Planificación y modelos

Planificación y PDDL, Parte I (Capítulos 1–3)

Francisco Martín Rico

Intelligent Robotics Lab, URJC



Índice

1. Planificar: decidir antes de actuar
2. Un primer modelo: STRIPS
3. Qué hace un planificador con tu modelo
4. Resumen

1 Planificar: decidir antes de actuar

2 Un primer modelo: STRIPS

3 Qué hace un planificador con tu modelo

4 Resumen

Decidir antes de actuar

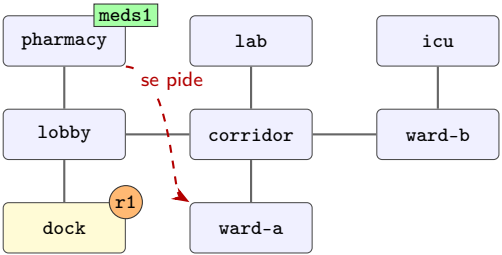
- Hay decisiones que no miran adelante: obstáculo a medio metro, el robot se para
- Otras dependen de lo que vendrá: para llevar medicamentos a la sala A, el mensajero antes debe *alejarse* de ella, hacia la farmacia
- **Planificar**: dada una situación, lo que se puede hacer y un objetivo, hallar qué acciones hacer, y en qué orden, para alcanzar el objetivo
- El resultado es un **plan**, encontrado *antes* de actuar, sobre una descripción del mundo

Qué hace de un problema un problema de planificación

- La respuesta es una forma de actuar, no un número ni una clase
- Las acciones interactúan: una hace posible otra (coger solo en la farmacia) o imposible (con una caja, no hay segunda recogida)
- Demasiadas secuencias para probarlas todas a ciegas

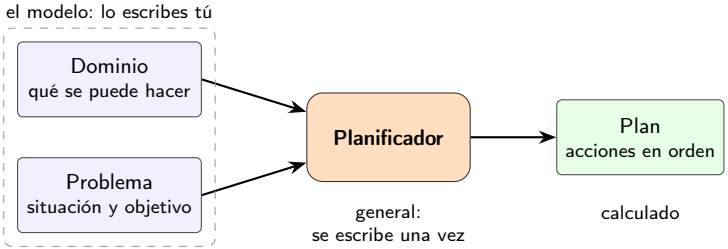
El mundo del mensajero: el ejemplo de todo el curso

- Una planta de hospital con **ocho lugares**
- A la UCI solo se llega por la sala B; toda conexión vale en los dos sentidos
- Robots `r1`, `r2`; artículos `meds1`, `meds2`, `sample1`, `meal1`
- Primera versión: **mover**, **coger**, **dejar**; un artículo cada vez
- Pequeño a propósito, y con sitio para crecer: puertas, batería, duraciones, horarios...



La flecha es el deseo, no el camino: el camino sigue las conexiones grises.

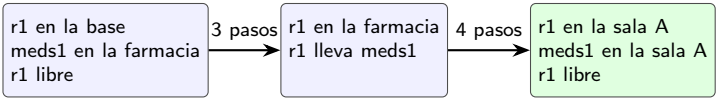
Un resolutor general más un modelo



- **Autonomía basada en modelos**. **Modelo**: qué existe, qué puede ser cierto, qué se puede hacer, qué se quiere; nunca *cómo*. **Planificador**: solo sabe buscar
- Para cambiar el comportamiento, se cambia el modelo: medicamentos en el laboratorio, un pasillo cerrado, un ala nueva
- Un planificador no es mejor que su modelo: «diez cajas por robot» da planes que apilan diez cajas en una bandeja

Tres ingredientes: estados, acciones, objetivo

- **Estado**: el mundo en un momento, todo lo que importa para decidir y nada más; el primero es el **estado inicial**
- **Acción**: una **precondición** (cuándo se puede hacer) y **efectos** (qué cambia)
- **Objetivo**: una condición sobre los estados; «meds1 en la sala A» describe muchos



Los estados son nodos; las acciones, aristas; un plan, un camino. Pero **nadie dibuja el grafo**: el modelo lo describe en una página, y el planificador construye la parte que necesita.

Los supuestos clásicos

Finito en el vestíbulo o no, nunca «a 2,37 m dentro»

Determinista un resultado por acción

Totalmente conocido el estado inicial, por completo

Estático solo el agente cambia el mundo

Instantáneo las acciones no tienen duración

Todos son falsos en un hospital; todos facilitan mucho la planificación. Un plan es entonces una simple secuencia, con éxito garantizado.

Supuesto	Qué lo levanta
Finito	números (Parte III)
Instantáneo	acciones durativas (Parte IV)
Estático	hechos con hora (Cap. 13), procesos y eventos (Parte VI)
Determinista	no en PDDL: ejecutar, observar, replanificar (Cap. 20)
Totalmente conocido	no en PDDL (Cap. 20)

Un modelo equivocado que es útil

A un modelo no hay que preguntarle «¿es verdadero?», sino «¿es lo bastante bueno para las decisiones que necesitamos de él?»

La planificación y sus alternativas

Enfoque	Tú aportas	Una situación nueva exige	Punto débil
Comportamiento a mano	Las decisiones	Código nuevo si no se previó	La variedad
Búsqueda a mano	Estados, sucesores, heurística, en código	Datos nuevos; código si cambian las reglas	La reutilización
Aprendizaje	Ejemplos o una recompensa	Más datos si se aleja de lo visto	Las garantías
Planificación	Un modelo	Un fichero de problema nuevo	El modelo debe existir

- Un planificador es la búsqueda a mano un paso más allá: el espacio de estados se *declara*; búsqueda y heurísticas se reutilizan en todo problema
- La planificación encaja con reglas claras, muchas situaciones, objetivos cambiantes
- En un robot conviven: percepción aprendida, control a mano, trayectorias y **planificación de tareas**: *qué* hacer y en qué orden

Dónde se usa la planificación

El espacio

Remote Agent controló la Deep Space 1 en mayo de 1999. Una herramienta de planificación ayudó a preparar las jornadas de los vehículos marcianos Spirit y Opportunity (2004).

Logística

Flotas, contenedores, grúas, fábricas. El dominio *logistics* es una prueba habitual desde la primera competición (Cap. 4).

Robótica

Robots de servicio con objetivos cambiantes. PlanSys2 conecta planificadores PDDL con robots ROS 2 (Cap. 20).

STRIPS y un robot llamado Shakey

En 1971 Fikes y Nilsson (Stanford Research Institute) publicaron STRIPS, el planificador de Shakey. Cada acción: los hechos que necesita, los que hace verdaderos, los que hace falsos. El planificador pronto quedó superado; esa forma de describir acciones sigue siendo el núcleo de PDDL.

Dominio y problema

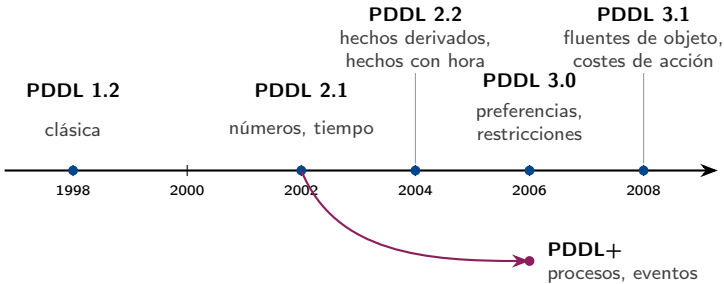
- **Dominio**: una clase de mundo. Qué puede ser cierto, qué se puede hacer. Ningún robot ni habitación concretos
- **Problema**: una situación y una petición. Objetos, mapa, inicio, objetivo
- **Un dominio, muchos problemas**: cada petición del día es un problema nuevo; hasta el mapa pertenece al problema

Solo para dar una idea: la acción `move` (el Cap. 2 explica cada símbolo)

```
(:action move
  :parameters (?r ?from ?to)
  :precondition (and (robot ?r) (location ?from) (location ?to)
                     (robot-at ?r ?from) (connected ?from ?to))
  :effect (and (not (robot-at ?r ?from))
               (robot-at ?r ?to)))
```

Un nombre, parámetros, una condición, un efecto. Las palabras con ? son variables: cualquier robot, cualquier par de lugares conectados.

PDDL y sus versiones

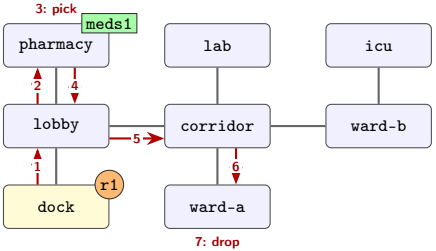


- 1998: McDermott y un comité, para la primera competición de planificación (IPC)
- Cada versión añade lo que los supuestos clásicos dejan fuera: nuestra ruta
- Un modelo declara sus **requisitos**; cada planificador acepta algunos: de ahí la *portabilidad*

Primer contacto con Kairos

```
$ kairos courier-domain.pddl courier-p01.
    pddl
; cost = 7
(move r1 dock lobby)
(move r1 lobby pharmacy)
(pick r1 meds1 pharmacy)
(move r1 pharmacy lobby)
(move r1 lobby corridor)
(move r1 corridor ward-a)
(drop r1 meds1 ward-a)
```

- Primero el dominio, después el problema
- Un **paso** por línea: la acción y luego los objetos, en el orden de los parámetros
- Sin costes en el modelo: coste = pasos



Los pasos 1–2 se alejan de la sala A: primero hay que coger la caja. Nadie le habló al planificador de farmacias.

Validar, otro problema, sin plan

```
$ kairos validate courier-domain.pddl courier
      -p01.pddl \
    courier-p01.plan
plan valid, makespan 7
```

Validar: simular el plan sobre el modelo, paso a paso.

Mismo dominio, otro día (p02):

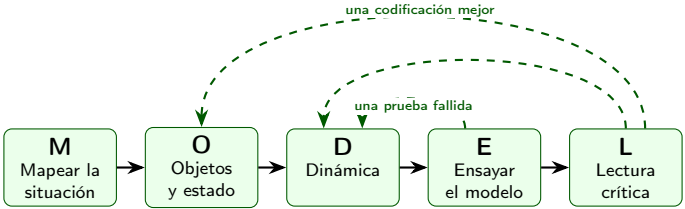
```
$ kairos courier-domain.pddl courier-p02.pddl
; cost = 8
(move r1 ward-a corridor)
...
(drop r1 sample1 lab)
```

p03: sala A cerrada, su conexión no aparece.

```
$ kairos courier-domain.pddl courier-p03.pddl
note: the goal needs atoms that no action can reach
(item-at meds1 ward-a) can never be reached:
  drop adds it, but needs (robot-at ?r - object ward-a), which
       never holds
    move adds it, but needs (connected ?from - object ward-a),
      which never holds
      no action adds it, and the initial state does not have it
       [orchestrator.unreachable_goal]
kairos: unsolvable (exhausted)
```

- «**Unsolvable**» es una respuesta: ninguna secuencia, por larga que sea, sirve
- La nota lo explica, con el vocabulario del modelo

El método MODEL: modelar es elegir



Paso	Pregunta	Artefacto
M	¿Cuál es el problema, quién actúa, qué cuenta como resuelto, qué se deja fuera?	ficha del modelo
O	¿Qué cosas existen, y qué tiene que saber de ellas el planificador?	tabla de estado
D	¿Qué se puede hacer, cuándo, y qué cambia?	tabla de acciones
E	¿Hace el planificador lo que el mundo permitiría, y nada más?	batería de pruebas
L	¿Qué otra codificación serviría, y cuál es mejor aquí?	comparativa

Paso M en el mundo del mensajero: la ficha del modelo

Paso M: una primera ficha del modelo

Historia	Un robot lleva artículos entre las habitaciones de una planta de hospital, a petición.
Agentes	Un robot mensajero. Es el único que actúa.
Éxito	Cada artículo pedido está en la habitación donde se pidió.
Alcance	Dentro: las habitaciones y sus conexiones; dónde están el robot y los artículos; qué lleva el robot. Fuera: distancias, tiempo, batería, puertas, personas, cómo agarra el brazo.
Supuestos	Los cinco clásicos. El robot lleva un artículo cada vez.

- En el paso M no hay PDDL. La mayoría de los malos modelos fallan aquí: resuelven otro problema
- Cada línea del Alcance es una decisión: «Fuera: batería» permite recorridos que la agotan (el Cap. 8 cambia la ficha)

Los cinco criterios

El paso L juzga las alternativas con cinco criterios, siempre en este orden:

1. **Fidelidad**: ningún plan imposible (diez cajas a la vez) y ninguna solución perdida (no cruzar nunca dos veces el vestíbulo)
2. **Tamaño**: cuántos hechos y acciones concretos resultan al aplicar el dominio a todos los objetos
3. **Guía**: cuánto facilita la búsqueda: callejones sin salida, formas equivalentes, rodeos, lo que ven las estimaciones del planificador
4. **Claridad**: fácil de leer, cambiar y explicar, también a ti mismo dentro de seis meses
5. **Portabilidad**: qué planificadores aceptan las características usadas

La fidelidad, primero

Un modelo equivocado da respuestas equivocadas muy deprisa. Los demás se compensan entre sí, así que el paso L termina con una decisión *y su motivo*.

Alternativas comparadas: dónde están las cosas

Criterio	A: ocho lugares	B: mil celdas (40 por 25 m)
Fidelidad	Los planes dicen qué habitaciones cruzar, no cómo	Rutas exactas; dejan de valer en cuanto se mueve una silla
Tamaño	8 lugares, 14 movimientos posibles	1000 celdas, unos 4000 movimientos posibles
Guía	Una entrega son 7 pasos	Decenas de pasos, muchísimas rutas equivalentes
Claridad	Los nombres que usa la gente	La celda $(17, 4)$ no le dice nada a una enfermera
Portabilidad	Clásica	Clásica

Veredicto

A es mejor en todos los criterios en que difieren cuando el planificador decide *qué* hacer; B responde a otra pregunta, cómo cruzar una habitación, que ya resuelve el software de navegación. El nivel de detalle adecuado es el más bajo que todavía distingue las decisiones que necesitas.

Errores comunes de un primer contacto

Ficheros en el orden equivocado

Primero el dominio, después el problema. Intercambiados: `error: expected 'domain', found name 'problem'`. Los diagnósticos dan fichero, línea, columna y un código; lee primero el primer error.

Tomar «unsolvable» por un fallo

Ejecutar más tiempo no cambia nada: el modelo no tiene plan. O el mundo no lo tiene, o el modelo está mal. Empieza por la nota.

Esperar lo que el modelo no dice

El robot se queda en la sala A: el objetivo no pedía la base. Lo que no está en el modelo no existe.

Tratar un plan como un programa

Un plan vale para un estado inicial y un objetivo. El plan de `p01` en `p02`: `unknown object 'meds1'`, `plan invalid`. Se reutiliza el dominio, nunca el plan.

1 Planificar: decidir antes de actuar

2 Un primer modelo: STRIPS

3 Qué hace un planificador con tu modelo

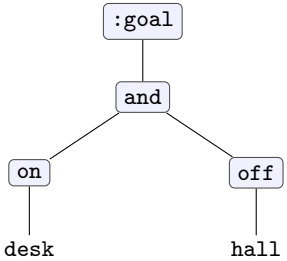
4 Resumen

Listas dentro de listas: la forma de PDDL

Todo es una lista entre paréntesis; su primera palabra dice qué es (**expresiones S**, de Lisp).

```
(:goal (and (on desk) (off hall)))
```

- La disposición es libre: el planificador cuenta paréntesis
- Comentarios: desde ; hasta el final de la línea
- Nombres: letras, dígitos, -, _; sin mayúsculas
- Las palabras clave empiezan por :, las variables por ?
- Cada fichero es una lista `(define ...)`



Cada par de paréntesis es un nodo, con el nombre de su primera palabra: así lo ve el planificador.

Predicados, hechos y estados

Tres lámparas, los **objetos**: `hall`, `desk`, `porch`.

```
(:predicates
  (on ?l)       ; lamp ?l is lit
  (off ?l))     ; lamp ?l is dark
```

- **Predicado**: una propiedad o una relación
- **Hecho**: un predicado aplicado a objetos, `(on desk)` (Kairos: *atom*)
- **Estado**: el conjunto de los hechos ciertos, {`(on hall)`, `(off desk)`, `(off porch)`}
- **Hipótesis del mundo cerrado**: lo que no está en el estado es falso

Los nombres no significan nada para el planificador

{`(on hall)`, `(off hall)`} es un estado válido. Solo el modelo, con su estado inicial y sus acciones, mantiene coherentes los hechos.

¿Para qué `off`?

En STRIPS una condición solo pide hechos ciertos. «La lámpara está apagada» necesita un hecho que sea cierto justo entonces. El Cap. 5 levanta este límite.

Acciones: precondiciones y efectos

```
(define (domain switch)
  (:requirements :strips)

  (:predicates
    (on ?l)       ; lamp ?l is lit
    (off ?l))     ; lamp ?l is dark

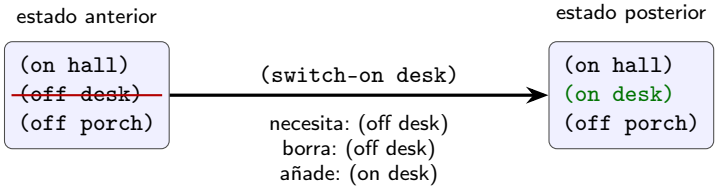
  (:action switch-on
    :parameters (?l)
    :precondition (off ?l)
    :effect (and (on ?l)
                 (not (off ?l))))

  (:action switch-off
    :parameters (?l)
    :precondition (on ?l)
    :effect (and (off ?l)
                 (not (on ?l)))))
```

- `:requirements :strips`: las características usadas; el nivel básico, todos los planificadores lo admiten
- `:parameters`: sobre qué actúa la acción; la acción es un patrón
- `:precondition`: hechos que deben ser ciertos, unidos con `and`
- `:effect`: los hechos tal cual pasan a ser ciertos (**adición**); `(not ...)`, falsos (**borrado**)
- Objetos en lugar de parámetros: una **acción instanciada**, `(switch-on desk)`

Aplicar una acción

1. Una acción instanciada es **aplicable** si todos los hechos de su precondición están en el estado
2. **Aplicarla**: quitar los borrados, poner las adiciones. *Todo lo demás se queda*



- Una acción dice solo lo que cambia: las otras lámparas no hace falta mencionarlas
- Pero tiene que decir *todo* lo que cambia: sin su borrado, la lámpara del escritorio estaría encendida y apagada a la vez

El problema: objetos, estado inicial y objetivo

```
(define (problem switch-p01)
  (:domain switch)
  (:objects hall desk porch)
  (:init (on hall) (off desk) (off porch))
  (:goal (and (on desk) (off hall))))
```

- `:objects`: no existe nada más
- `:init` es **completo**: lo que no enumera es falso
- `:goal` es **parcial**: la lámpara del porche puede acabar de cualquier manera

```
$ kairos check switch-domain.pddl switch-p01.pddl
valid, plannable
$ kairos switch-domain.pddl switch-p01.pddl
; cost = 2
(switch-off hall)
(switch-on desk)
```

- *Valid*: PDDL correcto, ficheros que concuerdan. *Plannable*: Kairos lo admite
- **Plan**: cada acción instanciada, aplicable tras las anteriores; el estado final contiene el objetivo

Paso 0: objetos y estado

Propiedad	Sobre	Clase	Codificada como
Es un robot / un lugar / un artículo	un objeto	estática	`(robot ?x)` etc.
Se puede ir de un lugar a otro	dos lugares	estática	`(connected ?from ?to)`
Dónde está un robot	un robot, un lugar	dinámica	`(robot-at ?r ?l)`
Dónde está un artículo	un artículo, un lugar	dinámica	`(item-at ?i ?l)`
Qué artículo lleva un robot	un robot, un artículo	dinámica	`(carrying ?r ?i)`
El robot no lleva nada	un robot	dinámica	`(free ?r)`

- **Estática**: ninguna acción la cambia, datos del problema. **Dinámica**: alguna acción la cambia; hay que revisar cada acción frente a ella
- **Predicados de tipo**: STRIPS no tiene clases; sin ellos, nada impide «mover el vestíbulo» (el Cap. 4 trae los tipos)
- `connected` tiene dirección: una puerta de doble sentido son dos hechos. `free`: redundante para nosotros, necesario en STRIPS, acompasado con `carrying`

Paso D: dinámica

Acción	Parámetros	Se aplica cuando	Cambia
`move`	robot, origen, destino	el robot está en *origen*; *origen* está conectado con *destino*	el robot está en *destino*; ya no está en *origen*
`pick`	robot, artículo, lugar	el robot y el artículo están los dos en el lugar; el robot está libre	el robot lleva el artículo; el artículo ya no está en el lugar; el robot ya no está libre
`drop`	robot, artículo, lugar	el robot está en el lugar y lleva el artículo	el artículo está en el lugar; el robot está libre; ya no lleva el artículo

La rutina del paso D

- Para cada acción y cada fila dinámica de la tabla de estado: ¿la hace cierta, falsa o la deja igual?
- Parámetros: todo objeto que mencionen la condición y los cambios (`pick` necesita el lugar para borrar `(item-at ?i ?l)`)

El dominio del mensajero en STRIPS: predicados y move

```
(define (domain courier)
  (:requirements :strips)

  (:predicates
    (robot ?x)                ; ?x is a robot
    (location ?x)             ; ?x is a room of the floor
    (item ?x)                 ; ?x is something to carry
    (connected ?from ?to)     ; the robot can go from ?from to ?to
    (robot-at ?r ?l)          ; robot ?r is in ?l
    (item-at ?i ?l)           ; item ?i lies in ?l
    (carrying ?r ?i)          ; robot ?r carries item ?i
    (free ?r))                ; robot ?r carries nothing

  (:action move
    :parameters (?r ?from ?to)
    :precondition (and (robot ?r) (location ?from) (location ?to)
                       (robot-at ?r ?from) (connected ?from ?to))
    :effect (and (not (robot-at ?r ?from))
                 (robot-at ?r ?to)))
```

- Tabla de estado → `:predicates`, uno por fila
- Tabla de acciones → acciones
- Precondiciones: primero los predicados de tipo, después las condiciones de la tabla
- Efectos: los borrados y las adiciones de la columna «Cambia»

El dominio del mensajero en STRIPS: pick y drop

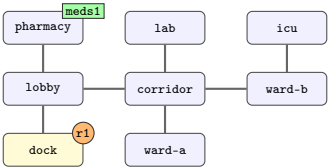
```
(:action pick
  :parameters (?r ?i ?l)
  :precondition (and (robot ?r) (item ?i) (location ?l)
                     (robot-at ?r ?l) (item-at ?i ?l) (free ?r))
  :effect (and (not (item-at ?i ?l)) (not (free ?r))
               (carrying ?r ?i)))

(:action drop
  :parameters (?r ?i ?l)
  :precondition (and (robot ?r) (item ?i) (location ?l)
                     (robot-at ?r ?l) (carrying ?r ?i))
  :effect (and (not (carrying ?r ?i))
               (item-at ?i ?l) (free ?r))))
```

- `pick` y `drop` son imágenes especulares: lo que una añade, la otra lo borra
- Esa simetría es buena señal; su ausencia merece una segunda mirada
- `free` y `carrying` cambian juntos, como exigía el paso O

Un problema para el dominio

```
(define (problem courier-p01)
  (:domain courier)
  (:objects r1 meds1
            lobby corridor pharmacy lab ward-a ward-b icu dock)
  (:init
    (robot r1) (item meds1)
    (location lobby) (location corridor) (location pharmacy)
    (location lab) (location ward-a) (location ward-b)
    (location icu) (location dock)
    ; the map: every connection in both directions
    (connected lobby corridor) (connected corridor lobby)
    (connected lobby pharmacy) (connected pharmacy lobby)
    (connected lobby dock) (connected dock lobby)
    (connected corridor lab) (connected lab corridor)
    (connected corridor ward-a) (connected ward-a corridor)
    (connected corridor ward-b) (connected ward-b corridor)
    (connected ward-b icu) (connected icu ward-b)
    ; where things are
    (robot-at r1 dock) (free r1)
    (item-at meds1 pharmacy))
  (:goal (item-at meds1 ward-a)))
```



- 10 objetos, 27 hechos en `:init`
- 24 estáticos: 10 de clase, 14 del mapa (7 conexiones, en los dos sentidos)
- Solo 3 dinámicos: lo que cambia un plan
- Objetivo: un hecho; el robot puede acabar en cualquier sitio

Paso E: comprobar, resolver, leer

```
$ kairos check courier-domain.pddl courier-
      p01.pddl
valid, plannable
```

Después, resolver (el plan de 7 pasos que conocemos) y leerlo sobre el estado: ¿podría el robot real dar cada paso?

Después de	Hechos dinámicos del estado
(inicio)	(robot-at r1 dock) (free r1) (item-at meds1 pharmacy)
(move r1 dock lobby)	(robot-at r1 lobby) (free r1) (item-at meds1 pharmacy)
(move r1 lobby pharmacy)	(robot-at r1 pharmacy) (free r1) (item-at meds1 pharmacy)
(pick r1 meds1 pharmacy)	(robot-at r1 pharmacy) (carrying r1 meds1)
(move r1 pharmacy lobby)	(robot-at r1 lobby) (carrying r1 meds1)
(move r1 lobby corridor)	(robot-at r1 corridor) (carrying r1 meds1)
(move r1 corridor ward-a)	(robot-at r1 ward-a) (carrying r1 meds1)
(drop r1 meds1 ward-a)	(robot-at r1 ward-a) (free r1) (item-at meds1 ward-a)

- Un artículo cargado no está **en ninguna parte**: ningún `item-at` entre pick y drop, así que nada puede cogerlo del suelo
- `kairos validate` hace esta lectura sobre cualquier fichero de plan

La batería de pruebas: trivial y típica

Prueba 1, trivial (p02): plan conocido antes de ejecutar

```
    ; where things are
    (robot-at r1 pharmacy) (free r1)
    (item-at meds1 pharmacy))
  (:goal (item-at meds1 lobby)))
```

```
$ kairos courier-domain.pddl courier-p02.pddl
; cost = 3
(pick r1 meds1 pharmacy)
(move r1 pharmacy lobby)
(drop r1 meds1 lobby)
```

Caza los errores gruesos, fáciles de localizar.

Prueba 2, típica (p03): dos entregas, vuelta a la base

```
    (item-at meds1 pharmacy) (item-at sample1 icu))
  (:goal (and (item-at meds1 ward-a)
              (item-at sample1 lab)
              (robot-at r1 dock))))
```

```
$ kairos courier-domain.pddl courier-p03.pddl
; cost = 18
(move r1 dock lobby)
...
(move r1 lobby dock)
```

Pasos 1–7 medicamentos, 8–15 muestra, 16–18 vuelta. ¿Algún paso que una persona no daría?

Prueba 3: un problema que no debe tener plan

p04: la UCI está cerrada

```
    (connected corridor ward-b) (connected ward-b corridor)
    ; no (connected ward-b icu) and no (connected icu ward-b)
    ; where things are
    (robot-at r1 dock) (free r1)
    (item-at meds1 pharmacy))
  (:goal (item-at meds1 icu)))
```

- Un modelo puede estar mal por permitir demasiado; una prueba que espera un plan nunca lo nota
- Lee la nota hacia atrás desde el objetivo: drop necesita el robot en la UCI, move necesita una conexión con ella, y `connected` es estático

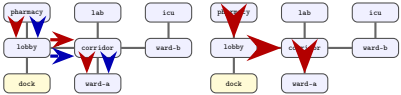
```
$ kairos courier-domain.pddl courier-p04.pddl
note: the goal needs atoms that no action can
      reach
(item-at meds1 icu) can never be reached:
  drop adds it, but needs (robot-at ?r -
      object icu), which never holds
    move adds it, but needs (connected ?from
      - object icu), which never holds
      no action adds it, and the initial
      state does not have it [orchestrator.
      unreachable_goal]
kairos: unsolvable (exhausted)
```

Atom = hecho; - `object` es un tipo (Cap. 4), aquí «cualquier objeto».

Prueba 4: un problema que tienta al planificador a hacer trampa

p05: dos cajas, las dos para la sala A

```
    ; where things are
    (robot-at r1 dock) (free r1)
    (item-at meds1 pharmacy) (item-at meds2 pharmacy))
 (:goal (and (item-at meds1 ward-a) (item-at meds2 ward-a))))
```



honrado: dos viajes, 15 pasos tramposo: un viaje, 9 pasos

- El planificador toma el camino más corto que permite el modelo
- Kairos: 15 pasos, dos viajes. El modelo aguanta

El plan tramposo de 9 pasos, a mano (el «redundante» `free` mantiene honrado al robot):

```
$ kairos validate courier-domain.pddl courier-p05.pddl \
    courier-p05-cheat.plan
courier-p05-cheat.plan:5: error: step 4 (pick r1 meds2 pharmacy) at 3.000: precondition false: (free r1) is false [
      validate.precondition_failed]
 5 | (pick r1 meds2 pharmacy)
plan invalid
```

La batería de pruebas del mundo del mensajero

p06 pide el propio estado prohibido: `(and (carrying r1 meds1) (carrying r1 meds2))`

```
$ kairos courier-domain.pddl courier-p06.pddl
note: no plan: every state reachable from the initial one was searched (1220 states) and none satisfies the goal. [
      orchestrator.why]
kairos: unsolvable (exhausted)
```

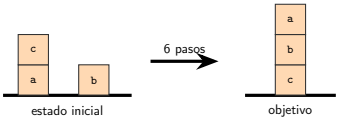
Otra clase de irresoluble: cada hecho del objetivo es alcanzable, nunca los dos juntos.

Paso E: la batería de pruebas

Prueba	**Fichero**	**Esperado y obtenido**
Trivial	`courier-p02`	3 pasos: pick, move, drop
Típica	`courier-p01, p03`	7 y 18 pasos; todos los pasos, sensatos
Sin plan	`courier-p04`	irresoluble: no se puede llegar a la UCI
Trampa	`courier-p05, p06`	15 pasos (no 9); irresoluble

La batería de pruebas se queda con el modelo: se vuelve a ejecutar tras cada cambio.

Un clásico: el mundo de bloques




```
(:action stack                    ; leave ?x on top of ?y
  :parameters (?x ?y)
  :precondition (and (holding ?x) (clear ?y))
  :effect (and (on ?x ?y) (clear ?x) (hand-empty)
               (not (holding ?x)) (not (clear ?y)))))
```

- También `pick-up`, `put-down`, `unstack`; `(hand-empty)` no tiene parámetros; `clear` trata de una ausencia
- La **anomalía de Sussman**: las partes del objetivo hay que intercalarlas

```
$ kairos blocks-domain.pddl blocks-p01.pddl
; cost = 6
(unstack c a)
(put-down c)
(pick-up b)
(stack b c)
(pick-up a)
(stack a b)
```

`(and (on a b) (on b a))`: irresoluble, 40 estados buscados. No hay ninguna ley escrita: las acciones nunca producen un estado así.

El método tras la primera pasada completa

- **Estático frente a dinámico**: dinámico = aparece en algún efecto. Mensajero: `robot-at`, `item-at`, `carrying`, `free`, lo previsto
- **La regla del marco**: los efectos dicen *todo lo que cambia, y nada de lo que no cambia*. La segunda mitad es gratis; la primera, obligación del modelador
- Gripper (dos pinzas): una capacidad se modela con objetos que sostienen cosas; el planificador llena las dos

Preguntas que este capítulo añade

M	¿Puedo escribir la ficha antes de nada de PDDL? ¿Qué reglas del mundo enuncia?
O	¿Estática o dinámica? ¿Trata de una ausencia, y pide un hecho propio? ¿Predicados que se solapan: quién los acompasa?
D	Cada acción, cada propiedad dinámica: ¿cierta, falsa o igual? ¿Cada hecho nuevo sustituye a uno que hay que borrar?
E	¿`check`, resolver y `validate` con pruebas trivial, típica, sin plan y de la trampa?
L	¿Qué otra codificación serviría?

Alternativas comparadas: tres codificaciones STRIPS

Criterio	A: `pick`, `drop`	B: `carry` (robot y artículo se mueven juntos)	C: un solo `at` para robots y artículos
Fidelidad	Un artículo cada vez, impuesto por `free`. Los planes dicen cuándo agarrar y soltar	Un artículo cada vez, por construcción. El agarre es invisible	Como A, pero solo mientras cada acción conserve sus predicados de tipo
Tamaño	8 predicados, 3 acciones; 46 acciones instanciadas	6 predicados, 2 acciones; 42 acciones instanciadas; crece con artículos por conexiones	7 predicados, 3 acciones; 46 acciones instanciadas
Guía	Plan de 15 pasos	Plan de 11 pasos; muchos relevos equivalentes	Como A
Claridad	Se lee como la historia	El fichero más corto; los planes hay que interpretarlos	Un nombre menos; cuesta más ver las clases
Portabilidad	`:strips`	`:strips`	`:strips`

- B lleva las cajas por relevos a través del pasillo; C sin predicados de tipo: los medicamentos van solos a la sala, (`move meds1 pharmacy lobby`)

Veredicto

B si solo importa qué artículos van adónde; C con tipos de verdad (Cap. 4). Nos quedamos con A: sus planes lo dicen todo, y (`carrying ?r ?i`) hará falta después (contaminación, carga sobre la batería, un carro).

Errores comunes en STRIPS

Un efecto de borrado olvidado

```
(:action move
  :parameters (?r ?from ?to)
  :precondition (and (robot ?r) (location ?from) (location ?to)
                     (robot-at ?r ?from) (connected ?from ?to))
  :effect (robot-at ?r ?to))
```

`check` no encuentra nada; la prueba de la trampa da 8 pasos donde hacen falta 15: el robot deja una copia de sí mismo en cada habitación. *Remedio*: toda adición de una propiedad de un solo valor borra el valor antiguo.

Un objetivo que ya es cierto

`; cost = 0`: conoce la longitud esperada antes de ejecutar.

Una precondición que falta

Sin `(free ?r)` en `pick`: la trampa de 9 pasos. Los problemas típicos dan los mismos planes que el dominio correcto.

Un predicado mal escrito

`robot-in`: un aviso, «never true», así que `move` nunca se aplica. Lee los avisos.

Un problema que no encaja

Aridad errónea, predicado desconocido: *el hecho se ignora*, luego es falso.

1 Planificar: decidir antes de actuar

2 Un primer modelo: STRIPS

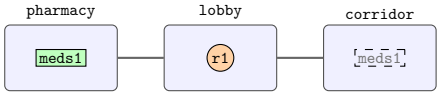
3 Qué hace un planificador con tu modelo

4 Resumen

Abrir la caja: un problema lo bastante pequeño para dibujarlo

Cuatro entregas, dos robots: el primer plan en milisegundos, después 3 segundos y medio millón de estados para demostrar que no hay uno mejor. Un modelador tiene que saber por qué.

```
(define (problem courier-p01)
  (:domain courier)
  (:objects r1 meds1 pharmacy lobby corridor)
  (:init
    (robot r1) (item meds1)
    (location pharmacy) (location lobby) (location corridor)
    (connected pharmacy lobby) (connected lobby pharmacy)
    (connected lobby corridor) (connected corridor lobby)
    (robot-at r1 lobby) (free r1)
    (item-at meds1 pharmacy))
  (:goal (item-at meds1 corridor)))
```



- 16 hechos en `:init`; 13 nunca cambian (tipos, mapa)
- Sin ellos: $s_0 = \{$`(robot-at r1 lobby)`, `(item-at meds1 pharmacy)`, `(free r1)`$\}$

Qué significa un modelo

- Acción instanciada a: $pre(a)$, $add(a)$, $del(a)$
- **Aplicable** en s: $pre(a) \subseteq s$
- **Sucesor**: $s' = (s \setminus del(a)) \cup add(a)$
- s **satisface** G: $G \subseteq s$
- **Plan**: $a_1, \ldots, a_n$, cada una aplicable a su vez, y el último estado satisface G. Puede ser largo, puede dar vueltas

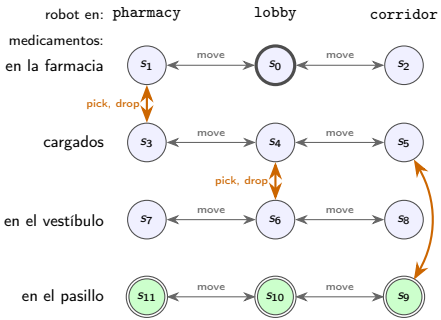
Tres conjuntos y una fórmula: todo el significado de una acción STRIPS.

`kairos validate` comprueba exactamente esto. Dos formas de fallar:

```
$ kairos validate courier-domain.pddl courier-p01.pddl
      courier-p01-shortcut.plan
courier-p01-shortcut.plan:4: error: step 3 (move r1 lobby
       corridor) at 2.000: precondition false: (robot-at
       r1 lobby) is false [validate.precondition_failed]
 4 | (move r1 lobby corridor)
plan invalid
```

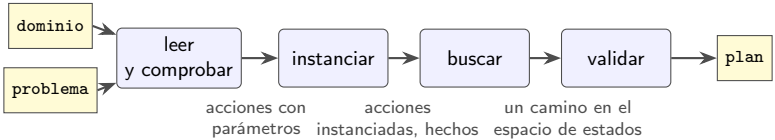
```
$ kairos validate courier-domain.pddl courier-p01.pddl
      courier-p01-halfway.plan
courier-p01-halfway.plan: error: the goal does not hold
      at the end of the plan: (item-at meds1 corridor)
      is false [validate.goal_not_reached]
plan invalid
```

El espacio de estados



- Aplica toda acción aplicable, una y otra vez: los **estados alcanzables** y sus flechas forman el **espacio de estados**
- Robot: 3 habitaciones; medicamentos: 3 o cargados. $3 \times 4 = 12$ estados
- Un plan es un camino de s_0 a un estado objetivo: aquí $s_0, s_1, s_3, s_4, s_5, s_9$
- Varios estados objetivo; aquí nada es irreversible
- El espacio no está en los ficheros: está *implícito*; el planificador construye lo que necesita

Instanciación: de variables a objetos



Contando a mano, problema diminuto (5 objetos):

- Todas las sustituciones: $3 \times 5^3 = 375$
- Los hechos estáticos podan: `move` necesita un robot y un hecho `connected`: $1 \times 4 = 4$
- `pick`: $1 \times 1 \times 3 = 3$; `drop`: 3
- **10 acciones instanciadas**; 8 átomos que pueden cambiar

- Análisis optimista: un hecho, una vez cierto, sigue siéndolo. Nunca desecha una acción necesaria
- Planta entera (p02): $14 + 8 + 8 = 30$ acciones, 18 átomos
- Torres de Hanói, 3 discos: $25 + 16 + 9 = 50$, incluida `(move d1 peg2 peg2)`, nunca aplicable

Lo que dice el planificador

```
$ kairos courier-domain.pddl courier-p01.pddl -v --threads 1
...
[   0.001s] [ground] grounded in 0.1 ms: 8 atoms, 0 numeric fluents, 10 actions, 0 durative actions, 0 axioms (50
      Datalog facts, 17 rules, 0 instances pruned)
[   0.001s] [sas] SAS+ in 0.1 ms: 3 variables (2 from 3 mutex groups, 1 binary), 3 invariants from 13 candidates, 0
      axiom layers
...
kairos: solved (proven optimal), cost 5
```

```
$ cat out/ground.txt
...
(:action (move r1 lobby corridor)
  :precondition (robot-at r1 lobby)
  :effect
    (not (robot-at r1 lobby))
    (robot-at r1 corridor))
...
```

- Han desaparecido 4 de 5 precondiciones: los hechos **estáticos** se comprueban una vez, al instanciar
- **Grupos mutex**: el robot está en una habitación, así que una variable con tres valores: 3 variables

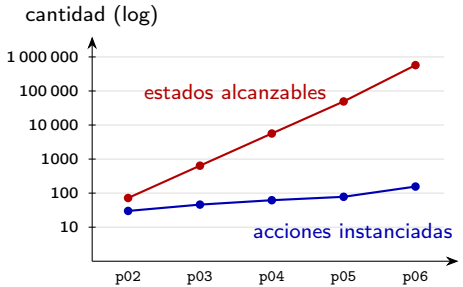
Por qué explota el espacio

Un robot, n artículos, 8 habitaciones:

$$\text{estados} = 8 \cdot \left(8^n + n \cdot 8^{n-1}\right)$$

Problema	Átomos	Acciones	Estados
p02: 1 artículo	18	30	72
p03: 2 artículos	27	46	640
p04: 3 artículos	36	62	5632
p05: 4 artículos	45	78	49 152
p06: 4 artículos, 2 robots	58	156	573 440

Puzle: 181 440 tableros alcanzables. Hanói: 3^n estados.



El modelo crece por suma; el espacio de estados, por multiplicación. Decidir si un problema STRIPS tiene plan es **PSPACE-completo** (Bylander, 1994): ningún planificador es rápido con todos los modelos.

Búsqueda hacia delante

Mantén una **lista abierta** y el conjunto de estados encontrados. Repite:

1. Saca un estado de la lista abierta; si está vacía, no hay plan
2. Si satisface el objetivo, el camino hasta él es un plan
3. **Expándelo**: acciones aplicables, sucesores
4. Añade los sucesores no encontrados antes

Todo está en el paso 1: *qué* estado sacar. El trabajo se mide en estados **expandidos** y **evaluados**.

Buscar a ciegas: en anchura

Saca el estado más antiguo. En el problema diminuto: 7 expansiones, un plan mínimo de 5. Se extiende como el agua.

El precio

En `courier-p05` (plan más corto: 28 acciones) recorrería la mayor parte de los 49 152 estados. Una búsqueda que no sabe nada del objetivo paga la explosión entera.

Heurísticas: ignorar los borrados

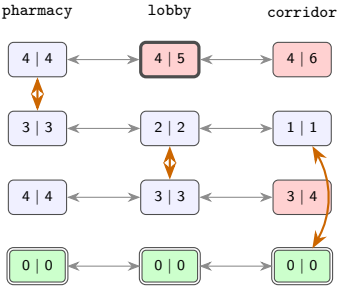
Una **heurística** h estima las acciones que aún faltan. **Voraz primero el mejor**: saca el estado con la h más pequeña. La estimación sale de una **relajación**: ignorar todo efecto de borrado.

Capa 0	Capa 1	Capa 2	Capa 3
robot-at lobby item-at pharmacy free	robot-at pharmacy robot-at corridor	carrying	item-at lobby item-at corridor
move lobby pharmacy move lobby corridor	pick pharmacy	drop corridor drop lobby	

```
(move r1 lobby pharmacy)
(pick r1 meds1 pharmacy)
(move r1 lobby corridor)      ; from the lobby: it never left
(drop r1 meds1 corridor)
```

Plan relajado, hacia atrás desde el objetivo: $h(s_0) = 4$, distancia real 5. Olvida la vuelta al vestíbulo. (HSP, FF; Kairos usa el FF y los hitos de LAMA.)

La heurística en todo el espacio



h | distancia real. Exacta en 9 estados; se queda corta en 3 (en rojo). A lo largo del plan: 4, 4, 3, 2, 1, 0, cuesta abajo. Dos sucesores con 4: una **meseta**.

```
$ kairos courier-domain.pddl courier-p02.pddl --debug=search --
      threads 1 --no-anytime
...
[   0.001s] [search] initial h: ff 6.000000, lmcount 6.000000
[   0.001s] [search] new best h 5 (1 expanded, 2 evaluated)
[   0.001s] [search] new best h 4 (4 expanded, 5 evaluated)
[   0.001s] [search] new best h 3 (5 expanded, 6 evaluated)
[   0.001s] [search] new best h 2 (6 expanded, 7 evaluated)
[   0.001s] [search] new best h 1 (7 expanded, 8 evaluated)
[   0.001s] [search] new best h 0 (8 expanded, 9 evaluated)
[   0.001s] [search] plan 0 with 7 actions, cost 7
...
```

Planta entera: 8 expansiones de 72 estados. `courier-p05`: primer plan tras 79 de 49 152. Por eso funcionan los planificadores.

Cuando la relajación ve poco

- Buena cuando la dificultad está en *conseguir* cosas; mala cuando las cosas *estorban*, porque estorbar se hace con borrados
- Torres de Hanói, 3 discos: el plan relajado mueve cada disco una vez, $h = 3$; la realidad es 7. Para n discos: n frente a $2^n - 1$

Discos	Estados (3^n)	h inicial	Primer plan	Plan más corto
3	27	3	14 acciones, 23 expandidos	7
4	81	4	18 acciones, 28 expandidos	15
6	729	6	77 acciones, 284 expandidos	63
8	6561	8	357 acciones, 3170 expandidos	255
10	59 049	10	1322 acciones, 22 333 expandidos	1023

Mensajero: 79 expansiones en 49 152 estados. Hanói: 22 333 en 59 049. Mismo tamaño; lo que cambia es cuánto ve la heurística, y eso depende de cómo se modela el problema.

Un plan, un plan mejor, el mejor plan

Satisfactoria: un plan, deprisa. **Óptima**: con garantía. Kairos es ***anytime***: primer plan enseguida, mejores después; el último es el mejor.

```
$ kairos puzzle-domain.pddl puzzle-p02.pddl --threads 1 -v
...
[   0.006s] [search] plan of cost 95 with 95 actions (440
      expanded, 441 evaluated)
[   0.010s] [search] plan of cost 51 with 51 actions (712
      expanded, 900 evaluated)
[   0.016s] [search] plan of cost 39 with 39 actions (1055
      expanded, 1477 evaluated)
[   0.050s] [search] plan of cost 31 with 31 actions (3239
      expanded, 5084 evaluated)
[   1.485s] [orchestrator] solved (proven optimal) in 1484.7
      ms
kairos: solved (proven optimal), cost 31
...
```

La demostración encontró los 181 440 tableros alcanzables.

	Primer plan		Demostración	
Problema	Acc.	Exp.	Acc.	Estados
p02	7	8	7	33
p03	13	21	13	397
p04	19	47	19	5054
p05	28	79	28	48 048
p06	31	393	27	544 039

- Encontrar, mejorar, demostrar: cada uno mucho más caro
- `--no-anytime`, `--timeout S` (30 por defecto), `--target-cost C`

No hay plan, o no se ha encontrado

- **Demostrado por la instanciación** (p07, sin enlace sala B–UCI): aun ignorando los borrados, el objetivo es inalcanzable; la nota es una cadena de razones
- **Demostrado por la búsqueda** (p08, la UCI de sentido único): la relajación ve un plan; la lista abierta se vacía tras 8 estados
- Puzle con el 7 y el 8 intercambiados: 181 440 estados. Ninguna heurística acorta una demostración
- Hanói, 12 discos: hay un plan de 4095 movimientos, pero 2 segundos no bastan

```
$ kairos hanoi-domain.pddl hanoi-p07.pddl --threads 1
      --timeout 2
...
kairos: timeout (timeout)
$ echo $?
11
```

Línea de estado	Salida	Significado
`solved (...)`	0	un plan
`unsolvable (exhausted)`	10	demostrado: no hay plan
`timeout (timeout)`	11	no se sabe nada

«Unsolvable» es una demostración. «Timeout» solo dice que la búsqueda se detuvo.

Para el modelador: los parámetros multiplican, las precondiciones estáticas podan

Un parámetro ocioso (`courier-extra-domain`)

```
(:action drop
  :parameters (?r ?i ?l ?to)      ; ?to is never used
  :precondition (and (robot ?r) (item ?i) (location ?l)
                     (robot-at ?r ?l) (carrying ?r ?i))
  :effect (and (not (carrying ?r ?i))
               (item-at ?i ?l) (free ?r))))
```

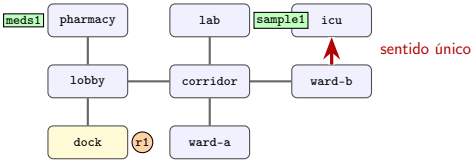
- Tres entregas: 62 acciones instanciadas pasan a 326 ($24 \times 12 = 288$ descargas)
- Planes aún correctos, `check` calla; síntoma: `(drop r1 meds1 ward-a dock)`

Sin `connected` **en** `move` (`courier-anywhere`)

```
$ kairos courier-anywhere-domain.pddl courier-
      anywhere-p01.pddl --threads 1 -v
...
[   0.001s] [ground] grounded in 0.1 ms: 18 atoms,
       0 numeric fluents, 80 actions, 0 durative
      actions, 0 axioms (166 Datalog facts, 16
      rules, 0 instances pruned)
...
; cost = 4
(move r1 dock pharmacy)
(pick r1 meds1 pharmacy)
(move r1 pharmacy ward-a)
(drop r1 meds1 ward-a)
```

Movimientos: de 14 a 64; acciones: de 30 a 80. El robot atraviesa paredes: **erróneo y más grande**.

Para el modelador: callejones sin salida y simetría



- **Callejón sin salida**: desde él no se alcanza el objetivo; acciones irreversibles
- p09: el plan de 16 pasos deja la UCI para el final; la relajación ve este callejón
- Los que no ve (p08) cuestan una búsqueda completa

Simetría: cuatro cajas idénticas para la sala A (p10)

Problema	Primer	Mejor	Estados (dem.)
1 caja	7	7	33
2 cajas	15	15	464
3 cajas	25	23	4670
4 cajas	33	31	43 595
1 caja, 3 robots	7	7	607

Misma situación para el hospital, estados distintos para el planificador. Daña la mejora y la demostración. No crees simetría: nada de robots de repuesto.

El método MODEL: tamaño y guía en cifras

- **Paso E**: ¿cómo de grande es el modelo, y es lo que esperaba? Estima acciones instanciadas y estados *antes* de ejecutar; lee después la línea `[ground]`. Muy por encima de la estimación: un error mientras no se demuestre lo contrario
- **Paso L**: ¿qué dicen las cifras? *Tamaño*: átomos y acciones instanciadas, y cómo crecen. *Guía*: expandidos hasta el primer plan, primero frente al mejor, estados encontrados para demostrar

Paso E en `courier-p05`: estimar y después ejecutar

Estimación. Movimientos $1 \times 14 = 14$; recogidas $1 \times 4 \times 8 = 32$; descargas 32: 78 acciones instanciadas. Átomos $8 + 4 \times 8 + 4 + 1 = 45$. Estados $8 \cdot (8^4 + 4 \cdot 8^3) = 49\,152$: se agotan en mucho menos de un segundo, así que podemos pedir la demostración.

Ejecución. `-v --threads 1`: 45 átomos, 78 acciones, primer plan de 28 tras 79 expansiones, `proven optimal` tras 48 048 estados.

Comparación. Todo cuadra. Con 326 acciones, habríamos buscado un parámetro sobrante antes de mirar ningún plan.

Alternativas comparadas: lo que ve el planificador

Tres entregas (medicamentos, muestra, comida). B, **movimientos cargados**: lo cargado sigue en algún sitio; `move-with` mueve también su `item-at`.

Plan de B: 18 acciones, termina *sin descarga*: entrar en el laboratorio con la muestra hace cierto `(item-at sample1 lab)`. C, **ir a cualquier parte**: un solo go, 56 pares `reachable`.

	A: base	B: movimientos cargados	C: ir a cualquier parte
Fidelidad	planes ejecutables tal cual	un artículo cargado cuenta como entregado	correcta solo si vale cualquier ruta
Tamaño	36 átomos, 62 acciones	36 átomos, 104 acciones; 14 movimientos más por artículo	36 átomos, 104 acciones; crece con el cuadrado de las habitaciones
Guía	primero: 19 acc., 47 exp.; demostr.: 5054 estados	primero: 18, 34 exp.; demostr.: 4567	primero: 15, 26 exp.; mejor: 12; demostr.: 4937
Claridad	un solo significado para `item-at`	`item-at` significa dos cosas	el problema enumera lo que el mapa implica
Portabilidad	STRIPS puro	STRIPS puro	STRIPS puro

Veredicto

Evita B: más grande, no más fácil, y su hecho de más dejó pasar un plan erróneo. C si la navegación halla la ruta igualmente; no si la ruta importa (puertas, batería). Nos quedamos con A.

Errores comunes con el planificador

Leer «timeout» como «no existe plan»

Salida 11: nada demostrado. Solo `unsolvable` (10) es una demostración. Mira el código de salida; da más tiempo antes de culpar al modelo.

Tomar el primer plan por el mejor

Puzle: primer plan de 95 deslizamientos, el mejor de 31. Lee el *último* plan y el estado: `proven optimal` es una garantía, `timeout` no.

Esperar una demostración que nadie necesita

p06: mejor plan a los 10 ms, demostración a los 3 s. Usa `--no-anytime`, `--timeout`, `--target-cost`.

Un parámetro que nada restringe

`(drop r1 meds1 ward-a dock)`, 326 acciones en vez de 62. Quítalo (Cap. 7: el *parámetro ocioso*).

Una precondición estática que falta

`(move r1 dock pharmacy)`, 80 acciones en vez de 30: más grande *y* erróneo. La estimación lo caza antes.

Juzgar un modelo con un problema diminuto

Un artículo: un milisegundo. Cuatro artículos, dos robots: 544 039 estados. Prueba con problemas dos o tres veces mayores.

1 Planificar: decidir antes de actuar

2 Un primer modelo: STRIPS

3 Qué hace un planificador con tu modelo

4 Resumen

Qué llevarse

- Planificar: decidir antes de actuar. Un **planificador** general más un **modelo** (dominio + problema); para cambiar el comportamiento, se cambia el modelo
- Planificación clásica: finito, determinista, totalmente conocido, estático, instantáneo; el curso levanta los supuestos uno a uno, con las versiones de **PDDL**
- STRIPS: hechos, mundo cerrado, acciones con precondición, adiciones y borrados; $s' = (s \setminus del(a)) \cup add(a)$; la **regla del marco**
- El **método MODEL**: Mapear, Objetos, Dinámica, Ensayar, Lectura crítica; ficha, tabla de estado, tabla de acciones, batería de pruebas (trivial, típica, sin plan, trampa), comparativa según fidelidad, tamaño, guía, claridad, portabilidad
- El planificador instancia (los parámetros multiplican, las precondiciones estáticas podan) y busca en un espacio que explota, guiado por una heurística que ignora los borrados
- Un plan, el mejor plan, una demostración: costes muy distintos. `unsolvable` es una demostración, `timeout` no

Ejercicios para practicar

- **1.1.2** Dos entregas en papel, en los dos órdenes: ¿por qué una es mucho mejor?
- **1.3.1** ¿Qué supuesto clásico rompe cada hecho del hospital?
- **1.7.1–1.7.3** Al teclado: medicamentos a la UCI, borrar `(free r1)`, validar un plan escrito a mano
- **1.8.1** Una ficha del modelo para un robot aspirador
- **2.4.2** Quita `(off porch)`: predice, ejecuta, lee la nota
- **2.7.3** El ferri: pasos M a E, a partir de una historia de tres líneas
- **3.2.1** Cuenta los átomos y las acciones instanciadas de `courier-p03`; compruébalo con `kairos -v`
- **3.5.1** Un plan relajado a mano, desde otro estado
- **3.9.2** Repara la codificación B para que un artículo cargado no cuente como entregado