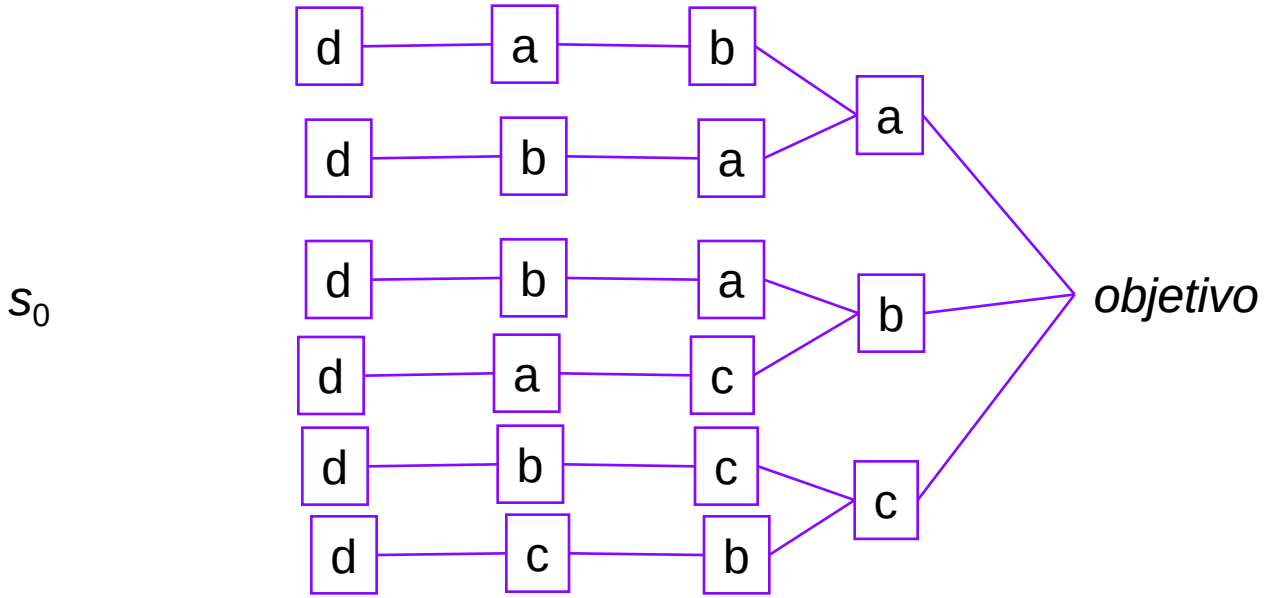
- Def: si a es relevante para g, entonces
  - $\gamma^{-1}(g,a) = (g - \text{efectos(a)}) \cup \text{precond}(a)$

  sino $\gamma^{-1}(g,a)$ esta indefinido

    - Ej: en el caso
      - g = {on(b1,b2), on(b2,b3)}
      - a = stack(b1,b2)
    - ¿Cual seria $\gamma^{-1}(g,a)$?

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, b *y* c independientes, una acción d que las ha de preceder siempre, y que no hay ningun camino desde s_0 al estado necesario como input de d
  - El algoritmo intenta todas las ordenaciones posibles de a, b y c antes de darse cuenta que no hay solución.

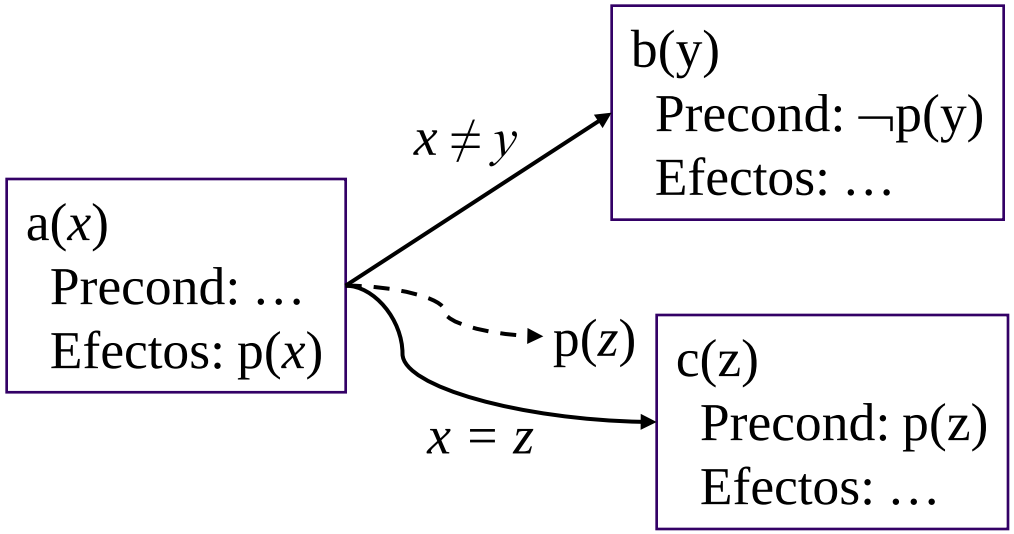


Planificación en el espacio de planes (Plan-space)

- 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.

Plan-Space Planning: restricciones

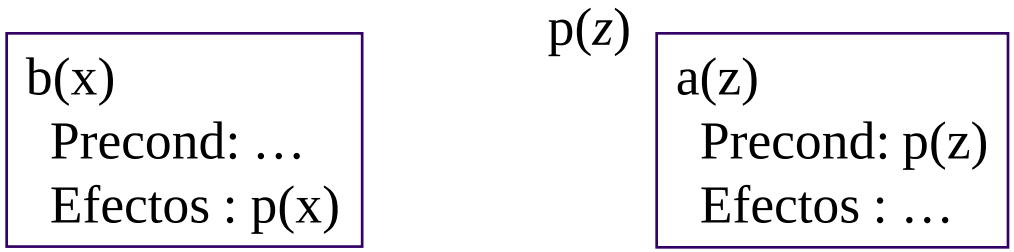
- 3 tipos de restricciones:
  - *restricciones de precedencia*: *a* debe preceder a *b*
  - *restricciones de asignación*:
    - restricciones de desigualdad: x ≠ *y*
    - Restricciones de igualdad: x = *z*
  - *enlaces causales (causal links)*:
    - usar la acción *a* para obtener la precondición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



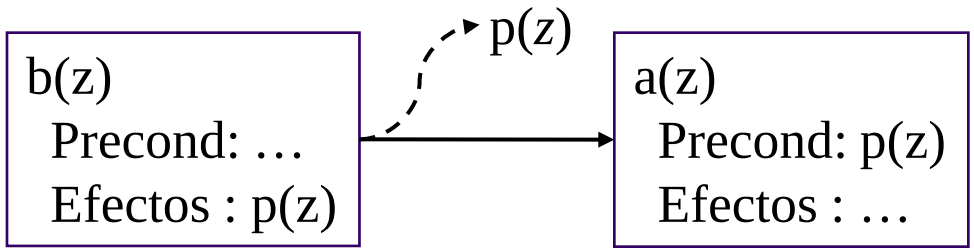
[Ejemplo de Dana Nau en Lecture slides for *Automated Planning.*]

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

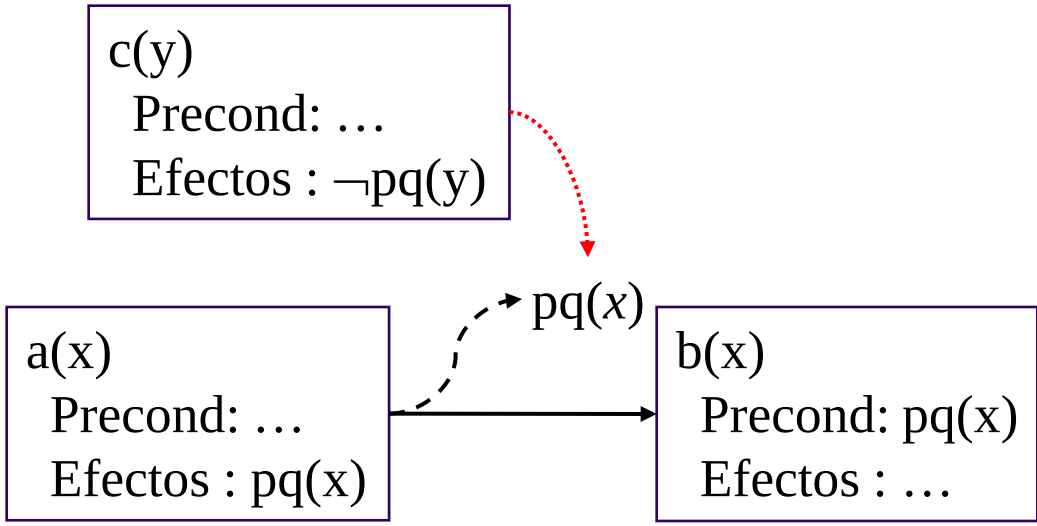


[Ejemplos de Dana Nau en Lecture slides for *Automated Planning*.]

Plan-Space Planning: Fallos - 2. Ataques

- Ataque: una interacción que elimina condiciones
  - la acción *a* genera la precondition (e.g., pq(*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*



[Ejemplo de Dana Nau en Lecture slides for *Automated Planning.*]

El algoritmo PSP

```
PSP(π)
    flaws ← OpenGoals(π) ∪ Threats(π)
    if flaws = ∅ then return(π)
    select any flaw φ ∈ flaws
    resolvers ← Resolve(φ, π)
    if resolvers = ∅ then return(failure)
    nondeterministically choose a resolver ρ ∈ resolvers
    π' ← Refine(ρ, π)
    return(PSP(π'))
end
```

- PSP es completo
- Devuelve un pla ordenado parcialmente
  - Cualquier orden total del plan satisfacerá los objetivos
  - Una ejecución paralela que mantenha el orden parcial también cumplirá los objetivos.

Ejemplo 1

Start y **Finish** son acciones ficticias que usaremos en vez del estado inicial y el objetivo

- Operadores:
  - **Start**

    Precond: none

    Effects: At(Home), sells(HWS,Drill), Sells(SM,Milk), Sells(SM,Banana)
  - **Finish**

    Precond: Have(Drill), Have(Milk), Have(Banana), At(Home)
  - **Go(*l*,*m*)**

    Precond: At(*l*)

    Effects: At(*m*), ¬At(*l*)
  - **Buy(*p*,*s*)**

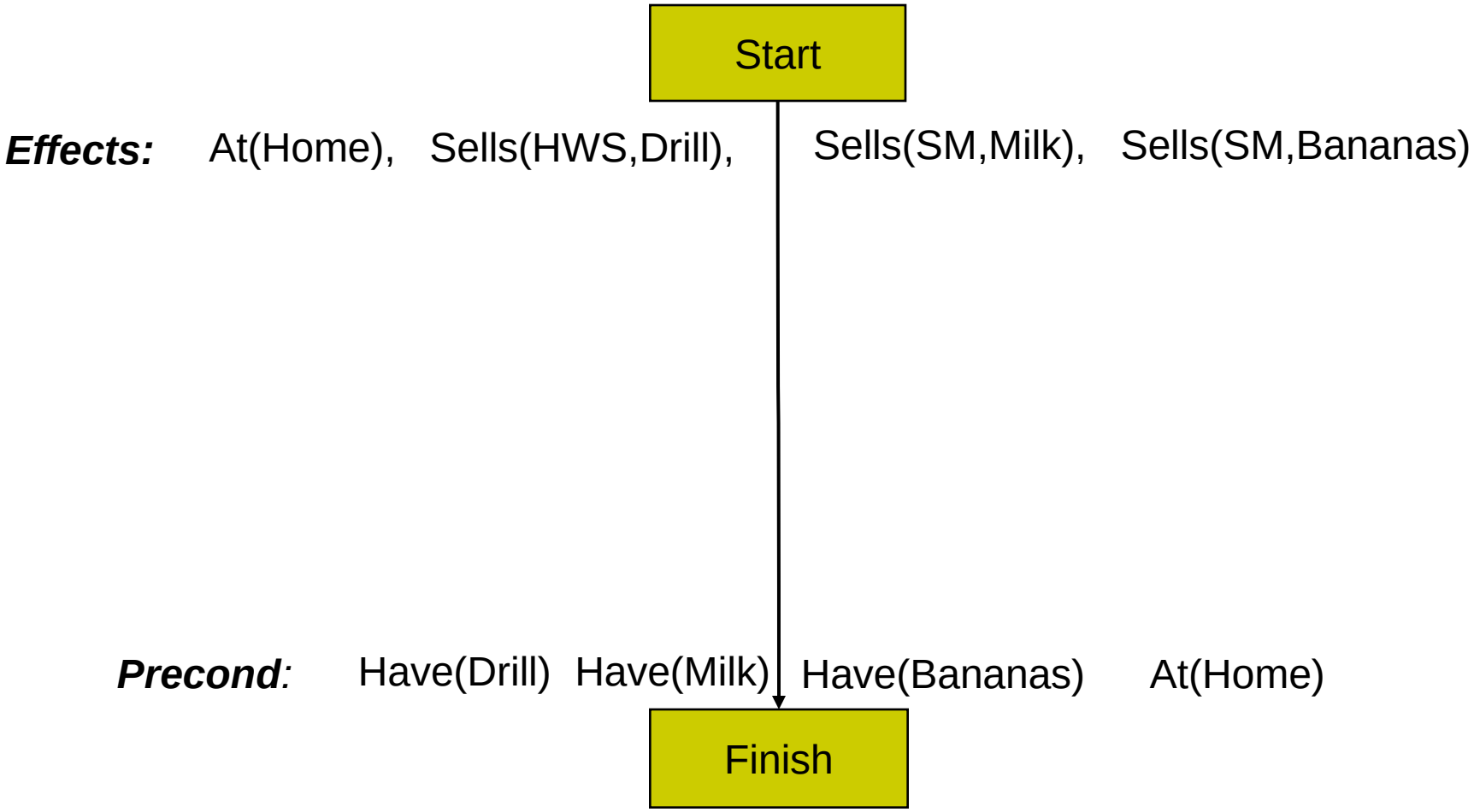
    Precond: At(*s*), Sells(*s*,*p*)

    Effects: Have(*p*)

[Ejemplo de Dana Nau en Lecture slides for *Automated Planning.*]

Ejemplo 1 (cont)

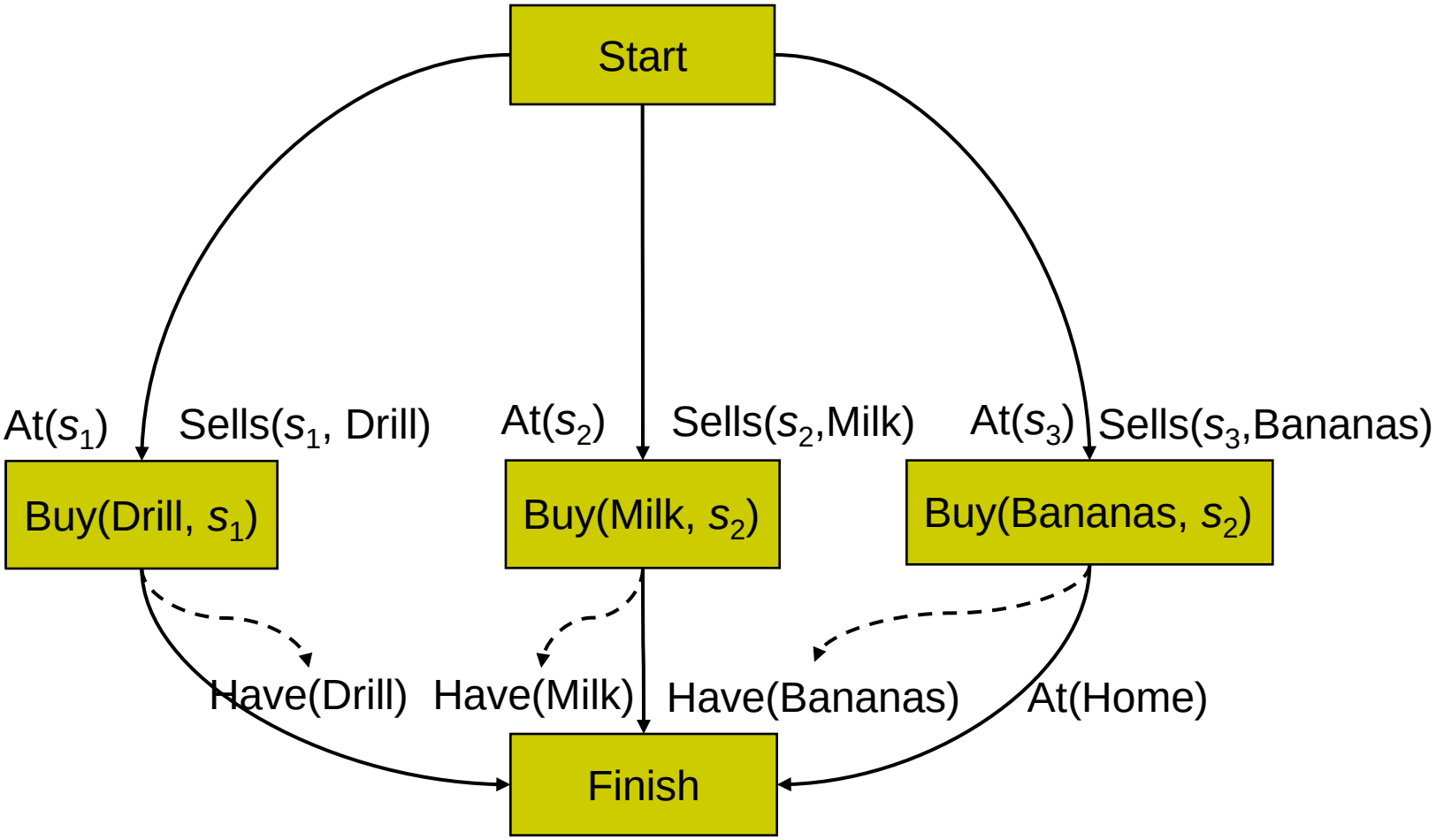
- Le damos a PSP un plan inicial π: **Start**, **Finish** y una restricción de orden.



[Ejemplo de Dana Nau en Lecture slides for *Automated Planning.*]

Ejemplo 1 (cont)

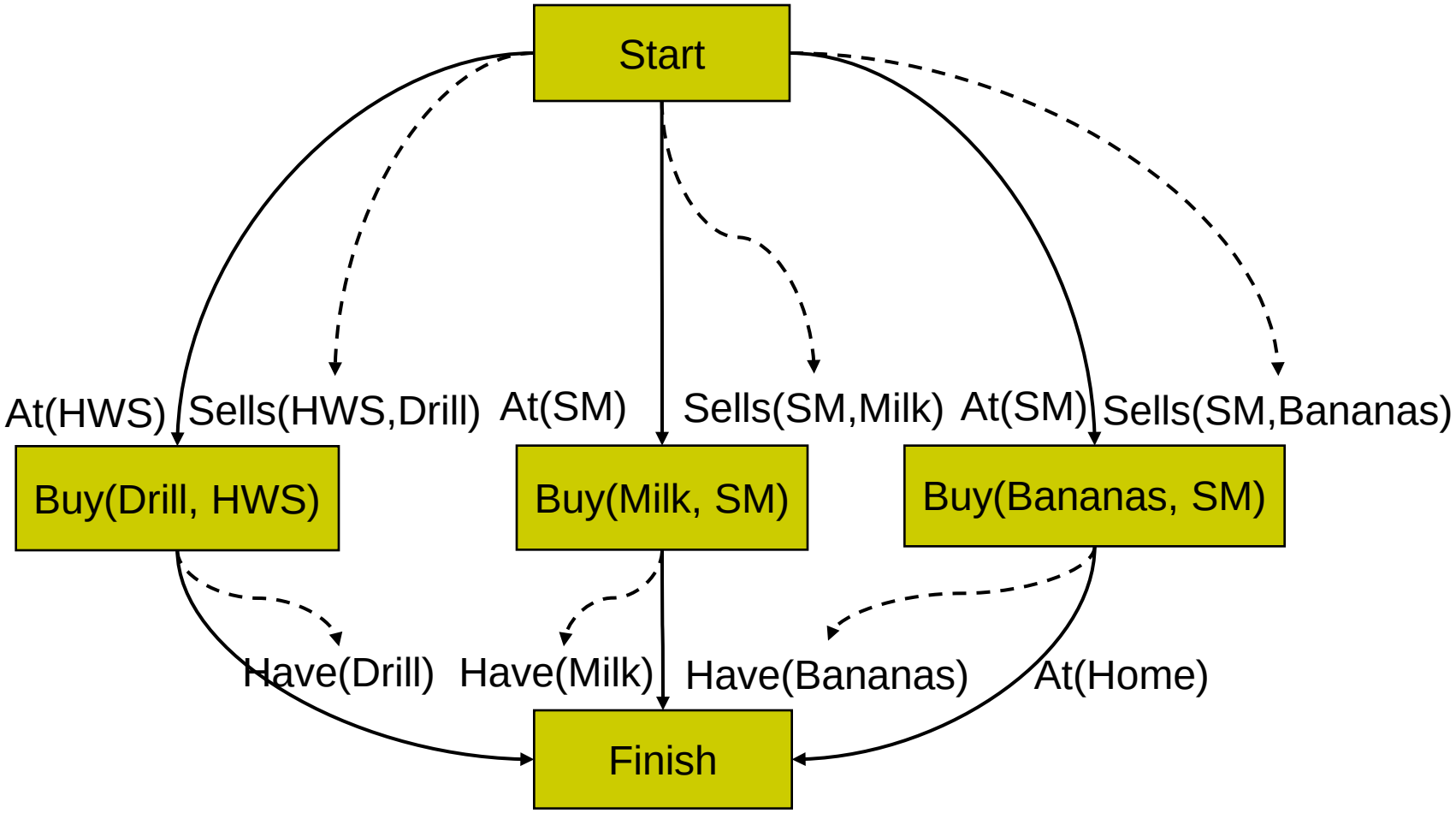
- Primeros tres refinamientos: las únicas formas de obtener las precondiciones de los Have



[Ejemplo de Dana Nau en Lecture slides for *Automated Planning.*]

Ejemplo 1 (cont)

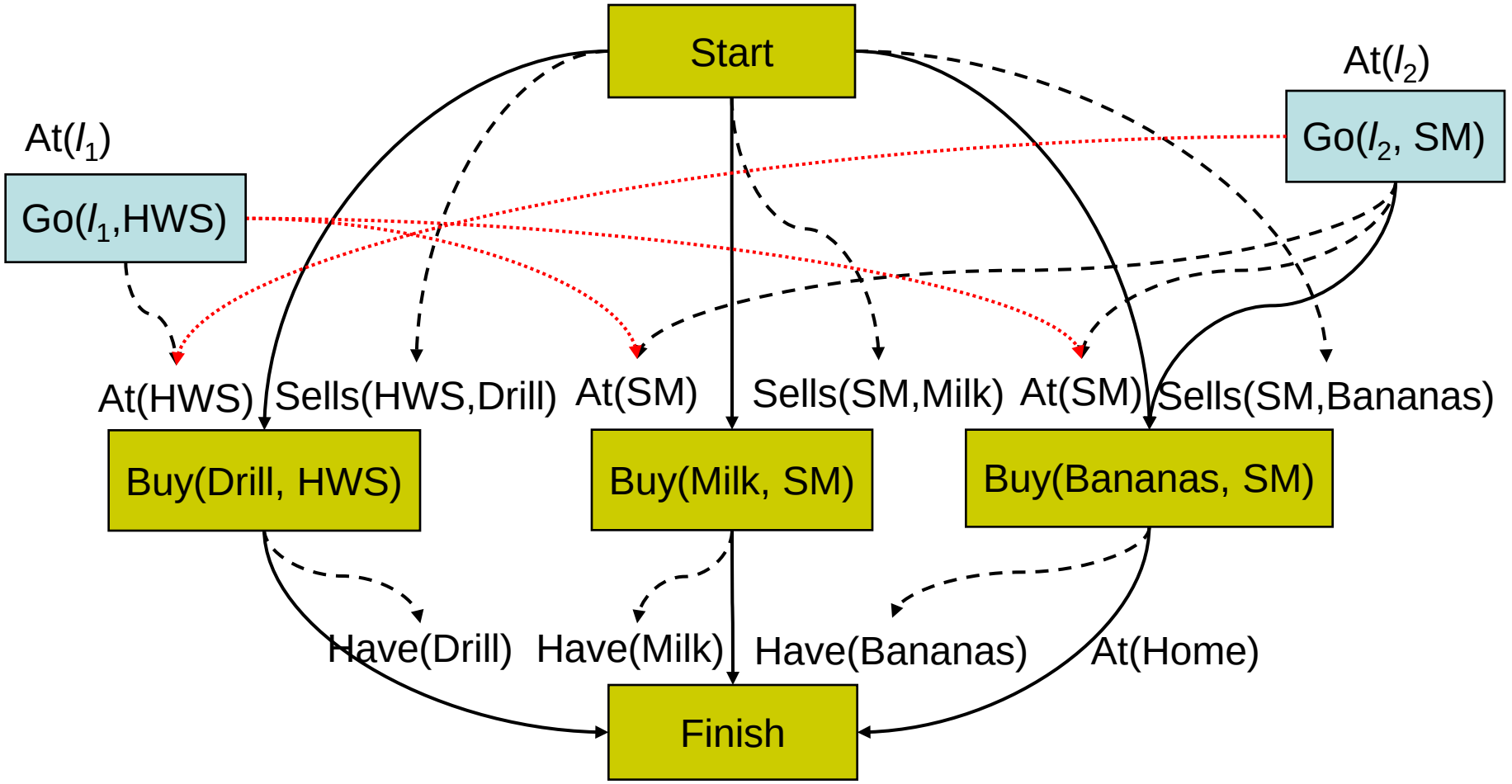
- Tres refinamientos más: las únicas formas de obtener las percondiciones de los Sells



[Ejemplo de Dana Nau en Lecture slides for *Automated Planning.*]

Ejemplo 1 (cont)

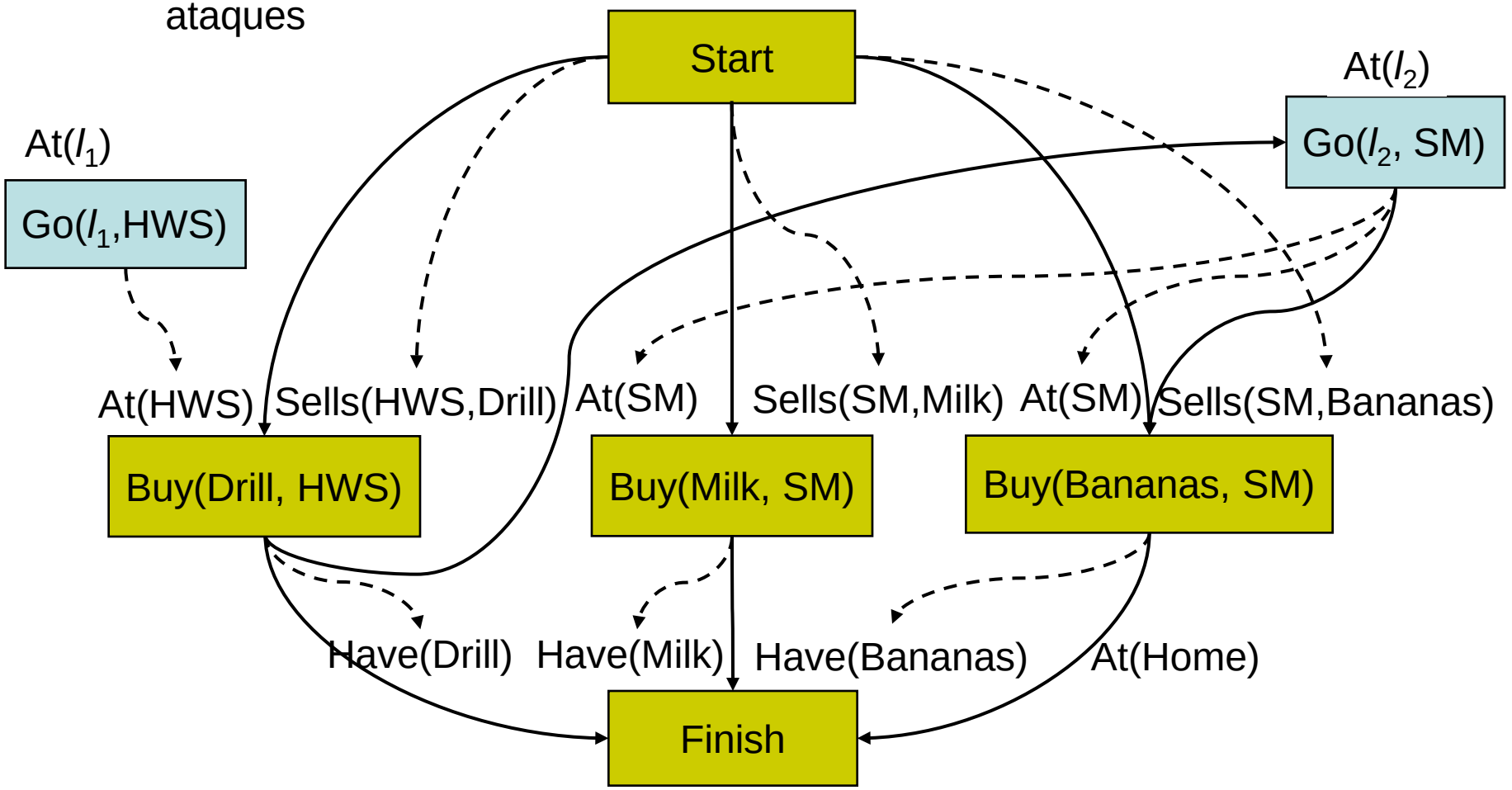
- Dos refinamientos más: las únicas formas de obtener At(HWS) y At(SM)
  - Esta vez aparecen varios ataques



[Ejemplo de Dana Nau en Lecture slides for *Automated Planning.*]

Ejemplo 1 (cont)

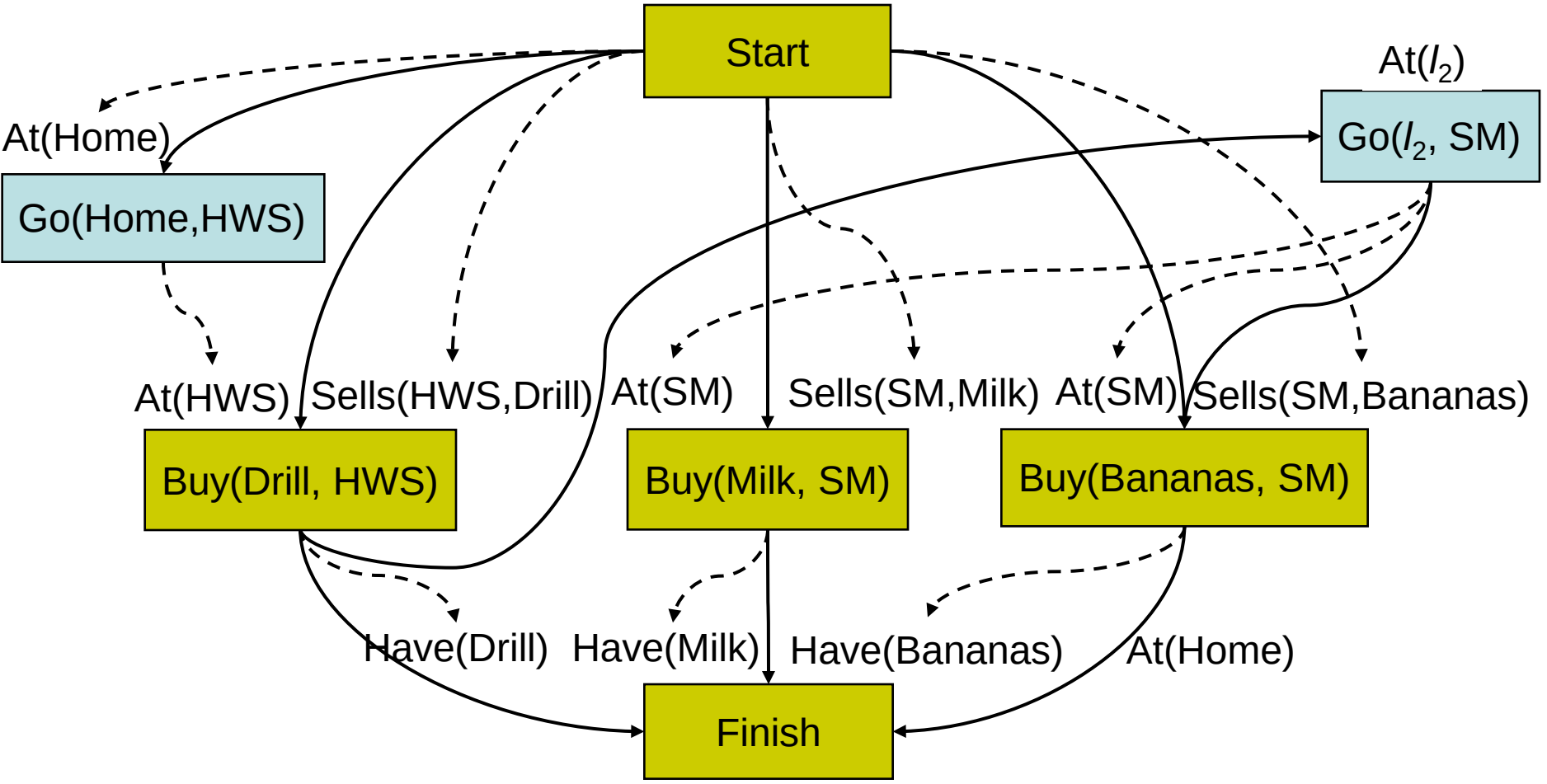
- Una elección no-determinista: ¿cómo resolver el ataque a At(s_1)?
  - Nuestra elección: que Buy(Drill) preceda a Go(SM) → resuleve otros ataques



[Ejemplo de Dana Nau en Lecture slides for *Automated Planning.*]

Ejemplo 1 (cont)

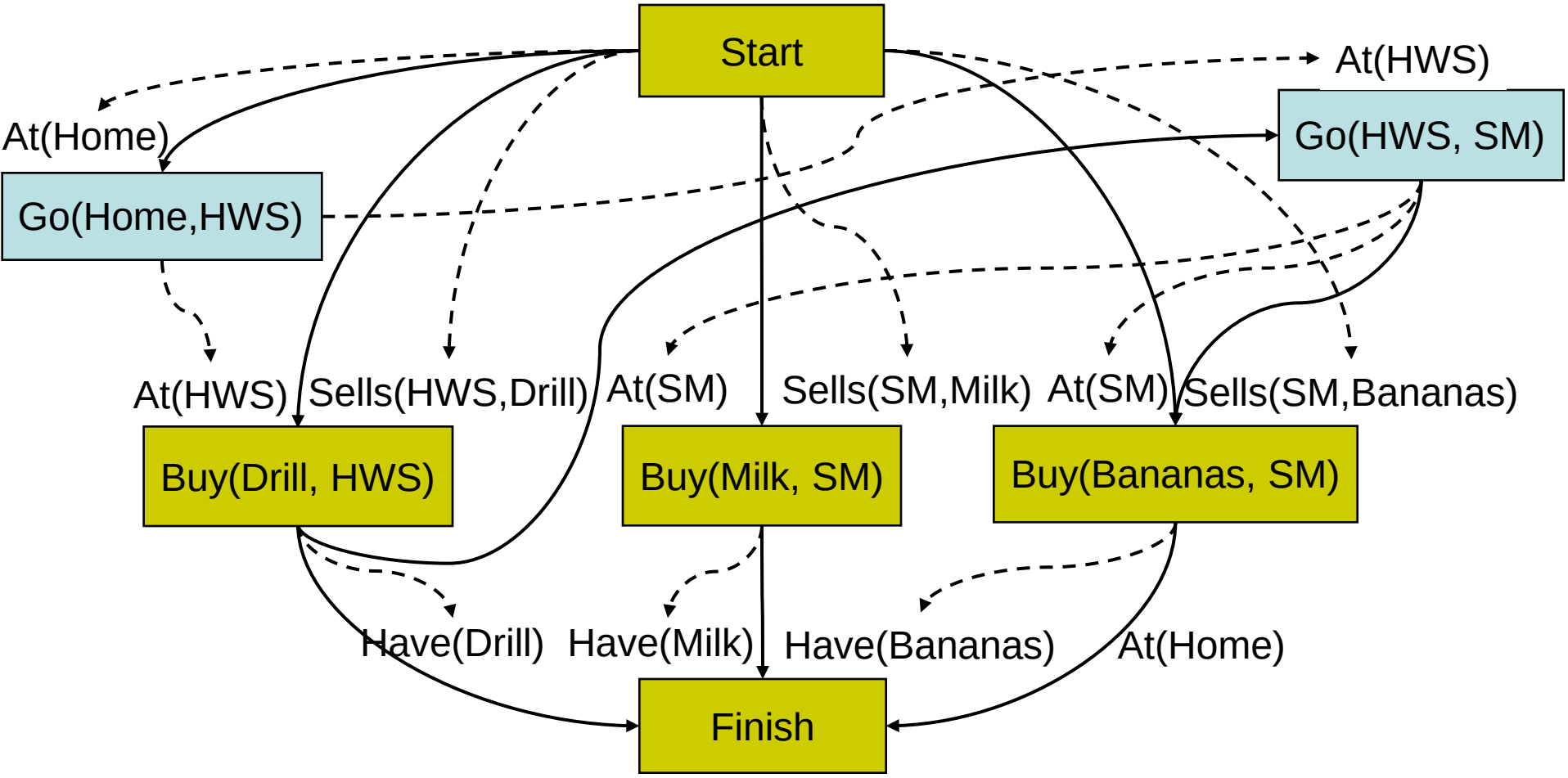
- Elección no determinista: ¿Cómo obtener At(l_1)?
  - Lo haremos desde Start, con l_1=Home



[Ejemplo de Dana Nau en Lecture slides for *Automated Planning.*]

Ejemplo 1 (cont)

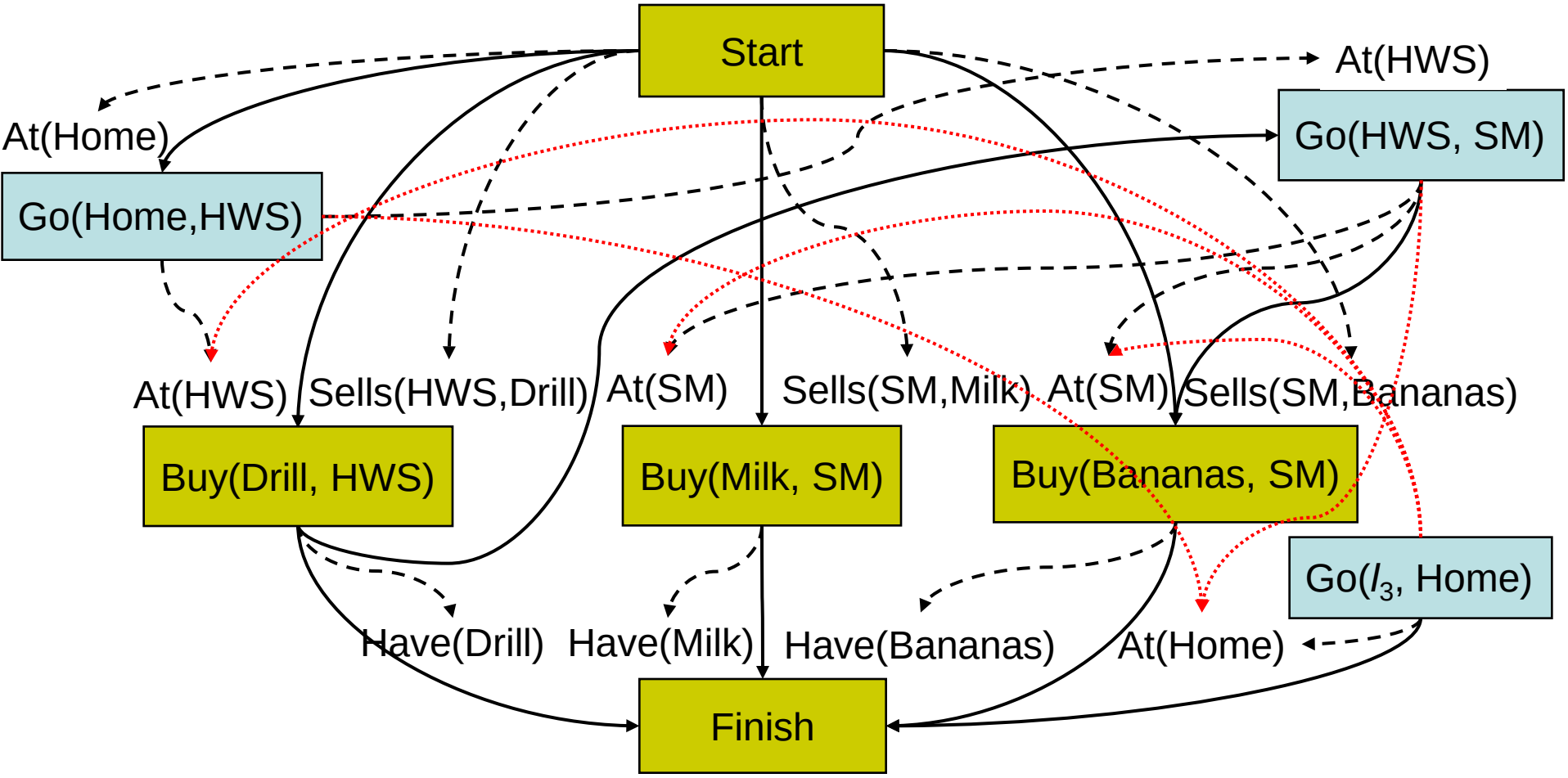
- Elección no determinista: ¿Cómo obtener At(l_2)?
  - Lo haremos a partir de Go(Home,HWS), con l_2= HWS



[Ejemplo de Dana Nau en Lecture slides for *Automated Planning.*]

Ejemplo 1 (cont)

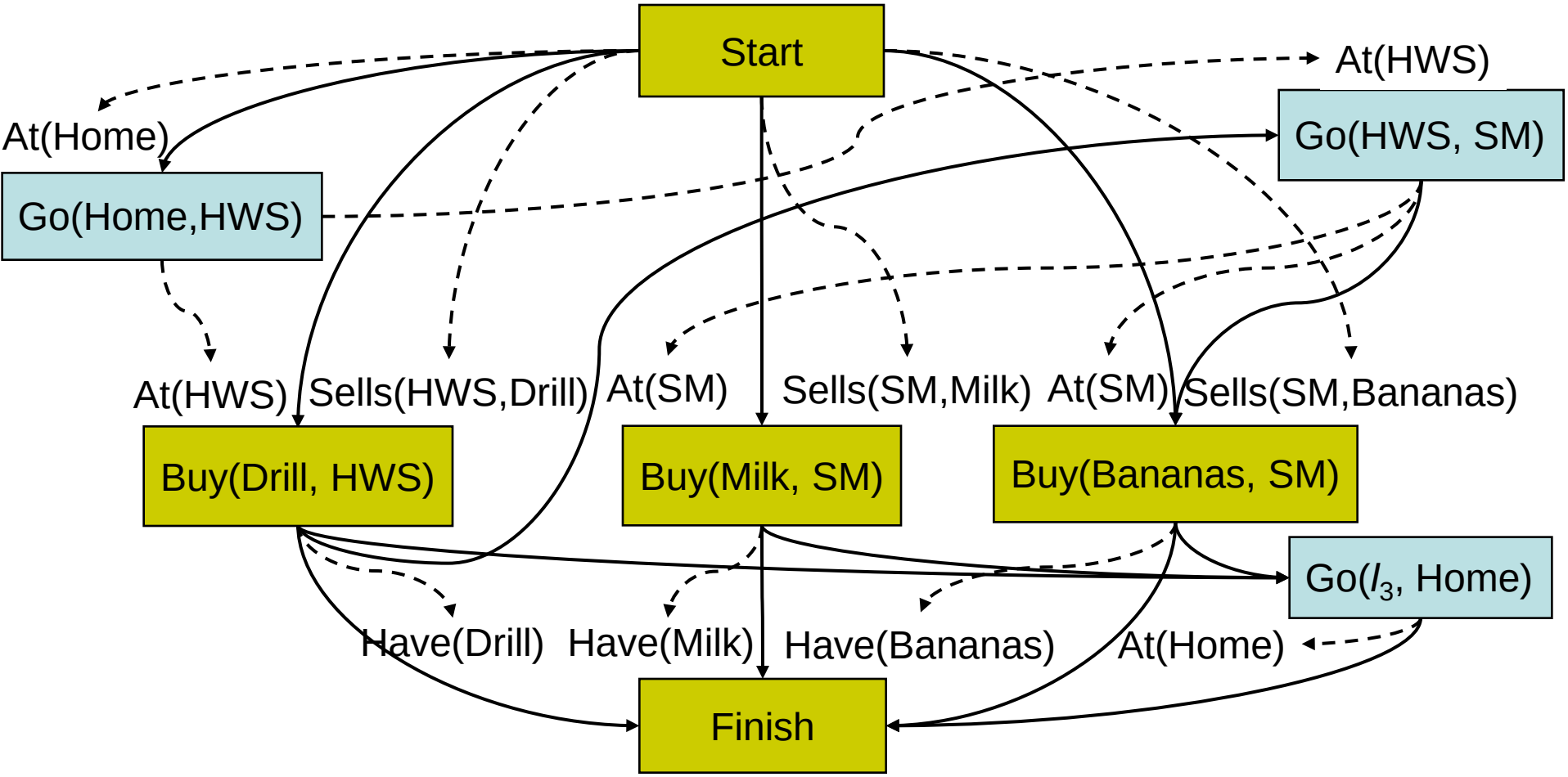
- La única forma de obtener At(Home) al final → esto crea varios ataques



[Ejemplo de Dana Nau en Lecture slides for *Automated Planning.*]

Ejemplo 1 (cont)

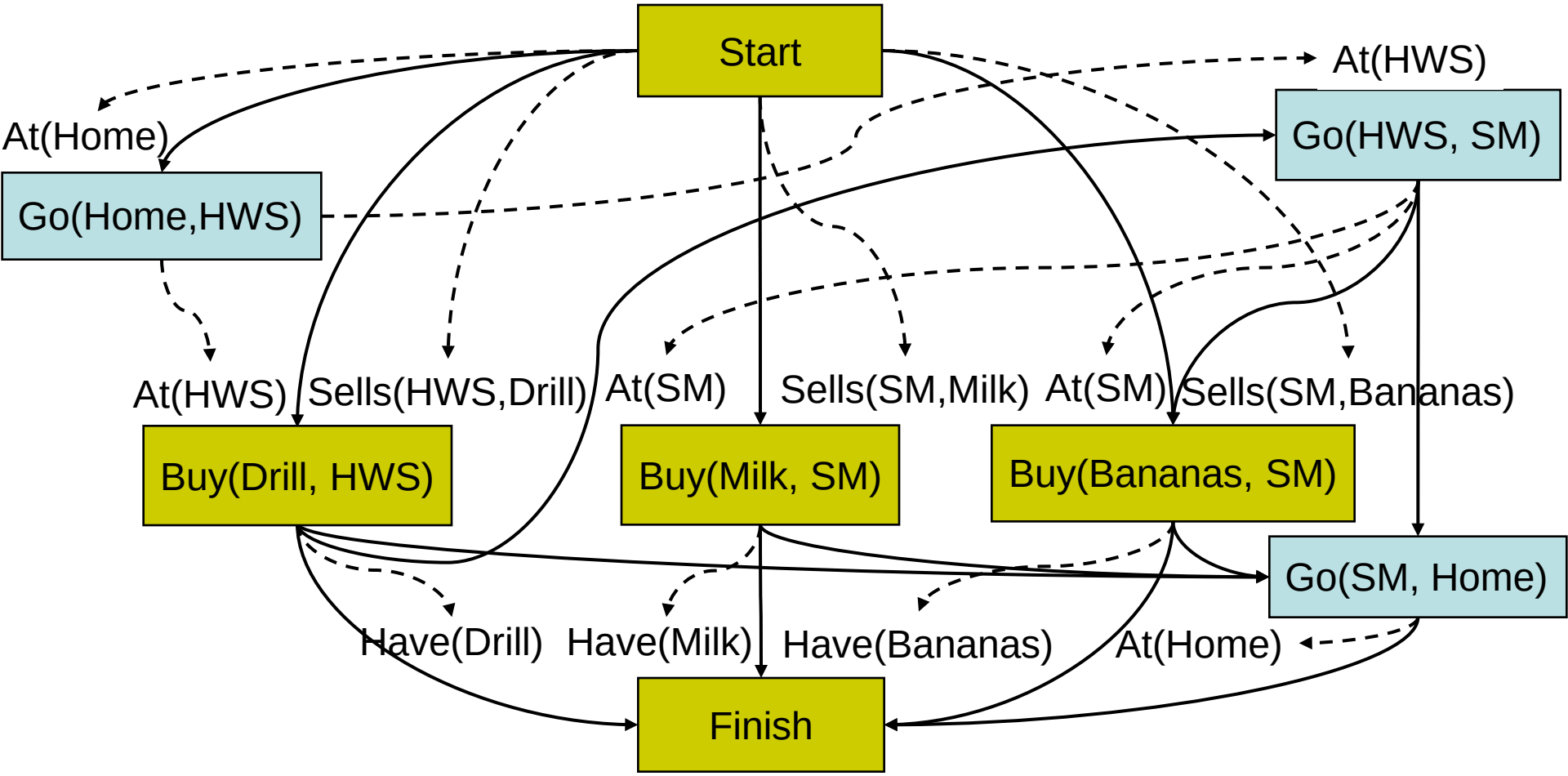
- Para eliminar los ataques a At(SM) y At(HWS), los haremos preceder Go(l_3,Home) → esto elimina los otros ataques



[Ejemplo de Dana Nau en Lecture slides for *Automated Planning.*]

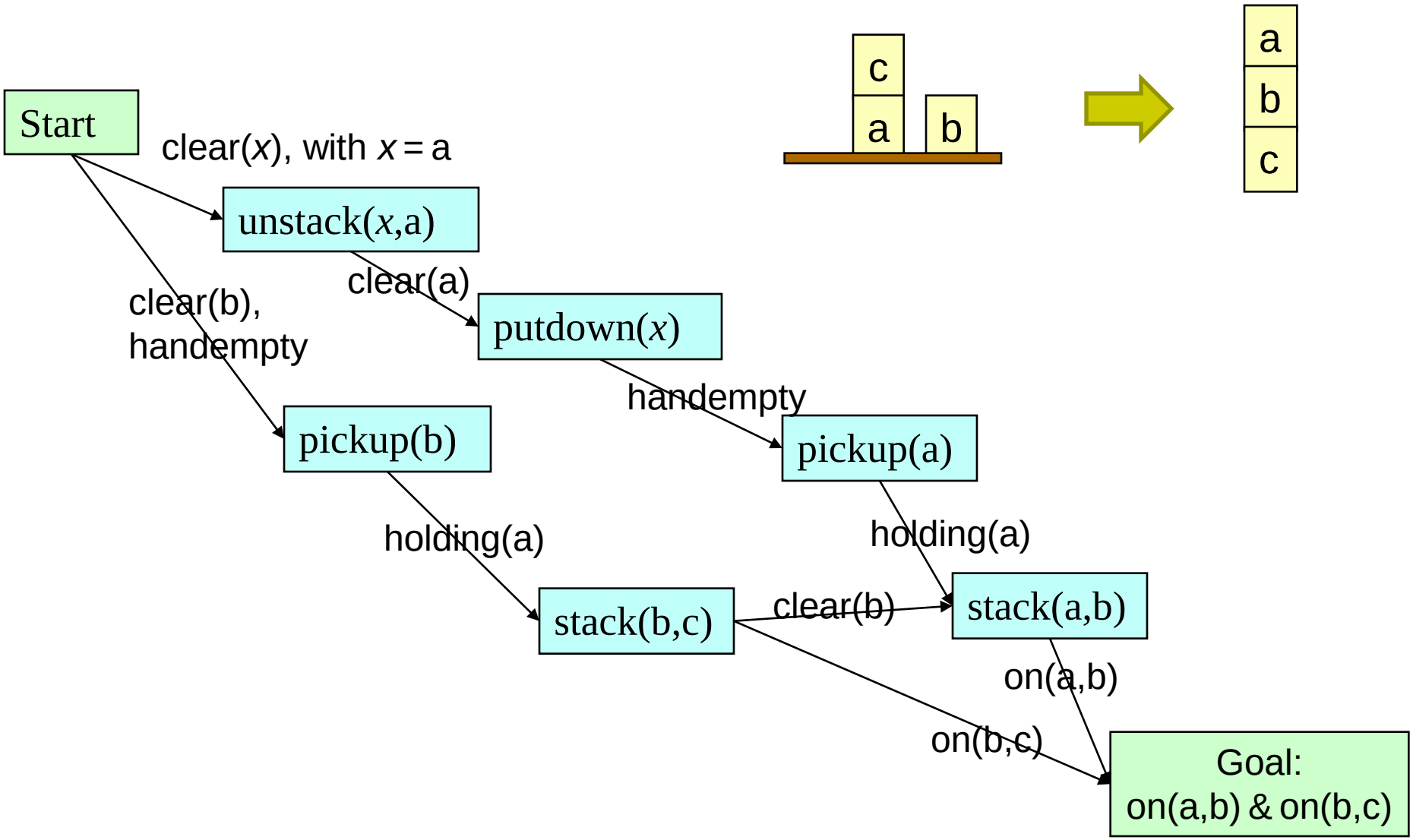
Ejemplo 1 – Plan final

- Establecer At(l_3) con l_3=SM



[Ejemplo de Dana Nau en Lecture slides for *Automated Planning.*]

Ejemplo 2


Start
clear(x), with x = a
unstack(x,a)
clear(a)
clear(b),
handempty
putdown(x)
handempty
pickup(b)
pickup(a)
holding(a)
holding(a)
stack(b,c)
clear(b)
stack(a,b)
on(a,b)
on(b,c)
Goal:
on(a,b) & on(b,c)
c
a
b
a
b
c

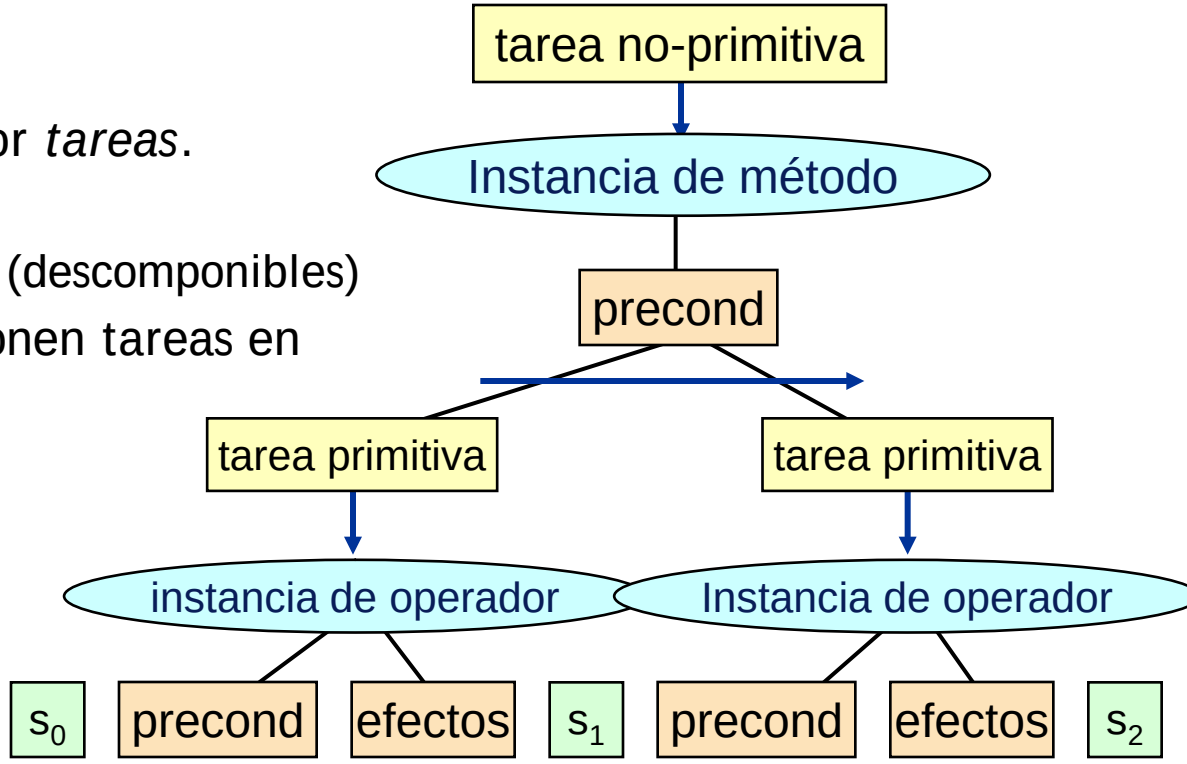
Planificación Jerárquica

- Hasta ahora todos los métodos que hemos visto trabajan en un único nivel de abstracción.
- Los humanos somos capaces de generar planes en diferentes niveles, desde planes de muy alto nivel a planes muy detallados.
- El plan de alto nivel a veces se llama receta y sirve como guia para la planificación a bajo nivel.

- **Planificación Jerárquica**:
  - Describe tareas y subtareas, y les asocia acciones.
  - Las tareas forman una red de tareas jerárquica (*Hierarchical Task Network* ó *HTN*)
  - El motor de planificación es capaz de explorar los diferentes niveles de (sub)tareas.

Planificación jerárquica simple

- Descripción del problema:
  - *Estados* y *operadores*: como en planificación clásica
  - No hay *objetivos*
  - Planificación guiada por *tareas*.
    - *Tareas primitivas*
    - *Tareas no primitivas* (descomponibles)
  - Los *métodos* descomponen tareas en sub-tareas



Planificación jerárquica simple: tareas y métodos

- *Tarea*: una expresión de la forma $t(u_1,\ldots,u_n)$
  - t es un *símbolo de tarea*, y cada u_i es un término
  - Dos tipos de símbolos de tarea:
    - *primitivos*: taear de las que sabemos cómo ejecutarlas directamente
      - El símbolo de tarea es un nombre de operador
    - *no-primitivos*: tareas que se han de descomponer en subtareas
      - usar *métodoss*

- *Método*: una tupla

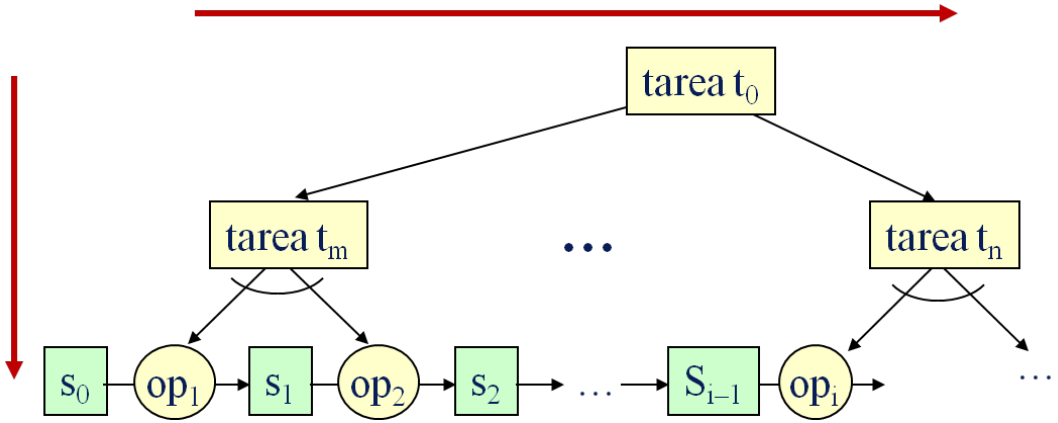
  m = (nombre(m), tarea(m), precond(m), subtareas(m))

  - nombre(m): una expresión de la forma $n(x_1,\ldots,x_n)$
    - $x_1,\ldots,x_n$ son parametros – simbolos de variables
  - tarea(m): una tarea no-primitiva
  - precond(m): precondiciones (literales)
  - subtareas(m): una secuencia parcialmente ordenada de las tareas $\langle t_1, \ldots, t_k \rangle$

Planificación jerárquica simple: dominio y problema

- **Dominio de planificación**: *métodos*, *operadores*
- **Problema de planificación**: *métodos*, *operadores*, *estado inicial*, *lista de tareas*

- **Solución**: cualquier plan ejecutable que se pueda generar por aplicar de forma recursiva
  - métodos para tareas no-primitivas
  - operadores para tareas primitivas



Ejemplo de PSP

- Un ejemplo simple de planificación:
  - Ir de casa al parque.

method travel-by-foot
  precond: $distance(x, y) \leq 2$
  task: travel(a, x, y)
  subtasks: walk(a, x, y)

method travel-by-taxi
  task: travel(a, x, y)
  precond: $cash(a) \geq 1.5 + 0.5 \times distance(x, y)$
  subtasks: $\langle$call-taxi(a, x), ride(a, x, y), pay-driver$(a, x, y)\rangle$

operator walk
  precond: $location(a) = x$
  effects: $location(a) \leftarrow y$

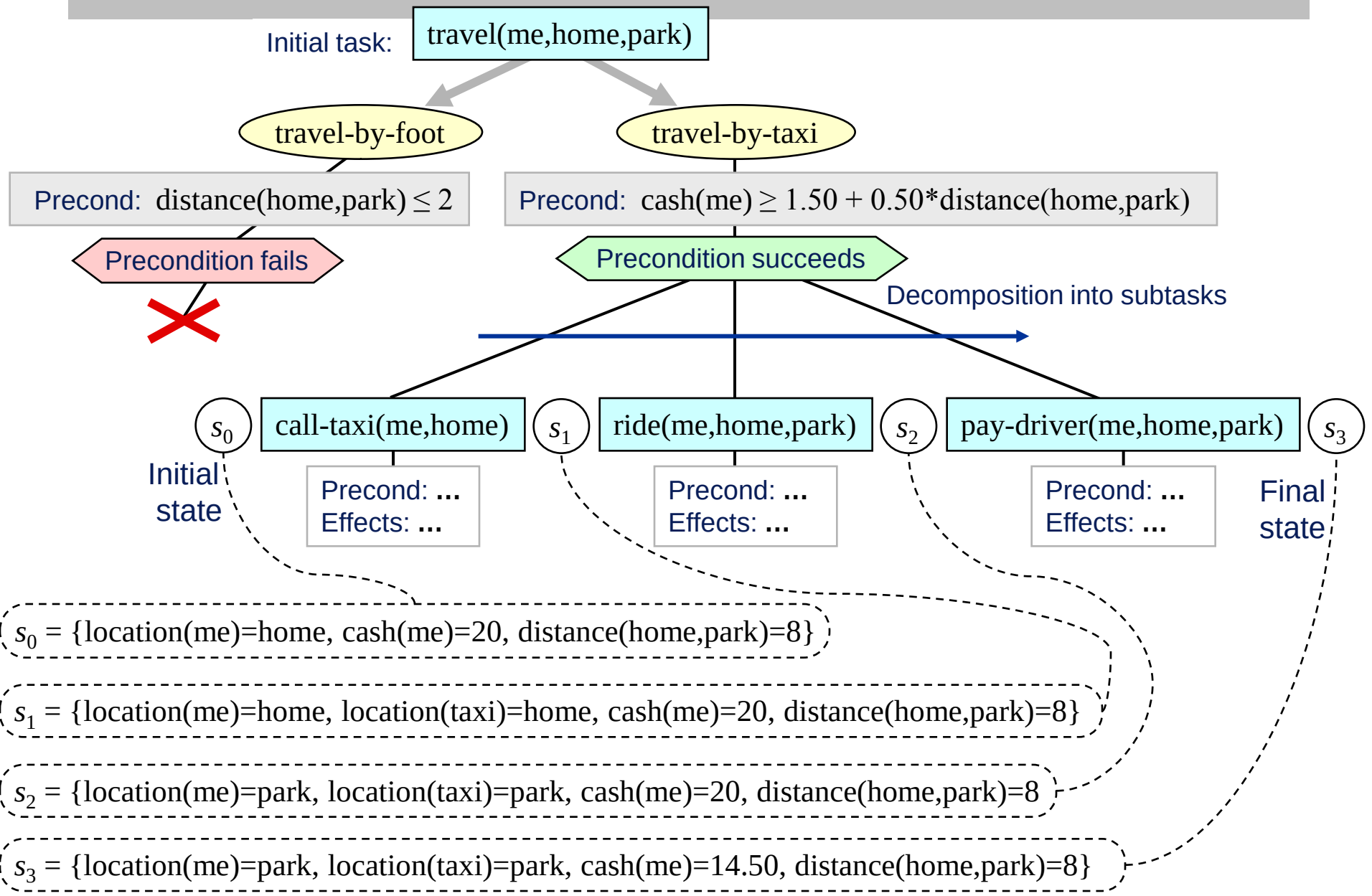
operator call-taxi(a, x)
  effects: $location(taxi) \leftarrow x$

operator ride-taxi(a, x)
  precond: $location(taxi) = x$, $location(a) = x$
  effects: $location(taxi) \leftarrow y$, $location(a) \leftarrow y$

operator pay-driver(a, x, y)
  precond: $cash(a) \geq 1.5 + 0.5 \times distance(x, y)$
  effects: $cash(a) \leftarrow cash(a) - 1.5 + 0.5 \times distance(x, y)$

[Ejemplo de Dana Nau en Lecture slides for *Automated Planning.*]

Ejemplo de PSP: estoy en casa con $20, quiero ir a un parque a 8 millas de aquí.



Otros resultados interesantes en planificacion

- Reutilització de Planes
- Planificación específica de un domínio (Domain-specific planning)

Reutilización de planes (Plan Reuse)

- Si planificar es algorítmicamente tan complejo…
- Idea: Reutilizar planes anteriores para resolver nuevos problemas de planificación
- Consiste en dos pasos
  - Reconocimiento (matching) del plan
  - Modificación del plan
- Resultado: según varios estudios…
  - En general, reutilizar planes suele ser aún más complejo a nivel algorítmico que planificar desde cero.
  - El cuello de botella es el plan matching
  - Da mejores resultados solo cuando dos problemas son lo suficientemente similares

Algoritmos especificos de un dominio

- Idea: podemos optimizar el algoritmo de planificación para resolver un problema concreto
- Ejemplo: Blocks World:

```
loop
   if there is a clear block x such that
            x needs to be moved and
            x can be moved to a place where it won't need to be moved
        then move x to that place
   else if there is a clear block x such that
            x needs to be moved
        then move x to the table
   else if the goal is satisfied
        then return the plan
   else return failure
repeat
```

[Ejemplo de Dana Nau en Lecture slides for *Automated Planning.*]

Heurísticos especificos de un dominio

- Originalmente se usaban heurísticos independientes del dominio
- ¿Y si se usan heurísticos dependientes del problema?
  - Problema: no podremos aplicarlo a otros problemas

- Resultado: normalmente los planificadores con heurísticos específicos del dominio mejoran su rendimiento respecto a los que son independientes del dominio

- Ejemplo:
  - En el N-puzzle, la estimación detallada de la distancia entre el estado actual y el estado objetivo puede acelerar bastante la búsqueda del plan.

La competición internacional de planificación

- Una forma de hacer que la tecnología de planificación avance más rápido
- Primera edición en 1998
- Creadora del lenguaje PDDL

- Efectivamente ha hecho evolucionar el área de investigación
  - Explosión de nuevos métodos
  - Hibridación de métodos
  - Optimización de métodos
- Todos los resultados están disponibles en la web
  `http://ipc.icaps-conference.org/`
- El código de muchos de los planificadores esta disponible