Tiempo: PDDL 2.1 y 2.2

Planificación y PDDL, Parte IV (Capítulos 10–13)

Francisco Martín Rico

Intelligent Robotics Lab, URJC



Índice

1. Acciones durativas
2. Coordinación en el tiempo
3. Cambio continuo
4. Predicados derivados y un mundo con calendario
5. Resumen

1 Acciones durativas

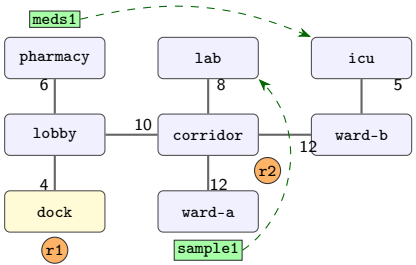
2 Coordinación en el tiempo

3 Cambio continuo

4 Predicados derivados y un mundo con calendario

5 Resumen

Lo que una secuencia no puede decir

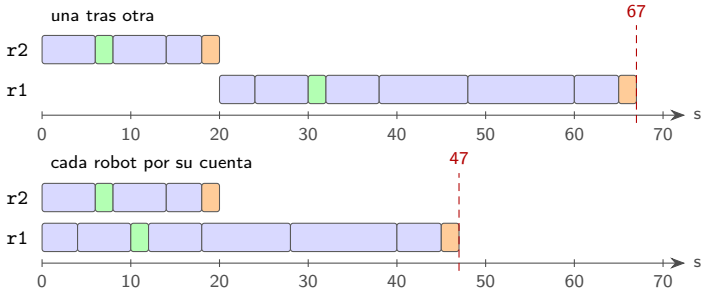


Dos robots, dos artículos, distancias en metros.

Un planificador secuencial devuelve **13 acciones**, una por línea. Falta:

- **Cuánto tarda** cada acción (12 m de pasillo frente a levantar una caja)
- **Qué puede ocurrir a la vez** (los robots no se necesitan entre sí)
- **Cuándo se alcanza el objetivo** (lo que quiere saber la enfermera)

Dos lecturas de las mismas trece acciones

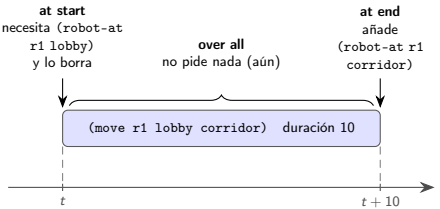


- Un **cronograma** (diagrama de Gantt): el tiempo hacia la derecha, una fila por cosa que trabaja. Azul, desplazamientos; verde, recogidas; naranja, descargas.
- Un coste (Capítulo 9) suma las duraciones: 67 en los dos dibujos. No ve el trabajo en paralelo.

Una primera acción durativa

```
(:durative-action move
  :parameters (?r - robot ?from ?to - location)
  :duration (= ?duration 10)
  :condition (and (at start (robot-at ?r ?from))
                  (at start (connected ?from ?to)))
  :effect (and (at start (not (robot-at ?r ?from)))
               (at end (robot-at ?r ?to))))
```

- `:duration`: una restricción sobre `?duration`
- `:condition`: cada una `at start`, `over all` o `at end`
- `:effect`: solo `at start` o `at end`
- Requisito `:durative-actions`



Durante el intervalo el robot no está en **ninguna habitación**. PDDL no tiene unidad de tiempo: leemos segundos.

Leer un plan temporal

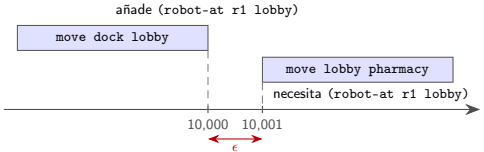
```
$ kairos courier-fixed-domain.pddl courier-fixed-p01.pddl
; cost = 52.006
; makespan = 52.006
0.000: (move r1 dock lobby) [10.000]
10.001: (move r1 lobby pharmacy) [10.000]
20.002: (pick r1 meds1 pharmacy) [2.000]
20.003: (move r1 pharmacy lobby) [10.000]
30.004: (move r1 lobby corridor) [10.000]
40.005: (move r1 corridor ward-a) [10.000]
50.006: (drop r1 meds1 ward-a) [2.000]
```

- Líneas `t: (action) [d]`: inicio, duración; por orden de inicio, *nada más*
- **Makespan**: final de la última acción, $50{,}006 + 2$

1. 10,001, no 10: correcto (siguiente diapositiva)
2. recoge en 20,002, sale en 20,003: defecto de *nuestro modelo*

Qué ocurre en cada instante

- El mundo solo cambia en los inicios y finales de acciones: los **acontecimientos**
- *No moving targets*: dos acontecimientos **interfieren** si uno cambia un hecho que el otro necesita o cambia
- Los que interfieren van separados por ϵ (Kairos: 0,001, `-epsilon`); los demás pueden compartir instante. Sin ella, se rechaza:




```
$ kairos validate courier-time-domain.pddl courier-time-p01.pddl \
    courier-time-p01-tight.plan
courier-time-p01-tight.plan:3: error: step 2 (move r1 lobby pharmacy) at 4.000
    : at start condition false: (robot-at r1 lobby) is false [validate.
    precondition_failed]
```

Condiciones en el tiempo: invariantes

Lo que debe seguir cierto *mientras* dura la acción es un **invariante**: `over all`.

```
(:durative-action pick
  :parameters (?r - robot ?i - item ?l - location)
  :duration (= ?duration (handling-time ?r))
  :condition (and (at start (robot-at ?r ?l))
                  (over all (robot-at ?r ?l))
                  (at start (item-at ?i ?l))
                  (at start (free ?r)))
  :effect (and (at start (not (item-at ?i ?l)))
               (at start (not (free ?r)))
               (at end (carrying ?r ?i))))
```

Paso D: tres preguntas por condición

1. ¿Hace falta para *empezar*? `at start`
2. ¿Fallaría la acción si parase a medias? También `over all`
3. ¿Debe cumplirse al *completarse*? También `at end`

Especificador	Se comprueba	Uso típico
`at start`	en t	lo necesario para empezar: dónde está el robot, una pinza libre
`over all`	en el intervalo **abierto** $(t, t+d)$	lo que no debe cambiar: el robot sigue ahí, la puerta abierta
`at end`	en $t+d$	lo necesario para completarse: el ascensor sigue en la planta

Duraciones que dependen de la acción

```
(:durative-action move
  :parameters (?r - robot ?from ?to - location)
  :duration (= ?duration (/ (distance ?from ?to) (speed ?r)))
```

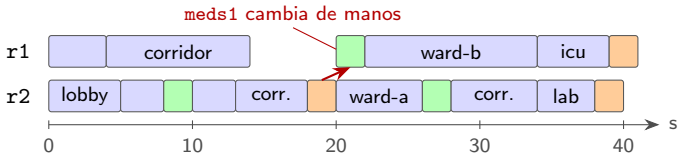
- Cualquier expresión numérica: distancia entre velocidad, evaluada **cuando la acción empieza**
- Funciones estáticas speed, handling-time; requisito :numeric-fluents
- (:metric minimize (total-time)): el makespan, en todos los problemas
- Ahora honrado: la recogida acaba en 12,002 y el desplazamiento empieza en 12,002; el intervalo de over all es abierto

```
$ kairos courier-time-domain.pddl
    courier-time-p01.pddl
; cost = 42.005
; makespan = 42.005
0.000: (move r1 dock lobby) [4.000]
4.001: (move r1 lobby pharmacy) [6.000]
10.002: (pick r1 meds1 pharmacy) [2.000]
12.002: (move r1 pharmacy lobby) [6.000]
18.003: (move r1 lobby corridor) [10.000]
28.004: (move r1 corridor ward-a) [12.000]
40.005: (drop r1 meds1 ward-a) [2.000]
```

Trabajar en paralelo

No hace falta nada nuevo: las acciones se solapan salvo que el modelo las mantenga separadas.

- `r1` solo: 95 s. Dos robots, una entrega cada uno: 47,006. Si sigue buscando: 41,007



- Un relevo que nadie pidió: el rápido `r2` pasa `meds1` en el pasillo.
- **El makespan no es el trabajo**: 67 robot-segundos en el plan de 47, 75 en el de 41.
- Sin demostración de optimalidad del makespan. Las pausas son parte del plan: no hay acción `wait`.

Una cosa cada vez

Añadimos una lámpara UV, `disinfect` (20 s), solo con su invariante:

```
$ kairos courier-uv-domain.pddl
    courier-uv-p01.pddl
; cost = 28.001
; makespan = 28.001
0.000: (pick r1 meds1 pharmacy) [2.000]
0.000: (disinfect r1 pharmacy) [20.000]
20.000: (move r1 pharmacy lobby) [6.000]
26.001: (drop r1 meds1 lobby) [2.000]
```

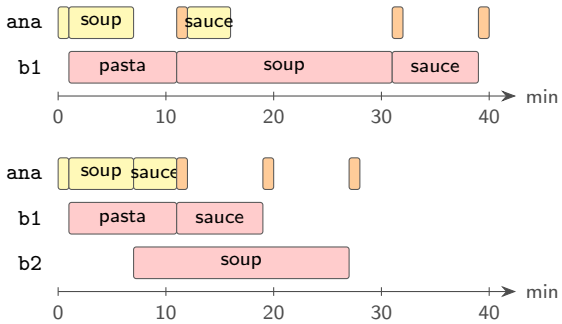
El robot recoge mientras desinfecta. Patrón *cerrojo de recurso* (Capítulo 7), ahora para una sola acción. Los invariantes dicen en qué se apoya la acción; el testigo dice que el agente es uno.

El robot es un **recurso**: un testigo `(idle ?r)`, que se toma al inicio y se devuelve al final.

```
(:durative-action disinfect
  :parameters (?r - robot ?l - location)
  :duration (= ?duration 20)
  :condition (and (at start (idle ?r))
                  (at start (robot-at ?r ?l))
                  (over all (robot-at ?r ?l)))
  :effect (and (at start (not (idle ?r)))
               (at end (disinfected ?l))
               (at end (idle ?r)))))
```

Con `idle` en todas las acciones: 30,003, una acción tras otra.

Otros dominios: una cocina y un ascensor



Un fogón: 40 min. Dos fogones: 28. Amarillo, preparar; rojo, cocer; naranja, servir.

- `cook` no tiene cocinero: el fogón trabaja solo (testigo `burner-free`)
- La fila **sin huecos** es el cuello de botella: con un fogón, un segundo cocinero no ayudaría
- Ascensor: `board` pide `(lift-at ?e ?f)` al inicio, durante y al final. Un invariante deja a los pasajeros **compartir** el ascensor; un testigo separa a los usuarios

El método MODEL: tiempo

Lo que añade el Capítulo 10

Paso M. *Éxito*: ¿importa *cuándo* se alcanza el objetivo? Entonces, tiempo y (`total-time`).
Agentes: ¿quién puede actuar a la vez que quién?
Paso D. Cuatro columnas nuevas en la tabla de acciones:

- ¿Importa la duración, y de qué depende?
- ¿Qué debe cumplirse al inicio, durante todo el intervalo y al final?
- ¿Cuándo ocurre cada efecto? Lo que se quita, al inicio; lo que se produce, al final
- ¿Qué puede ejecutarse en paralelo con ella? Lo que no deba necesita un testigo

Paso E. La prueba de la trampa se convierte en una familia: un agente en dos sitios a la vez; un objetivo cuyo horario honrado más corto conoces; cada par de líneas solapadas, contrastado con el mundo.

Alternativas comparadas

Acciones instantáneas con coste, o acciones durativas (dos robots, dos entregas)

	Coste (`courier-cost`)	Durativas (`courier-time`)
Fidelidad	Trabajo total, 52,5 s, óptimo demostrado; `r2` lo hace todo	Horario: makespan 41,007; usa todo solapamiento abierto
Guía	3336 estados, óptimo en 13 ms	primer plan en 2 ms, mejor a los 0,9 s; sin demostración
Portabilidad	cualquier planificador numérico	solo planificadores temporales

Dónde va un efecto (`courier-walk`; en negrita, planes que el mundo no permite)

	Ambos al final	Ambos al inicio	Repartidos
Mientras se mueve, el robot está	aún en `?from`	ya en `?to`	en ningún sitio
Recorrido de tres desplazamientos	30,002	**10,002**	30,002
En dos sitios a la vez	**un plan, 10,001**	sin plan	sin plan

Veredicto. Coste para el menor trabajo total; durativas si las cosas se solapan e importa cuándo acaban. **Tomar al inicio, dar al final.**

Errores comunes

Renombrar las partes de una acción

```
$ kairos check wrong-untimed-domain.
    pddl
wrong-untimed-domain.pddl:15:21:
    error: in durative action 'move'
    : conditions of durative actions
     must be under at start, at end
    or over all [semantic.
    untimed_condition]
...
```

Cada especificador es una decisión; el comprobador no va a tomarla por ti.

At start para lo que debe cumplirse siempre

Un agente empieza algo y se va. Añade `over all` y valida el plan sospechoso.

Un testigo olvidado en una acción

Esa acción se solapa con todas; si no lo devuelves, el agente no vuelve a actuar. Ningún comprobador avisa.

Leer las líneas como una secuencia

Dibuja el cronograma, con una fila por agente o recurso.

1 Acciones durativas

2 Coordinación en el tiempo

3 Cambio continuo

4 Predicados derivados y un mundo con calendario

5 Resumen

Planes que solo existen si las acciones se solapan

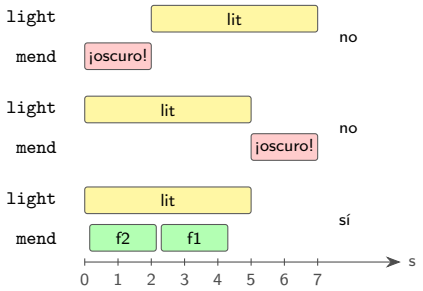
```
;; There is light from the start of the action to its end.
(:durative-action light
  :parameters (?m - match)
  :duration (= ?duration 5)
  :condition (at start (unused ?m))
  :effect (and (at start (not (unused ?m)))
               (at start (lit ?m))
               (at end (not (lit ?m)))))

;; Mending takes 2 seconds, all of them with light.
(:durative-action mend
  :parameters (?f - fuse ?m - match)
  :duration (= ?duration 2)
  :condition (and (at start (hand-free))
                  (at start (lit ?m))
                  (over all (lit ?m)))
  :effect (and (at start (not (hand-free)))
               (at end (mended ?f))
               (at end (hand-free)))))
```

- Una cerilla arde 5 s; reparar un fusible, 2 s y con luz
- `light` añade `(lit ?m)` al inicio y lo borra al final, adrede: cierto **exactamente mientras se ejecuta**
- `mend` lo pide al inicio y todo el tiempo

```
$ kairos cellar-domain.pddl
    cellar-p01.pddl
; cost = 5
; makespan = 5.000
0.000: (light m1) [5.000]
0.001: (mend f1 m1) [2.000]
```

Concurrencia requerida



- Ninguna secuencia de acciones completas sirve: antes o después de la luz, la reparación queda a oscuras
- **Concurrencia requerida**: todo plan contiene acciones que se solapan
- Una cerilla sirve para dos fusibles (4 s caben en 5), no para tres. La salida 14 es más débil que `unsolvable` (10): no hay demostración

```
$ kairos cellar-domain.pddl
    cellar-p03.pddl
...
kairos: no plan found (exhausted)
$ echo $?
14
```

La acción envolvente: una puerta sujeta

Una **acción envolvente** añade un hecho al inicio y lo borra al final; su **contenido** necesita el hecho al inicio y durante todo el intervalo.

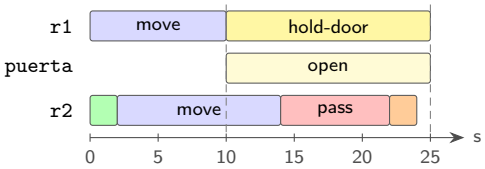
```
(:durative-action hold-door
  :parameters (?r - robot ?d - door ?l ?other - location)
  :duration (= ?duration 15)
  :condition (and (at start (idle ?r))
                  (at start (robot-at ?r ?l))
                  (over all (robot-at ?r ?l))
                  (at start (joins ?d ?l ?other)))
  :effect (and (at start (not (idle ?r)))
               (at start (open ?d))
               (at end (not (open ?d)))
               (at end (idle ?r))))
```

```
(:durative-action pass
  :parameters (?r - robot ?d - door ?from ?to - location)
  ; ...
  :condition (and (at start (idle ?r))
                  (at start (robot-at ?r ?from))
                  (at start (joins ?d ?from ?to))
                  (at start (open ?d))
                  (over all (open ?d)))
  :effect (and (at start (not (idle ?r)))
               (at start (not (robot-at ?r ?from)))
               (at end (robot-at ?r ?to))
               (at end (idle ?r))))
```

Paso D: una acción que proporciona algo mientras se ejecuta

Pregunta qué hace cierto cada acción *solo mientras dura*. Después: quién ejecuta la acción envolvente, quién el contenido, ¿y pueden ser el mismo agente? Aquí `idle` responde: no.

El plan de la puerta



- La fila central es el hecho `(open lab-door)`; pass queda dentro
- Makespan 25,001: fin de la acción envolvente, aunque la muestra llega al laboratorio en 24,003
- La sujeción empieza 4 s antes: nada dice «lo más tarde posible»
- Un robot solo: `unsolvable`, con demostración

```
$ kairos courier-door-domain.pddl courier-door-p02.pddl
note: no plan: not even without durations and times (happenings in any order,
    the over all conditions kept) does the goal hold: 196 states of that
    relaxation searched, with no durative action overlapping itself (as in
    POPF). [orchestrator.why]
kairos: unsolvable (exhausted)
```

Temporalmente simple, temporalmente expresivo

- Lenguaje **temporalmente expresivo**: enuncia concurrencia requerida; PDDL 2.1 lo es
- **Primero la secuencia, luego el horario**: rápido, búsqueda clásica; no resuelve el sótano
- **Buscar sobre inicios y finales**: lo resuelve, espacio mayor (POPF, OPTIC)

Qué hace Kairos

Motor **M2**: comprime cada acción durativa y asigna tiempos. Motor **M3**: busca sobre acontecimientos, tiempos en una red temporal. `-v` muestra la elección; `-engine m2` en el sótano: 2 estados, sin plan.

```
$ kairos cellar-domain.pddl cellar-p01.pddl -v
...
[   0.001s] [orchestrator] engine M3: (light m1) sets (lit m1) only while it
    runs, and other actions can use that
...
```

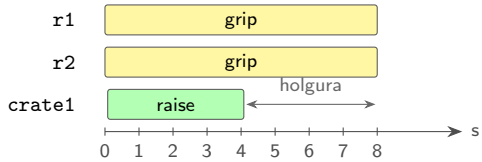
Trabajar juntos: dos robots y un cajón

Frágil: cada `lift` pide `over all` el agarre del otro. Únicos planes: ambos empiezan *exactamente* a la vez.

- validar a la vez: `plan valid, makespan 4`
- búsqueda: `no plan found`; empiece quien empiece, falla el otro invariante

Con holgura: cada agarre, acción envolvente de 8 s; `raise` (4 s), dentro de ambos.

```
(:durative-action raise
  :parameters (?c - crate ?r1 ?r2 - robot ?l - location)
  :duration (= ?duration 4)
  :condition (and (at start (not (= ?r1 ?r2)))
                  (at start (crate-at ?c ?l))
                  (at start (gripping ?r1 ?c))
                  (at start (gripping ?r2 ?c))
                  (over all (gripping ?r1 ?c))
                  (over all (gripping ?r2 ?c)))
  :effect (and (at start (not (crate-at ?c ?l)))
               (at end (on-cart ?c)))))
```



Un robot que hace los dos papeles

Sin `(at start (not (= ?r1 ?r2)))`: `(raise crate1 r1 r1 dock)`. Siempre que un modelo diga «otro», hace falta un testigo o una desigualdad.

Duraciones que elige el planificador

```
(:durative-action charge
  :parameters (?r - robot ?l - location)
  :duration (and (>= ?duration 1)
                 (<= ?duration (/ (- (battery-capacity ?r)
                                     (battery ?r))
                                  (charge-rate ?r))))
  :condition (and (at start (idle ?r))
                  (at start (robot-at ?r ?l))
                  (at start (has-charger ?l))
                  (over all (robot-at ?r ?l)))
  :effect (and (at start (not (idle ?r)))
               (at end (increase (battery ?r)
                                 (* ?duration (charge-rate ?r))))
               (at end (idle ?r)))))
```

- Cotas en lugar de un valor: `:duration-inequalities`
- `?duration` en un efecto: la elección tiene una consecuencia
- 20 unidades, 31 m hasta la UCI: hacen falta 5,5 s. A mano: válido, 36,504

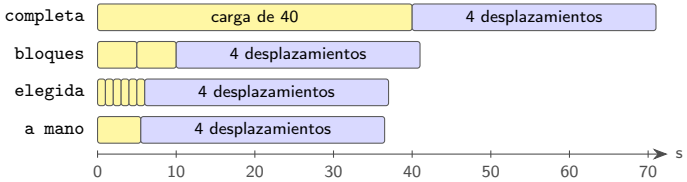
```
$ kairos courier-charge-domain.
    pddl courier-charge-p01.pddl
; cost = 71.00399999999999
; makespan = 71.004
0.000: (charge r1 dock) [40.000]
...
; better plan 2
; cost = 37.00899999999999
; makespan = 37.009
...
5.005: (charge r1 dock) [1.000]
6.006: (move r1 dock lobby)
    [4.000]
```

Cómo elige un planificador una duración

- Kairos prueba cada duración en sus **cotas**: carga completa, mínima repetida
- Si falla, **pasos de tiempo** (0,1 s, o el instante en que terminar hace cierta una condición)
- El validador acepta cualquier duración dentro de las cotas

Una cafetera: 25 ml/s, un expreso, de 40 a 50 ml, cotas de 1 a 10 s.

```
$ kairos coffee-domain.pddl
    coffee-p01.pddl
; cost = 1.6
; makespan = 1.600
0.000: (pour cup1) [1.600]
```



tres codificaciones de la carga, y el plan a mano

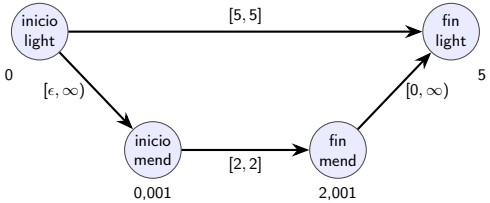
Cómo ordenan el tiempo los planificadores

Redes temporales simples

Una variable por punto temporal (cada inicio, cada final); restricciones $l \leq t_j - t_i \leq u$:

- duración 5: $t_{fin} - t_{ini} = 5$
- duración elegida: $1 \leq t_{fin} - t_{ini} \leq 40$
- después de otro acontecimiento: $t_j - t_i \geq \epsilon$

Consistente si y solo si ningún ciclo es una contradicción: caminos mínimos, sin búsqueda.



La STN de `cellar-p01`: tiempos más tempranos 0; 0,001; 2,001 y 5. Una reparación de 5 s necesitaría $\epsilon + 5$ donde la flecha directa permite 5: no hay plan.

El método MODEL: coordinación

Lo que añade el Capítulo 11

Paso D. *¿Debe solaparse con otra acción?* (acción envolvente, contenido, trabajo conjunto: cuál contiene a cuál, cuánta holgura). *¿Es su duración una decisión?* Entonces, cotas, y un efecto o una condición que dependa de `?duration`. **Paso E.** Un contenido que no cabe; el agente de la acción envolvente como su propio contenido; un objeto en dos papeles. «No plan found» no es una demostración.

	Acción envolvente (`hold-door`)	Cerrojo (`open-door`, `release-door`)
Fidelidad	la puerta siempre se cierra, a los 15 s exactos (25,001)	abierta lo que haga falta (24,003); se queda abierta salvo que el objetivo diga otra cosa
Guía	concurrencia requerida, 1919 estados; sin demostración	temporalmente simple: motor M2
Portabilidad	planificadores con concurrencia requerida	cualquier planificador temporal

Veredicto. Acción envolvente cuando termina sola (una cerilla, el pase de un satélite); cerrojo cuando el agente decide cuándo parar; ambas cosas: una acción envolvente con `(<= ?duration 15)`.

Errores comunes

Una acción envolvente por accidente

Un indicador de «ocupado» añadido al inicio y borrado al final, que se lee en otra parte. Usa un testigo borrado al inicio y añadido al final, como `idle`.

Esperar la mejor duración

Un planificador que prueba los extremos da planes en los extremos.

«No plan found» como «unsolvable»

```
note: engine M3 found no plan; as
    POPF, it keeps one partial
    plan per state, so this is
    strong evidence but not a
    proof [orchestrator.
    m3_exhausted]
```

Duración elegida sin consecuencias

La magnitud debe crecer con `?duration`, y la cota superior, detenerla.

1 Acciones durativas

2 Coordinación en el tiempo

3 Cambio continuo

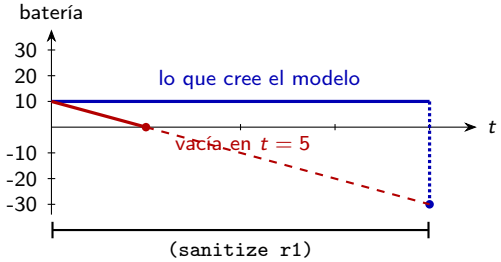
4 Predicados derivados y un mundo con calendario

5 Resumen

Una escalera donde el mundo tiene una rampa

- Batería a 10 de 30; una lámpara UV de 20 s gasta 2/s (40 en total); un cargador da 5/s
- Modelo discreto **prudente** (se paga al inicio, la carga se abona al final): `no plan found`
- Modelo **descuidado** (la lámpara se paga al final): la lámpara sola. Peor que no tener plan

```
$ kairos lamp-careless-domain.pddl
    lamp-careless-p01.pddl
; cost = 20
; makespan = 20.000
0.000: (sanitize r1) [20.000]
```



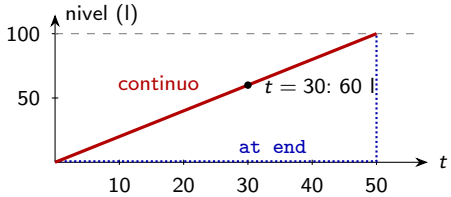
Cambio discreto y cambio continuo

Un **cambio discreto** ocurre en un instante; un **cambio continuo**, durante un intervalo, a un **ritmo**. Ritmo constante: **lineal** (todo este capítulo).

Efectos continuos

```
(:durative-action fill
  :parameters (?t - tank)
  :duration (and
    (>= ?duration 1)
    (<= ?duration (/ (- (capacity ?t) (level ?t)) (flow ?t))))
  :condition (at start (tap-closed ?t))
  :effect (and
    (at start (not (tap-closed ?t)))
    (at end (tap-closed ?t))
    (increase (level ?t) (* #t (flow ?t))))))
```

- **Sin etiqueta de tiempo**: actúa durante toda la acción
- `#t`: el tiempo desde que empezó la acción; su factor es el **ritmo**
- Solo `increase` o `decrease`; requisito `:continuous-effects`
- El mismo plan que con un efecto al final, pero el nivel tiene valor en cada segundo

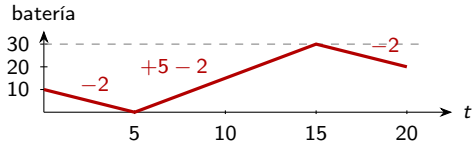



```
$ kairos tank-domain.pddl
    tank-p01.pddl
; cost = 50
; makespan = 50.000
0.000: (fill t1) [50.000]
```

Leer un número mientras cambia

```
(:durative-action charge
  :parameters (?r - robot ?l - location)
  :condition (and
    (at start (idle ?r))
    (at start (has-charger ?l))
    (over all (robot-at ?r ?l))
    (over all (<= (battery ?r) (battery-capacity ?r))))
  :effect (and
    (at start (not (idle ?r)))
    (increase (battery ?r) (* #t (charge-rate ?r)))
    (at end (idle ?r))))
(:durative-action sanitize
  :parameters (?r - robot)
  :duration (= ?duration 20)
  :condition (and
    (at start (lamp-off ?r))
    (over all (>= (battery ?r) 0)))
  :effect (and
    (at start (not (lamp-off ?r)))
    (decrease (battery ?r) (* #t (lamp-rate ?r)))
    (at end (lamp-off ?r))
    (at end (sanitized ?r)))))
```

```
$ kairos lamp-continuous-domain.
    pddl lamp-continuous-p01.pddl
; cost = 20
; makespan = 20.000
0.000: (sanitize r1) [20.000]
5.000: (charge r1 dock) [10.000]
...
```



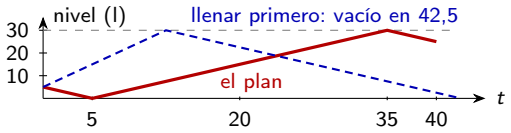
Agua que viene y va: el invernadero

```
(:durative-action fill
  :parameters (?t - tank)
  :duration (and (>= ?duration 1) (<= ?duration 120))
  :condition (and
    (at start (tap-closed ?t))
    (over all (<= (level ?t) (capacity ?t))))
  :effect (and
    (at start (not (tap-closed ?t)))
    (increase (level ?t) (* #t (flow ?t)))
    (at end (tap-closed ?t))))
```

Una regla general

Una cota calculada al inicio de una acción solo es correcta si nada más cambia el número mientras tanto. Si no: una cota constante y un límite over all.

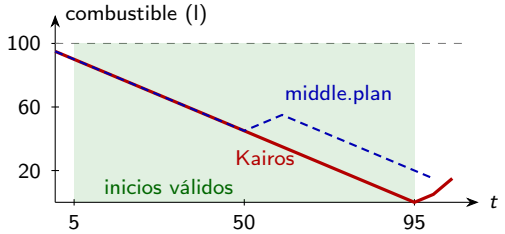
```
$ kairos greenhouse-domain.pddl
    greenhouse-p01.pddl
; cost = 40
; makespan = 40.000
0.000: (irrigate b1 t1) [40.000]
5.000: (fill t1) [30.000]
...
```



El generador: cuándo empezar

- `generate`: 100 s a 1 l/s, nunca vacío (`over all`)
- `refuel`: 10 s a 2 l/s, sin rebosar nunca (`over all`)
- 95 de 100 l, un bidón: si se vierte antes de 5, rebosa; después de 95, el depósito está vacío
- Duraciones fijas: el planificador debe encontrar un **instante**

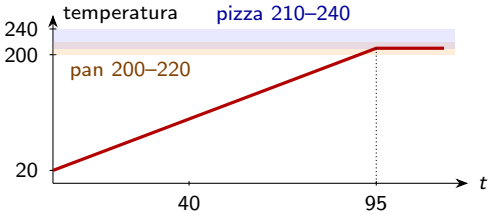
```
$ kairos generator-domain.pddl
    generator-p01.pddl
; cost = 105
; makespan = 105.000
0.000: (generate gen) [100.000]
95.000: (refuel gen j1) [10.000]
```



Kairos vierte en el último instante. Con 60 l: `no plan found`, salida 14; la demostración es una resta.

El horno: cuándo parar

```
(:durative-action bake
  :parameters (?d - dish)
  :duration (= ?duration 10)
  :condition (and
    (at start (raw ?d))
    (at start (oven-empty))
    (over all (>= (temperature) (min-temp ?d)))
    (over all (<= (temperature) (max-temp ?d))))
```




```
$ kairos oven-domain.pddl oven-p02
    .pddl
...
; better plan 2
; cost = 114.90100000000001
; makespan = 114.901
0.000: (heat) [95.000]
94.900: (bake bread) [10.000]
104.901: (bake pizza) [10.000]
```

- Resistencia de 2 grados/s, de 1 a 300 s. Solo pan: 90 s exactos
- Dos platos: parar en 210, dentro de ambos rangos; el pan entra en 94.9, aún subiendo

Lo que debe encontrar el planificador

Kairos ante el cambio continuo

Como cambios discretos: un número que solo se lee como «bastante grande» o solo como «bastante pequeño»: un salto en el instante más seguro; duraciones en sus cotas. **En pasos de tiempo**: 0,1 s, o un salto a donde una condición va a cambiar. **Validación**: cada plan se comprueba en tiempo continuo.

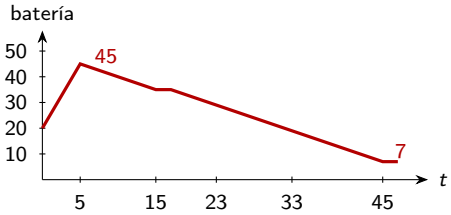
- Duraciones en sus cotas, o donde una condición las fija
- Los planes esperan hasta el último momento
- Sin plan: a menudo `no plan found` (14), no `unsolvable` (10)
- Hábito: escribe el plan en que crees y pásale `kairos validate` (exacto para cambio lineal)

```
$ kairos generator-domain.pddl generator-p01.pddl -v
...
[   0.001s] [orchestrator] durations: (generate gen): a continuous effect on a
     variable read in other ways than "large enough" or "small enough" alone;
    planned as PDDL+
```

El mundo del mensajero: una batería que se descarga y se carga

```
(:durative-action move
  :parameters (?r - robot ?from ?to - location)
  :duration (= ?duration (/ (distance ?from ?to) (speed ?r)))
  :condition (and
    (at start (robot-at ?r ?from))
    (at start (idle ?r))
    (over all (connected ?from ?to))
    (over all (>= (battery ?r) 0)))
  :effect (and
    (at start (not (robot-at ?r ?from)))
    (at start (not (idle ?r)))
    (decrease (battery ?r) (* #t (drain-rate ?r)))
    (at end (robot-at ?r ?to))
    (at end (idle ?r))))
```

`charge`: de 5 s al tiempo hasta la batería llena, +5/s. Una cota calculada al inicio es correcta aquí: `idle` deja fuera los desplazamientos.



`courier-battery-p02`: 20 unidades, hacen falta 38. Carga de 5 s (el mínimo), llega con 7.

- Prueba de la trampa (p05): un desplazamiento de 12 m empezado con 8 unidades se rechaza, `-4 >= 0 is false`
- Ronda de 90 m (p03): el mejor plan carga 8 s, hasta llenar; bastaría con 6

El método MODEL: cambio continuo

Preguntas nuevas para los pasos O y D

O: ¿cambia la magnitud *mientras* se ejecuta una acción? ¿A qué ritmo? **D**: ¿la lee alguien mientras tanto? ¿Puede otra acción cambiarla a la vez? ¿Qué acota una duración libre? *Si nadie la lee mientras tanto y nadie más la cambia, basta con un cambio discreto*: al inicio si se consume, al final si se produce.

	Discreto descuidado	Discreto prudente	Continuo
Fidelidad	planes imposibles: batería en -30	pierde planes que usan la energía a medida que llega	exacto para ritmos constantes
Guía	fácil, y equivocada	búsqueda temporal corriente	instantes y duraciones; a menudo sin demostración
Claridad	esconde la regla	totales a mano	ritmos y límites como en el mundo
Portabilidad	cualquier planificador temporal con números	Kairos, OPTIC, POPF	Kairos y POPF

Veredicto. Nunca el descuidado. Continuo cuando productores y consumidores se solapan (lámpara y cargador, grifo y bancales); procesos (Capítulo 17) cuando cambia sin el agente.

Errores comunes

#t bajo una etiqueta de tiempo

```
$ kairos check tank-wrong-domain.
    pddl
tank-wrong-domain.pddl:29:39: error:
     in durative action 'fill': #t
    can only appear in increase/
    decrease effects of durative
    actions and processes [semantic.
    misplaced_continuous]
...
```

Un límite comprobado solo al principio

El número se mueve y nadie mira: `lamp-alone.plan` es válido para el modelo descuidado. Los límites durante una acción son `over all`.

Leer `no plan found` como «no hay plan»

La salida 14 no es una demostración; la 10, sí.

Un requisito olvidado

Declara `:continuous-effects`; otros planificadores se detienen ahí.

1 Acciones durativas

2 Coordinación en el tiempo

3 Cambio continuo

4 Predicados derivados y un mundo con calendario

5 Resumen

Hechos que se siguen de otros hechos

- `(free ?r)` es redundante con `carrying`, acompasado a mano
- «Desde la base de carga se llega a la UCI»: mantenerlo con efectos es imposible
- **Predicado básico**: se enuncia (estado inicial, efectos). **Predicado derivado**: lo calcula una regla

```
(:derived (free ?r - robot)
  (not (exists (?i - item) (carrying ?r ?i))))
```

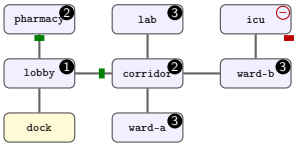
Cabeza y cuerpo; `:derived-predicates`. `pick` y `drop` ya no tocan `free`.

- Se **lee** en cualquier parte: precondiciones, condiciones, objetivos, otras reglas
- **Nunca se escribe**: ni en efectos ni en el estado inicial
- Varias reglas para una cabeza: un `or`

```
$ kairos check
    courier-free-wrong-domain.pddl
courier-free-wrong-domain.pddl:22:18:
    error: in action 'drop': derived
    predicate 'free' cannot be changed
    by an effect [semantic.
    derived_in_effect]
...
```

Reglas que se llaman a sí mismas

```
(:derived (passable ?a ?b - location)
  (and (connected ?a ?b)
       (forall (?d - door)
         (imply (or (joins ?d ?a ?b) (joins ?d ?b ?a))
                (open ?d)))))
(:derived (reachable ?a ?b - location)
  (or (passable ?a ?b)
      (exists (?c - location)
        (and (passable ?a ?c) (reachable ?c ?b)))))
(:action go
  :parameters (?r - robot ?from ?to - location)
  :precondition (and (robot-at ?r ?from) (reachable ?from ?to))
  :effect (and (not (robot-at ?r ?from))
               (robot-at ?r ?to)))
```



`(reachable dock ?b)` por rondas; `d-icu` cerrada (rojo).

- Una **regla recursiva**: caminos de cualquier longitud, imposible solo con `exists`
- Se construye de cero, ronda a ronda, hasta que una no añade nada
- go cruza cualquier número de habitaciones: navegar no es cosa del planificador

Las puertas como medio, la alcanzabilidad como consecuencia

Todas las puertas cerradas (`courier-reach-p02`):

```
$ kairos courier-reach-domain.pddl
    courier-reach-p02.pddl
; cost = 9
(go r1 dock lobby)
(open-door r1 d-corridor lobby corridor)
(open-door r1 d-pharmacy lobby pharmacy)
(go r1 lobby pharmacy)
(pick r1 meds1 pharmacy)
(go r1 pharmacy ward-b)
(open-door r1 d-icu ward-b icu)
(go r1 ward-b icu)
(drop r1 meds1 icu)
```

Ninguna acción añade `reachable`; cada `open-door` cambia qué go es aplicable.

Un predicado derivado en un objetivo, negado (p03):

```
(:goal (and (item-at meds1 icu)
            (robot-at r1 dock)
            (not (reachable lobby icu)))))
```

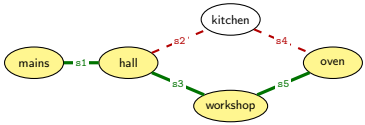
- Dos planes cierran puertas distintas: la del pasillo o la de la UCI
- El objetivo dice *qué*; el planificador encuentra *cómo*

Torres y circuitos

```
(:derived (linked ?a ?b - point)
  (or (wire ?a ?b) (wire ?b ?a)
      (exists (?s - switch)
        (and (closed ?s)
             (or (switched ?s ?a ?b) (switched ?s ?b ?a))))))

(:derived (powered ?p - point)
  (or (source ?p)
      (exists (?q - point) (and (powered ?q) (linked ?q ?p)))))
```

- El horno con corriente, la cocina no (trabaja allí un electricista)
- También debe abrirse s4: `linked` es simétrico y el horno alimentaría la cocina. Está en las reglas, no en ninguna acción
- Bloques: `above` como regla recursiva; el planificador elige la disposición
- Las reglas deben estar **estratificadas**: ningún predicado depende negativamente de sí mismo




```
$ kairos circuit-domain.pddl
      circuit-p01.pddl
; cost = 5
(open-switch s2)
(open-switch s4)
(close-switch s3)
(close-switch s1)
(close-switch s5)
```

Qué hace un planificador con las reglas

Un estado guarda solo hechos básicos; en cada estado nuevo las reglas se aplican hasta que no sale nada nuevo.

```
$ kairos courier-reach-domain.pddl courier-reach-p02.pddl -v
...
[   0.001s] [ground] grounded in 0.4 ms: 291 atoms, 0 numeric fluents, 92
    actions, 0 durative actions, 518 axioms (1031 Datalog facts, 28 rules, 0
    instances pruned)
...
```

- Dos reglas, 518 reglas instanciadas (*axiomas*): tres variables sobre n objetos, del orden de n^3
- Las heurísticas razonan a través de las reglas
- Menos planificadores (Kairos, LPG-td, LAMA en la prueba del libro)

¿Por qué no compilar las reglas?

Para `free`, escribe el `exists` en `pick`. Para una regla recursiva no hay nada que sustituir: quitar los axiomas agranda el dominio o los planes exponencialmente en el peor caso (Thiébaux, Hoffmann y Nebel).

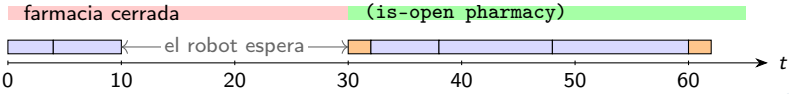
Un mundo con calendario: literales iniciales temporizados

En el `:init` del problema:

```
(at 30 (is-open pharmacy))
```

- En el instante 30 el literal se hace cierto, haga lo que haga el plan; `(at 50 (not ...))` lo hace falso
- Requisito `:timed-initial-literals`
- `pick` y `deliver` piden `(over all (is-open ?l))`

```
$ kairos courier-hours-domain.pddl
    courier-hours-p01.pddl
; cost = 62.00399999999999
; makespan = 62.004
0.000: (move r1 dock lobby) [4.000]
4.001: (move r1 lobby pharmacy) [6.000]
30.000: (pick r1 meds1 pharmacy) [2.000]
32.001: (move r1 pharmacy lobby) [6.000]
...
60.004: (deliver r1 meds1 ward-a) [2.000]
```



Ventanas y plazos

- **Ventana**: un literal abre, otro posterior cierra; las acciones necesitan el hecho over `all`
- **Plazo**: un hecho «aún a tiempo», que borra un literal temporizado y que deliver necesita `(at end (in-time ?i))`

```
(in-time meds1) (in-time meal1)
(at 50 (not (in-time meds1))))
```

Servir antes la comida da la misma longitud, e incumple el plazo:

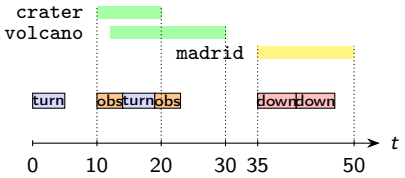
```
$ kairos validate courier-hours-domain.
    pddl courier-hours-p03.pddl
    courier-hours-mealfirst.plan
courier-hours-mealfirst.plan:14: error:
    step 13 (deliver r1 meds1 ward-a) at
    88.012: at end condition false: (
    in-time meds1) is false [validate.
    precondition_failed]
...
```

- Un plazo de 20 (p04): `unsolvable`, con demostración. Incluso ignorando los borrados y con las duraciones más cortas, `(delivered meds1)` llega después de que desaparezca `(in-time meds1)`

Un satélite y sus ventanas

```
$ kairos satellite-domain.pddl
    satellite-p01.pddl
; cost = 47.001
; makespan = 47.001
0.000: (turn volcano crater) [5.000]
10.000: (observe crater) [4.000]
14.001: (turn crater volcano) [5.000]
19.002: (observe volcano) [4.000]
35.000: (downlink volcano madrid)
    [6.000]
41.001: (downlink crater madrid) [6.000]
```

```
(at 10 (visible crater))
(at 20 (not (visible crater)))
(at 12 (visible volcano))
(at 30 (not (visible volcano)))
(at 35 (in-sight madrid))
(at 50 (not (in-sight madrid)))
```

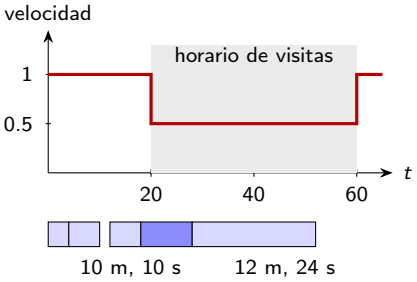


- Gira primero hacia el cráter, cuya ventana se cierra antes
- Cráter visible 3 s, exposición de 4 s (p02): `unsolvable`

Números en el calendario

```
(= (speed r1) 1)
(at 20 (= (speed r1) 0.5))    ; a timed initial fluent
(at 60 (= (speed r1) 1))
```

- Un **fluente inicial temporizado**: no es PDDL 2.2, sino una extensión con `:timed-initial-fluents`
- El último desplazamiento, de 12 m, tarda 24 s
- El desplazamiento que empieza en 18,004 conserva sus 10 s: una duración se calcula **cuando la acción empieza**



El método MODEL: consecuencias y calendario

Preguntas nuevas para los pasos M y O

M: ¿cambia el mundo en instantes conocidos, haga lo que haga el agente? Un horario en *Supuestos*; literales temporizados en el problema; `over all` o `at end` en las acciones. **O**: ¿es una propiedad consecuencia de otras? Un predicado derivado: un número fijo de hechos (`free`) o una cadena de longitud desconocida (`reachable`).

`(free ?r)`	Mantenido por efectos	Derivado	Escrito en línea
Fidelidad	correcto si cada acción lo acompasa	por construcción	por construcción
Guía	la mejor: las heurísticas lo ven	a través de la regla `:derived-predicates`	a través de un `exists` negado precond. negativas y existenciales
Portabilidad	cualquier planificador		

- `free`: con efectos sigue siendo buena opción. `reachable`: con efectos, imposible
- Un plazo: (`within` 50 (`delivered meds1`)) (PDDL 3) es más claro; los literales temporizados, más portables y con ventanas

Errores comunes

Una regla que solo mira un paso más allá

```
(:derived (reachable ?a ?b - location)
  (or (passable ?a ?b)
      (exists (?c - location)
        (and (passable ?a ?c) (passable ?c ?b)))))
```

Legal, y significa otra cosa. Prueba a más profundidad que la regla.

Escribir un predicado derivado

Un efecto o un hecho inicial antiguo que se ha quedado: bórralo.

El hecho ya es cierto

`(is-open pharmacy)` y `(at 30 (is-open pharmacy))`: el robot recoge en 10. Escribe el primer literal temporizado como el primer *cambio*.

Una ventana comprobada solo al inicio

Una acción empieza antes del cierre y termina en una habitación cerrada. Decide con las tres preguntas; prueba con una ventana más corta que la acción.

Un requisito olvidado

`:timed-initial-literals` va con los demás; Kairos avisa.

1 Acciones durativas

2 Coordinación en el tiempo

3 Cambio continuo

4 Predicados derivados y un mundo con calendario

5 Resumen

Qué llevarse

- Una **acción durativa**: condiciones `at start`, `over all`, `at end`; efectos al inicio y al final. Un plan es un horario; su makespan es (`total-time`).
- Las acciones se solapan salvo que el modelo las separe: invariantes, testigos (`idle`), ϵ entre acontecimientos que interfieren. Tomar al inicio, dar al final.
- **Concurrencia requerida**: acciones envolventes. Kairos M2 frente a M3; `no plan found` (14) no es `unsolvable` (10). **Duraciones elegidas**: cotas y `?duration` en los efectos; una STN convierte órdenes en tiempos.
- **Efectos continuos**: ritmos con `#t`, que se suman; límites `over all`. Úsalos cuando alguien lee el número a mitad de la acción.
- **Predicados derivados**: hechos calculados por reglas, recursivas para caminos. **Literales iniciales temporizados**: un calendario que sigue el mundo; ventanas y plazos.

Ejercicios para practicar

- **10.1.1** Secuencia frente a cada robot por su cuenta: dibuja el cronograma
- **10.7.1** El testigo olvidado en una acción: encuéntralo con una prueba
- **11.1.1** Una reparación tan larga como la cerilla: ¿qué código de salida, y por qué no 10?
- **11.7.2** El sótano de las cerillas con un cerrojo: qué sale mal
- **12.4.1** El generador empieza con 90 litros: la ventana de inicios válidos
- **12.4.3** Sin verter en un generador en marcha: ¿qué problemas conservan un plan?
- **12.6.2** Quita la cota inferior de `charge`: ¿qué planes salen para `p02`?
- **13.3.4** «Encendido si no apagado» y «apagado si no encendido» como predicados derivados
- **13.6.3** El satélite, primero el volcán: ¿qué condición falla, y cuándo?