Números: PDDL 2.1

Planificación y PDDL, Parte III (Capítulos 8–9)

Francisco Martín Rico

Intelligent Robotics Lab, URJC



Índice

1. Fluentes numéricos
2. Calidad del plan: métricas y costes de acción
3. Resumen

1 Fluentes numéricos

2 Calidad del plan: métricas y costes de acción

3 Resumen

Lo que un hecho no puede decir

- Un hecho es cierto o falso: sirve para pocos valores
- Una batería es una **magnitud**: un número sobre el que el mundo hace aritmética
- 12 m a 1,5 unidades/m quitan 18 unidades, fuera cual fuera la carga
- Con hechos: un hecho por carga, una tabla de restas a mano
- Es magnitud si el planificador debe sumarlo, restarlo o compararlo



Recurso gastado y repuesto
batería, combustible, dinero, agua



Capacidad compartida
5 kg: 2 + 2 caben, comida de 4 sola



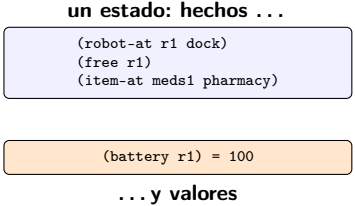
Datos que son números
fijos, las acciones los necesitan

Funciones: números en el estado

```
(:functions
  (battery ?r - robot)              ; charge left, 0 to 100
  (battery-capacity ?r - robot)     ; the most it can hold
  (consumption ?r - robot)          ; charge used per meter
  (distance ?from ?to - location)) ; meters between rooms
```

```
(= (battery r1) 100)
(= (battery-capacity r1) 100)
(= (consumption r1) 1)
```

- Una **función** aplicada a objetos es un **fluente numérico**: un número en el estado
- El problema da los valores iniciales en `:init`, con =
- Sin unidades ni rango: mantener 0–100 es trabajo de las acciones



- **Estática**: ninguna acción la cambia (distancia, capacidad); datos
- **Dinámica**: alguna acción la cambia (carga); estado
- Requisito `:numeric-fluents` (`:fluents` en PDDL 2.1)

Comparar y cambiar números

Condición numérica: < <= = >= >, el operador primero

```
(>= (battery ?r)
    (* (distance ?from ?to) (consumption ?r))))
```

- Expresiones: números, fluentes, + - * / en forma prefija
- Donde pueda ir un hecho, objetivos incluidos
- No hay «distinto de»: `(not (= (load ?r) 0))`

Efecto numérico: cambia un fluente

- `(increase f e)` suma e
- `(decrease f e)` lo resta
- `(assign f e)` sustituye el valor

Se lee el estado anterior a la acción

Todos los efectos ocurren a la vez; cada expresión lee los valores anteriores a la acción. El orden en que se escriben los efectos no significa nada.

La batería en la ficha del modelo

Paso M: lo que la batería cambia

Historia. Moverse gasta carga en proporción a los metros. La base de carga recarga, y una recarga llena la batería.
Alcance. Dentro: la carga, la longitud de cada conexión, el cargador. Fuera: cuánto tarda una recarga, la carga que gastan recoger y dejar, el desgaste.
Supuestos. El mismo consumo por metro lleve lo que lleve. Un robot nunca empieza un movimiento que no puede terminar.

- Un robot varado entre habitaciones no es un estado del modelo
- Así que el modelo debe prohibir el movimiento que lo dejaría varado
- Eso es una **precondición sobre un número**

El mensajero con batería

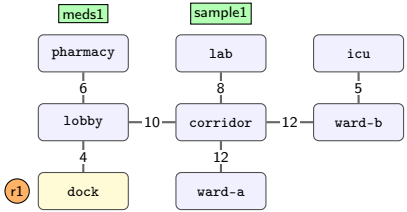
```
(:functions
  (battery ?r - robot)              ; charge left, 0 to 100
  (battery-capacity ?r - robot)     ; the most it can hold
  (consumption ?r - robot)          ; charge used per meter
  (distance ?from ?to - location)) ; meters between rooms

(:action move
  :parameters (?r - robot ?from ?to - location)
  :precondition (and
    (robot-at ?r ?from) (connected ?from ?to)
    ; enough charge for the whole stretch
    (>= (battery ?r)
        (* (distance ?from ?to) (consumption ?r))))
  :effect (and
    (not (robot-at ?r ?from)) (robot-at ?r ?to)
    (decrease (battery ?r)
              (* (distance ?from ?to) (consumption ?r)))))
```

```
(:action charge
  :parameters (?r - robot ?l - location)
  :precondition (and
    (robot-at ?r ?l) (has-charger ?l)
    (< (battery ?r) (battery-capacity ?r)))
  :effect (and
    (assign (battery ?r) (battery-capacity ?r))))
```

- La precondición y el efecto de `move` usan la **misma expresión**; nada en PDDL las ata
- `charge` asigna la capacidad
- `(< (battery ?r) ...)`: no se carga una batería llena, sin estados inútiles
- `pick`, `drop`: el mensajero con tipos

Ejecutarlo: una entrega



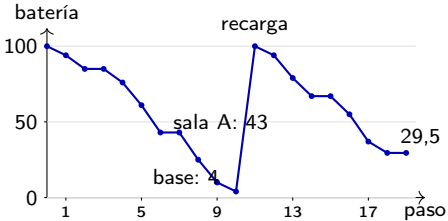
- $4 + 6 + 6 + 10 + 12 = 38$ m, carga para 100
- Paso E, prueba trivial: los números no han roto nada

```
$ kairos check courier-battery-domain
    .pddl courier-battery-p01.pddl
valid, plannable
  typing: full
  numeric-fluents: full
$ kairos courier-battery-domain.pddl
    courier-battery-p01.pddl
; cost = 7
(move r1 dock lobby)
(move r1 lobby pharmacy)
(pick r1 meds1 pharmacy)
...
(drop r1 meds1 ward-a)
```

Dos entregas: el planificador recarga

```
$ kairos courier-battery-domain.pddl
     courier-battery-p02.pddl
; cost = 19
(move r1 dock lobby)
...
(drop r1 meds1 ward-a)
(move r1 ward-a corridor)
(move r1 corridor lobby)
(move r1 lobby dock)
(charge r1 dock)
(move r1 dock lobby)
...
(drop r1 sample1 icu)
```

Consumo 1,5, dos entregas.



- Nadie la pidió: sin ella, un movimiento posterior no es aplicable
- Sala A con 43; la segunda entrega necesita 67,5

Seguir los números

`--dump=plan-trace` escribe el estado tras cada paso (solo lo que puede cambiar):

```
$ kairos courier-battery-domain.pddl courier-battery-p02.pddl \
    --no-anytime --dump=plan-trace --dump-dir trace > plan.txt
$ grep -A5 "step 1[01]:" trace/plan-trace.txt
; step 10: (move r1 lobby dock)
  ...
  (= (battery r1) 4.000000)
; step 11: (charge r1 dock)
  ...
  (= (battery r1) 100.000000)
```

Ensayar el modelo: la batería de pruebas

Paso E: la batería de pruebas de la carga

Trivial	p01: una entrega, carga de sobra. Plan de 7 acciones, sin recarga.
Típica	p02: dos entregas. Plan de 19 acciones, una visita a la base donde la traza dice que debe estar.
Sin plan	p03: robot en la UCI con 20 unidades, artículo para el laboratorio a 25 m. No debe haber plan.
Trampa	p04: objetivo, batería por encima de su capacidad, `(> (battery r1) 100)`. No debe haber plan.

El validador también hace la aritmética. p02 sin su recarga:

```
$ kairos validate courier-battery-domain.pddl \
    courier-battery-p02.pddl courier-battery-p02-nocharge.plan
courier-battery-p02-nocharge.plan:12: error: step 11 (move r1 dock lobby) at
    10.000: precondition false: (>= (battery ?r) (* (distance ?from ?to) (
    consumption ?r))): 4 >= 6 is false [validate.precondition_failed]
 12 | (move r1 dock lobby)
plan invalid
```

Los dos extremos de una magnitud

```
$ kairos courier-battery-domain.pddl courier-battery-p03.pddl
note: no plan: every state reachable from the initial one was searched (22
    states) and none satisfies the goal. [orchestrator.why]
kairos: unsolvable (exhausted)
```

Un primer borrador: el cargador sumaba una cantidad fija

```
:precondition (and (robot-at ?r ?l) (has-charger ?l))
:effect (and (increase (battery ?r) 50)))
```

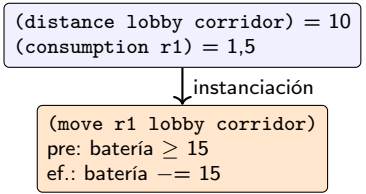
```
$ kairos courier-battery-naive-domain.pddl \
    courier-battery-naive-p04.pddl
; cost = 1
(charge r1 dock)
```

- Prueba de la trampa: 110 unidades en una batería de 100
- Con `assign`: sin plan
- **Los números no tienen cotas propias**

Números que nunca cambian

```
$ kairos courier-battery-domain.pddl courier-battery-p02.pddl -v
...
[   0.001s] [ground] grounded in 0.1 ms: 27 atoms, 1 numeric fluents, 47
    actions, 0 durative actions, 0 axioms (97 Datalog facts, 9 rules, 0
    instances pruned)
```

- 17 números en `:init`, **1 fluente numérico**
- Las funciones estáticas desaparecen dentro de las acciones instanciadas, como los hechos estáticos
- Los datos deben ser datos: un solo `move`, el mapa en el problema



Una capacidad de carga

```
(:functions
  (load ?r - robot) - number       ; kg it carries now
  (capacity ?r - robot) - number   ; kg it can carry
  (weight ?i - item) - number      ; kg of one item
(:action pick
  :parameters (?r - robot ?i - item ?l - location)
  :precondition (and
    (robot-at ?r ?l) (item-at ?i ?l)
    ; the item must still fit
    (<= (+ (load ?r) (weight ?i)) (capacity ?r)))
  :effect (and
    (not (item-at ?i ?l)) (carrying ?r ?i)
    (increase (load ?r) (weight ?i))))

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

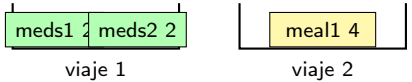
- `free` desaparece: su papel lo asume una comparación
- `weight`, `capacity`: estáticas; solo cambia `load`
- `(load ?r)` **resume** los pesos que lleva (patrón del capítulo 7)

La obligación de un resumen

Toda acción que cambia lo resumido actualiza el resumen. Un `increase` sin el `decrease`: una bandeja que se llena para siempre.

Tres artículos, 8 kg, un robot para 5

```
    (robot-at r1 lobby)
    (= (load r1) 0) (= (capacity r1) 5)
    (item-at meds1 lobby) (= (weight meds1) 2)
    (item-at meds2 lobby) (= (weight meds2) 2)
    (item-at meal1 lobby) (= (weight meal1) 4))
  (:goal (and (item-at meds1 ward-a) (item-at meds2 ward-a)
              (item-at meal1 ward-a))))
```



- La única agrupación con la que bastan dos viajes (con `free`: tres)
- Comida de 6 kg (p02): irresoluble, 1296 estados

```
$ kairos courier-load-domain.pddl
     courier-load-p01.pddl
; cost = 12
(pick r1 meds2 lobby)
(pick r1 meds1 lobby)
...
(drop r1 meds2 ward-a)
(drop r1 meds1 ward-a)
...
(pick r1 meal1 lobby)
...
(drop r1 meal1 ward-a)
```

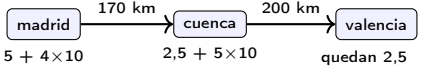
Un camión y su combustible

```
(:action drive
  :parameters (?t - truck ?from ?to - city)
  :precondition (and
    (truck-at ?t ?from) (road ?from ?to)
    (>= (fuel ?t) (/ (distance ?from ?to) (km-per-liter ?t))))
  :effect (and
    (not (truck-at ?t ?from)) (truck-at ?t ?to)
    (decrease (fuel ?t)
              (/ (distance ?from ?to) (km-per-liter ?t)))))

(:action refuel
  :parameters (?t - truck ?c - city)
  :precondition (and
    (truck-at ?t ?c) (has-station ?c)
    (<= (+ (fuel ?t) 10) (tank ?t)))   ; 10 more must fit
  :effect (increase (fuel ?t) 10)))
```

- Consumo: kilómetros entre kilómetros por litro
- Una capacidad mantenida por una precondición: tienen que caber 10 más

```
$ kairos truck-domain.pddl
    truck-p01.pddl
; cost = 11
(refuel t1 madrid)
...
(drive t1 madrid cuenca)
(refuel t1 cuenca)
...
(drive t1 cuenca valencia)
```



Las jarras de agua

```
; everything in ?from fits in ?to
(:action pour-all
  :parameters (?from ?to - jug)
  :precondition (and
    (not (= ?from ?to))
    (> (amount ?from) 0)
    (<= (+ (amount ?from) (amount ?to)) (volume ?to)))
  :effect (and
    (increase (amount ?to) (amount ?from))
    (assign (amount ?from) 0)))

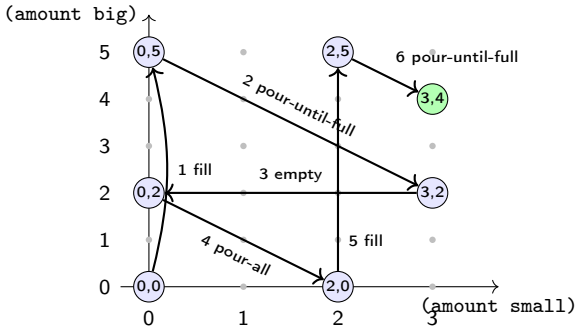
; ?to gets full and something stays in ?from
(:action pour-until-full
  :parameters (?from ?to - jug)
  :precondition (and
    (not (= ?from ?to))
    (< (amount ?to) (volume ?to))
    (> (+ (amount ?from) (amount ?to)) (volume ?to)))
  :effect (and
    (decrease (amount ?from) (- (volume ?to) (amount ?to)))
    (assign (amount ?to) (volume ?to)))))
```

- Jarras de 3 y 5 litros, sin marcas; medir exactamente 4
- Verter tiene dos resultados: **dos acciones**
- `pour-all`: el primer efecto lee `(amount ?from)`, el segundo lo pone a 0
- Los dos leen la cantidad antigua: intercambiar las líneas no cambia nada
- Objetivo: una condición numérica, `(= (amount big) 4)`

Un plan como camino entre pares de números

```
$ kairos jugs-domain.pddl
    jugs-p01.pddl
; cost = 6
(fill big)
(pour-until-full big small)
(empty small)
(pour-all big small)
(fill big)
(pour-until-full big small)
```

Jarras de 4 y 6, objetivo 3 (p02): toda cantidad es par; 20 estados, irresoluble.



El estado no tiene hechos: es un par (pequeña, grande).

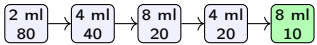
Escalar: una dilución

```
(:functions
  (concentration ?t - tube)    ; mg per ml
  (volume ?t - tube)           ; ml of liquid in the tube
  (size ?t - tube))            ; ml the tube holds

(:action dilute
  :parameters (?t - tube)
  :precondition (<= (* 2 (volume ?t)) (size ?t))
  :effect (and (scale-up (volume ?t) 2)
               (scale-down (concentration ?t) 2)))
(:action discard-half
  :parameters (?t - tube)
  :precondition (>= (volume ?t) 2)
  :effect (scale-down (volume ?t) 2)))
```

- `(scale-up f e)` multiplica, `(scale-down f e)` divide
- Para proporciones: intereses, duplicaciones, descuentos

```
$ kairos dilution-domain.pddl
    dilution-p01.pddl
; cost = 4
(dilute t1)
(dilute t1)
(discard-half t1)
(dilute t1)
```



Objetivo: `(<= (concentration t1) 10)`, `(>= (volume t1) 4)`; tubo de 10 ml. Una desigualdad acepta muchos estados.

Contar: existencias en un almacén

```
(:functions
  (stock ?p - product)     ; units on the shelf
  (batch ?p - product)     ; units that one restocking brings
  (packed ?p - product)    ; units already in the box
  (box-space))             ; units that still fit in the box

(:action pack
  :parameters (?p - product)
  :precondition (and (>= (stock ?p) 1) (>= (box-space) 1))
  :effect (and (decrease (stock ?p) 1)
               (increase (packed ?p) 1)
               (decrease (box-space) 1)))

; the supplier only comes when the shelf is empty
(:action restock
  :parameters (?p - product)
  :precondition (= (stock ?p) 0)
  :effect (increase (stock ?p) (batch ?p))))
```

```
$ kairos warehouse-domain.
    pddl warehouse-p01.
    pddl
; cost = 6
(pack masks)
(pack masks)
(pack gloves)
(restock gloves)
(pack gloves)
(pack gloves)
```

- Se cuentan, no se nombran

Valores que nunca se dieron

Omitido en `:init`: **indefinido**, no cero

Las acciones que lo leen nunca se aplican, en silencio

Borra `(= (battery r1) 100)` del primer problema:

```
$ kairos check courier-battery-domain.pddl \
    courier-battery-undefined.pddl
valid, plannable
  typing: full
  numeric-fluents: full
$ kairos courier-battery-domain.pddl courier-battery-undefined.pddl
note: no plan: every state reachable from the initial one was searched (2
    states) and none satisfies the goal. [orchestrator.why]
kairos: unsolvable (exhausted)
```

Una puerta de un solo sentido por olvidar una línea

Borra `(= (distance ward-a corridor) 12)`:

```
$ kairos courier-battery-domain.pddl
    courier-battery-oneway.pddl
; cost = 22
...
(drop r1 meds1 corridor)
...
(pick r1 meds1 corridor)
(move r1 corridor ward-a)
(drop r1 meds1 ward-a)
```

Medicamentos en el pasillo hasta el final.



sin distancia: nunca se aplica

¿Por qué no cero?

Una distancia cero hace gratis un movimiento. Un valor indefinido bloquea acciones: hacer demasiado poco se nota más que hacer demasiado.

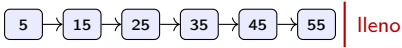
- Escribe los valores junto a sus hechos
- Una prueba de alcanzabilidad en el paso E

Por qué los números dificultan la planificación

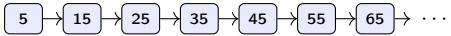
- Hechos: estados finitos, así que se puede demostrar «irresoluble»
- Un fluente puede tomar infinitos valores
- El camión sin el límite del depósito:

```
(:action refuel
  :parameters (?t - truck ?c - city)
  :precondition (and (truck-at ?t ?c) (has-station ?c))
  :effect (increase (fuel ?t) 10)))
```

Con un depósito de 60 litros



Sin límite



La batería, las jarras y el camión con depósito no pueden crecer: una búsqueda que recuerda los estados llega a un final.

Acotado o sin fin

Una carretera de 40 litros, 5 en el depósito, objetivo: exactamente 12. Con el depósito de 60:

```
$ kairos truck-domain.pddl truck-p03.pddl
note: no plan: every state reachable from the initial one was searched (16
    states) and none satisfies the goal. [orchestrator.why]
kairos: unsolvable (exhausted)
```

Sin el límite, siempre un estado más; nada demuestra que más tiempo no servirá:

```
$ kairos truck-endless-domain.pddl truck-endless-p03.pddl \
    --timeout 5
note: no plan within the time limit of 5 s (1487655 states searched). More
    time may find one: --timeout (SolveOptions::timeout). [orchestrator.why]
kairos: timeout (timeout)
```

No es culpa de un planificador concreto

La planificación numérica es indecidible

Para los fluentes numéricos, Malte Helmert demostró en 2002 que ningún algoritmo puede decir en todos los casos si existe un plan. Dos contadores no acotados, sumar uno, restar uno y una comprobación de cero imitan cualquier programa: preguntar por un plan es preguntar si un programa termina.

- Números **acotados**: terreno clásico, se puede demostrar «irresoluble»
- **No acotados**: un planificador puede encontrar un plan, pero su silencio no demuestra nada
- **Más estados**: un fluente los multiplica por sus valores
- **Guía más débil**: la relajación numérica ignora las reducciones; en ella el mensajero nunca se queda sin batería

Acota toda magnitud que el mundo acota; prefiere pocos valores a muchos.

El método MODEL: magnitudes

Paso O: tres preguntas para cada propiedad en la que hay un número

¿Es una magnitud? ¿La suma, la resta o la compara el modelo? Un número de habitación o una prioridad entre tres es un nombre o un nivel.
¿Solo importa un umbral? Si las acciones solo preguntan «¿hay para un movimiento más?», puede bastar un hecho (`charged`, `full`). Conserva el número si acciones distintas consumen cantidades distintas, o si un objetivo o una métrica hablan de él.
¿Está acotada? El valor más pequeño y el más grande que permite el mundo, y qué precondición o `assign` mantiene el fluente entre ellos.

- Paso D: ¿necesita o cambia cada acción cada magnitud?; ¿se actualiza todo resumen?
- Paso E: dos pruebas por magnitud: más de lo que hay, y un objetivo más allá de una cota

La tabla de estado gana una clase de valor

Propiedad	De qué	Clase	Codificación y cotas
Carga	un robot	dinámica	`(battery ?r)`, de 0 a la capacidad; la mantienen la precondición de `move` y el `assign` de `charge`
Capacidad de batería	un robot	estática	`(battery-capacity ?r)`
Consumo	un robot	estático	`(consumption ?r)`, por metro
Longitud	dos habitaciones	estática	`(distance ?from ?to)`, en los dos sentidos
Carga transportada	un robot	dinámica	`(load ?r)`, de 0 a `(capacity ?r)`; la mantiene `pick`
Peso	un artículo	estático	`(weight ?i)`

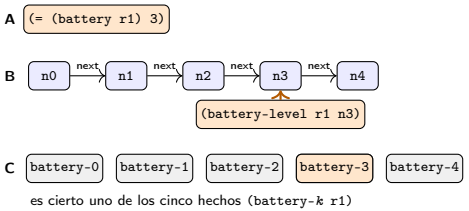
Cada magnitud dinámica indica sus cotas y las acciones que las mantienen.

Alternativas: una batería contada en movimientos

Cada movimiento gasta una unidad, una batería llena tiene cuatro, de la farmacia a la sala A hay cinco movimientos.

- **A, un número**: `(>= (battery ?r) 1)`, `(decrease (battery ?r) 1)`
- **B, una cadena de niveles** (cadena de sucesores, capítulo 7): `move` toma `?before ?after - level`
- **C, un predicado por nivel**: `battery-4` ... `battery-0`, con pasos por `when`

El mismo plan de 10 acciones en las tres (B: `(move r1 dock lobby n4 n3)`).



es cierto uno de los cinco hechos `(battery-k r1)`

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

Tres codificaciones comparadas

	A: número	B: cadena	C: indicadores
Fidelidad	Cualquier cantidad: metros por consumo, fracciones, robots distintos	Solo pasos enteros; 15 unidades: 15 niveles, o una acción por cantidad	Como B, y los niveles fijados en el dominio
Tamaño	31 acciones instanciadas con toda capacidad; 1 fluente numérico	76 acciones instanciadas con 5 niveles, 1516 con 101	31 acciones instanciadas, un efecto condicional por nivel
Guía	Finito solo si se acota; la relajación numérica no ve que se agota	Finito; una variable de estado, fácil para heurísticas clásicas	Como B
Claridad	Se lee como la historia	Dos parámetros más por acción; `next` en cada problema	Largo; una capacidad nueva obliga a editar el dominio
Portabilidad	Necesita `:numeric-fluents`: no en los clásicos	STRIPS y tipos (más una igualdad negada en `charge`)	Efectos condicionales y precondiciones negativas

Veredicto. Un número si las cantidades son datos (distancias, pesos, fracciones, rangos amplios). Una cadena para un recuento pequeño que cambia de uno en uno, sobre todo con un clásico. Indicadores para dos o tres niveles con nombre: un umbral, no un recuento.

Errores comunes (1)

El consumo sin la precondición

Demasiado bueno: p03 (20 unidades, 25 m) tiene plan.

```
$ kairos courier-battery-negative-domain.
    pddl \
    courier-battery-negative-p03.pddl
; cost = 5
...
```

Batería 15, 3, −5. Corrección: una precondición, la misma expresión.

Un valor inicial olvidado

Ningún error: las acciones que lo leen nunca se aplican. Sin plan, o uno extraño.

- Da un valor a todo fluente que se lea
- Siete conexiones, en los dos sentidos: catorce distancias

Errores comunes (2)

Sintaxis de hechos para números

Un valor como un argumento más:

```
(battery r1 100)
```

`kairos check`: *expected term, found number '100'*. Un efecto con =: *expected effect*. Usa `(assign f e)`.

Igualdad con fracciones

Un gotero añade 0,1 ml; objetivo `(= (dose) 0.3)`.

```
$ kairos drip-domain.pddl drip-p01.pddl
note: no plan: every state reachable from
    the initial one was searched (8 states)
     and none satisfies the goal. [
    orchestrator.why]
...
```

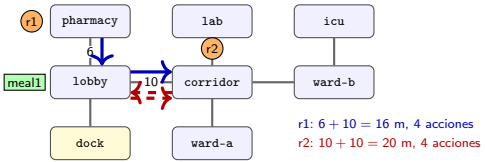
$3 \times 0{,}1 = 0{,}30000000000000004$. Un intervalo, o unidades enteras.

1 Fluentes numéricos

2 Calidad del plan: métricas y costes de acción

3 Resumen

Cuando no basta con un plan cualquiera




```
$ kairos courier-cost-domain.pddl
    courier-cost-p01-nometric.pddl
; cost = 4
(move r2 corridor lobby)
(pick r2 meal1 lobby)
(move r2 lobby corridor)
(drop r2 meal1 corridor)
```

- Dos planes de cuatro acciones: el mismo plan para todos los modelos vistos
- Los planificadores satisfactorios se detienen en el primer plan
- Sin métrica, coste = **longitud del plan**
- Correcto solo si todas las acciones valen lo mismo

Costes de acción

```
(:requirements :strips :typing :action-costs)
(:functions
  (total-cost) - number                    ; meters so far
  (distance ?from ?to - location) - number) ; static data

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

```
  (= (total-cost) 0)
  (robot-at r1 pharmacy) (free r1)
  (robot-at r2 corridor) (free r2)
  (item-at meal1 lobby))
(:goal (item-at meal1 corridor))
(:metric minimize (total-cost)))
```

- Dominio: una función (`total-cost`), que incrementa toda acción que cuesta
- El coste de `move` se lee de la estática `distance`
- `pick`, `drop`: sin `increase`, coste cero
- Problema: empieza en cero, termina con la **métrica**
- La métrica es del problema: cada problema dice si le importa

Con métrica, la respuesta cambia

```
$ kairos courier-cost-domain.pddl
    courier-cost-p01.pddl
; cost = 20
(move r2 corridor lobby)
(pick r2 meal1 lobby)
(move r2 lobby corridor)
(drop r2 meal1 corridor)
; better plan 2
; cost = 16
(move r1 pharmacy lobby)
(pick r1 meal1 lobby)
(move r1 lobby corridor)
(drop r1 meal1 corridor)
```

- El coste impreso es ahora el valor de la métrica
- Kairos siguió buscando: `; better plan 2`
- El último plan impreso es el mejor

```
$ kairos validate
    courier-cost-domain.pddl
    courier-cost-p01.pddl \
    courier-cost-p01-r2.plan
plan valid, makespan 4, metric 20
```

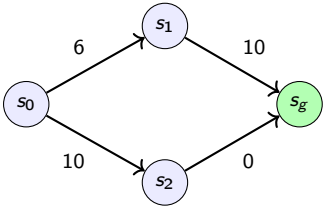
Válido: funciona, no que sea bueno.

El requisito :action-costs

Un fragmento de la planificación numérica deliberadamente pequeño (PDDL 3.1, competición de 2008):

- solo cambia (total-cost), una función sin parámetros
- solo con (increase (total-cost) e), e un número no negativo o una función estática de los parámetros
- ninguna precondición ni objetivo lee un número
- la métrica es (:metric minimize (total-cost))

Kairos avisa de que :numeric-fluents no está declarado, y planifica con normalidad.



El espacio de estados clásico, finito, mismos estados y planes: los costes solo pesan las transiciones. Un planificador clásico solo necesita una búsqueda que sume pesos.

Costes a partir de datos: una cocina con precios

```
(:types
  ingredient dish - object
  egg fat sweetener - ingredient
  omelette cake - dish)
(:action add-egg
  :parameters (?e - egg ?d - dish)
  :precondition (in-stock ?e)
  :effect (and (not (in-stock ?e)) (with-egg ?d)
               (increase (total-cost) (price ?e))))
(:action fry
  :parameters (?d - omelette)
  :precondition (and (with-egg ?d) (with-fat ?d))
  :effect (and (ready ?d) (increase (total-cost) 1)))

(:action bake
  :parameters (?d - cake)
  :precondition (and (with-egg ?d) (with-fat ?d)
                     (with-sweetener ?d))
  :effect (and (ready ?d) (increase (total-cost) 2))))
```

```
(:init
  (= (total-cost) 0)
  (in-stock farm-egg) (= (price farm-egg) 3)
  (in-stock egg1) (= (price egg1) 1)
  (in-stock egg2) (= (price egg2) 1)
  (in-stock butter1) (= (price butter1) 4)
  (in-stock oil1) (= (price oil1) 2)
  (in-stock oil2) (= (price oil2) 2)
  (in-stock honey1) (= (price honey1) 5)
  (in-stock sugar1) (= (price sugar1) 1))
(:goal (and (ready om1) (ready cake1)))
(:metric minimize (total-cost)))
```

- Un coste a partir de datos, `(price ?e)`, o una constante, el gas
- Los tipos hacen el trabajo de las recetas: sirve cualquier `fat`

La comida más barata

```
$ kairos kitchen-domain.pddl
    kitchen-p01.pddl
; cost = 18
(add-egg farm-egg cake1)
(add-egg egg1 om1)
(add-fat butter1 cake1)
(add-fat oil1 om1)
(add-sweetener honey1 cake1)
(bake cake1)
(fry om1)
```

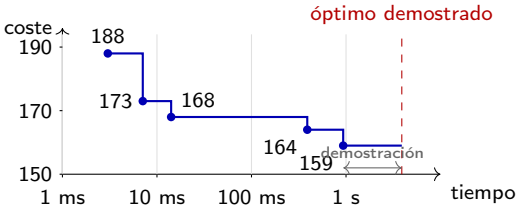
Siete acciones cada uno: la longitud no puede distinguirlos.

```
; better plan 2
; cost = 10
(add-fat oil1 cake1)
(add-fat oil2 om1)
(add-egg egg1 cake1)
(add-egg egg2 om1)
(add-sweetener sugar1 cake1)
(bake cake1)
(fry om1)
```

Acciones de coste cero: gratis para la búsqueda. ¿Y en el mundo?

El planificador en acción: búsqueda anytime

- **Óptimo**: descarta todo plan más barato antes de responder
- **Anytime**: un primer plan rápido, luego otros estrictamente mejores
- Cuatro entregas, dos robots (p02)




```
$ kairos courier-cost-domain.pddl courier-cost-p02.pddl -v 2>&1 \
    >/dev/null | grep -E "plan [0-9]|^kairos: solved"
[   0.003s] [search] plan 0 with 29 actions, cost 188
...
[   0.944s] [search] plan 4 with 28 actions, cost 159
kairos: solved (proven optimal), cost 159
```

Qué significa proven optimal

```
(move r1 dock lobby)
(move r1 lobby pharmacy)
(pick r1 meds1 pharmacy)
...
(move r1 corridor ward-b)
(move r1 ward-b icu)
(drop r1 meds2 icu)
```

El mejor plan, 159 m (`courier-cost-p02.plan`).

- No existe un plan de 158 metros **en el modelo, para la métrica tal como está escrita**
- Todas las acciones son de `r1`; `r2` no sale nunca de la UCI
- Correcto para los metros: cada metro de `r2` se suma
- Para terminar pronto, el modelo necesita el tiempo (Parte IV); para repartir el desgaste, otra medida
- Un plan óptimo responde a la pregunta que se escribió

Lo bastante bueno

```
$ kairos courier-cost-domain.pddl courier-cost-p02.pddl \
    --no-anytime -v 2>&1 | grep -E "cost =|^kairos: solved"
; cost = 188
kairos: solved (no improvement), cost 188
```

Arriba, solo el primer plan. `--target-cost C` se detiene en coste `C` o menos (14 ms):

```
$ kairos courier-cost-domain.pddl courier-cost-p02.pddl \
    --target-cost 170 -v 2>&1 | grep -E "cost =|^kairos: solved"
; cost = 188
; cost = 173
; cost = 168
kairos: solved (target reached), cost 168
```

Un presupuesto, y qué significan las palabras de estado

`--timeout S`: un presupuesto en segundos; con medio segundo:

```
$ kairos courier-cost-domain.pddl courier-cost-p02.pddl \
    --timeout 0.5 -v 2>&1 | grep -E "cost =|^kairos: solved"
; cost = 188
...
kairos: solved (timeout), cost 164
```

- Siempre `solved`; solo `proven optimal` es una garantía
- Los planificadores **óptimos** prometen coste mínimo; los **satisfactorios**, un plan

Estado	Significado
`proven optimal`	no hay nada mejor
`target reached`	suficiente según `C`
`timeout`	el mejor a tiempo
`no improvement`	solo el primer plan

Métricas sobre cualquier fluente

```
(:functions
  (distance ?from ?to - location)  ; meters, static
  (consumption ?r - robot)         ; energy per meter, static
  (meters)                         ; covered by all the robots
  (energy-used))                   ; used by all the robots

(:action move
  :parameters (?r - robot ?from ?to - location)
  :precondition (and (robot-at ?r ?from) (connected ?from ?to))
  :effect (and
    (not (robot-at ?r ?from)) (robot-at ?r ?to)
    (increase (meters) (distance ?from ?to))
    (increase (energy-used)
              (* (distance ?from ?to) (consumption ?r)))))
```

```
(:metric minimize (meters)))
(:metric minimize (energy-used)))
```

- La métrica puede ser cualquier expresión sobre fluentes numéricos
- Cada medida en su propio **contador**; cada problema elige
- Incumple `:action-costs` dos veces: cambian dos fluentes, uno con un producto
- Ninguna precondición los lee
- `r1` ligero (1) en la base de carga; `r2` pesado (3) en la sala B

Metros o energía

```
$ kairos courier-quality-domain.pddl
     courier-quality-meters.pddl
; cost = 68
...
(drop r1 meds1 ward-a)
```

- Metros: cada robot se encarga de la entrega que tiene cerca (30 m, 38 m)
- Energía: 128, 92, 88: planes más largos, más baratos

```
$ kairos courier-quality-domain.pddl courier-quality-energy.pddl \
    -v 2>&1 >/dev/null | grep -E "plan [0-9]|^kairos: solved"
[   0.002s] [search] plan 0 with 13 actions, cost 128
...
kairos: solved (proven optimal), cost 88
```

El optimizador encuentra los resquicios

```
(move r1 lobby corridor)
(drop r1 meds1 corridor)
(move r1 corridor ward-b)
...
(drop r1 sample1 lab)
(move r1 lab corridor)
(pick r1 meds1 corridor)
(move r1 corridor ward-a)
(drop r1 meds1 ward-a)
```

- Esperábamos 92; el planificador ahorró 4
- Deja los medicamentos en el suelo del pasillo y entra en la sala A una sola vez
- Válido en el modelo. ¿Pueden esperar sin vigilancia? Pregunta para la ficha
- Si no: una precondición, o las restricciones del capítulo 15

Lee el mejor plan con desconfianza

Un planificador que optimiza explora los rincones del modelo y halla lo que este olvidó prohibir.

Maximizar: (:metric maximize (battery r1))

```
$ kairos courier-battery-domain.pddl courier-battery-max.pddl
; cost = 62
...
note: the search minimizes the number of actions; the metric of the problem is
     reported for each plan but does not guide the search yet [orchestrator.
    metric]
```

```
$ kairos validate courier-battery-domain.pddl \
    courier-battery-max.pddl courier-battery-max.
    plan
plan valid, makespan 11, metric 100
```

Consejo

Usa **sumas de cosas que solo se acumulan**.

Varios deseos en un solo número

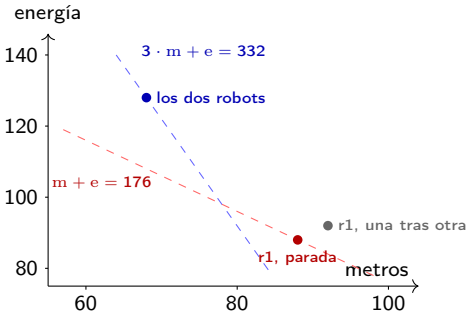
Suma ponderada: un metro pesa tres unidades de energía (`mix1`):

```
(:metric minimize (+ (* 3 (meters)) (energy-used)))
```

Mejor: los dos robots, $3 \times 68 + 128 = 332$. Con `(+ (meters) (energy-used))`:

```
$ kairos courier-quality-domain.pddl
      courier-quality-mix2.pddl \
    | grep cost
; cost = 196
; cost = 176
```

Gana `r1` con su parada: $88 + 88 = 176$.

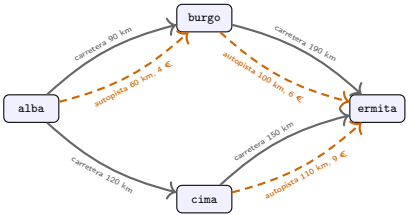


La razón entre los pesos elige el punto. Con el dinero como unidad, los pesos son precios.

Un viaje por carretera: peajes y combustible

```
(:action drive-highway
  :parameters (?c - car ?from ?to - town)
  :precondition (and (car-at ?c ?from) (highway ?from ?to))
  :effect (and
    (not (car-at ?c ?from)) (car-at ?c ?to)
    (increase (tolls-paid) (toll ?from ?to))
    (increase (fuel-used)
              (/ (highway-km ?from ?to) (km-per-liter ?c))))))
```

```
; money: the tolls plus the fuel at 2 euros per liter
(:metric minimize (+ (tolls-paid) (* 2 (fuel-used))))
```




```
$ kairos road-trip-domain.pddl
    road-trip-p01.pddl
; cost = 27
(drive-road car1 alba cima)
(drive-road car1 cima ermita)
; better plan 2
; cost = 25
(drive-road car1 alba burgo)
(drive-highway car1 burgo
    ermita)
```

Túnel: ahorra 9 € de combustible por 6 de peaje. Una furgoneta (p02): las dos autopistas.

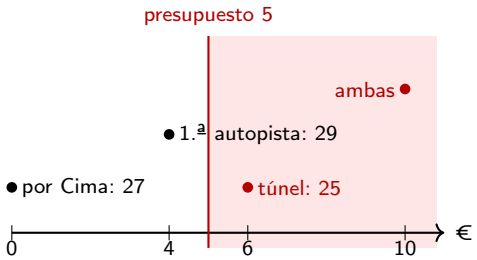
Un límite o una métrica

Un **límite estricto** divide los planes; una métrica los ordena.

```
:precondition (and
  (car-at ?c ?from) (highway ?from ?to)
  (<= (+ (tolls-paid) (toll ?from ?to)) (toll-budget)))
```

```
$ kairos road-trip-budget-domain.
    pddl road-trip-budget-p01.pddl
; cost = 27
(drive-road car1 alba cima)
(drive-road car1 cima ermita)
```

Presupuesto 5 €: el plan de 25 necesita un peaje de 6. Un límite solo quita planes.



- Las dos autopistas: inválido, `10 <= 5 is false`
- Un fallo: un **límite**. Solo peor: una **métrica**

El método MODEL: qué hace mejor a un plan

Paso M: dos preguntas sobre la calidad

¿Qué hace que un plan sea mejor que otro? Nombra la medida y su unidad: metros, euros, energía, manipulaciones. Si es «nada», dilo: sin métrica.
¿Es un solo número? Una unidad común hace de los pesos precios; si no, alguien decide cuánto de una vale una unidad de otra. Escribe los pesos con su motivo. Y para cada deseo: ¿una preferencia (métrica) o una condición de éxito (límite)?

- Ficha: *Éxito* gana «cuantos menos metros, mejor»; *Supuestos*: coger es gratis, un robot quieto no cuesta nada
- Paso O: cada medida, un contador dinámico desde cero que no lee ninguna precondición salvo si es límite
- Paso D: una columna de coste, ceros justificados. Paso E: dos planes de coste conocido; un coste olvidado

Alternativas: tres maneras de decir «mejor»

	Longitud del plan	**Costes de acción**	**Métrica sobre fluentes**
Fidelidad	Correcta solo si todas las acciones valen lo mismo. Aquí: 13 acciones, 68 m, 128 de energía	Una suma de costes fijos por acción. Aquí: 68 m, óptimo	Cualquier expresión: energía por robot, sumas ponderadas. Aquí: 88 de energía, o las mezclas
Tamaño	92 acciones instanciadas, ningún fluente	92, un fluente que ninguna condición lee	92, dos fluentes contadores
Guía	Búsqueda clásica	Espacio clásico con pesos; el mejor soportado	Buena mientras los contadores crezcan en cantidades fijas
Claridad	Nada que escribir, y nada dicho	Una línea por acción, una en el problema	Contadores que mantener; la métrica documenta los pesos
Portabilidad	Todos los planificadores	`:action-costs`: también los clásicos	`:numeric-fluents`: solo planificadores numéricos

Veredicto. Nada cuando nada importa. Costes de acción cuando la calidad es una suma de costes fijos de las acciones: casi siempre. Métricas sobre fluentes para costes que dependen del estado, medidas ponderadas por problema o un contador que es también un límite.

Alternativas: un límite o una métrica

	Solo métrica	Solo límite	Límite y métrica
Fidelidad	No se pierde ningún plan; puede gastar más de lo que tiene el conductor. Mejor: 25 €, peaje 6	Nunca por encima del presupuesto; cualquier plan dentro de él, bueno o malo, quizá ninguno	Nunca por encima, y el más barato dentro. Mejor: 27 €
Tamaño	Solo contadores	Una precondición numérica por acción limitada	Ambos
Guía	Los costes guían la búsqueda	Callejones sin salida: un peaje pagado pronto puede bloquear el único camino posterior	Ambos efectos
Claridad	El compromiso está en los pesos	La regla donde se aplica, en la acción	Dos sitios que leer
Portabilidad	Métrica sobre fluentes	Precondiciones numéricas	Ambas

Veredicto. No son rivales: el límite para lo que no debe ocurrir, la métrica para lo que debe ser pequeño. En caso de duda, empieza solo con la métrica y lee el mejor plan; si hace algo inaceptable, faltaba un límite.

Errores comunes (1)

Una acción que olvidó su coste

`add-fat` sin `increase`: gratis. Mejor comida 6, con mantequilla.

```
$ kairos kitchen-free-domain.pddl
    kitchen-free-p01.pddl
...
; cost = 6
...
(add-fat butter1 cake1)
```

Corrección: un coste para cada acción, con los ceros escritos.

La métrica en el fichero equivocado

`(:metric ...)` en el dominio: un error de sintaxis.

```
$ kairos check
    kitchen-wrong-domain.pddl
kitchen-wrong-domain.pddl:22:4:
    error: unknown domain section
    :metric [parse.unknown_section
    ]
invalid
```

Al revés: ningún error.

Errores comunes (2)

Costes negativos y mejora sin fin

Un reembolso, o `maximize` sobre algo que crece para siempre. Kairos deja de optimizar y lo dice (azúcar a -1: primer plan, 18). Un ciclo de coste total negativo: no existe el mejor plan. Corrección: costes no negativos; acota lo que se maximiza.

El valor inicial del coste

Sin `(= (total-cost) 0)` el fluente está indefinido. Kairos es indulgente; un planificador que siga la regla no halla aplicable ninguna acción con coste. Inicializa cada contador.

Fiarse del primer plan, o de la palabra equivocada

«El modelo envía al robot lejano», cuando un plan posterior no lo hace: en la ronda, 188 el primero, 159 el mejor. Lee el último plan y la línea de estado con `-v`. La salida anytime guardada en un fichero no es un plan válido: usa `--output` y valida el último fichero.

1 Fluentes numéricos

2 Calidad del plan: métricas y costes de acción

3 Resumen

Qué llevarse

Fluentes numéricos

- Las funciones ponen números en el estado; las expresiones leen el estado anterior a la acción
- Las estáticas desaparecen en las acciones instanciadas; las dinámicas son estado
- Indefinido no es cero: acciones que faltan en silencio
- Números no acotados: no se puede demostrar «sin plan»
- Paso O: ¿magnitud? ¿umbral? ¿acotada?

Calidad del plan

- Sin métrica: el coste es la longitud del plan
- Costes de acción: portables, espacio de estados clásico
- Métricas sobre fluentes: contadores y sumas ponderadas
- Anytime: lee el último plan y el estado
- Paso M: ¿qué hace mejor a un plan? ¿un solo número? ¿límite o métrica?

Ejercicios para practicar

Capítulo 8

- 8.3.1: predice el plan de p02 con 50 unidades
- 8.3.2: una reserva de 10 unidades
- 8.5.1: reparte el vino con jarras de 8, 5 y 3
- 8.6.1: borra el peso de la comida
- 8.7.2: el camión sin fin con 1, 2 y 4 segundos
- 8.8.3: el robot de riego, pasos M, O, D, E

Capítulo 9

- 9.2.1: una enfermera en cada recogida y entrega
- 9.4.2: haz que un plan óptimo use los dos robots
- 9.5.1: energía más 5 por manipulación
- 9.6.2: el precio del combustible al que compensan las dos autopistas
- 9.7.1: el combustible del viaje como límite estricto
- 9.8.2: el dron, batería y riesgo