El método en acción

Planificación y PDDL, Parte VII (Capítulos 19–20)

Francisco Martín Rico

Intelligent Robotics Lab, URJC



Índice

1. El método MODEL, completo
2. Caso de estudio: una misión robótica, de la historia a la ejecución
3. Resumen

1 El método MODEL, completo

2 Caso de estudio: una misión robótica, de la historia a la ejecución

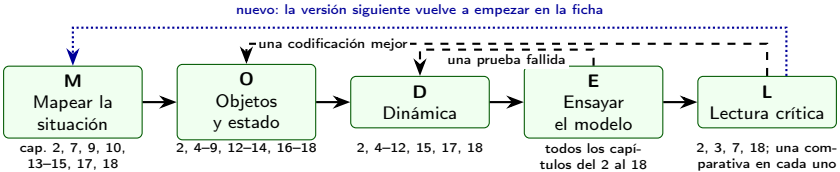
3 Resumen

El método en una página

Una lavandería, tres coladas, una lavadora, una secadora, un jersey de lana, una mañana que no es infinita. ¿De dónde sale la primera línea? El Capítulo 19 no enseña PDDL nuevo: reúne el método en un solo sitio.

Paso	Pregunta	Artefacto
M Mapear la situación	¿Cuál es el problema, quién actúa, qué cuenta como resuelto, qué se deja fuera?	**Ficha del modelo**: Historia, Agentes, Éxito, Alcance, Supuestos; luego la métrica, los deseos, las reglas, el calendario
O Objetos y estado	¿Qué cosas existen, y qué debe saber de ellas el planificador para decidir?	**Tabla de estado**: cada propiedad, estática o dinámica, cuántos valores, cómo se codifica
D Dinámica	¿Qué se puede hacer o puede ocurrir, cuándo, y qué cambia?	**Tabla de acciones**: cada acción, proceso y evento; cuándo, qué cambia, cuánto dura
E Ensayar el modelo	¿Hace el planificador lo que el mundo permitiría, y nada más?	**Batería de pruebas**: trivial, típica, sin plan, trampa; cada plan, leído y validado
L Lectura crítica	¿Qué otra codificación serviría, y cuál es mejor aquí?	**Comparativa**: alternativas según los cinco criterios, una decisión y su razón

Cinco pasos, tres bucles, cinco criterios



Criterio	Cómo lo mide el libro
Fidelidad	Ningún plan imposible (pruebas de la trampa, planes a mano que rechaza `kairos validate`); ninguna solución perdida (planes razonables a mano, aceptados)
Tamaño	Átomos y acciones instanciadas de la línea `[ground]` de `kairos -v`
Guía	Estados hasta el primer plan y el último; el primero frente al mejor; grupos mutex de `[sas]`; el motor
Claridad	Fichero y plan se leen como la historia; sitios que cambian cuando cambia una regla
Portabilidad	Requisitos frente a la tabla de planificadores del Apéndice B

Las cinco listas de comprobación (1): M y O

Una **lista de comprobación**: preguntas que se responden por escrito, en orden, cada vez. Un «no» rápido es una decisión: va al *Alcance*.

Lista M: la ficha del modelo

- **Agentes**: quién actúa, quién a la vez (7, 10)
- **Éxito**: obligatorio frente a deseado; precio de un deseo (14)
- **Calidad**: un número, ¿en qué unidad? ¿Importa *cuándo* (9, 10)?
- **Supuestos**: ¿un calendario (13)? ¿Cambia cuando nadie actúa (17)?
- **Reglas**, aparte de los objetivos: ¿una acción o todo el plan; puede ceder (15)?

Lista O: la tabla de estado

- ¿Estática o dinámica (2)? ¿Una constante (4)?
- ¿Cuántos valores: uno, a lo sumo uno (5, 16)?
- ¿Trata de una ausencia? ¿Gemelos acompasados (2, 5)?
- ¿Deducible; una regla sobre una cadena (6, 13)?
- ¿Una magnitud: umbral, cotas, contador (8, 9)?
- ¿Cambia por sí sola: interruptor y medida (17)?

Las cinco listas de comprobación (2): D, E y L

D: la tabla de acciones

- **¿Quién lo causa?** El agente: acción. El mundo, mientras: proceso. El mundo, en un instante: evento. *Imagina que el agente se queda dormido* (17)
- ¿Lo necesita, o es una regla (15)?
- Todo lo que cambia (2)
- Duración, solapamientos (10, 11)

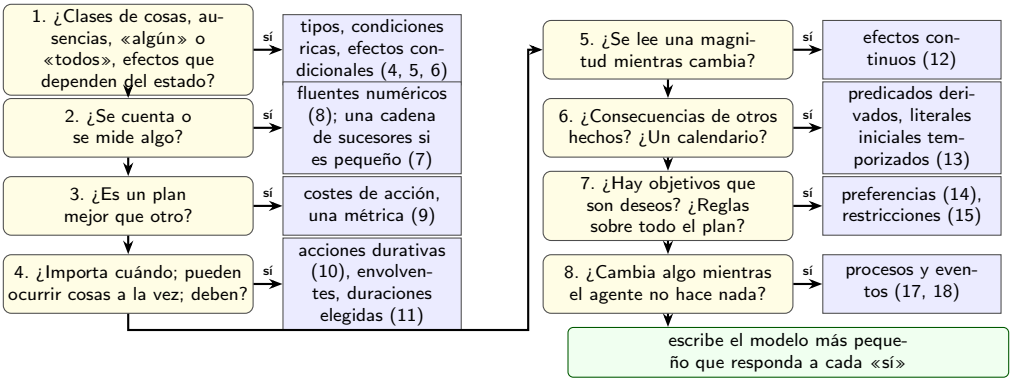
E: la batería de pruebas

- Primero estimar; lápiz (3, 18)
- `check`, resolver, leer, `validate`
- Cuatro pruebas, coste esperado conocido
- Trampas como ficheros de plan (7)
- Una prueba por promesa
- Sin plan: ¿una demostración? (11, 18)

L: la comparativa

- Todo patrón tiene un rival
- El **nivel más bajo** que no miente (18)
- Dos o tres, ejecutadas, con cifras
- Decisión y razón en la ficha

Qué característica para qué necesidad: la guía de decisión



Un «sí» añade una característica y la lectura sigue. Orden del libro, y del coste: **quédate en el nivel más bajo que no mienta sobre aquello de lo que depende el problema**.

De la necesidad a la característica (1): estado, condiciones, efectos

El problema necesita	Característica	Requisito	Cap.
clases de cosas, unas tratadas igual	tipos, jerarquía, `either`	`:typing`	4
un objeto que nombran las reglas	una constante	ninguno	4
dos parámetros que deben diferir	`(not (= ?a ?b))`	`:equality`	4
una condición sobre una ausencia	`not`	`:negative-preconditions`	5
varias formas de cumplir una condición	`or`, `imply`	`:disjunctive-preconditions`	5
una condición sobre algún objeto	`exists`	`:existential-preconditions`	5
condición u objetivo sobre todos los objetos	`forall`	`:universal-preconditions`	5
resultado según el estado; muchos objetos	`when`, `forall` en efectos	`:conditional-effects`	6
una magnitud que se suma, gasta, compara	funciones, `increase`, ...	`:numeric-fluents`	8
un plan mejor, suma de costes fijos	`(total-cost)`, `:metric`	`:action-costs`	9
un plan mejor, una medida del estado	`:metric` sobre fluentes	`:numeric-fluents`	9
una propiedad de un solo valor	un fluente de objeto	`:object-fluents`	16

`:adl` declara de una vez todos los requisitos por encima de la primera línea.

De la necesidad a la característica (2): tiempo, deseos, un mundo que se mueve

El problema necesita	Característica	Requisito	Cap.
acciones que duran y se solapan; hora de fin	acciones durativas, `(total-time)`	`:durative-actions`	10
algo cierto solo mientras dura una acción	una acción envolvente	sin indicador propio	11
una duración que es una decisión	cotas sobre `?duration`	`:duration-inequalities`	11
una magnitud que se lee mientras cambia	`#t` en una acción durativa	`:continuous-effects`	12
hecho que se sigue de otros, a toda profundidad	`:derived`	`:derived-predicates`	13
un calendario: horarios, ventanas, plazos	`(at t (fact))` en `:init`	`:timed-initial-literals`	13
números que cambian a horas conocidas	`(at t (= (f) v))`	`:timed-initial-fluents`	13
objetivos que son deseos, con un precio	`preference`, `is-violated`	`:preferences`	14
una regla sobre todo el plan	`always`, `within`, ...	`:constraints`	15
cambio mientras se cumple una condición	`:process`	`:time`	17
cambio cuando una condición se hace cierta	`:event`	`:time`	17
condición que ningún instante puede romper	`:constraint` en el dominio	`:time`	18

Lo que cuesta cada característica (1)

Característica	Compra (Fidelidad)	Cobra	Portabilidad
Tipos, constantes, =	mezclar clases se vuelve un error de `check`	nada: el mismo problema instanciado	7; `either` 5
Negación, `or`	ausencias y alternativas dichas directamente	`or` puede duplicar acciones	5
`forall`, `exists`	reglas que siguen a los objetos	se expanden sobre los objetos	5
Efectos condicionales	una acción para una sola cosa del mundo	los planificadores que no los tienen los compilan en más acciones	3
Predicados derivados	las consecuencias no pueden desacompasarse	se evalúan en cada estado	3
Fluentes numéricos	las cantidades, como datos	el espacio de estados deja de ser finito	5
Costes de acción; métricas	qué plan es mejor	el mejor plan, tarde; demostrado, más tarde	4; 4
Fluentes de objeto	un valor único no puede romperse	nada	1

Portabilidad: planificadores, de los ocho que compara la documentación de Kairos (Apéndice B), que devolvieron un plan válido para un problema pequeño con la característica, 10 s cada uno.

Lo que cuesta cada característica (2)

Característica	Compra (Fidelidad)	Cobra	Portabilidad
Acciones durativas	solapamiento, *makespan*	planes válidos, no los más cortos	5
Concurrencia requerida	lo que solo existe durante una acción	«no plan found» sin demostración	4
Duraciones elegidas	«todo lo que haga falta»	duraciones en sus cotas	3
Efectos continuos	un número leído mientras se mueve	los instantes se vuelven decisiones	2
Literales temporizados	el calendario del mundo	acciones: deben caber en ventanas	3
Preferencias	deseos con precio	optimización: el primer plan, quizá vacío	4
Restricciones	reglas dichas una vez, en todo estado	un monitor en cada estado	2
Procesos, eventos	un mundo que no espera	pasos de tiempo; reproducir planes	1

- Los que faltan no son los mismos: POPF y OPTIC no tienen negación, `or` ni efectos condicionales; LAMA, números
- A menudo compensa cambiar una característica baja por un patrón alto: predicados complementarios, una cadena de sucesores, bloques, un autómata

El catálogo de patrones: cuarenta nombres, cada uno con un rival

Clásicos (Capítulo 7, y 2–6)

Patrón	Rival
Tabla estática	regla en una acción
Hechos espejo	un hecho y un `or`
Invariante de a lo sumo uno	un fluente de objeto
Cadena de sucesores	huecos; un número
Testigo de fase, de turno	precondiciones
Cerrojo de recurso	envolvente (tiempo)
Par inverso, sin vuelta atrás	el uno del otro
Acción fusionada	acciones separadas
Predicado complementario	un `not`
Una acción por caso	un `when`
Deducido, no almacenado	un hecho almacenado

Números, tiempo, PDDL 3, híbridos (8–18)

Patrón	Rival
Magnitud acotada	un hecho de umbral
Fluente contador	costes de acción
Testigo	ninguno: se solapan
Envolvente y contenido	un cerrojo
Holgura	simultaneidad exacta
Duración elegida, bloques	el uno del otro
Ventana, plazo	restricciones
Objetivo blando	un objetivo estricto
Monitor	una restricción
Inicio-proceso-parada	una acción durativa
Restricción global	un evento y un objetivo

El nombre dice qué es una fila de la tabla de estado o de acciones; el rival es una columna del paso L.

Antipatrones: PDDL válido, detectado por el síntoma

Pasan `kairos check`: `valid, plannable`. Cuatro familias de síntomas, y la familia dice dónde mirar.

Demasiado fácil: el modelo permite demasiado

Demasiado corto, barato o pronto. Borrado olvidado (2, 7); consumo sin la precondición (8); una acción que olvidó su coste (9); un testigo olvidado (10); un límite comprobado solo al principio (12)

Demasiado difícil: el modelo permite poco

No hay plan, o rodeos extraños. Un valor inicial olvidado (8, 16); gemelos desacompasados (5, 7); una regla más fuerte que el mundo (15); igualdad con fracciones (8) o sobre un flujo (18)

Demasiado grande: algo fuera de control

Más acciones instanciadas o estados de los estimados. Parámetro ocioso (3, 7); un horizonte que nadie acotó (18)

No es lo que se pidió

Correcto para el modelo, otra pregunta. Preferencias sin métrica (14); pesos en la unidad equivocada (14); una restricción donde hacía falta un evento (18)

Depurar: primero, leer el veredicto

Veredicto	Código	Qué dice, y un ejemplo
un plan	0	Válido para el modelo. `solved` no es `proven optimal` (9)
`unsolvable`	10	**Una demostración**: no existe plan (3). `laundry-v1-p03`
`no plan found`	14	La búsqueda terminó sin plan y sin demostración (11, 12, 17). `laundry-v5-p03`
`timeout`	11	Nada: la búsqueda seguía en marcha (3)
`invalid`	2	Los ficheros no dicen lo que se quería decir: lee el primer error (1)

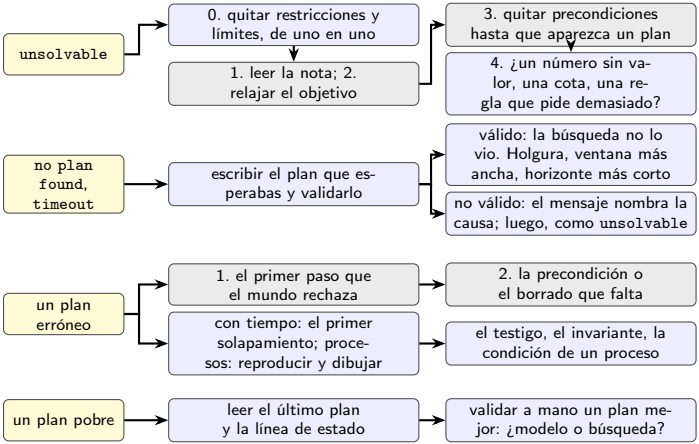
Solo un veredicto es una demostración

Los guiones que ejecutan una batería de pruebas (`tests.txt`) indican el código de salida que espera cada prueba.

Sin plan, sin demostración

Escribe el plan que esperabas y ejecuta `kairos validate`. **No válido**: el mensaje nombra la causa. **Válido**: la búsqueda no lo vio.

El procedimiento de depuración, ampliado



Gris: el procedimiento del Capítulo 7

Azul: lo que añaden los números, el tiempo, las restricciones y los procesos

La «cosa» errónea es un **paso** (secuencial), un **solapamiento** (temporal), un **tramo de tiempo** (procesos)

Lo que no encuentra el planificador de este libro

Cuando el plan escrito a mano es válido, el modelo está bien y la búsqueda no lo vio. Con Kairos 1.0.6:

Cuatro planes que se le escapan

- **Una duración entre sus cotas**: solo se prueban las cotas. *Cómo sortearlo*: que algo la necesite, o bloques (11)
- **La misma envolvente usada dos veces** (11). *Cómo sortearlo*: escribe el plan a mano y valídalo
- **Dos inicios en el mismo instante** (11). *Cómo sortearlo*: holgura
- **Entre dos pasos de 0,1**: ventanas estrechas que se pasan de largo (17, 18). *Cómo sortearlo*: ventanas $\geq$ ritmo $\times$ 0,1; desigualdades

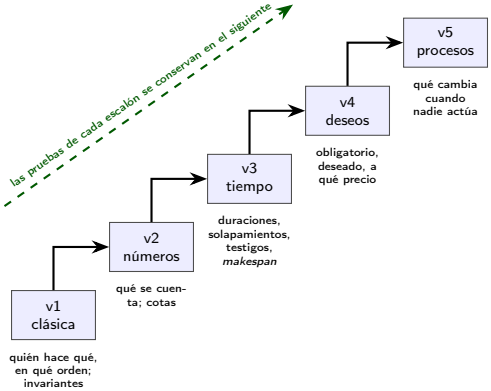
Dos que tienen que ver con los planes buenos

- Un plan temporal es un horario válido, no el más corto (11)
- En PDDL+ la búsqueda minimiza el número de acciones, diga lo que diga la métrica: los planes esperan y cargan más (18)

En todos los casos

El plan que imprime Kairos es válido. Lo que puede faltar es un plan, o uno mejor. Una prueba «sin plan» necesita un **horizonte acotado**.

Hacer crecer un modelo por versiones



Reglas para hacer crecer un modelo

1. **Una dimensión por versión**
2. **En verde antes del salto**: todas las pruebas, como se esperaba
3. **Primero cambia la ficha**, luego las tablas, luego el PDDL
4. **Las pruebas antiguas se traducen y se conservan**; un veredicto que cambia lo explica una línea de la ficha
5. **Pruebas nuevas**: un «sin plan» y una trampa que la versión anterior no podía enunciar
6. **Cada versión es un fichero**
7. **Para** cuando el modelo ya no mienta

La lavandería: historia y versión 1

Sábado en la lavandería. Coladas que se lavan, secan y doblan. La etiqueta dice a qué máquinas va una prenda; la lana, nunca a la secadora. Una lavadora, una secadora.

Paso M: la ficha, versión 1

Agentes	La persona; las máquinas, herramientas
Éxito	Cada colada del objetivo, doblada
Alcance	Dentro: sucia, mojada, doblada; dónde; etiquetas. Fuera: tiempo, kilos, monedas, tendedero
Sup.	Clásicos. Una colada por máquina. Un ciclo nunca falla

Patrones: `allowed` tabla estática; `in` a lo sumo uno; `empty` cerrojo de recurso; `put/take-out` par inverso; el resto, sin vuelta atrás

```
(:predicates
  (dirty ?l - load)
  (wet ?l - load)
  (folded ?l - load)
  (in-basket ?l - load)               ; out of every machine
  (in ?l - load ?m - machine)
  (empty ?m - machine)
  (allowed ?l - load ?m - machine))  ; static: the label

(:action put
  :parameters (?l - load ?m - machine)
  :precondition (and (in-basket ?l) (empty ?m) (allowed ?l ?m))
  :effect (and (not (in-basket ?l)) (not (empty ?m))
               (in ?l ?m)))
(:action wash
  :parameters (?l - load ?w - washer)
  :precondition (and (in ?l ?w) (dirty ?l))
  :effect (and (not (dirty ?l)) (wet ?l)))
;; Only what is clean and dry is folded
(:action fold
  :parameters (?l - load)
  :precondition (and (in-basket ?l) (not (dirty ?l))
                     (not (wet ?l)))
  :effect (folded ?l)))
```

Versión 1: las cuatro pruebas

Trivial: una colada, siete pasos, escritos antes de ejecutar.

```
$ kairos laundry-v1-domain.pddl
    laundry-v1-p01.pddl
; cost = 7
(put towels w1)
(wash towels w1)
(take-out towels w1)
(put towels d1)
(dry towels d1)
(take-out towels d1)
(fold towels)
```

Típica: tres coladas, 21 pasos, óptimo demostrado. **Sin plan**: el jersey.

```
$ kairos laundry-v1-domain.pddl laundry-v1-p03.
    pddl
...
kairos: unsolvable (exhausted)
```

Trampas como ficheros de plan: dos coladas en un lavado; doblar toallas mojadas.

```
... step 2 (put colors w1) at 1.000:
    precondition false: (empty w1) is false ...
plan invalid
... step 4 (fold towels) at 3.000: precondition
    false: (wet towels) is true ...
plan invalid
```

Versión 2: números

```
(:functions
  (weight ?l - load)                    ; kg, static
  (room ?m - machine))                  ; kg still free in the drum

(:action put
  :parameters (?l - load ?m - machine)
  :precondition (and (in-basket ?l) (allowed ?l ?m)
                     (>= (room ?m) (weight ?l)))
  :effect (and (not (in-basket ?l)) (in ?l ?m)
               (decrease (room ?m) (weight ?l))))
;; One cycle washes every load in the drum
(:action wash
  :parameters (?w - washer)
  :effect (forall (?l - load)
            (when (in ?l ?w)
              (and (not (dirty ?l)) (wet ?l)))))
```

- Ficha: entran los kilos; un tambor admite hasta su capacidad; un ciclo trata todo lo de dentro
- `empty` desaparece: `room`, una magnitud acotada
- Un salto arrastró una característica del Capítulo 6
- La trampa antigua (dos coladas) ahora es legal: regla 4

Coladas de 3, 4 y 5 kg, tambores de 8 kg:

```
; better plan 2
; cost = 19
(put towels w1)
(wash w1)
...
(put whites w1)
(put colors w1)
...
```

Edredón de 9 kg: `unsolvable`. Tres coladas en un lavado:

```
... 1 >= 5 is false ...
plan invalid
```

Versión 3: tiempo

```
(:durative-action put
  :parameters (?l - load ?m - machine)
  :duration (= ?duration 1)
  :condition (and (at start (in-basket ?l))
                  (at start (hands-free))
                  (at start (>= (room ?m) (weight ?l)))
                  (over all (allowed ?l ?m))
                  (over all (stopped ?m)))
  :effect (and (at start (not (in-basket ?l)))
               (at start (not (hands-free)))
               (at start (decrease (room ?m) (weight ?l)))
               (at end (in ?l ?m))
               (at end (hands-free))))
;; The machine works alone: the person is free meanwhile
(:durative-action wash
  :parameters (?w - washer)
  :duration (= ?duration (cycle ?w))
  :condition (and (at start (stopped ?w))
                  (at start (exists (?l - load) (in ?l ?w))))
  :effect (and (at start (not (stopped ?w)))
               (at end (stopped ?w))
               (forall (?l - load)
                 (when (at end (in ?l ?w))
                   (at end (and (not (dirty ?l)) (wet ?l)))))))
```

Ficha: *lo antes posible*; una cosa cada vez; las máquinas funcionan solas; minutos.

Acción	Dur.	Debe cumplirse
`put, take-out`	1	parada, todo el tiempo
`wash, dry`	ciclo	parada al inicio; algo dentro
`fold`	5	limpia y seca al inicio

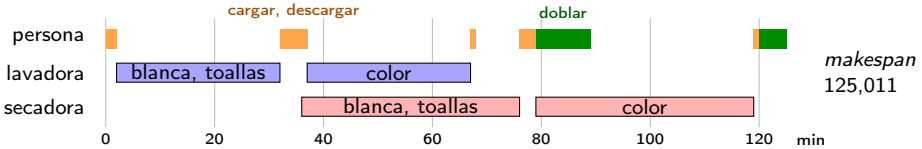
- Dos **testigos**: `(hands-free)`, `(stopped ?m)`
- Un **invariante**: nadie abre una máquina en marcha

Versión 3: un ciclo vacío, y luego el horario honrado

La primera versión 3 (`laundry-v3-idle`):

```
84.010: (fold whites) [5.000]
90.011: (wash w1) [30.000]
119.009: (take-out colors d1)
    [1.000]
```

- Un ciclo sin nada dentro: termina antes que el plan, así que no cuesta nada
- El salto cambió aquello que le da igual al planificador
- Plan erróneo: falta una precondición, *algo en el tambor*



La secadora es el cuello de botella: ningún horario puede ser mucho más corto que 125. Conoce el *makespan* honrado antes de fiarte del planificador.

Versión 4: deseos

Deseo	Condición	Peso
(obligatorio)	ropa blanca doblada	
`colors-done`	ropa de color doblada	60
`towels-done`	toallas dobladas	30

```
(:goal (and (folded whites)
            (preference colors-done (folded colors))
            (preference towels-done (folded towels))))
(:metric minimize (+ (total-time)
                     (* 60 (is-violated colors-done))
                     (* 30 (is-violated towels-done)))))
```

- Pasos O, D sin cambios; `:preferences`; el salto está en el problema
- El color en el tambor: 9 min por 60. Las toallas: una segunda tanda, 37 min por 30
- Prueba del precio: color a 5 (p04): fuera, coste 93,010. Equilibrio en 9

```
; cost = 169.006
; makespan = 79.006
...
; better plan 2
; cost = 118.01
; makespan = 88.010
```

Un plan pobre (p03, toallas a 40):

```
; cost = 128.01
plan valid, makespan 125.011, metric
     125.011
```

Plan a mano mejor: el modelo está bien, la búsqueda (M2) se quedó corta.

Versión 5: procesos

```
;; Nobody chooses this: it runs while its condition holds
(:process sun-dry
  :parameters (?l - load)
  :precondition (and (hanging ?l) (wet ?l) (sunny))
  :effect (decrease (moisture ?l) (* #t (sun-rate))))

;; ... and this happens at the instant the load is dry
(:event dried
  :parameters (?l - load)
  :precondition (and (wet ?l) (<= (moisture ?l) 0))
  :effect (not (wet ?l))))
```

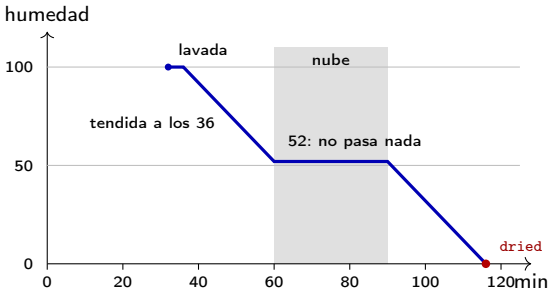
- Ficha: *el tendedero entra*
- Lavar y la secadora siguen siendo durativas: duración conocida, nada las interrumpe
- El lavado pone la humedad a 100

Lista D: ¿quién lo causa?

- La persona se queda dormida tras tender: el jersey se seca igual. No es una acción
- Se seca mientras está tendido y brilla el sol: un **proceso**
- Seco en un instante: un **evento**
- El sol sigue un calendario: **literales iniciales temporizados**

Interruptor: `(hanging ?l)`, `(sunny)`. Medida: `moisture`. La antigua prueba «sin plan» cambia de veredicto: sol toda la mañana (`p01`), doblado a los 90.

Versión 5: una nube, una reproducción, un veredicto



p02: ropa blanca, lana, una nube del 60 al 90. Plan de 169 min; la lana se recoge a los 116, cuando se disparó el evento. La ropa blanca espera mojada del 33 al 122.

```
$ kairos validate laundry-v5-domain.
    pddl laundry-v5-p02.pddl
    laundry-v5-p02-hand.plan
plan valid, makespan 122, metric 122
```

- Trampa: secar a la sombra (`p02-shade`), rechazada: `(wet wool) is true`
- Sin plan: nublado (`p03`)

```
kairos: no plan found (exhausted)
```

Salida 14: el lápiz da la demostración; sin sol, la humedad se queda en 100.

Cinco versiones, una al lado de otra

	Añade a la ficha	Prueba típica (p02)	Búsqueda (`kairos -v`)
v1	orden del trabajo; una colada por máquina	21 pasos, óptimo	808 estados, 3 ms
v2	kilos; un ciclo trata todo el tambor	19 pasos, óptimo	2106 estados, 12 ms
v3	duraciones; qué puede solaparse; el *makespan*	125,011 min	101 696 estados, 2,1 s
v4	obligatorio y deseado; precios en minutos	coste 118,010	5358 estados, 20 ms
v5	el tendedero, el sol, la nube	169 min (122 a mano)	168 296 estados, 2,0 s

- Cada versión dice algo cierto que la anterior no podía, cuesta más de buscar y funciona en menos planificadores (v2: 3 de 8; v5: 1)
- Las versiones **no son borradores**: qué coladas caben juntas, mejor en v2, con demostración en milisegundos; una mañana corriente, v3; v5 solo los días con lana
- Problemas distintos: la última columna compara órdenes de magnitud

Paso L: ¿es el tendedero un proceso?

Tres codificaciones sobre la versión 3, la misma mañana (lana, ropa blanca, nube del 60 al 90):

	A: duración fija (50 min)	B: dentro de una ventana	C: proceso y evento
Fidelidad	plan imposible: lana doblada mojada; correcto solo sin nubes	ningún plan imposible; pierde soluciones: el secado no hace pausas	exacta: se detiene bajo la nube, se reanuda
Guía	10 407 estados; *makespan* 91,005	9535 estados; 146,002	168 296 estados; 169, 122 a mano
Claridad	una acción más	lo mismo, una condición	la ley del tendedero, aparte; hay que reproducir los planes
Portabilidad	durativas, literales temporizados	lo mismo	procesos: solo Kairos

- A: donde uno se puede fiar del sol, o una primera versión. B: honrado en el nivel temporal, si los ratos de sol son largos. C: cuando el secado se detiene y se reanuda
- La decisión depende de una pregunta que nadie hizo: *¿hasta qué punto es fiable el sol?* Vuelta a la ficha

Documentar un modelo

```
;; Story    One person washes, dries and folds loads of clothes
;;          with a washer, a dryer and a clothesline.
;; Decides  The person: put, take-out, wash, dry, fold, hang,
;;          collect. The world: sun-dry (process), dried (event).
;; Out      Detergent, programs, ironing, a second person.
;; Units    Minutes, kilograms, moisture in percent.
;; Rules    R1 a drum holds up to its capacity     (put)
;;          R2 a cycle treats what is in the drum  (wash, dry)
;;          R3 the door stays shut during a cycle  (stopped)
;;          R4 the person does one thing at a time (hands-free)
;;          R5 only clean, dry loads are folded    (fold)
;;          R6 the line dries only in the sun      (sun-dry)
;; Keeps    A load is in the basket, in one machine or on the
;;          line. room = capacity minus the weights inside.
;; Problem  Gives weight, room, cycle, sun-rate and allowed;
;;          moisture 0 for every load; the sun as timed
;;          literals; the wishes and their weights.
;; Tests    tests.txt. p01 trivial, p02 typical, p03 no plan,
;;          p04 wishes; p02-shade.plan is the cheat.
;; History  v1 classical. v2 capacity in kilograms. v3 time.
;;          v4 wishes. v5 the clothesline.
```

- **Cabecera** para quien lee el fichero. *Decides*: acciones frente a mundo. *Units*: el error silencioso. *Rules*: dónde se impone cada una. *Problem*: el contrato
- **Ficha del modelo** para el dueño del problema; añade Calidad, Deseos, Reglas, **Nivel**, **Decisiones**, **Límites conocidos**, **Versiones**
- **Fichero de pruebas** para quien lo cambie: `tests.txt`, una línea por prueba con su código de salida

Límites conocidos (lavandería)

Los planes de v4 pueden mejorar cuando una segunda tanda podría solaparse; los de v5 no tienen prisa.

Errores comunes en la manera de trabajar

Una trampa que pasa por la razón equivocada

La trampa de v2 ejecutada en v3 se rechaza por `duration 0.000000 does not satisfy`, no por los 12 kg. En verde y no prueba nada. *Corrección*: traduce los ficheros de plan; lee cada mensaje

Una cabecera de una versión anterior

`laundry-v2-stale`: `warning: uses conditional-effects but :requirements does not include` el requisito. *Corrección*: copia los requisitos de la lista que imprime `check`

Una prueba «sin plan» con el horizonte abierto

Sol hasta el 50 (p05): 3 s y 371 765 estados, todo el tiempo que se le dé. Nublado (p03): 8093 estados, menos de 0,1 s. *Corrección*: acota el horizonte

Un veredicto que sigue aunque cambie la ficha

El jersey esperó la salida 10 durante cuatro versiones; en v5 tiene un plan de 90 min. *Corrección*: un veredicto solo cambia con una línea de la ficha

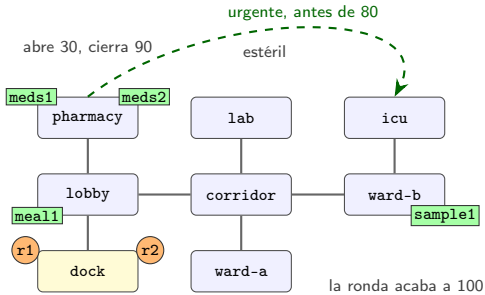
1 El método MODEL, completo

2 Caso de estudio: una misión robótica, de la historia a la ejecución

3 Resumen

La ronda de la mañana, según la describe la responsable

- Dos mensajeros, `r1` y `r2`, en la base de carga, **un cargador**; baterías 100 y 30; 1 unidad por metro; +5 por s; *se queda hasta llenarse*
- Urgente: `meds1` a la UCI **antes de 80**. Rutinarias: `meds2` a la sala A; `sample1` de la sala B al laboratorio; `meal1` del vestíbulo a la sala B
- La farmacia **abre a 30** y cierra a 90
- Laboratorio estéril: muestras **selladas**; entregar solo en el destino
- La ronda acaba a 100. Después, medicamentos, muestra, comida; robots en casa *si es posible*



un cargador; batería `r1` 100, `r2` 30

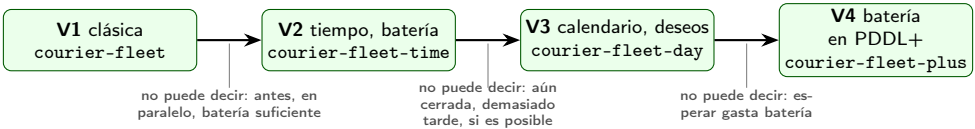
El paso M es una conversación

- *¿Dos cosas a la vez?* No: un compartimento
- *¿Dejar el cargador antes de llenarse?* No. **Decide una codificación**
- *¿Pasar una caja, o dejarla por el camino?* No: «nada de cajas en los pasillos»
- *¿Una muestra sin sellar en el laboratorio?* «No puede pasar»: no se hace, no se pierde
- *«Si es posible», ¿comparado con qué?* En casa vale menos que cualquier entrega
- *¿Batería mientras espera?* «Un poco»: fuera al principio

Cada respuesta es una línea de la ficha del modelo.

Un plan para el modelado

Crecer por versiones; saltar solo cuando la actual no puede decir algo que importa.



Paso M: la ficha del modelo de la ronda de la mañana

Ficha del modelo

Agentes	`r1`, `r2`: a la vez, una cosa cada uno. Farmacia, cargador, reloj: un horario o una ley
Éxito	Obligatorio: `meds1` en la UCI antes de 80. Deseado: las demás entregas antes de 100; los robots en la base de carga
Alcance	Dentro: habitaciones, distancias, posiciones, qué se lleva, sellado, duraciones, batería, horarios, plazos. Fuera: rutas dentro de una habitación, personas, agarre, fallos, incertidumbre
Supuestos	Un artículo por robot; entrega en el destino; un robot en el cargador, hasta llenarse; 1 por metro, nada parado (revisado en v4); farmacia 30–90, plazo 80, fin 100

Deseo	Condición	Peso
`meds2-done`	`meds2` en la sala A	5
`sample-done`	`sample1` en el laboratorio	4
`meal-done`	`meal1` en la sala B	2
`r1-home, r2-home`	el robot acaba en la base	1 cada uno

El orden de la responsable en números: $5 > 4 > 2 > 1$. Comprobado con ella: nunca se renuncia a una entrega para llevar los robots a casa.

Versión 1: quién lleva qué

```
; Nobody enters a sterile room with a sample that is not
; sealed: the rule of the lab, asked of every move.
(:action move
  :parameters (?r - robot ?from ?to - location)
  :precondition (and
    (robot-at ?r ?from) (connected ?from ?to)
    (or (not (sterile ?to))
        (forall (?s - sample)
          (or (not (carrying ?r ?s)) (sealed ?s)))))
  :effect (and (not (robot-at ?r ?from))
               (robot-at ?r ?to)))
; Items are handed over at their destination and nowhere else:
; there is no drop.
(:action deliver
  :parameters (?r - robot ?i - item ?l - location)
  :precondition (and (robot-at ?r ?l) (carrying ?r ?i)
                     (destination ?i ?l))
  :effect (and (not (carrying ?r ?i)) (free ?r)
               (item-at ?i ?l) (delivered ?i))))
```

- Tipos `medicine`, `sample`, `meal` bajo `item`
- `destination`: una **tabla estática**; las peticiones son datos
- Nada de «contaminada»: el estado malo es inalcanzable, no necesita nombre
- «Nada de cajas en los pasillos» eliminó `drop`: `deliver` es una **acción sin vuelta atrás**, a propósito
- `(not (delivered ?i))` en `pick`
- La regla del laboratorio: una precondición de `move`; el `forall` no depende de que haya un solo compartimento

Versión 1: la batería de pruebas

La ronda (p02), robots en casa:

```
; better plan 2
; cost = 31
(move r2 dock lobby)
(move r2 lobby pharmacy)
(pick r2 meds2 pharmacy)
...
(move r2 ward-b corridor)
(seal r2 sample1)
(move r2 corridor lab)
(deliver r2 sample1 lab)
...
```

- Sellada un movimiento antes del laboratorio: correcto
- La caja urgente es la tercera entrega de cuatro
- Las 31 acciones son de r2: `solved (proven optimal)`, `cost 31`. El modelo cuenta acciones

Prueba	Fichero	Esperado y obtenido
Trivial	p01	5 pasos
Típica	p02	31 pasos, óptimo demostrado
Sin plan	p03	`unsolvable`: puerta de la UCI cerrada
Trampa	p04, p05	`unsolvable`; se rechaza el plan sin `seal`

```
... step 3 (move r1 corridor lab) at 2.000:
    precondition false: (or (not (sterile ?to)) ...
plan invalid
```

Versión 1, paso L: tres maneras de escribir la regla del laboratorio

A, la física: un efecto condicional marca `contaminated`. **B**, una precondición (el dominio). **C**, una restricción:

```
; Never in a sterile room with a sample that is not sealed.
(:constraints
  (always (forall (?r - robot ?s - sample ?l - location)
    (imply (and (carrying ?r ?s) (robot-at ?r ?l) (sterile ?l))
           (sealed ?s)))))
```

Las tres: 31 pasos, p04 irresoluble, el plan sin sellar rechazado, cada una con sus palabras.

Veredicto

Tamaño y Guía no deciden. A, fuera por Fidelidad. Nos quedamos con **B**, que viaja a un ejecutor; **C**, como prueba: valida cualquier plan de B con el dominio de C. B vigila la puerta; C, la habitación.

	A: efecto condicional	B: precondición	C: restricción
Fidelidad	las muestras se estropean	lo que dijo ella; vigila la puerta	cada estado
Tamaño	54 acciones instanciadas, 40 hechos	80 (el `or` divide `move`), 41 hechos	54, 3 axiomas
Guía	10 749 estados; un callejón sin salida tardío	10 432 estados	10 749 estados
Portabilidad	`:conditional-effects`	precondiciones ADL	`:constraints`: pocos

Versión 2: tiempo y batería

```
(:durative-action move
  :parameters (?r - robot ?from ?to - location)
  :duration (= ?duration (/ (distance ?from ?to) (speed ?r)))
  :condition (and
    (at start (idle ?r))
    (at start (robot-at ?r ?from))
    (at start (connected ?from ?to))
    (at start (>= (battery ?r)
                  (* (distance ?from ?to) (consumption ?r))))
    (at start (or (not (sterile ?to))
                  (forall (?s - sample)
                    (or (not (carrying ?r ?s)) (sealed ?s))))))
  :effect (and
    (at start (not (idle ?r)))
    (at start (not (robot-at ?r ?from)))
    (at start (decrease (battery ?r)
                (* (distance ?from ?to) (consumption ?r))))
    (at end (robot-at ?r ?to))
    (at end (idle ?r))))
```

El Alcance añade duraciones, trabajo en paralelo, batería, carga.

Nuevo en el estado	Codificado como
no hace nada	`(idle ?r)`
un cargador; libre	`(has-charger ?l)`
	`(charger-free ?l)`
metros; velocidad	`(distance ?f ?t)`
	`(speed ?r)`
carga, capacidad	`(battery ?r)`
por metro; por s	`(consumption ?r)`
	`(charge-rate ?r)`

- Testigo `(idle ?r)`: se toma al inicio, se devuelve al final
- La batería de un movimiento, entera al inicio: basta si nadie la lee mientras tanto

Versión 2: una carga llena la batería

```
(:durative-action charge
  :parameters (?r - robot ?l - location)
  :duration (= ?duration (/ (- (battery-capacity ?r)
                               (battery ?r))
                            (charge-rate ?r)))
  :condition (and
    (at start (idle ?r))
    (at start (has-charger ?l))
    (at start (charger-free ?l))
    (at start (< (battery ?r) (battery-capacity ?r)))
    (over all (robot-at ?r ?l)))
  :effect (and
    (at start (not (idle ?r)))
    (at start (not (charger-free ?l)))
    (at end (assign (battery ?r) (battery-capacity ?r)))
    (at end (charger-free ?l))
    (at end (idle ?r)))))
```

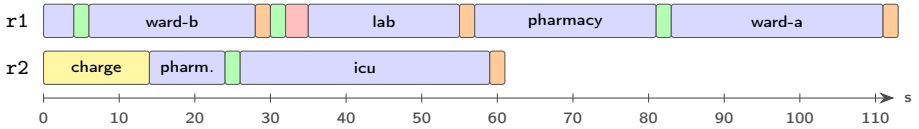
El cargador es un **cerrojo de recurso**: se borra al inicio, se añade al final.

Acción	Duración	Lo nuevo con el tiempo
`move`	distancia / velocidad	batería al inicio; en ningún sitio mientras se mueve
`pick`	2 s	se queda (`over all`); cargado al final
`seal`	3 s	sellada al final
`deliver`	2 s	entregado, libre al final
`charge`	lo que falta / ritmo	cargador al inicio; llena al final

Versión 2: la ronda en el tiempo

```
; better plan 2
; cost = 113.01700000000002
; makespan = 113.017
0.000: (move r1 dock lobby) [4.000]
0.000: (charge r2 dock) [14.000]
4.001: (pick r1 meal1 lobby) [2.000]
...
111.017: (deliver r1 meds2 ward-a)
    [2.000]
```

- Los dos robots trabajan desde el primer segundo
- r2 carga de 30 a 100 en $70/5 = 14$ s, lleva la caja urgente 43 m, llega con 57
- r1: comida, muestra sellada en la sala B, caja rutinaria: 98 m con 100
- De «31 acciones» a 113 s: el modelo puede decir *antes*
- Azul: movimientos; verde: pick; naranja: deliver; rojo: seal; amarillo: charge



Versión 2: la prueba de la trampa se convierte en una familia

Sin plan (p03): 20 de batería en la sala B, la base de carga a 26 m.

```
kairos: no plan found (exhausted)
```

Salida 14, no 10: los motores temporales no lo demuestran; nuestra aritmética, sí.

Dos trampas a mano: moverse mientras recoge; dos robots en un cargador.

```
... at start condition false: (idle r1) is
     false ...
plan invalid
... at start condition false: (
    charger-free dock) is false ...
plan invalid
```

Turnos (p04): los dos robots con 10, una entrega cada uno.

```
; makespan = 65.605
0.000: (move r2 dock lobby) [4.000]
0.000: (charge r1 dock) [18.000]
4.001: (pick r2 meal1 lobby) [2.000]
6.002: (move r2 lobby dock) [4.000]
18.001: (move r1 dock lobby) [4.000]
18.001: (charge r2 dock) [19.600]
...
```

r2 va a por la comida mientras r1 carga, y vuelve en el instante en que el cargador queda libre. No habíamos pensado en ello, y es correcto.

Versión 2, paso L: ¿cuánto dura una carga?

Duración **elegida** (de 1 s hasta llenarse):

```
; makespan = 107.015
0.000: (charge r2 dock) [1.000]
0.000: (move r1 dock lobby)
    [4.000]
1.001: (charge r2 dock) [1.000]
2.002: (charge r2 dock) [1.000]
3.003: (charge r2 dock) [1.000]
...
```

Cuatro cargas de 1 s: duraciones en sus cotas. Válido, 6 s más corto. Continua: el mismo plan.

	Completa	Elegida	Continua
Fid.	lo que dice la ficha	cargas parciales, prohibidas	como la elegida
Tam.	82	84	84
Guía	113,017	107,015, cuatro cargas	107,015
Port.	durativas + números	Kairos, OPTIC, POPF	Kairos, POPF

Veredicto: la carga completa

«Se queda hasta llenarse» hace imposible el plan más corto: la codificación que perdía soluciones en el Capítulo 11 es aquí la fiel. Si ella hubiera dicho «cuando quiera», el planificador entraría en la decisión (bloques).

Versión 3: un calendario y deseos

V2 recoge a 24 en una farmacia que abre a 30, entrega después de la ronda y solo tiene objetivos. `(is-open ?l)`, `(in-time ?i)`: solo los cambia el mundo, en momentos conocidos.

```
(:durative-action pick
  :parameters (?r - robot ?i - item ?l - location)
  :duration (= ?duration 2)
  :condition (and
    (at start (idle ?r))
    (at start (item-at ?i ?l))
    (at start (free ?r))
    (at start (not (delivered ?i)))
    (over all (robot-at ?r ?l))
    (over all (is-open ?l)))
(:durative-action deliver
  :parameters (?r - robot ?i - item ?l - location)
  :duration (= ?duration 2)
  :condition (and
    (at start (idle ?r))
    (at start (carrying ?r ?i))
    (at start (destination ?i ?l))
    (over all (robot-at ?r ?l))
    (at end (in-time ?i)))
```

```
  ; the calendar of the day
  (is-open lobby) (is-open ward-b)
  (at 30 (is-open pharmacy))
  (at 90 (not (is-open pharmacy)))
  (in-time meds1) (in-time meds2)
  (in-time sample1) (in-time meal1)
  (at 80 (not (in-time meds1)))
  (at 100 (not (in-time meds2)))
  (at 100 (not (in-time sample1)))
  (at 100 (not (in-time meal1))))
(:goal (and
  (delivered meds1)                    ; urgent: required
  (preference meds2-done (delivered meds2))
  (preference sample-done (delivered sample1))
  (preference meal-done (delivered meal1))
  (preference r1-home (robot-at r1 dock))
  (preference r2-home (robot-at r2 dock))))
(:metric minimize (+ (* 5 (is-violated meds2-done))
                     (* 4 (is-violated sample-done))
                     (* 2 (is-violated meal-done))
                     (* 1 (is-violated r1-home))
                     (* 1 (is-violated r2-home)))))
```

Versión 3: las pruebas

El plan antiguo es una prueba de la versión nueva:

```
... step 8 (pick r2 meds1 pharmacy) at
    24.003: over-all condition broken ...
    : (is-open pharmacy) is false ...
plan invalid
```

Trivial (p01): ante la persiana a 10, espera 20 s.

```
4.001: (move r1 lobby pharmacy) [6.000]
30.000: (pick r1 meds1 pharmacy) [2.000]
...
65.005: (deliver r1 meds1 icu) [2.000]
```

Sin plan, plazo 30 (p03): demostrado.

```
note: no plan: the timed initial
    literals take away what the goal
    needs ... [orchestrator.deadlines]
kairos: unsolvable (exhausted)
```

Sin plan, abre a 30, plazo 60 (p04):

```
kairos: timeout (timeout)
```

Salida 11, nos toca completarla:
$30 + 2 + 33 + 2 = 67 > 60$. Prefiere pruebas cuya respuesta puedas calcular.

Versión 3: la ronda, y un límite

La ronda (p02), 3 s:

```
; better plan 2
; cost = 12
; makespan = 67.005
```

El plan de la prueba trivial: un deseo de cinco (r2-home). 25 s: coste 7. Relajar: la ronda *sin su calendario* (p07):

```
; better plan 7
; cost = 0
; makespan = 211.030
...
kairos: solved (exhausted), cost 0
```

- Todos los deseos, en menos de 0,1 s: el modelo de los deseos es correcto
- Sin calendario: motor M2, casi clásico. Con literales temporizados: M3, sobre inicios y finales; las preferencias lo multiplican
- Un hecho sobre Kairos 1.0.6, no sobre PDDL. No podemos quitar el calendario; podemos cambiar lo que pedimos en una llamada

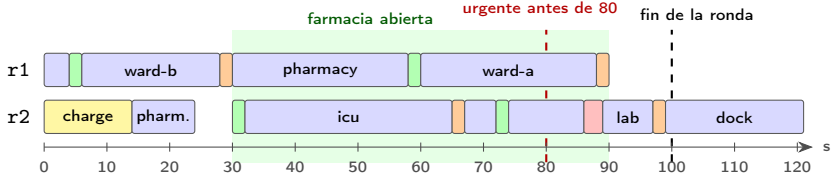
Cuando el planificador decepciona

Relaja el problema hasta que deje de hacerlo: la característica cuya eliminación arregla la respuesta es la que cuesta.

Versión 3, paso L: tres maneras de planificar los deseos

- **A** un problema (p02): coste 12 en 3 s
- **B** un robot cada vez; juntar; validar contra A. r1 primero: métrica 2; r2 primero: métrica 1
- **C** una escalera de objetivos estrictos: tercer peldaño (p08) timeout, aunque tiene plan

```
$ kairos validate ...
    courier-fleet-day-p02-merged.plan
plan valid, makespan 121.014, metric 2
$ kairos validate ...
    courier-fleet-day-p02-merged2.plan
plan valid, makespan 121.014, metric 1
```



El segundo plan conjunto: todas las entregas a tiempo; solo r1 no llega a casa (18 de batería, a 26 m).

Versión 3, paso L: la comparativa

	A: un problema	B: un robot cada vez	C: escalera de objetivos estrictos
Fidelidad	exacta: la ficha, línea a línea	pierde planes en los que el otro robot debería elegir primero; sus planes son válidos para A	pierde los pesos: solo un orden
Tamaño	76 acciones durativas, 5 preferencias	la mitad, dos veces	como A sin preferencias, varias veces
Guía	mala con calendario: 12 en 3 s, 7 en 25 s	buena: menos de un segundo por llamada; 2, luego 1	mala: el tercer peldaño agota el tiempo
Claridad	un fichero es la ronda	dos ficheros y una fusión; quién va primero es una decisión nueva	muchos ficheros

Veredicto: A es el modelo, B es cómo planificamos con él

A se queda como especificación, y el validador pone precio a cualquier plan contra ella. B es honesta solo gracias a ese último paso: los planes conjuntos pueden fallar donde se encuentran los robots (el cargador). Quién va primero: dos órdenes para dos robots, 120 para cinco.

Versión 4: la batería es cosa del mundo

```
(:action plug
  :parameters (?r - robot ?l - location)
  :precondition (and (idle ?r) (robot-at ?r ?l)
                     (has-charger ?l) (charger-free ?l)
                     (< (battery ?r) (battery-capacity ?r)))
  :effect (and (not (idle ?r)) (not (charger-free ?l))
               (plugged ?r ?l)))
(:process charging
  :parameters (?r - robot ?l - location)
  :precondition (plugged ?r ?l)
  :effect (increase (battery ?r) (* #t (charge-rate ?r))))

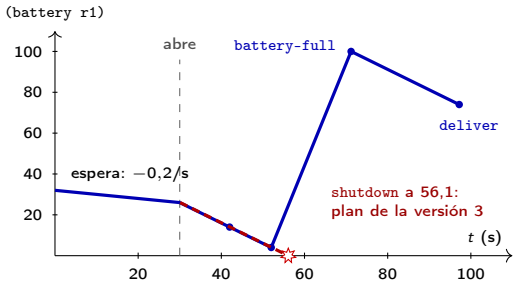
; The charger lets go by itself when the battery is full.
(:event battery-full
  :parameters (?r - robot ?l - location)
  :precondition (and (plugged ?r ?l)
                     (>= (battery ?r) (battery-capacity ?r)))
  :effect (and (not (plugged ?r ?l)) (charger-free ?l)
               (idle ?r)))
```

Ni `unplug` ni duración que calcular: «se queda hasta llenarse» encuentra su forma natural.

Cambio	Lo causa
habitación siguiente	robot: `move`
se enchufa	robot: `plug`
se descarga (0,2/s; +0,8 en marcha)	mundo: `drain`, `motor-drain`
se llena (5,2/s)	mundo: `charging`
el cargador lo suelta	mundo: `battery-full`
se apaga	mundo: `shutdown`
abre la farmacia	reloj: literal temporizado

Fuera, por decisión moldeada por el planificador: manipulación instantánea; **sin muestras**. *La versión 4 trata de la batería; el laboratorio se prueba en las versiones 1 a 3.*

Versión 4: un robot y una persiana



Prueba de la trampa (p04): 32 ante la persiana, abre a 30, 28 m por recorrer. V3: esperar, recoger a 30. Lápiz: 30 s cuestan 6, sale con 26, necesita 28. Continua: recoge a 42, se enchufa a 52.

```
... step 5 (deliver r1 meds2 ward-a)
    at 58.100: precondition false: (
    powered r1) is false ...
plan invalid
```

El planificador: a la base de carga con la caja, carga, entrega con 74. `battery-full` no está en el plan.

```
; makespan = 97.200
52.000: (plug r1 dock) [0.000]
71.200: (move r1 dock lobby) [4.000]
```

Parado hasta 30, un paseo inútil (las menos acciones). Típica: 94; 84,1 a mano. p03: $5/0{,}2 = 25 < 30$.

Las versiones como modelos de la misma página

	Versión 2: tiempo, batería	Versión 3: calendario, deseos	Versión 4: batería PDDL+
Fidelidad	farmacia cerrada, entregas tarde; sin deseos	la ficha; esperar es gratis	la batería tal como es; sin deseos, sin laboratorio
Tamaño	82 acciones durativas	76, y 5 preferencias	17 para un robot
Guía	los dos robots, 113,017, en menos de 1 s	por robot: coste 1, cuatro llamadas de 3 s	un robot, un objetivo: menos de 1 s
Claridad	horarios	el problema se lee como la ficha	los planes necesitan una gráfica
Portabilidad	la mayoría de los planificadores temporales	literales temporizados y preferencias	Kairos

Veredicto: ninguna versión sustituye a las demás

V3 planifica la ronda; **V4** la audita: robots parados 8 y 17 s cuestan 1,6 y 3,4 unidades; `r2` llega a casa con menos de 7, no con 10. La lección vuelve como un margen (10 de reserva). **V2**, para una noche con todo abierto. **V1** prueba y demuestra las reglas en milisegundos.

Del plan a la ejecución: qué es un plan para un robot

Para el planificador, una demostración; para un robot, instrucciones. Cada línea necesita:

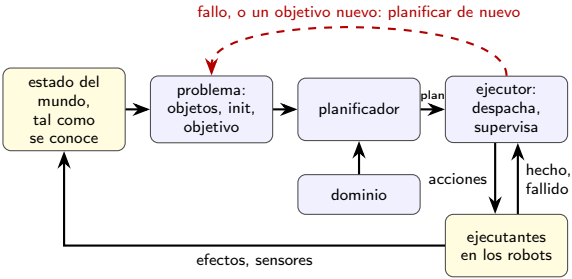
- **Alguien que la lleve a cabo**: (`move r1 lobby pharmacy`) es una tarea de navegación; informa de hecho, fallido, progreso
- **Un motivo para empezar**: los instantes son una manera de cumplir las **dependencias**
- **Un modelo que siga siendo cierto**

Despacho

Enviar cada acción a quien la realiza en el momento justo. **Disparado por tiempo**: el plan como horario. **Disparado por eventos**: empezar cuando ha terminado aquello de lo que depende y se cumplen sus condiciones. Las redes temporales simples guardan las dependencias.

Qué ocurre	En la ronda	Una reacción razonable
Una acción tarda más	un movimiento de 12 s tarda 15	nada, mientras se cumplan los plazos
Una acción falla	la caja se escurre al recogerla	reintentar; luego, planificar de nuevo
Una condición se rompe durante la acción	la farmacia cierra antes de hora	parar, planificar de nuevo
El estado no es el previsto	el cargador dio 92, no 100	comprobar lo que queda
La tarea cambia	una petición urgente nueva a 30	planificar de nuevo con el objetivo nuevo

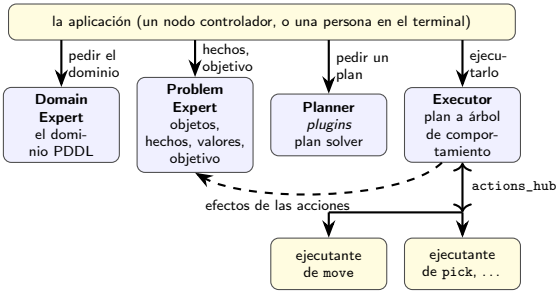
Planificar de nuevo



Barato gracias a la separación del Capítulo 1: el dominio es fijo, el problema es nuevo.

- **El estado intermedio tiene que ser un estado del modelo**: `r1` llevando `meds2`, la sala A bloqueada. Sin `drop`: un robot perdido para el día. Conserva **acciones de recuperación**
- **Ha pasado el tiempo**: desplaza el calendario; el mundo envejece mientras el planificador piensa
- **Siempre hay algo en marcha**: esperar, cancelar, o planificar desde donde termine

PlanSys2: cuatro nodos



El ROS 2 Planning System, tal como está en sus fuentes en la 4.0.0 (la flota no se ejecutó sobre él). Cuatro nodos con ciclo de vida:

- **Domain Expert**: `model_file`; tres comprobaciones, la última con el comprobador de Kairos como biblioteca
- **Problem Expert**: el conocimiento que el sistema tiene del mundo; lo escribe como un problema PDDL
- **Planner**: *plugins* plan solver, Kairos por defecto, también POPF; `plan_solver_timeout` 15 s
- **Executor**: la acción de ROS 2 `execute_plan`

Un plan se convierte en un árbol de comportamiento

```
<Sequence name="ACTION_ID">
WAIT_PREV_ACTIONS
  <WaitAtStartReq action="ACTION_ID"/>
  <ApplyAtStartEffect action="ACTION_ID"/>
  <ReactiveSequence name="ACTION_ID">
    <CheckOverAllReq action="ACTION_ID"/>
    <ExecuteAction action="ACTION_ID"/>
  </ReactiveSequence>
  <CheckAtEndReq action="ACTION_ID"/>
  <ApplyAtEndEffect action="ACTION_ID"/>
</Sequence>
```

La acción durativa del Capítulo 10, ejecutada: tres lugares de una condición y dos de un efecto son cinco nodos. Una condición mal colocada se comprueba en el momento equivocado.

- Constructor por defecto `STNBTBuilder`: árbol a partir de una red temporal simple
- Un fallo cancela lo que está en marcha: `FAILURE`. Replanificar es cosa de la aplicación
- **Ejecutantes de acciones**: `ActionExecutorClient`, `do_work`; `specialized_arguments`: cada robot, los suyos
- `actions_hub`: petición, respuesta, confirmar/rechazar, realimentación, fin, cancelar

Lo que esto pide al modelo

Acciones: lo que saben hacer los ejecutantes. `(over all (is-open ?l))` se comprueba mientras se mueve el brazo: alguien debe informar. La carga completa es una acción. El estado son hechos: un robot con una caja imposible de entregar no tenía acción.

Lo que PDDL deja fuera

Determinismo

Una acción tiene un resultado; un plan es una secuencia, no una estrategia. En la práctica: planificar con el resultado más probable y un margen, vigilar, planificar de nuevo. Falla cuando hay que *prepararse*: ningún plan determinista lleva una segunda bolsa por si la primera se rompe.

Conocimiento completo

El mundo cerrado. Las acciones de **percepción** cambian lo que el robot sabe: los planes se ramifican, **observabilidad parcial**. Nuestros modelos lo esconden en el ejecutante.

Fuera por motivos más modestos: geometría, otros agentes, magnitudes que se aprenden, objetos nuevos.

Lo que se deja fuera no desaparece

Acaba en un ejecutante, en el supervisor o en el margen de un número. Al final de un proyecto, vuelve a leer el *Alcance*: para cada cosa en «Fuera», ¿quién se ocupa de ella ahora?

Errores comunes del caso

Escribir primero la última versión

Un `timeout` no dice entonces nada: ¿una regla mal, un número mal, o solo es difícil? *Solución*: versiones

Quedarse con la primera respuesta con deseos

Coste 12, salida 0, un deseo de cinco. *Solución*: conoce un buen coste (como mucho 2) antes de ejecutar

Leer un límite de tiempo como «no hay plan»

p08 tiene plan. 10, 14 y 11 son tres respuestas. *Solución*: un plan a mano o un cálculo

Un plan válido sin holgura

La muestra se entrega a 99,011, plazo 100. *Solución*: márgenes en el modelo (95 para 100); la holgura más pequeña, como prueba

Un modelo que no puede empezar desde la mitad

Sin `drop`: un robot cargado con el destino cerrado está perdido. *Solución*: enumera los estados interrumpibles, añade acciones de recuperación con un coste

El método MODEL: una historia, cuatro modelos

Lo que el caso añade

M	¿Qué pregunté a la autora? ¿Qué versiones, en qué orden?
O, D	¿Qué se añade? «Casi todo»: un salto demasiado grande
E	¿Planes antiguos validados por la versión nueva? Pruebas sin plan: ¿demostración, se rinde, límite de tiempo?
L	¿Con mi planificador y mi límite de tiempo? ¿Qué frase cambiaría el veredicto?
Después	¿Quién ejecuta cada acción? ¿Holgura más pequeña? ¿Un estado desde el que no puede partir?

- **La historia decide**: «no puede pasar», «se queda hasta llenarse», «un poco». La Fidelidad es fidelidad *a una ficha*
- **El planificador**: dos veces un modelo correcto obtuvo una respuesta pobre. *Relajar* para encontrar el coste; conservar el modelo completo como *especificación*; planificar una pregunta más fácil
- **El validador** es el punto fijo: restricciones, planes antiguos, planes conjuntos, tiempo continuo

1 El método MODEL, completo

2 Caso de estudio: una misión robótica, de la historia a la ejecución

3 Resumen

Qué llevarse

Capítulo 19

- Cinco listas de comprobación con todas las preguntas; el paso M decide la mayoría
- Guía de decisión: necesidad → característica → coste; quédate en el nivel más bajo que no mienta
- Cuarenta patrones, cada uno con un rival; antipatrones por síntoma
- `unsolvable` es una demostración; plan a mano y validador separan modelo y búsqueda
- Versiones, primero la ficha, pruebas antiguas conservadas; documentación en tres sitios

Capítulo 20

- Una página, cuatro modelos probados, cada uno para una pregunta
- 31 acciones → 113,017 s → métrica 1 → una auditoría de la batería
- El planificador forma parte de la decisión; el validador es el juez
- Un plan aún no es comportamiento: ejecutantes, dependencias, supervisor, replanificación

El resolutor es trabajo de otros y mejora cada año. El modelo es tuyo.

Ejercicios para practicar

Capítulo 19 (proyectos: ficha, tablas, pruebas, comparativa)

- 19.2.1 La biblioteca (clásico, una cesta de tres)
- 19.3.1 El camión cisterna: ¿qué preguntas fueron «no»?
- 19.3.3 La escala: primero el *makespan* a mano
- 19.3.6 La esclusa: ¿quién causa cada cambio?
- 19.7.1 La lavandería, con monedas

Capítulo 20 (tres segundos por problema temporal)

- 20.3.2 Donde la precondición y la restricción no están de acuerdo
- 20.4.2 Una carga en bloques de 5 s
- 20.5.3 La comida vale 6
- 20.7.1 Replanificar a 30 con `sample2`
- 20.10.1 El almacén con dos carretillas