Sistemas híbridos: PDDL+

Planificación y PDDL, Parte VI (Capítulos 17–18)

Francisco Martín Rico

Intelligent Robotics Lab, URJC



Índice

1. Procesos y eventos
2. Modelar sistemas híbridos
3. Resumen

1 Procesos y eventos

2 Modelar sistemas híbridos

3 Resumen

Un mundo que no espera

- Hasta ahora el mundo solo cambiaba por el **agente**
- Cap. 12: la batería solo baja en un `move`; parado, se conserva
- Mundo real: una muestra se calienta, un depósito se llena, una pelota cae
- Literales temporizados (cap. 13): horas fijas. Aquí: *mientras* el robot esté encendido, *cuando* el depósito se llene

¿De quién es el cambio?

Acción durativa: del agente, no sobrevive a la acción. Mundo: el agua fluye *con la válvula abierta*; el agente solo decide cuándo abrir y cerrar.

PDDL+ (Fox y Long, 2006)

Para **sistemas híbridos**: estado discreto (válvula) y estado continuo (nivel). Todo PDDL 2.1, más:

- **proceso**: cambia números de forma continua mientras se cumple su condición
- **evento**: cambia el estado en el primer instante en que su condición es cierta

Las acciones siguen ahí: son las únicas decisiones del agente.

El depósito: dos acciones, un proceso y un evento

```
;; What the agent decides
(:action open-valve
  :parameters ()
  :precondition (not (valve-open))
  :effect (valve-open))

(:action close-valve
  :parameters ()
  :precondition (valve-open)
  :effect (not (valve-open)))

;; What the world does while the valve is open
(:process fill
  :parameters ()
  :precondition (valve-open)
  :effect (increase (level) (* #t (inflow))))

;; What the world does at the instant the tank is full
(:event overflow
  :parameters ()
  :precondition (and (>= (level) (capacity)) (not (flooded)))
  :effect (flooded))
```

- Léelo en tres bloques:
  - lo que decide el **agente**: `open-valve`, `close-valve`
  - el mundo **mientras** se cumple una condición: `fill`
  - el mundo **en un instante**: `overflow`
- Las acciones son **instantáneas**: son las únicas decisiones
- Un plan da esas decisiones y sus instantes; lo que hace el mundo entre medias no está en él

Procesos: cambio mientras se cumple una condición

```
(:process fill
  :parameters ()
  :precondition (valve-open)
  :effect (increase (level) (* #t (inflow))))
```

- `:precondition`: no es algo que conseguir, sino cuándo el proceso **está en marcha**
- `:effect`: solo `increase`/`decrease` con `#t`; ni hechos ni `assign`
- `#t`: tiempo en marcha; `(inflow)`: el **ritmo**
- Nunca en un plan. Requisito `:time`

Control indirecto

El agente no controla `fill`: controla el hecho `(valve-open)`.

Las acciones cambian hechos, los hechos habilitan procesos, los procesos cambian números.

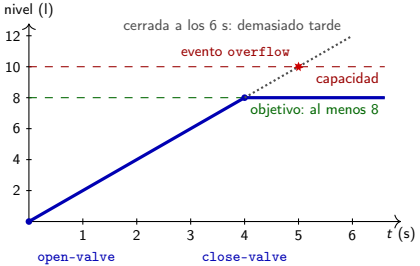
En una línea

Un proceso es una acción que nadie elige: su precondición dice cuándo funciona; su efecto, siempre con `#t`, a qué ritmo.

El primer plan

Objetivo de `tank-p01`: al menos 8 l, válvula cerrada, sin inundación

```
$ kairos tank-domain.pddl tank-p01.pddl
; cost = 4
; makespan = 4.000
0.000: (open-valve) [0.000]
4.000: (close-valve) [0.000]
4.000: @PlanEND
```



La pendiente es el ritmo de `fill`. De puntos: la válvula cerrada dos segundos después.

- **El hueco**: de 0 a 4 s funciona `fill`: $2 \times 4 = 8$ l
- `@PlanEND`: puede pasar tiempo tras la última acción
- Coste = duración, **no** mínima: pocas acciones

Los instantes son el plan

```
$ cat tank-p01-wait.plan
; A plan written by hand: closing at 4.5 s
    is as good as at 4 s.
0.000: (open-valve)
4.500: (close-valve)
$ kairos validate tank-domain.pddl tank-p01.
    pddl tank-p01-wait.plan
plan valid, makespan 4.5
```

El validador de Kairos reproduce cualquier plan en tiempo continuo.

- Vale cualquier instante de cierre en $[4, 5)$
- Clásica: *qué* acciones; durativa: también *orden y solapamiento*
- PDDL+: los **propios instantes**: válido a los 4,5 s, inválido a los 3 o a los 6
- Buscar entre instantes es caro: los problemas siguen siendo pequeños

Eventos: cambio en un instante

```
(:event overflow
  :parameters ()
  :precondition (and (>= (level) (capacity)) (not (flooded)))
  :effect (flooded))
```

- Efectos de una acción: hechos, `assign`, saltos
- **Se dispara** en el primer instante en que se cumple su precondición: ni antes, ni después, ni se omite
- Solo se evita manteniendo falsa su condición
- Deja una marca, que el objetivo rechaza
- **Debe hacer falsa su propia precondición**: `(not (flooded))`

```
$ kairos validate tank-domain.pddl tank-p01.pddl tank-p01-late.plan
tank-p01-late.plan: error: the goal does not hold at the end of the plan: (
    flooded) is true [validate.goal_not_reached]
plan invalid
```

Un evento como objetivo

`tank-p02`: el objetivo es la inundación

```
$ kairos tank-domain.pddl tank-p02.
    pddl
; cost = 5
; makespan = 5.000
0.000: (open-valve) [0.000]
5.000: @PlanEND
```

- Una acción y luego cinco segundos en los que el agente no hace nada
- El plan termina en `@PlanEND`, cuando el evento se ha disparado
- Aquí el final importa: `open-valve` sola, terminando en el instante cero, no alcanza el objetivo

Sin `(not (flooded))`

A los 5 s el evento se dispara y su condición sigue cumpliéndose (el nivel está en la capacidad): se dispararía una y otra vez en el mismo instante. PDDL+ lo prohíbe.

Un problema sin plan

`tank-p03`: 12 l en un depósito de 10, sin inundación. Ninguna temporización de la válvula lo consigue.

```
$ kairos tank-domain.pddl tank-p03.pddl
note: no plan in steps of time; a plan that needs finer timing than the steps
    would be missed, so this is evidence but not a proof [orchestrator.
    steps_exhausted]
...
kairos: no plan found (exhausted)
```

- No es el `unsolvable` de los capítulos clásicos: código de salida 14, no 10
- Kairos probó todas las temporizaciones que considera, no todos los instantes: no afirma demostrar nada
- Paso E: «sin plan» pasa a ser «Kairos no halla ninguno», *más* nuestro razonamiento

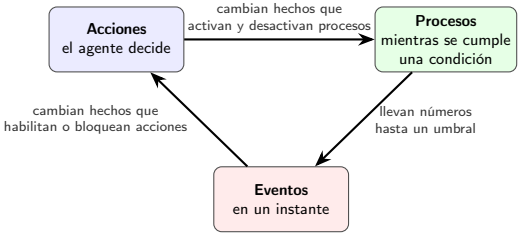
Tres clases de cambio

	Acción	Proceso	Evento
Quién lo causa	el agente	el mundo	el mundo
Cuándo	cuando el planificador decide, si se cumple la precondición	todo el tiempo que se cumple la precondición	en el primer instante en que se cumple la precondición
Cuánto dura	un instante	lo que dure la condición	un instante
Efectos	hechos y números, a saltos	solo números, de forma continua (`#t`)	hechos y números, a saltos
En el plan	sí, con su instante	no	no

El patrón inicio-proceso-parada

Para dar al agente una actividad que pueda empezar y terminar: una **acción** que añade un hecho, un **proceso** que funciona mientras se cumple el hecho, una **acción o evento** que lo borra. Depósito: `open-valve`, `fill`, `close-valve`. En todos los modelos del capítulo.

Cómo se alimentan unas a otras



Solo la caja de la izquierda está bajo el control del planificador.

Qué ocurre entre dos acciones

1. Ninguna condición de evento es cierta: **el tiempo fluye**, cada proceso activo cambia sus números a su ritmo
2. Una condición de evento se vuelve cierta: el tiempo se detiene, el evento se dispara y otros pueden seguirle en el mismo instante
3. Después el tiempo vuelve a fluir

Dos procesos sobre el mismo número: **los ritmos se suman**. Eventos que se disparan sin fin en un instante: el modelo está mal.

La pelota que rebota

```
;; The hand waits at 2 m: the ball must pass slowly enough
(:action catch
  :parameters ()
  :precondition (and (falling)
                     (>= (height) 1.9) (<= (height) 2.1)
                     (>= (velocity) -2) (<= (velocity) 2))
  :effect (and (not (falling)) (held)
               (assign (velocity) 0)))

;; Two quantities change at once: the height with the velocity,
;; the velocity with gravity
(:process fall
  :parameters ()
  :precondition (falling)
  :effect (and (increase (height) (* #t (velocity)))
               (decrease (velocity) (* #t (gravity)))))

;; On the floor and going down: the velocity turns around
(:event bounce
  :parameters ()
  :precondition (and (falling) (<= (height) 0)
                     (< (velocity) 0))
  :effect (and (assign (velocity)
                       (* -1 (* (restitution) (velocity))))
               (increase (bounces) 1))))
```

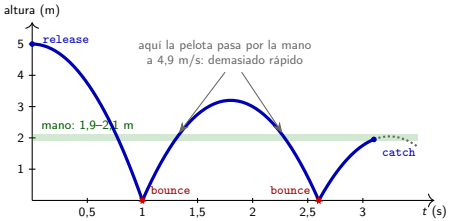
- **Un proceso, dos efectos**: el ritmo de la altura es la velocidad, que a su vez cambia: una parábola
- **Un evento que asigna**: `bounce` conserva el 80 % de la velocidad, hacia arriba; la velocidad pasa a ser positiva, así que se desactiva a sí mismo
- **Una ventana estrecha**: `catch` solo a menos de 10 cm de la mano y a menos de 2 m/s

La pelota: escribimos las leyes, el planificador halla cuándo

`ball-p01`: desde 5 m, gravedad 10, atrapar tras dos rebotes

```
$ kairos ball-domain.pddl
    ball-p01.pddl
; cost = 3.1
; makespan = 3.100
0.000: (release) [0.000]
3.100: (catch) [0.000]
3.100: @PlanEND
```

- Suelo en $t = 1$ s, a -10 m/s; rebote: $+8$
- Suelo otra vez en $t = 2{,}6$, a -8; rebote: $+6{,}4$
- En $t = 3{,}1$: 1,95 m, 1,4 m/s: dentro de la ventana



Cada arco es el proceso `fall`; las estrellas son `bounce`. Solo cerca de la cima del tercer arco va la pelota lo bastante despacio.

El coche: el conductor solo cambia la aceleración

```
;; Parking needs the car at rest and the pedals released
(:action park
  :parameters ()
  :precondition (and (running) (= (acceleration) 0)
                     (>= (velocity) -0.05) (<= (velocity) 0.05))
  :effect (and (not (running)) (parked)
               (assign (velocity) 0)))

(:process moving
  :parameters ()
  :precondition (running)
  :effect (and
    (increase (distance-run) (* #t (velocity)))
    (increase (velocity) (* #t (acceleration)))))

(:event radar
  :parameters ()
  :precondition (and (> (velocity) (speed-limit))
                     (not (fined)))
  :effect (fined)))
```

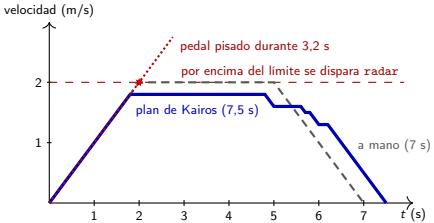
- `accelerate`, `brake`: ± 1 de aceleración, en $\{-1,0,1\}$
- `moving`: aceleración $\rightarrow$ velocidad $\rightarrow$ distancia
- `radar`: marca `fined` por encima del límite

Paso D: quién causa cada cambio

Agente: pedales, aparcar (3 acciones). *Mundo, mientras*: velocidad y distancia (1 proceso). *Mundo, en un instante*: el radar (1 evento). `drive-ten-meters` daría al agente un poder que no tiene.

El coche: válido no es bueno

```
$ kairos car-domain.pddl car-p01.
    pddl
; cost = 33.8
0.000: (accelerate) [0.000]
0.300: (brake) [0.000]
...
; better plan 2
; cost = 7.5
0.000: (accelerate) [0.000]
1.800: (brake) [0.000]
...
7.500: (park) [0.000]
```



- Plan 1: cinco acciones, el mínimo; avanza a 0,3 m/s durante medio minuto
- Plan 2: válido; un profesor de autoescuela no lo firmaría
- A mano (discontinua), 2 + 3 + 2 s: válido, *makespan* 7
- Pedal pisado 3,2 s: `(fined) is true`

El mensajero: un desplazamiento es una acción, un proceso y un evento

```
;; --- The decisions of the agent: instantaneous actions ---
(:action start-move
  :parameters (?r - robot ?from ?to - location)
  :precondition (and (powered ?r) (robot-at ?r ?from)
                     (connected ?from ?to))
  :effect (and (not (robot-at ?r ?from)) (moving ?r ?from ?to)
               (assign (traveled ?r) 0)))
;; --- What the world does: processes and events ---
;; The robot advances while it is on its way and on
(:process travel
  :parameters (?r - robot ?from ?to - location)
  :precondition (and (moving ?r ?from ?to) (powered ?r))
  :effect (increase (traveled ?r) (* #t (speed ?r))))

;; It arrives at the instant it has covered the distance
(:event arrive
  :parameters (?r - robot ?from ?to - location)
  :precondition (and (moving ?r ?from ?to)
                     (>= (traveled ?r) (distance ?from ?to)))
  :effect (and (not (moving ?r ?from ?to)) (robot-at ?r ?to)))
```

- La batería se descarga ahora **siempre que el robot está encendido**
- `start-move`: la decisión; pone a cero el odómetro; en camino, en ninguna habitación
- `travel`: mientras se mueve *y está encendido*
- `arrive`: al cubrir la distancia; borra `moving` y detiene ambos
- Un evento a propósito: cubierta la distancia, *está* allí

La batería: un proceso y un evento

```
;; The battery drains whenever the robot is on
(:process drain
  :parameters (?r - robot)
  :precondition (powered ?r)
  :effect (decrease (battery ?r) (* #t (drain-rate ?r))))

;; An empty battery switches the robot off, wherever it is
(:event shutdown
  :parameters (?r - robot)
  :precondition (and (powered ?r) (<= (battery ?r) 0))
  :effect (not (powered ?r))))
```

- A `drain` no le importa si se mueve
- `shutdown` borra `(powered ?r)`: se detiene a sí mismo, a `drain` y a `travel`
- Vacía en el camino: atascado para siempre
- Sin precondición «batería para el viaje»: de eso se encargan las leyes

Paso 0: el estado de un modelo híbrido

Dos clases nuevas de estado:

- hechos que solo activan procesos: `(powered ?r)`, `(moving ?r ?f ?t)`
- números que miden un proceso: `(traveled ?r)`

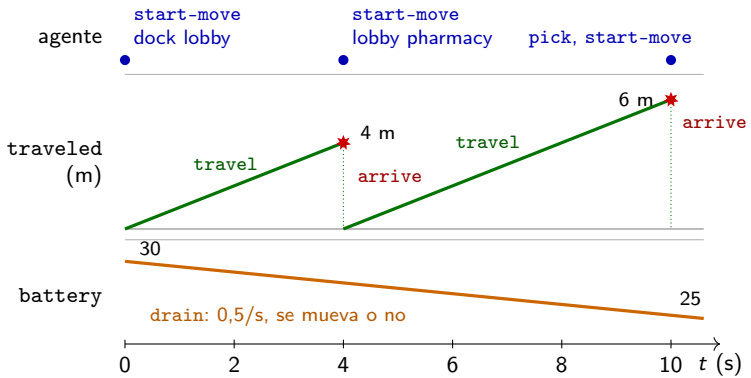
Para cada proceso: ¿qué hecho lo arranca y lo detiene, y qué número dice cuánto ha avanzado?

Ensayar el modelo

```
$ kairos courier-battery-domain.pddl
    courier-battery-p01.pddl
; cost = 38
0.000: (start-move r1 dock lobby) [0.000]
4.000: (start-move r1 lobby pharmacy) [0.000]
10.000: (pick r1 meds1 pharmacy) [0.000]
10.000: (start-move r1 pharmacy lobby) [0.000]
16.000: (start-move r1 lobby corridor) [0.000]
26.000: (start-move r1 corridor ward-a)
    [0.000]
38.000: (drop r1 meds1 ward-a) [0.000]
38.000: @PlanEND
```

- Batería 30, descarga 0,5/s: 60 s encendido; viaje de 38 m a 1 m/s
- Solo los **inicios** de los desplazamientos: las llegadas son eventos, se deducen de las distancias
- Cada `start-move` en el instante de la llegada anterior
- Final: $30 - 0{,}5 \times 38 = 11$
- p02, batería 8 (16 s): `no plan found` tras 221 664 estados

Los diez primeros segundos, en tres carriles



Arriba: decisiones. En medio: `travel` sube el odómetro hasta que se dispara `arrive`; el siguiente `start-move` lo pone a cero. Abajo: a `drain` le da igual.

La prueba de la trampa: un robot que avanza por inercia

```
$ kairos courier-battery-coast-domain.pddl
    courier-battery-coast-p02.pddl
; cost = 38
...
10.000: (start-move r1 pharmacy lobby) [0.000]
15.400: (power-off r1) [0.000]
16.000: (power-on r1) [0.000]
16.000: (start-move r1 lobby corridor) [0.000]
16.500: (power-off r1) [0.000]
26.000: (power-on r1) [0.000]
...
38.000: (drop r1 meds1 ward-a) [0.000]
```

- Error: `travel` solo con `(moving ?r ?from ?to)`, sin `(powered ?r)`
- Resuelve `p01` con el mismo plan
- En `p02` (batería 8, sin plan): inicia un viaje, se apaga y el proceso erróneo sigue empujando
- Batería 0,3 al primer `power-off`: basta para el día
- Una máquina de movimiento perpetuo, porque el modelo contenía una

El modelo correcto rechaza el viaje gratis

```
$ kairos validate courier-battery-domain.pddl courier-battery-p02.pddl
    courier-battery-coast-p02.plan
courier-battery-coast-p02.plan:8: error: step 7 (start-move r1 lobby corridor)
     at 16.000: precondition false: (robot-at r1 lobby) is false [validate.
    precondition_failed]
 8 | 16.000: (start-move r1 lobby corridor) [0.000]
plan invalid
```

Apagado, el robot se quedó a 60 cm del vestíbulo. Modelos clásicos: la trampa era una precondición que faltaba; modelos híbridos: falta una **condición de un proceso**.

El planificador busca viajes gratis

Un proceso que funciona cuando no debe, o que no funciona cuando debe. Prueba con un problema cuyos únicos planes necesitarían uno.

Una muestra que se calienta

```
;; Picking an item takes it out of the fridge, if it was in one
(:action pick
  :parameters (?r - robot ?i - item ?l - location)
  :precondition (and (robot-at ?r ?l) (item-at ?i ?l) (free ?r))
  :effect (and (not (item-at ?i ?l)) (not (free ?r))
               (carrying ?r ?i) (not (chilled ?i))))
;; Dropping a sample into the fridge of the room
(:action store
  :parameters (?r - robot ?i - sample ?l - location)
  :precondition (and (robot-at ?r ?l) (carrying ?r ?i)
                     (has-fridge ?l))
  :effect (and (not (carrying ?r ?i))
               (item-at ?i ?l) (free ?r) (chilled ?i)))
;; Nobody does this: it happens to every sample out of a fridge
(:process warm-up
  :parameters (?i - sample)
  :precondition (and (not (chilled ?i)) (not (spoiled ?i)))
  :effect (increase (temperature ?i) (* #t (warming-rate))))

(:event spoil
  :parameters (?i - sample)
  :precondition (and (>= (temperature ?i) (max-temperature ?i))
                     (not (spoiled ?i)))
  :effect (spoiled ?i)))
```

- `warm-up` funciona en toda muestra no `(chilled ?i)`: ninguna acción lo «arranca»
- El agente lo controla con los hechos de su condición: `pick` la saca, `store` la mete en una nevera
- `courier-cold-p01`: robot en el pasillo, medicamentos para la sala A; muestra en el laboratorio a 4°, 0,1°/s, estropeada a 8: 40 s fuera de una nevera; neveras en laboratorio y UCI

Primero los medicamentos: falla

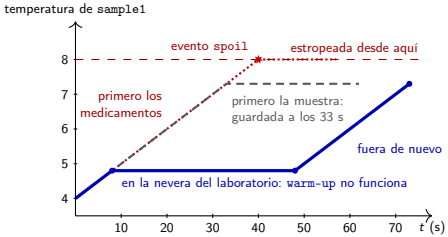
Un planificador que cuenta acciones tiene la tentación de entregar primero los medicamentos: diez acciones. Escribimos ese plan a mano y lo validamos.

```
$ kairos validate courier-cold-domain.pddl courier-cold-p01.pddl
    courier-cold-p01-meds-first.plan
courier-cold-p01-meds-first.plan: error: the goal does not hold at the end of
    the plan: (spoiled sample1) is true [validate.goal_not_reached]
plan invalid
```

- Todas las acciones son aplicables; falla por algo que **ninguna acción hizo**
- El robot llega al laboratorio a los 32 s y a la UCI a los 57; la muestra se estropeó a los 40

El planificador aparca la muestra

```
$ kairos courier-cold-domain.pddl
    courier-cold-p01.pddl
...
0.000: (start-move r1 corridor lab)
    [0.000]
8.000: (pick r1 sample1 lab) [0.000]
8.000: (store r1 sample1 lab)
    [0.000]
...
48.000: (pick r1 sample1 lab)
    [0.000]
...
73.000: (store r1 sample1 icu)
    [0.000]
```



- En la nevera *del laboratorio* de 8 a 48 s; UCI a 7,3°: lo que haría una enfermera sensata
- Primero la muestra: 62 s, once acciones, también válido

Cómo planifica Kairos con PDDL+

Búsqueda en pasos de tiempo

Kairos añade acciones propias que solo dejan pasar el tiempo:

- una **espera corta**: 0,1 s
- una **espera larga**: pasos de 0,1 s hasta que una condición numérica está a punto de cambiar

Los procesos se integran con **Runge-Kutta**; los eventos se disparan entre pasos. Las esperas son los huecos entre las líneas del plan.

Validación en tiempo continuo

Un plan por pasos es solo un candidato. Antes de imprimirlo, Kairos integra de forma exacta y dispara cada evento en el instante en que se cruza su condición. Los candidatos rechazados se descartan. `kairos validate` aplica la misma comprobación a cualquier plan.

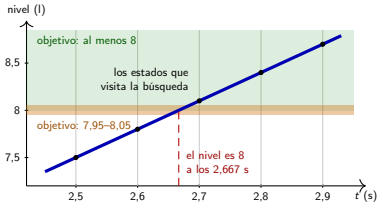
- Instantes múltiplos de 0,1: válvula a los 4,0, no a los 4,5
- Robot por inercia apagado a los 15,4: último paso antes de vaciarse
- Plan 1 del coche: pocos pasos de búsqueda, mucho tiempo

Lo que se le puede escapar

`tank-p04`: caudal de 3 l/s; 8 l a los 2,667 s

```
$ kairos tank-domain.pddl tank-p04.
    pddl
; cost = 2.7
; makespan = 2.700
0.000: (open-valve) [0.000]
2.700: (close-valve) [0.000]
2.700: @PlanEND
```

- «Al menos 8»: un objetivo ancho, un paso cae en él (8,1 l)
- Un objetivo de 7,95–8,05 l: `no plan found`, aunque existe un plan



Otras formas de planificar con PDDL+

Paso fijo (ENHSP), o ecuaciones para un resolutor de restricciones; todos comprueban contra la semántica continua (VAL). De los ocho planificadores del Apéndice B, solo Kairos planificó el dominio de prueba con procesos: los modelos menos portables del libro.

El método MODEL: quién causa cada cambio

Paso	Preguntas nuevas
M	¿Cambia el mundo cuando el agente no hace nada? ¿Esperar cuesta algo, o consigue algo?
O	Para cada cantidad que cambia por sí sola: ¿qué hecho enciende y apaga el cambio, y qué número mide cuánto ha avanzado?
D	Para cada cambio: ¿quién lo causa? El agente, por una decisión (acción); el mundo, mientras se cumple una condición (proceso); el mundo, en el instante en que una condición se vuelve cierta (evento).
D	Para cada evento: ¿qué vuelve a hacer falsa su condición una vez que se ha disparado?
E	¿Hay algún tramo de tiempo en el que un proceso bueno funcione gratis, o uno malo esté apagado? Escribe el problema que lo necesita.

Una prueba rápida para el paso D: el agente se queda dormido

Lo que siga ocurriendo es un proceso o un evento; lo que deje de ocurrir era una acción. El agua sigue entrando (proceso), el depósito se desborda (evento), la válvula se queda como estaba (acciones).

Alternativas comparadas: el final de un desplazamiento

	Llegada como evento	**Llegada como acción**	**Desplazamiento durativo**
Fidelidad	Buena; un robot que se apaga por el camino se queda allí	Puede quedarse entre habitaciones (plan de 44 s)	Buena; un apagado rompe `over all`: plan inválido
Tamaño	32 acciones, 15 procesos, 15 eventos	46 acciones, 15 procesos, 1 evento	18 acciones, 14 durativas, 1 proceso, 1 evento
Guía	76 139 estados, 0,4 s	290 833 estados, 1,4 s	279 estados, 0,01 s
Claridad	Tres piezas; el plan solo muestra inicios	Inicios y paradas, como un registro	Una pieza; el plan muestra duraciones
Portabilidad	Solo PDDL+	Solo PDDL+	Planificadores PDDL+ que acepten acciones durativas

Veredicto

Una actividad que termina sola: no hagas de su final una decisión. Prefiere la **acción durativa** si la duración se conoce al empezar y nada la interrumpe (búsqueda dos órdenes de magnitud menor). Quédate con el **evento** si el final depende de cómo evoluciona el mundo.

Errores comunes (1)

Un evento que no se desactiva a sí mismo

`overflow` solo con `(>= (level) (capacity))`: `kairos check` lo acepta; el validador dice `events keep firing at 5.000000 (overflow): an event must make its own condition false`. Solución: un hecho que añade y niega, un hecho que borra o un número que vuelve a cruzar el umbral.

Un proceso sin #t, o #t fuera de un proceso

`(increase (level) (inflow))` en un proceso: `processes can only have continuous effects (with #t)`. `#t` en `open-valve`: `#t can only appear in increase/decrease effects of durative actions and processes`. Solución: la acción añade un hecho y un proceso hace el llenado.

Un proceso que funciona cuando no debe

El robot que avanza por inercia: nada rechaza el dominio, los problemas corrientes dan planes corrientes. Solución: escribe en la precondición todo lo que debe cumplirse para que el cambio continúe; ejecuta la prueba de la trampa.

Errores comunes (2)

Leer «no plan found» como «unsolvable»

`kairos: no plan found (exhausted)`, código de salida 14: todas las temporizaciones de su rejilla, no todas las temporizaciones. Escribe a mano el plan esperado y dáselo a `kairos validate`, que no tiene rejilla.

Olvidar `:time`, o usar las palabras clave de una acción durativa

Sin `:time`: se acepta con un aviso (`uses processes but :requirements does not include :time`); otros planificadores pueden rechazar el fichero. Procesos y eventos llevan `:precondition` y `:effect`; `:condition` es un error.

1 Procesos y eventos

2 Modelar sistemas híbridos

3 Resumen

Reglas que se cumplen en todo momento: restricciones globales

```
(:constraint keep-the-limit
  :parameters ()
  :condition (<= (velocity) (speed-limit)))
```

Sustituye al evento radar + objetivo `(not (fined))`

- Se cumple en **cada instante**; sin efecto, solo prohíbe
- Viene de ENHSP, no del PDDL+ original; no es el `(:constraints ...)` de PDDL 3

```
$ kairos validate car-limit-domain.pddl car-limit-p01.pddl car-limit-p01-fast.
    plan
car-limit-p01-fast.plan: error: constraint 'keep-the-limit' broken at 2.000001
    : (<= (velocity) (speed-limit)): 2 <= 2 is false [validate.
    global_constraint]
plan invalid
```

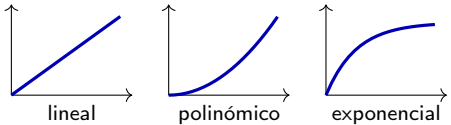
Pedal pisado durante 3,2 s: qué regla se rompió y cuándo. **Restricción**: lo que no debe ocurrir nunca.

Evento: algo malo con consecuencias (una multa, una muestra estropeada).

Cambio que no es lineal

Efecto (`* #t` *ritmo*): el ritmo decide la forma

- **constante**: una recta (**lineal**, Capítulo 12)
- **otra cantidad que cambia**: distancia con la velocidad como ritmo: una parábola
- **la propia cantidad**: enfriamiento, resistencia del aire: se acerca a un límite



Por qué no basta una escalera

Ritmo fijo durante cada paso: exacto para el cambio lineal, erróneo en otro caso. La pelota desde 5 m en pasos de 0,1 s: a 0,5 m al cabo de un segundo, cuando ya está en el suelo. Kairos integra con **Runge-Kutta**: planifica y valida de forma exacta; las heurísticas linealizan, lo que cuesta esfuerzo de búsqueda, pero nunca validez.

Portabilidad: los planificadores del Capítulo 12 suelen esperar ritmos constantes.

Un coche con resistencia del aire

```
(:process moving
  :parameters ()
  :precondition (running)
  :effect (and
    (increase (distance-run) (* #t (velocity)))
    (increase (velocity) (* #t (acceleration)))))

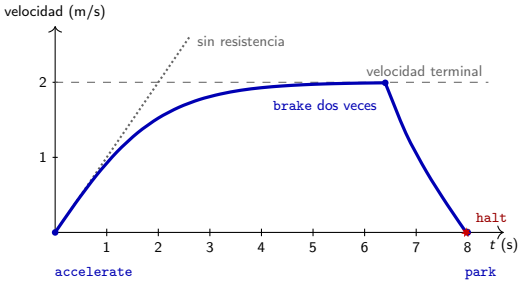
;; A second process on the same quantity: the two rates add up
(:process air-drag
  :parameters ()
  :precondition (and (running) (> (velocity) 0))
  :effect (decrease (velocity)
            (* #t (* (drag) (* (velocity) (velocity))))))

;; A braking car stops at zero: it does not start to go back
(:event halt
  :parameters ()
  :precondition (and (running) (<= (velocity) 0)
                     (< (acceleration) 0))
  :effect (and (assign (velocity) 0)
               (assign (acceleration) 0))))
```

- Dos procesos sobre la velocidad: los ritmos se suman, $a - kv^2$
- **Velocidad terminal** $\sqrt{a/k}$: con $a = 1$, $k = 0{,}25$, 2 m/s. La física pone el límite
- `halt` pone velocidad y aceleración exactamente a 0: `park` puede pedir `(= (velocity) 0)` en cualquier instante posterior
- Un evento en un cruce convierte una **ventana estrecha en un estado estable**

El plan con resistencia

```
$ kairos car-drag-domain.
    pddl car-drag-p01.pddl
; cost = 8
; makespan = 8.000
0.000: (accelerate) [0.000]
6.400: (brake) [0.000]
6.400: (brake) [0.000]
8.000: (park) [0.000]
8.000: @PlanEND
```



Con resistencia, la velocidad se curva hacia la velocidad terminal.

- Dos `brake`: soltar y frenar
- 6,4 s: primer paso a 10,03 m; `halt` a 7,97 s

Procesos que interactúan: un invernadero en invierno

```
;; The plants die of frost below 10 degrees and burn above 30
(:constraint plants-alive
  :parameters ()
  :condition (and (>= (temperature) 10) (<= (temperature) 30)))
(:process heating
  :parameters ()
  :precondition (heater-on)
  :effect (increase (temperature) (* #t (heater-power))))

;; Always active: the warmer inside, the faster the heat leaves
(:process losing-heat
  :parameters ()
  :precondition (> (temperature) (outside))
  :effect (decrease (temperature)
            (* #t (* (loss) (- (temperature) (outside))))))

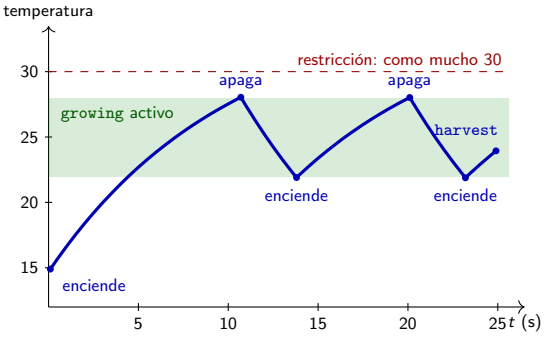
(:process growing
  :parameters ()
  :precondition (and (>= (temperature) 22)
                     (<= (temperature) 28))
  :effect (increase (growth) (* #t 1))))
```

- **Dos procesos, una cantidad**: `heating` $+3$/s, `losing-heat` $-0{,}1(T-5)$/s; los ritmos se suman
- **Un proceso cambia su propio ritmo**: la pérdida disminuye al bajar T
- **Activado por una cantidad**: `growing` solo funciona con $22 \leq T \leq 28$; el agente solo puede dirigir
- Restricción: $10 \leq T \leq 30$

Primero el lápiz, después el plan: *bang-bang*

Calefactor siempre encendido: $0{,}1(T-5)=3$, $T=35$. Siempre apagado: hacia 5. Los dos matan las plantas.

```
$ kairos greenhouse-domain.pddl
    greenhouse-p01.pddl
; cost = 24.9
0.100: (switch-on) [0.000]
10.700: (switch-off) [0.000]
13.800: (switch-on) [0.000]
20.100: (switch-off) [0.000]
23.200: (switch-on) [0.000]
24.900: (harvest) [0.000]
```



Nadie le dijo al planificador que mantuviera T en la banda: solo ahí hay crecimiento.

Un dron que aterriza: eventos excluyentes

```
(:event touch-down
  :parameters ()
  :precondition (and (airborne) (<= (altitude) 0)
                     (>= (velocity) (- 0 (max-impact))))
  :effect (and (not (airborne)) (not (rotors-on)) (landed)
               (assign (velocity) 0)))

(:event crash
  :parameters ()
  :precondition (and (airborne) (<= (altitude) 0)
                     (< (velocity) (- 0 (max-impact))))
  :effect (and (not (airborne)) (not (rotors-on)) (crashed)
               (assign (velocity) 0))))
```

Regla

Eventos que pueden dispararse en el mismo instante y tocan los mismos hechos: haz sus condiciones **excluyentes**.

```
$ kairos drone-domain.pddl
    drone-p01.pddl
; cost = 3.3
; makespan = 3.300
1.100: (rotors-up) [0.000]
2.000: (rotors-down) [0.000]
2.100: (rotors-up) [0.000]
3.300: @PlanEND
```

- Caída libre de 1,1 s hasta -11 m/s; `touch-down` a los 3,28 s, a 1,58 m/s
- Pausa a los 2,0: el paso de 0,1 s es la única resolución de la búsqueda

Eventos en cascada: fichas de dominó

```
(:action push
  :parameters (?d - domino)
  :precondition (and (within-reach ?d) (not (fallen ?d)))
  :effect (fallen ?d))

;; A fallen domino knocks down the next one. The condition
;; (not (fallen ?b)) is what stops the event: it fires once
(:event topple
  :parameters (?a ?b - domino)
  :precondition (and (fallen ?a) (next ?a ?b)
                     (not (fallen ?b)))
  :effect (fallen ?b)))
```

```
$ kairos dominoes-domain.pddl
    dominoes-p01.pddl
; cost = 0
; makespan = 0.000
0.000: (push d1) [0.000]
0.000: @PlanEND
```

- Una cadena de eventos en un instante: una **cascada**
- Cuatro eventos, sin que pase el tiempo; se detiene porque cada evento se desactiva a sí mismo y la fila se acaba
- `dominoes-p02` (d2 caída, d3 en pie): sin plan. No hay instante para una acción dentro de una cascada

En una línea

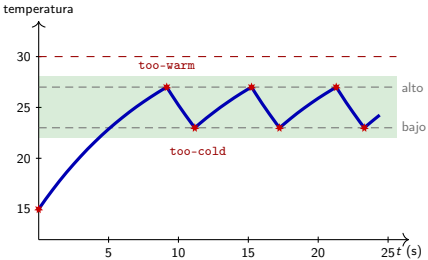
Una cascada debe terminar: cada evento se desactiva a sí mismo y ningún evento vuelve a activar a otro.

Eventos como dispositivos: el termostato

```
;; The thermostat: what the agent decided by hand, now as events
(:event too-cold
  :parameters ()
  :precondition (and (thermostat-on) (not (heater-on))
                     (<= (temperature) (low)))
  :effect (heater-on))

(:event too-warm
  :parameters ()
  :precondition (and (thermostat-on) (heater-on)
                     (>= (temperature) (high)))
  :effect (not (heater-on))))
```

```
$ kairos greenhouse-auto-domain.pddl
    greenhouse-auto-p01.pddl
...
; better plan 2
0.000: (enable-thermostat) [0.000]
24.400: (harvest) [0.000]
```



- Siete eventos, fuera de toda rejilla
- 8 030 estados en lugar de 432 899
- **Dónde está la inteligencia**: ¿hay un termostato en el sistema? Modélalo

Eventos que no paran: el intermitente

Erróneo a propósito

```
(:event light-up
  :parameters ()
  :precondition (and (blinking) (not (lit)))
  :effect (lit))

(:event go-dark
  :parameters ()
  :precondition (and (blinking) (lit))
  :effect (not (lit))))
```

- Cada evento por separado es correcto; juntos se disparan sin fin en el instante 0
- `kairos check`: válido; planificador: sin plan; validador: `events keep firing at 0.000000 (go-dark)`

Arreglado: tiempo entre los eventos

```
(:event light-up
  :parameters ()
  :precondition (and (blinking) (not (lit))
                     (>= (timer) (half-period)))
  :effect (and (lit) (assign (timer) 0)
               (increase (flashes) 1)))

(:event go-dark
  :parameters ()
  :precondition (and (blinking) (lit)
                     (>= (timer) (half-period)))
  :effect (and (not (lit)) (assign (timer) 0))))
```

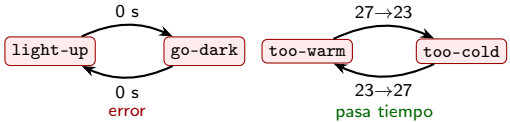
Kairos: `(switch-on)` en 0, `@PlanEND` en 2,5; la lámpara se enciende a los 0,5, 1,5 y 2,5 s. Proceso temporizador + evento que lo reinicia: la expresión idiomática para todo lo periódico o retardado.

Un solo umbral: comportamiento de Zenón

- Umbrales 23 y 27: **histéresis**, tiene que pasar tiempo entre eventos
- `greenhouse-auto-p02`: ambos a 25: infinitos eventos en torno a un instante
- Planificador: `kairos: timeout (timeout)`, con 3 s o con 20
- Validador: acepta «activar y cosechar a los 40 s» (precisión finita)
- Ningún mensaje claro: **herramientas lentas o en desacuerdo**

Paso D: las preguntas para un conjunto de eventos

Dibuja una flecha de A a B si un efecto de A puede hacer cierta la condición de B. Una cadena que termina: una cascada, bien. Un ciclo: bien solo si *tiene que pasar tiempo* en algún punto (histéresis, un temporizador). Un ciclo en tiempo cero es un error.



Procesos con acciones durativas y literales temporizados

Cada construcción, para su cambio:

- **acción durativa**: una actividad del agente que sigue su curso
- **proceso**: cambio según el estado
- **evento**: consecuencia que saca el mundo
- **literal temporizado**: cambio a una hora fija

`courier-plus`: el `move` durativo marca `(driving ?r)`; el agente enchufa y desenchufa, el cargador hace el resto. Encendido: $-0{,}2$/s; en marcha: -1; enchufado: $+1{,}8$.

```
;; Three processes change the same battery; their rates add up
(:process drain
  :parameters (?r - robot)
  :precondition (powered ?r)
  :effect (decrease (battery ?r) (* #t (drain-rate ?r))))

(:process motor-drain
  :parameters (?r - robot)
  :precondition (and (powered ?r) (driving ?r))
  :effect (decrease (battery ?r) (* #t (motor-rate ?r))))

(:process charging
  :parameters (?r - robot)
  :precondition (plugged ?r)
  :effect (increase (battery ?r) (* #t (charge-rate ?r))))

;; The charger cuts off by itself when the battery is full
(:event battery-full
  :parameters (?r - robot)
  :precondition (and (plugged ?r)
                     (>= (battery ?r) (battery-capacity ?r)))
  :effect (not (plugged ?r)))
```

El robot recarga

```
$ kairos courier-plus-domain.pddl
    courier-plus-p02.pddl
; cost = 95.3
0.000: (move r1 lobby pharmacy) [6.000]
6.000: (pick r1 meds1 pharmacy) [0.000]
6.000: (move r1 pharmacy lobby) [6.000]
12.000: (move r1 lobby dock) [4.000]
16.000: (plug r1 dock) [0.000]
69.300: (unplug r1) [0.000]
69.300: (move r1 dock lobby) [4.000]
...
95.300: (drop r1 meds1 ward-a) [0.000]
```

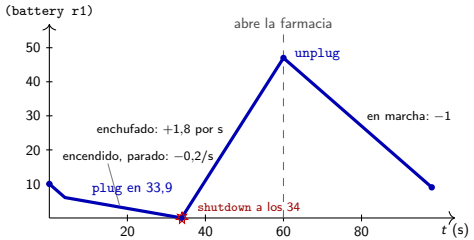
- p02: batería 20; 34 m por recorrer
- En la base de carga con 4 unidades; enchufado 53,3 s; desenchufa a 99,9, un paso antes de `battery-full`
- Necesitaba 26 para el resto: válido y derrochador (la espera larga)
- `(over all (powered ?r))`: un apagado durante un desplazamiento rompe el plan
- 736 estados: duraciones y literales temporizados en instantes exactos

Esperar no sale gratis

p03: batería 10, `(at 60 (is-open pharmacy))`

```
$ kairos courier-plus-domain.pddl
    courier-plus-p03.pddl
; cost = 98
0.000: (move r1 lobby dock) [4.000]
33.900: (plug r1 dock) [0.000]
60.000: (unplug r1) [0.000]
60.000: (move r1 dock lobby) [4.000]
...
98.000: (drop r1 meds1 ward-a)
    [0.000]
```

Esperar un minuto cuesta 12; el robot tiene 10. En la Parte IV solo esperaría en el vestíbulo.



- Se enchufa con 0,02 de carga, un paso antes del apagado; carga hasta los 60 exactos
- Enchufarse al llegar es 10 s más corto; los dos pasan la espera en el cargador

Un sistema, tres modelos

Robot en el vestíbulo, batería 20, medicamentos en la farmacia, la sala A a 34 m, un cargador en la base de carga.

1. Números (cap. 8)

`courier-discrete`: un desplazamiento resta distancia × consumo; `charge` llena la batería.

Nueve acciones en un milisegundo, óptimo demostrado. Sin duraciones, ni la de la carga.

2. Durativo (cap. 12)

`courier-durative`: la batería baja durante un desplazamiento y sube durante un `charge` de duración elegida.

86,008 s; carga 42 s, el límite. Un robot parado no gasta nada.

3. PDDL+

`courier-plus`: descarga siempre que está encendido, cargar es un estado, batería vacía es un evento.

95,3 s; enchufado 53,3 s.

Lo que cada nivel no puede decir

La farmacia abre en un minuto, batería 10: el nivel 1 no puede enunciar la hora; el nivel 2 espera en el vestíbulo y entrega; solo el nivel 3 sabe que esperar vacía la batería. Cuarenta entregas al día: el nivel 1 lo planifica; los otros dos, con casi todos los planificadores, no.

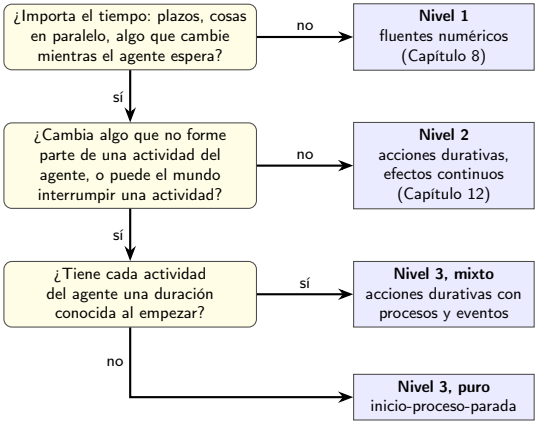
Alternativas comparadas: tres niveles

	1. Números	2. Durativas, continuas	3. PDDL+
Fidelidad	Orden y energía de los desplazamientos. Sin tiempo: carga instantánea, la espera no existe.	Tiempo, solapamiento, energía en las acciones. El cambio se para con ellas: esperar es gratis.	Cambio siempre que el estado lo dice: descarga en reposo, apagado, cargador que se corta.
Tamaño	31 acciones instanciadas	31 acciones durativas	11 acciones, 14 durativas, 3 procesos, 2 eventos
Guía	220 estados, óptimo demostrado	9 629 estados; duraciones probadas en sus extremos	736 estados aquí; en general, el más difícil
Claridad	El más sencillo de leer y de explicar	Una acción por actividad; los planes son cronogramas	Leyes aparte de las decisiones; hay que reproducir los planes
Portabilidad	Todo planificador numérico: 5 de los 8 del Apéndice B	Efectos continuos: dos de los ocho	Procesos: uno de los ocho

Veredicto

Elige el **nivel más bajo que no mienta** sobre aquello de lo que depende el problema. Dentro del nivel 3, conserva las acciones durativas allí donde el nivel 2 las tenía.

Elegir el nivel de un modelo



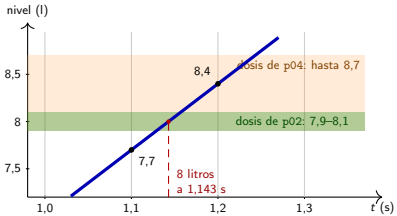
- **Nivel 1**: qué acciones y en qué orden; las cantidades son presupuestos
- **Nivel 2**: el tiempo forma parte de la pregunta, todo cambio se atribuye a una actividad
- **Nivel 3**: algo cambia mientras el agente no hace nada, o el mundo interrumpe
- Cada «sí» compra fidelidad y la paga en esfuerzo de búsqueda y en planificadores que leen el modelo

Límites: una ventana más estrecha que un paso

doser: 8 l $\pm$ 0,1; p02 bombea 7 l/s

```
$ kairos doser-domain.pddl doser-p02.
    pddl
...
kairos: no plan found (exhausted)
$ kairos validate doser-domain.pddl
    doser-p02.pddl doser-p02-hand.plan
plan valid, makespan 1.143
```

- En la dosis de 1,129 a 1,157 s: 29 ms; la búsqueda ve 7,7 l a 1,1 s y 8,4 l a 1,2
- El mismo síntoma que un modelo erróneo



El nivel está en la dosis estrecha entre dos estados de la búsqueda (puntos), separados 0,7 l.

Un evento no rescata a la búsqueda

- p03 arma el interruptor de flotador a 8 l: válido en tiempo continuo, pero `no plan found`
- La búsqueda dispara los eventos *entre pasos*: el interruptor se cierra a 1,2 s con 8,4 l
- El evento hace preciso el **mundo**, no la búsqueda
- p04 acepta hasta 8,7 l (un paso de caudal): el validador lo reproduce y el interruptor se cierra a 1,143 s con 8 l

```
$ kairos doser-domain.pddl
    doser-p04.pddl
; cost = 1.2
; makespan = 1.200
0.000: (open-valve) [0.000]
1.200: @PlanEND
```

Regla para el modelador

Toda ventana en la que deba acertar un plan debería medir al menos **el ritmo por 0,1 s**. Si no: ¿la necesita la historia? Frena el proceso, o planifica de otra forma y valida.

El método MODEL: probar y elegir modelos híbridos

Paso E para un modelo híbrido

1. **Primero, el lápiz**: adónde va cada cantidad con cada posición del interruptor
2. **Comprobar cada evento**: un problema cuyo único plan pase por él; dibujar las flechas
3. **Acotar el horizonte**: márgenes pequeños para los problemas «sin plan»
4. **Validar en tiempo continuo**: el plan que esperas, fuera de la rejilla de 0,1, y el plan que temes
5. **Reproducir y dibujar**: un viaje gratis se ve en una gráfica
6. **Medir las ventanas**: ancho frente a ritmo $\times$ 0,1 s

Paso	Preguntas nuevas
M	¿Hay algo que no deba ocurrir nunca? ¿Un mal resultado está prohibido, o solo tiene consecuencias?
O	Cada ritmo: ¿una constante, otra cantidad que cambia o la propia cantidad?
D	¿Qué procesos tocan la misma cantidad? ¿Qué eventos disparan a cuáles, y dónde tiene que pasar tiempo? ¿Qué dispositivos son eventos?
E	Los seis puntos de la izquierda
L	El nivel más bajo que no mienta

Errores comunes en modelos híbridos (1)

Dos eventos que se activan el uno al otro

`blinker-wrong`: `kairos check` dice válido, el planificador dice sin plan y solo el validador lo nombra (`events keep firing`). La causa es la pareja. Solución: tiempo dentro de cada ciclo de eventos, con un temporizador o dos umbrales.

Un termostato con un solo umbral

Un modelo de Zenón; nada lo avisa. Síntomas: un planificador lento y herramientas que no se ponen de acuerdo. Solución: histéresis, umbrales separados varios pasos.

Una ventana más estrecha que un paso

`no plan found (exhausted)` tras 100 estados aunque existe un plan. Una ventana que se pasa por alto tarda pocos estados; un modelo demasiado estricto, muchos. Confírmalo con un plan a mano; ensancha hasta ritmo $\times$ 0,1 s.

Errores comunes en modelos híbridos (2)

Igualdad sobre una cantidad que fluye

`(= (level) 8)`: una ventana de ancho cero. El validador es exacto: `(= (level) 8): 8.001 = 8 is false`. Solución: dos desigualdades, o un evento que *asigne* el valor exacto (como `halt`).

Esperar el plan más corto

`courier-plus` carga hasta el 99,9 % y espera junto al cargador. En PDDL+, Kairos minimiza acciones; las esperas largas se detienen donde algo cambia. Solución: presión en el modelo (un plazo, una batería más pequeña), o mejorar a mano y validar.

Una restricción donde hacía falta un evento, y al revés

«La batería nunca está vacía» prohíbe los planes de rescate; un límite de velocidad con evento y objetivo falla cuando un problema olvida el objetivo. Imposible: una ley del mundo; prohibido: una restricción; costoso: un evento con consecuencias.

1 Procesos y eventos

2 Modelar sistemas híbridos

3 Resumen

Qué llevarse

- PDDL+ separa lo que el agente **decide** (acciones instantáneas) de lo que hace el mundo: **procesos** (continuos, mientras se cumple una condición) y **eventos** (instantáneos, cuando una condición se vuelve cierta)
- Las acciones controlan los procesos a través de hechos: **inicio-proceso-parada**. Un evento debe desactivarse a sí mismo; ningún ciclo de eventos en tiempo cero
- Los planes enumeran decisiones y `@PlanEND`: hay que reproducirlos para leerlos
- Restricciones globales para lo prohibido; eventos para lo que tiene consecuencias
- Los procesos sobre una cantidad suman ritmos; el cambio puede ser no lineal; se combinan con acciones durativas y literales temporizados
- Kairos: pasos de 0,1 s, Runge-Kutta, validación en tiempo continuo; `no plan found` (14) no es `unsolvable`. Ventanas $\geq$ ritmo $\times$ 0,1 s, ninguna igualdad sobre lo que fluye
- El **nivel más bajo que no mienta**; prueba si hay viajes gratis

Ejercicios para practicar

Capítulo 17

- 17.2.2: un depósito con una fuga
- 17.3.2: un interruptor de boya para el depósito
- 17.4.3: acciones, procesos o eventos en un ascensor
- 17.5.4: los tres planes de la cadena de frío
- 17.6.2: cerrar a los 4,05 s
- 17.7.1: una cafetera

Capítulo 18

- 18.1.1: evento radar frente a restricción, cinco criterios
- 18.2.2: cuándo beberse el café
- 18.4.1: flechas entre eventos en cuatro modelos
- 18.5.1: enchufarse al llegar
- 18.6.1: elige un nivel para cuatro historias
- 18.7.1: el dosificador a 5 l/s