Preferences and constraints: PDDL 3

Planning and PDDL, Part V (Chapters 14–16)

Francisco Martín Rico

Intelligent Robotics Lab, URJC



Outline

1. Soft goals and preferences
2. Constraints on the whole plan
3. Object fluents
4. Summary

1 Soft goals and preferences

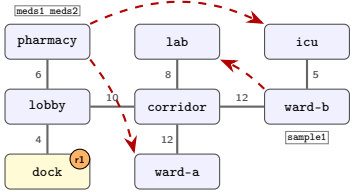
2 Constraints on the whole plan

3 Object fluents

4 Summary

More wishes than battery

- Typed courier + battery: a move costs its meters
- Battery of 100, no `charge` today
- Morning list: `meds1` to ward A, `meds2` to the ICU, `sample1` to the lab



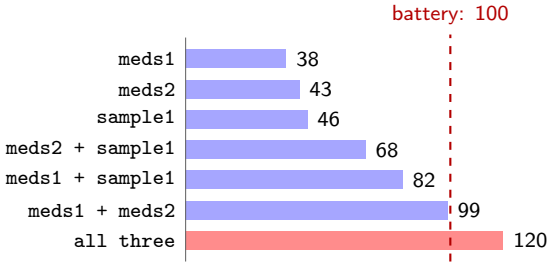
The three deliveries as one goal:

```
(:goal (and (item-at meds1 ward-a)
            (item-at meds2 icu)
            (item-at sample1 lab)))
```

```
$ kairos courier-battery-domain.pddl
    courier-battery-p01.pddl
...
kairos: unsolvable (exhausted)
```

Correct and useless: the robot stays at the dock all morning

Over-subscription



Battery needed by each set of deliveries, from the dock

- Any two deliveries fit; the three need 120
- **Over-subscribed**: more goals than the resources allow
- The normal case: many requests, one robot
- Deleting goals by hand: the modeler does the planner's work

The question of the hospital

Which deliveries should be made with the battery there is?

A preference in the goal

`courier-battery-p02.pddl`: the same list as wishes (requirement `:preferences`)

```
(:goal (and (preference meds-a (item-at meds1 ward-a))
            (preference meds-icu (item-at meds2 icu))
            (preference sample-lab (item-at sample1 lab))))
(:metric minimize (+ (is-violated meds-a)
                     (is-violated meds-icu)
                     (is-violated sample-lab))))
```

- **Preference**: a named condition, `(preference name C)`
- In a goal: a **soft goal**; the others are **hard goals**
- Violating breaks no rule: it only costs what the metric says
- `(is-violated meds-a)`: 0 if satisfied, 1 if not
- The metric: deliveries *not* made, minimized

The anytime search at work

```
$ kairos courier-battery-domain.pddl
    courier-battery-p02.pddl
; cost = 3
; better plan 2
; cost = 2
(move r1 dock lobby)
...
(drop r1 meds1 ward-a)
; better plan 3
; cost = 1
(move r1 dock lobby)
...
(drop r1 meds1 ward-a)
...
(drop r1 sample1 lab)
```

- Plan 1 has cost 3 and *no actions*: with only soft goals, doing nothing is a plan, the worst one
- Plan 2 delivers `meds1`: cost 2
- Plan 3 adds `sample1`: cost 1, and the search stops by itself
- The cost printed is the value of the metric

Valid is not the same as good

A hand-written plan that delivers only `meds1`:

```
$ kairos validate courier-battery-domain.pddl courier-battery-p02.pddl
    courier-battery-p03-lazy.plan
plan valid, makespan 7, metric 2
```

- Validity: preconditions and hard goals. Everything about wishes lives in the metric

Step M: required or wished

The *Success* line of the model card splits in two: **Required** (a plan must achieve it to be acceptable at all) and **Wished** (achieved if possible, each wish with its worth). Test: imagine the plan that leaves the condition out. *Unusable*: required. Only *worse*: a wish.

What each wish is worth

The head nurse: the ICU comes first. p03 only changes the metric:

```
(:metric minimize (+ (* 3 (is-violated meds-a))
                     (* 5 (is-violated meds-icu))
                     (* 1 (is-violated sample-lab)))))
```

- A **weight**: the price of violating that preference
- Kairos prints costs 9, 4, 3 and 1: four rows of the table
- Last plan: both medicines, arrives at the ICU with 1 unit left

Deliveries made	Battery	(1, 1, 1)	(3, 5, 1)
none	0	3	9
meds1	38	2	6
meds2	43	2	4
sample1	46	2	8
meds2, sample1	68	1	3
meds1, sample1	82	1	5
meds1, meds2	99	1	**1**

Equal weights (p02): three sets tie at 1. With weights (p03): only one is best

A wish about the robot, and a goal that must hold

p04: end at the dock, weight 4

```
(:goal (and (preference meds-a (item-at meds1 ward-a))
            (preference meds-icu (item-at meds2 icu))
            (preference sample-lab (item-at sample1 lab))
            (preference parked (robot-at r1 dock))))
(:metric minimize (+ (* 3 (is-violated meds-a))
                     (* 5 (is-violated meds-icu))
                     (* 1 (is-violated sample-lab))
                     (* 4 (is-violated parked)))))
```

p05: the lab must have its sample

```
(:goal (and (item-at sample1 lab)
            (preference meds-a (item-at meds1 ward-a))
            (preference meds-icu (item-at meds2 icu))))
(:metric minimize (+ (* 3 (is-violated meds-a))
                     (* 5 (is-violated meds-icu)))))
```

- Best, cost 3: ICU and lab, back at the dock with 10 units
- Both medicines, stranded in the ICU: $1 + 4 = 5$
- The empty plan costs 9, not 13
- Hard and soft goals mix in one and
- No empty plan; costs 8, 5 and 3
- The lazy plan: `plan invalid`

Choosing weights

- Only the relation between weights matters: (3, 5, 1) and (30, 50, 10) are the same problem
- Hard goals stay outside the bargain

Three patterns

- **Equal weights**: maximize the number of wishes granted
- **Worth**: each weight is what the wish is worth in a common unit (euros, minutes, meters); two small wishes can beat a large one
- **Strict priority**: a higher class beats *any number* of lower ones. Each class weighs more than the sum of all weights below it: $5 > 3 + 1$

Wishes against effort

`courier-cost`: a move adds its meters to `total-cost`; `pick`, `drop` add 1

```
(:goal (and (preference meds-a (item-at meds1 ward-a))
            (preference meal-icu (item-at meal1 icu))))
(:metric minimize (+ (total-cost)
                     (* 60 (is-violated meds-a))
                     (* 20 (is-violated meal-icu)))))
```

```
$ kairos courier-cost-domain.pddl
    courier-cost-p01.pddl
; cost = 80
; better plan 2
; cost = 60
(move r1 dock lobby)
...
(drop r1 meds1 ward-a)
```

- Weights *in meters*: the medicine is worth a trip of up to 60
- Nothing: $60 + 20 = 80$. Medicine: $40 + 20 = 60$
- The meal adds 63 to save 20: a bad deal
- p02, meal worth 100: both, cost 103

Net benefit

Worth achieved against the cost of achieving it: the planner achieves as much as pays

One preference, many instances

`courier-cost-p03.pddl`: two medicines and a meal in the pharmacy

```
(:goal (forall (?m - medicine)
         (preference meds-icu (item-at ?m icu))))
(:metric minimize (+ (total-cost)
                     (* 50 (is-violated meds-icu)))))
```

```
$ kairos courier-cost-domain.pddl
    courier-cost-p03.pddl
; cost = 100
; better plan 2
; cost = 95
(move r1 dock lobby)
...
(drop r1 meds2 icu)
```

- A `forall` *around* a preference: one preference per medicine, all named `meds-icu`
- `(is-violated meds-icu)`: the *number* of medicines not in the ICU
- Empty plan: $2 \times 50 = 100$
- Best: one medicine, $45 + 50 = 95$; the second needs 68 more to save 50
- The meal is not a `medicine`: no preference for it

Where the forall goes

	`forall` outside	`forall` inside
Written as	`(forall (?m) (preference p ...))`	`(preference p (forall (?m) ...))`
Preferences	one per object	one
`(is-violated p)`	number of objects that fail	1 if any object fails
Partial success	counts	does not count

- The same counting whenever a name is used more than once: two separate preferences named `meds` make `(is-violated meds)` 0, 1 or 2
- A name identifies a **class** of wishes with one price, not a single wish

Wishes about the way: preferences in preconditions

`courier-lobby`: do not cross the lobby with a sample

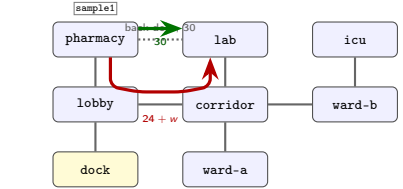
```
(:action move
  :parameters (?r - robot ?from ?to - location)
  :precondition (and (robot-at ?r ?from) (connected ?from ?to)
    (preference clean-lobby
      (or (not (= ?to lobby))
          (not (exists (?s - sample) (carrying ?r ?s))))))
  :effect (and (not (robot-at ?r ?from))
               (robot-at ?r ?to)
               (increase (total-cost) (distance ?from ?to))))
```

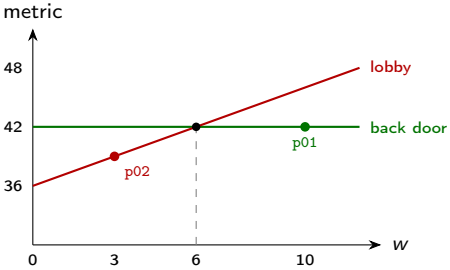
- Applicable anyway; applying it without the preference: one violation
- (`is-violated clean-lobby`): how many *times*
- A soft goal cannot say it: at the end the robot is not in the lobby

Not a precondition

Broken whenever cheaper, and for free if the metric does not mention it. A rule that must never be broken goes without `preference`: hard, and with no back door, Kairos answers `unsolvable`

The price decides the route





- The problems open a back door, 30 m; through the lobby, 24 m
- Sample must (hard goal) reach the lab
- p01, $w = 10$: back door, cost 42 (lobby: $36 + 10 = 46$)
- p02, $w = 3$: through the lobby, $36 + 3 = 39$
- The two routes cross at $w = 6$
- p03, no back door, two samples: cost 106, two violations paid

Two more worlds: a tourist's day

240 minutes, four sights, a bus in 5 minutes between any two places

```
(:goal (and (at hotel)
            (preference see-museum (seen museum))
            (preference see-cathedral (seen cathedral))
            (preference see-market (seen market))
            (preference see-tower (seen tower))
            (preference on-foot (not (used-bus)))))
(:metric minimize (+ (* 5 (is-violated see-museum))
                     (* 3 (is-violated see-cathedral))
                     (* 2 (is-violated see-market))
                     (* 4 (is-violated see-tower))
                     (* 1 (is-violated on-foot)))))
```

- A hard goal, wishes of different worth, and a wish that something did *not* happen
- `on-foot` costs the same for one ride or four: the last state only knows `(used-bus)`

```
$ kairos tourist-domain.pddl
    tourist-p01.pddl
...
; better plan 4
; cost = 3
(walk hotel cathedral)
(visit cathedral)
(walk cathedral museum)
(visit museum)
(walk museum tower)
(visit tower)
(bus tower hotel)
```

- Skips the market (2), one bus ride (1)

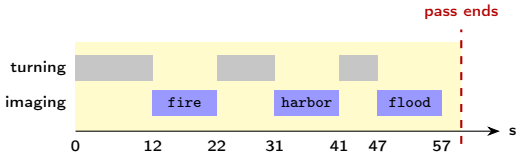
Two more worlds: a satellite with more requests than time

- Durative `slew` and `take-image` (10 s) in a pass of 60 s: `(at 60 (not (in-pass)))`
- Five requests, two urgent

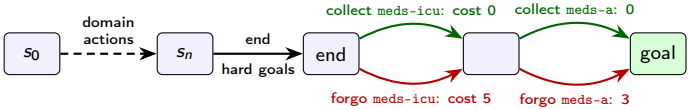
```
(:goal (and
  (forall (?t - urgent)
    (preference urgent-done (imaged ?t)))
  (forall (?t - routine)
    (preference routine-done (imaged ?t)))))
(:metric minimize (+ (* 5 (is-violated urgent-done))
                     (is-violated routine-done))))
```

- Three routine targets: $5 > 3 \times 1$, strict priority

- Kairos: costs 13, 3, 2
- Best plan: both urgent images, and `harbor` on the way from `fire` to `flood`
- Cost and makespan are now different numbers



What the planner does with a preference



- Soft goals compiled into costs: an *end* action, then **collect** (free, if the condition holds) or **forgo** (pays the weight) for each preference
- Kairos compiles goal preferences (also under `forall`) and up to four per action at the top of a precondition (copies of the action); the rest are *counted*: each plan is priced exactly, only improving plans are kept
- The search starts from the plan cheapest to *find*, often the empty one; when a sequential search ends by itself the last plan is `proven optimal`

The MODEL method: wishes and their worth

New questions for step M (Map the situation)

- **Which goals are required and which are wishes?**
- **What is each wish worth?** In which unit; against other wishes or also against effort
- **Is any wish about the way and not about the end?**
- **What is the plan when nothing is granted?** If the empty plan is not acceptable, a hard goal is missing

The model card gains a *Wishes* table: name, condition, weight, reason for the weight.

- Step E: run with no metric; then the **price test**: raise each weight until the plan changes; does it make sense?

Alternatives compared: do not cross the lobby with a sample

	Hard precondition	Preference	Action cost (conditional effect)
Fidelity	Loses solutions: without the back door, no plan	Exact: allowed, at a price	Exact, the same plans and prices
Size	No growth	Copies of `move`, collect and forgo actions	No growth
Guidance	Best: forbidden moves pruned	Good when compiled	Kairos does not use a conditional cost to guide the search
Clarity	A rule, read as a rule	Wish in the domain, price in each problem	Price buried in the domain
Portability	Negative, disjunctive preconditions	`:preferences`: Kairos, OPTIC, POPF, ENHSP	`:conditional-effects` + `:action-costs`

- All three: the same five actions through the back door, cost 42. **Hard**: breaking it makes the plan unusable (safety, physics, law); a huge weight is not a substitute
- **Preference**: acceptable at a price, above all if the price changes per problem
- **Action cost**: the action itself is to avoid, always (a toll)

Common mistakes (1)

Preferences without a metric

p02 without its `:metric`:

```
$ kairos courier-battery-domain.pddl
    courier-battery-nometric.pddl
; cost = 0
```

An empty plan and no warning. *Fix*: every preference appears in the metric. Same for a preference without a name

A name that no preference has

Caught by `kairos check`: run it first

```
$ kairos check courier-battery-domain.
    pddl courier-battery-typo.pddl
courier-battery-typo.pddl:36:24: error:
    no preference named 'sample-lap' [
    semantic.unknown_preference]
...
invalid
```

The same typo in the goal *and* the metric is no error at all

Common mistakes (2)

Maximizing the violations

`maximize` asks for the plan that fails the most: the empty plan, cost 3. *Fix*: `minimize` a sum of violations; a benefit is a constant minus the violations

Weights in the wrong unit

A weight of 3 in `courier-cost` means "worth three meters": the robot never moves for it, and the plan is `proven optimal`. *Fix*: the price test; write the unit next to each weight

Taking the first plan

With soft goals the first plan is often the empty one:

```
$ kairos courier-battery-domain.pddl
    courier-battery-p03.pddl
    --no-anytime
; cost = 9
```

Fix: let the anytime search run and take the last plan, or use `--target-cost`. A short timeout may return a plan far from the best

1 Soft goals and preferences

2 Constraints on the whole plan

3 Object fluents

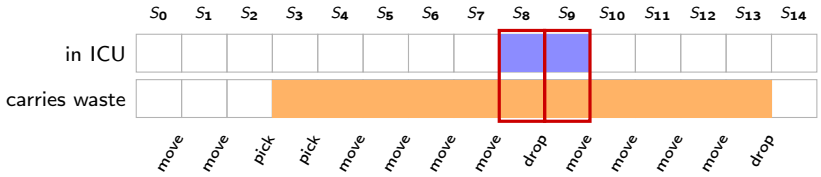
4 Summary

A rule that the plan breaks

```
$ kairos courier-rules-domain.pddl
    courier-rules-p01.pddl
; cost = 14
(move r1 dock lobby)
(move r1 lobby pharmacy)
(pick r1 waste1 pharmacy)
(pick r1 meds1 pharmacy)
...
(move r1 ward-b icu)
(drop r1 meds1 icu)
...
(drop r1 waste1 dock)
```

- `courier-rules`: a basket (any number of items), items `medicine`, `sample`, `waste`; `sanitize` cleans the gripper, `pick` dirties it
- Medicines to the ICU, waste to the dock, end at the dock
- The shortest plan rides into the ICU with waste
- Missing: *no robot is ever in the ICU while it carries waste*
- Not a goal; as preconditions, scattered over every action

The states of a plan



- A plan of n actions goes through $S_0, S_1, \ldots, S_n$: its **trajectory**, *both ends included*
- A goal looks only at S_{14}; the rule looks at every column and fails in S_8, S_9
- **Trajectory constraint**: a condition on the trajectory, in a section `:constraints` (requirement `:constraints`)

always

courier-rules-p02.pddl = p01 + the rule

```
(:goal (and (item-at meds1 icu) (item-at waste1 dock)
            (robot-at r1 dock)))
(:constraints
  (forall (?r - robot ?w - waste)
    (always (not (and (robot-at ?r icu) (carrying ?r ?w)))))))
```

- (always C): C holds in every state
- The forall: one constraint per robot and waste
- Kairos: 16 actions; the waste is dropped at the dock before going up to the ICU
- The domain did not change by a single character
- S_0 counts: an always false in it is unsolvable

The validator checks constraints too:

```
$ kairos validate
     courier-rules-domain.pddl
      courier-rules-p02.pddl
     courier-rules-p02-bad.
     plan
courier-rules-p02-bad.plan:
     error: the plan violates
     the problem :constraints
     [validate.
     constraint_violated]
plan invalid
```

sometime, at end, at-most-once

p03: the ICU nurse wants to see the robot pass by

```
(:goal (item-at waste1 dock))
(:constraints
  (sometime (robot-at r1 icu))))
```

- `(sometime C)`: in at least one state
- 10 actions instead of 8: steps into the ICU and out
- A goal that does not have to last
- `(at end C)`: in the last state, as a goal (so goals can be preferred in the same section)

p04: the dock door is slow and noisy

```
(:goal (and (item-at meds1 ward-a) (item-at waste1 dock)))
(:constraints
  (at-most-once (robot-at r1 dock))))
```

- `(at-most-once C)`: one stretch of consecutive states at most, of any length, or never
- Short plan: 11 actions, two visits
- With the rule: 12 actions, one visit (arrives, drops the waste, picks the medicines)

sometime-before and the cheat test

```
(:goal (item-at sample1 lab))
(:constraints
  (sometime-before (robot-at r1 lab) (sanitized r1))))
```

```
$ kairos courier-rules-domain.pddl
    courier-rules-p05.pddl
; cost = 8
(move r1 dock lobby)
(move r1 lobby corridor)
(sanitize r1 corridor)
(move r1 corridor ward-a)
(pick r1 sample1 ward-a)
(move r1 ward-a corridor)
(move r1 corridor lab)
(drop r1 sample1 lab)
```

- `(sometime-before A B)`: whenever `A` holds, `B` held in some *earlier* state
- Sanitized on the way *out*, the `pick` dirties it, enters the lab dirty
- Satisfied to the letter: the rule was written wrongly

Step E: the cheat test for a rule

The cheapest way to satisfy the words without doing what was meant? "Clean *when* it enters" is about one state:

```
(always (imply (robot-at r1 lab) (sanitized r1)))
```

sometime-after, and four operators as patterns

p06: after the ICU, disinfection at the dock

```
(:goal (item-at meds1 icu))
(:constraints
  (sometime-after (robot-at r1 icu) (robot-at r1 dock))))
```

- (`sometime-after A B`): whenever `A` holds, `B` holds then or *later*
- Goal reached after 7 actions; the plan has 11
- An obligation to discharge before the plan may end



Right: `C` comes back; the first `A` has no `B` before; the last `A` has no `B` after; the second `A` waits 4 steps

Rules with a clock

- (`within t C`): `C` in some state at time $\leq t$. A deadline
- (`always-within t A B`): whenever `A`, `B` then or at most t later. A deadline that restarts
- (`hold-during t1 t2 C`): `C` in every state with $t_1 \leq t_i < t_2$. A window
- (`hold-after t C`): `C` in every state with $t_i > t$. A window that never closes

The time of a state

t_i: when state S_i begins, $t_0 = 0$.

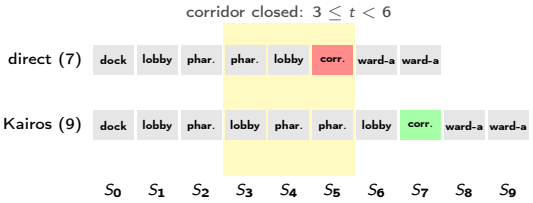
- Temporal plan: the time of the i-th happening, in the unit of durations
- Sequential plan: no clock; Kairos uses the position, S_i at time i

A deadline of 7 in a sequential plan: "within the first 7 actions"

Counting steps: a window in the corridor

```
(:goal (item-at meds1 ward-a))
(:constraints
  (hold-during 3 6 (not (robot-at r1 corridor)))))
```

```
$ kairos courier-rules-domain.pddl
    courier-rules-p09.pddl
; cost = 9
(move r1 dock lobby)
(move r1 lobby pharmacy)
(move r1 pharmacy lobby)
(move r1 lobby pharmacy)
(pick r1 meds1 pharmacy)
...
(drop r1 meds1 ward-a)
```



- A sequential plan cannot wait: the planner *invents* a wait (lobby and back)
- Also: p07 (`within 7 ...`), p08 (`always-within 3 ...`), p10 (`hold-after 4 ...`)

Counting seconds

`courier-timed`: durative moves (meters = seconds), pick and drop 2 s

```
(:goal (and (item-at meds1 ward-a) (item-at waste1 dock)))
(:constraints
  (within 45 (item-at meds1 ward-a)))
(:metric minimize (total-time)))
```

- Medicines in ward A at 44.007: in time
- Waste first: medicines at 54, rejected by `kairos validate`
- p02, (`hold-during` 20 40 ...): a temporal plan waits

```
$ kairos courier-timed-domain.pddl
    courier-timed-p01.pddl
; cost = 73.004
; makespan = 73.004
0.000: (move r1 dock lobby) [4.000]
4.001: (pick r1 waste1 lobby) [2.000]
6.002: (move r1 lobby pharmacy) [6.000]
12.003: (pick r1 meds1 pharmacy) [2.000]
14.004: (move r1 pharmacy lobby) [6.000]
20.005: (move r1 lobby corridor)
    [10.000]
30.006: (move r1 corridor ward-a)
    [12.000]
42.007: (drop r1 meds1 ward-a) [2.000]
...
```

The ten operators, side by side

Constraint	Holds for the trajectory $S_0, \ldots, S_n$ when
`(at end C)`	$S_n \models C$
`(always C)`	$S_i \models C$ for every i
`(sometime C)`	$S_i \models C$ for some i
`(at-most-once C)`	if $S_i \models C$, there is $j \geq i$ such that C holds in $S_i, \ldots, S_j$ and in no state after S_j
`(sometime-before A B)`	for every i with $S_i \models A$ there is $j < i$ with $S_j \models B$
`(sometime-after A B)`	for every i with $S_i \models A$ there is $j \geq i$ with $S_j \models B$
`(within t C)`	$S_i \models C$ for some i with $t_i \leq t$
`(always-within t A B)`	for every i with $S_i \models A$ there is $j \geq i$ with $S_j \models B$ and $t_j - t_i \leq t$
`(hold-during t1 t2 C)`	$S_i \models C$ for every i with $t_1 \leq t_i < t_2$
`(hold-after t C)`	$S_i \models C$ for every i with $t_i > t$

- `sometime-before`: $j < i$ strictly. `sometime-after`, `always-within`: $j \geq i$, the same state is enough
- "If `A` holds..." is satisfied by a plan where `A` never holds, the empty plan included

One problem per operator, two plans each

- `courier-rules-p02` to `p10`: one constraint each
- `-ok.plan` accepted, `-bad.plan` rejected; the bad plan is valid without the constraint

```
$ kairos validate courier-rules-domain.pddl
      courier-rules-p04.pddl
     courier-rules-p04-bad.plan
courier-rules-p04-bad.plan: error: the plan
      violates the problem :constraints [
     validate.constraint_violated]
plan invalid
```

The message names the section, not the constraint: validate against copies that keep one constraint each

What the operators cannot say

A small, fixed vocabulary. In PDDL 3.0 they cannot be nested (no `(always (sometime C))`); their conditions are about one state. "*Every time* it touches waste, it sanitizes before the *next* entry" needs a fact that remembers, constrained with `always`. The operators are a fragment of **linear temporal logic** (Pnueli, 1977)

Rules in the domain

`courier-safe`: the ICU rule in the domain (`icu` is a constant)

```
(:constraints
  (forall (?r - robot ?w - waste)
    (always (not (and (robot-at ?r icu) (carrying ?r ?w))))))
```

- Binds every problem; a plan must satisfy both sections
- p01: the same 16 actions
- p02: the waste lies *in* the ICU

```
$ kairos courier-safe-domain.pddl
    courier-safe-p02.pddl
...
kairos: unsolvable (exhausted)
```

- Read literally, it forbids cleaning the ICU; the hospital meant "no waste *into* the ICU"

Step M: whose rule is it

Scope: a law of the world (preconditions and effects), a rule of the organization (domain constraint), a condition of today (problem constraint). **Strength**: must it hold, or should it?

Constraints as preferences

`courier-rules-p11.pddl`: the hard ICU rule and two wishes about time

```
(:constraints
  (and
    (forall (?r - robot ?w - waste)
      (always (not (and (robot-at ?r icu) (carrying ?r ?w)))))
    (preference fast-meds (within 8 (item-at meds1 icu)))
    (preference early-waste (within 6 (item-at waste1 dock)))))
(:metric minimize (+ (* 3 (is-violated fast-meds))
                     (* 2 (is-violated early-waste)))))
```

A rulebook with exceptions

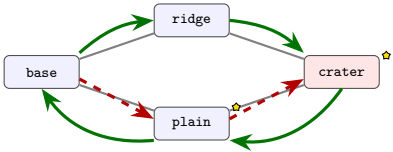
Rules that protect patients stay hard; rules of convenience get a price

- **Soft constraint**: a constraint in a `preference`; 1 in the metric if the trajectory breaks it
- 8 steps: medicines straight to the ICU; 6: waste straight to the dock. Both cannot be first
- Costs 5, 3, 2: the last plan delivers the medicines first and pays 2
- The hard rule holds in all three

Two more worlds: a rover and a crater

```
(:constraints (and
  ; look at the crater from the ridge before driving into it
  (sometime-before (at crater) (at ridge))
  ; drive into the crater only once
  (at-most-once (at crater))
  ; after the crater, come back to the base to be cleaned
  (sometime-after (at crater) (at base)))))
```

```
$ kairos rover-domain.pddl rover-p02.pddl
; cost = 6
(drive base ridge)
(drive ridge crater)
(collect crater)
(drive crater plain)
(collect plain)
(drive plain base)
```



- Without rules (p01, red): 4 actions through the plain
- Green: by the ridge, once through the crater, back home although no goal asks
- No fact “the crater has been surveyed”: the trajectory holds the history

Two more worlds: a workshop with an oven

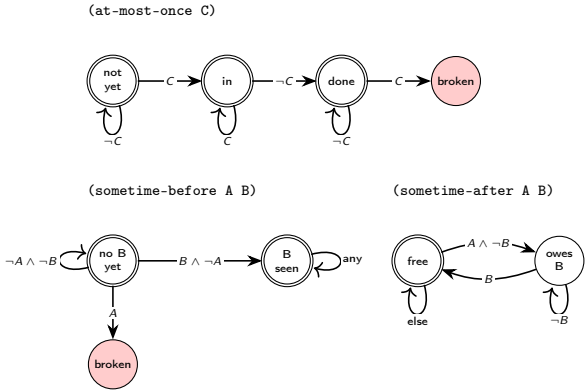
Two safety rules of the workshop, so constraints of the domain:

```
(:constraints (and
  ; never have the door open while the oven is hot
  (always (not (and (hot) (door-open))))
  ; one heating cycle per shift
  (at-most-once (hot))))
```

- Actions with physics only: `heat` and `open-door` ignore each other
- `close-door` before `heat`, `cool` before opening: rule 1. One batch: rule 2
- A hasty plan: `the plan violates the domain :constraints`

```
$ kairos workshop-domain.pddl
    workshop-p01.pddl
; cost = 11
(open-door)
(load lid)
(load gear)
(load bracket)
(close-door)
(heat)
(cool)
(open-door)
(unload lid)
(unload gear)
(unload bracket)
```

What the planner does with a constraint



A plan may end only in a double circle; a red state can never be left

- A **monitor**: a small automaton that remembers just enough of the past
- Kairos compiles them away: forbidden → precondition of every action; memory → new fact with conditional effects; at the end → goal
- Rules prune the search; the price is size
- Only Kairos and OPTIC returned valid plans

The MODEL method: rules about the whole plan

New questions for steps M and D

Step M. List the rules apart from the goals. For each one:

- **About one action or about the whole plan?**
- **One state or history?** One state at a time: an `always`. "Before", "after", "once", "by": memory
- **Whose rule is it, and can it bend?** Domain or problem; hard or preference

Step D. A new precondition: what the action *needs* (dynamics) or what somebody *wants* of the plan (a rule, perhaps better written once as a constraint)?

- The model card gains a table of rules: sentence, scope, strength, operator, and the cheat test run on each

Alternatives compared: a rule about the whole plan

	Constraint	Preconditions on every action	Automaton with extra predicates
Fidelity	Once, over all states; cannot leak. Can be *too* strong	Exact only if no action is forgotten; state rules only	Exact if the automaton is right; any rule
Size	Monitors on every action	Only the actions concerned	One or two facts per rule and object
Guidance	Good in Kairos	Good	Good
Clarity	Best: one line, can be made soft	Scattered, mixed with physics	Worst: hidden in bookkeeping
Portability	Poor	Good: ADL	Good: ADL, conditional effects

- `courier-icu-pre` (rule on `move` and `pick`) matches the constraint; `courier-icu-leaky` (`move` only) finds a plan that picks the waste up *in* the ICU
- `courier-dock-auto` adds `(left-dock ?r)`: the same 12 actions as `at-most-once`
- **Verdict**: write the rule as a constraint first, even if the final model cannot use one; it is the specification and, with `kairos validate`, the test

Common mistakes (1)

Forgetting that the initial state counts

Dock at most once, robot starting *at* the dock:

```
$ kairos courier-rules-domain.pddl
    courier-rules-once-start.pddl
...
kairos: unsolvable (exhausted)
```

The one visit is spent in S_0. *Fix*: decide what the rule means for the state the day starts in

An operator outside its section

In a goal, `sometime` is a predicate nobody declared:

```
$ kairos check courier-rules-domain.pddl
     courier-rules-ingoal.pddl
courier-rules-ingoal.pddl:20:15:
    warning: 'sometime' is not
    declared: never true [dialect.
    undeclared_predicate]
...
```

Then `unsolvable`. *Fix*: move it to `:constraints`; read the warnings

Common mistakes (2)

A window that the plan never reaches

`(hold-after 10 (robot-at r1 dock))`: a 7-action plan ending in ward A is valid. *Fix*: if it must be at the dock, say it in the goal too

Standing still through a window

Kairos checks a window at the times states *begin*: a robot standing in the corridor from 14 to 45 passes `(hold-during 20 40 ...)`. *Fix*: timed literals switch `(closed corridor)` at 20 and 40, and an `always`

Steps taken for seconds

`(within 45 ...)` in a sequential problem gives 45 *actions*. *Fix*: a number has the unit of the plan: positions or the time unit of durations

A rule stronger than the world

`unsolvable` on an easy problem. *Fix*: remove constraints one at a time until a plan appears, then validate it against each one alone

1 Soft goals and preferences

2 Constraints on the whole plan

3 Object fluents

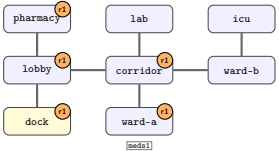
4 Summary

One robot in two rooms

courier-twin-domain.pddl: move lost one line

```
(:action move
  :parameters (?r - robot ?from ?to - location)
  :precondition (and (robot-at ?r ?from) (connected ?from ?to))
  :effect (robot-at ?r ?to))            ; the delete is missing
```

```
$ kairos courier-twin-domain.pddl
     courier-twin-p02.pddl
; cost = 6
(move r1 dock lobby)
(move r1 lobby pharmacy)
(pick r1 meds1 pharmacy)
(move r1 lobby corridor)
(move r1 corridor ward-a)
(drop r1 meds1 ward-a)
```



- 6 actions instead of 10; leaves *from the lobby*
- Check and validator accept it
- 2^8 situations, only 8 make sense

A function whose value is an object

Declared in `:functions`, with the type of the value:

```
(:functions
  (location-of ?r - robot) - location    ; the robot's room
  (holder ?i - item) - place)            ; a room, or a robot
```

- **Initial value**: `(= (location-of r1) dock)`
- **Test**: `(= (location-of ?r) ?from)` (needs `:equality`)
- **Change**: `(assign (location-of ?r) ?to)`; the old value is gone, nothing to delete
- Requirement: `:object-fluents`

Predicate: eight answers

lobby	corr.	phar.	lab	ward-a	ward-b	icu	dock
true	false	true	false	false	false	false	true

`(robot-at r1 _)`

Object fluent: one value

pharmacy `(location-of r1)`

one slot; assigning overwrites it

`(holder ?i)`: a room or a robot (supertype `place`); replaces `item-at` and `carrying`

The courier world with object fluents

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

(:action pick
  :parameters (?r - robot ?i - item ?l - location)
  :precondition (and (= (location-of ?r) ?l) (= (holder ?i) ?l)
                     (free ?r))
  :effect (and (assign (holder ?i) ?r) (not (free ?r))))

(:action drop
  :parameters (?r - robot ?i - item ?l - location)
  :precondition (and (= (location-of ?r) ?l) (= (holder ?i) ?r))
  :effect (and (assign (holder ?i) ?l) (free ?r))))
```

```
  (= (location-of r1) dock) (free r1)
  (= (holder meds1) pharmacy))
(:goal (= (holder meds1) ward-a)))
```

- `move`: one assignment, no delete to forget
- `pick`: the item changes holder, from the room to the robot
- `(free ?r)` stays a predicate: true or false, not a value
- The same 7 actions as the typed domain
- The bug cannot be written: no second value

Fluents as terms

`courier-terms`: a fluent where an object is expected

```
; no ?from: the robot leaves from wherever it is
(:action move
  :parameters (?r - robot ?to - location)
  :precondition (connected (location-of ?r) ?to)
  :effect (assign (location-of ?r) ?to))

; no ?l: the item must be held by the room the robot is in
(:action pick
  :parameters (?r - robot ?i - item)
  :precondition (and (= (holder ?i) (location-of ?r)) (free ?r))
  :effect (and (assign (holder ?i) ?r) (not (free ?r))))
```

- Parameters that only carried a value disappear
- Ground actions: 30 (`courier-of`) vs. 23

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

- The price:
  `(move r1 lobby)` no longer says where from

No value: undefined

`courier-cargo`: `(room-of ?i)` has no value while carried

```
(:action pick
  :parameters (?r - robot ?i - item ?l - location)
  :precondition (and (= (location-of ?r) ?l) (= (room-of ?i) ?l)
                     (free ?r))
  :effect (and (carrying ?r ?i) (not (free ?r))
               (assign (room-of ?i) undefined)))
```

- Not in `:init`, or assigned `undefined`; equal to nothing
- Prefer total fluents (`holder`) to partial ones

undefined cannot be compared

No `(= (room-of ?i) undefined)`: `kairos check` rejects it. *Fix*: keep a fact

Stopping one action early:

```
$ kairos validate
      courier-cargo-domain.pddl
      courier-cargo-p01.pddl
      courier-cargo-p01-carried.
      plan
courier-cargo-p01-carried.plan:
      error: evaluating the goal:
       (room-of sample1) is
      undefined [validate.
      evaluation]
plan invalid
```

Two more worlds: blocks with one fluent

```
(:types block surface - support)

(:constants table - surface)

(:functions
  (on-top-of ?b - block) - support)   ; what ?b rests on

; move a clear block onto another clear block
(:action stack
  :parameters (?b ?to - block)
  :precondition (and
    (not (= ?b ?to))
    (not (exists (?c - block) (= (on-top-of ?c) ?b)))
    (not (exists (?c - block) (= (on-top-of ?c) ?to))))
  :effect (assign (on-top-of ?b) ?to))

; move a clear block to the table
(:action unstack
  :parameters (?b - block)
  :precondition (and
    (not (= (on-top-of ?b) table))
    (not (exists (?c - block) (= (on-top-of ?c) ?b))))
  :effect (assign (on-top-of ?b) table)))
```

- `(on-top-of ?b)`: a block or the table
- *Clear* is not stored: no block has it as support
- Two actions, one effect each; nothing to go out of step

```
$ kairos blocks-of-domain.pddl
    blocks-of-p01.pddl
; cost = 3
(unstack c)
(stack b c)
(stack a b)
```

Two more worlds: coloring a map

```
(:types region color)

(:predicates
  (border ?a ?b - region))

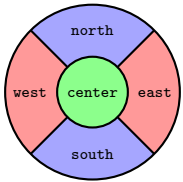
(:functions
  (color-of ?r - region) - color)   ; undefined until painted

(:action paint
  :parameters (?r - region ?c - color)
  :precondition (forall (?n - region)
                  (imply (border ?r ?n)
                         (not (= (color-of ?n) ?c))))
  :effect (assign (color-of ?r) ?c)))
```

Goal of `coloring-p01`: "every region has *some* color"

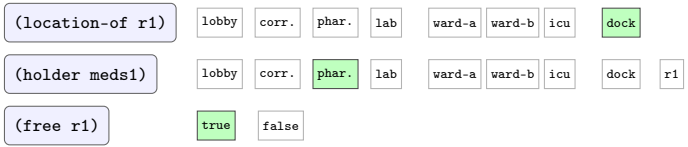
```
(:goal (forall (?r - region)
         (exists (?c - color) (= (color-of ?r) ?c)))))
```

- An uncolored neighbor passes: undefined is not equal to ?c



- Kairos: 5 `paint` actions, three colors
- Two colors (`p02`): 94 states, `unsolvable`: a small proof
- A CSP: one variable per region

State variables: what the planner sees



A state is one value per variable: $8 \times 9 \times 2 = 144$ combinations, against 2^{18} sets of facts. With predicates the planner must *discover* the variables (**mutex groups**):

```
$ kairos courier-domain.pddl courier-p01.pddl -v
kairos: 18 atoms, 30 actions (0 durative), 3 variables (2 mutex groups), ...
$ kairos courier-of-domain.pddl courier-of-p01.pddl -v
kairos: 1 atoms, 30 actions (0 durative), 3 variables (0 mutex groups), ...
```

- The fluents *were* the variables. `courier-twin-p02`: 10 variables (1 mutex groups): an invariant that does not hold

The MODEL method: one value or many

New questions for step O (Objects and state)

For every dynamic property, how many values does it have for one object?

- **Exactly one, always?** Room of a robot, holder of an item: an object fluent, or a predicate with a stated and tested invariant
- **At most one?** Color of a region: a fluent that may be undefined; who needs to ask?
- **Any number?** Rooms visited, items on a cart: a predicate
- **A number?** A numeric fluent

The state table gains a column, *values per object*. Every "one" written as a predicate is an invariant, and step E owes it a test.

Alternatives compared: where things are

	Predicates	Object fluents	Fluents as terms
Fidelity	Right only while every action keeps the invariant	The invariant cannot be broken	The same
Size	18 facts, 30 ground actions	1 fact, 2 fluents, 30 ground actions	23 ground actions
Guidance	3 variables, after finding 2 mutex groups	3 variables, given	The same
Clarity	Familiar; the invariant is implicit	The declaration documents the invariant	Shortest; plans say less
Portability	Every planner	`:object-fluents`: of those compared, only Kairos	The same

Verdict

Think in fluents always: "one value or many" belongs to step 0 whatever the syntax. Write fluents when the planner supports PDDL 3.1 fully; otherwise predicates, with each single-valued property recorded as an invariant and tested (the count of variables and mutex groups of `kairos -v` is one cheap test)

Common mistakes

Functions are not predicates

`(location-of r1 dock)` in `:init`: a warning, the fact is ignored, `unsolvable`. *Fix*: always compare or assign

Two initial values, or none

Two: a warning, the last one is used. None: `unsolvable` after one state. *Fix*: read the warnings; check initial values first

A value of the wrong type

```
$ kairos check courier-of-badtype-domain
    .pddl
courier-of-badtype-domain.pddl:28:18:
    error: in action 'pick': object
    function 'location-of' returns
    location [semantic.type_mismatch]
```

A fluent for a relation

The lobby has three neighbors: as a fluent, solutions are lost

1 Soft goals and preferences

2 Constraints on the whole plan

3 Object fluents

4 Summary

PDDL 3 at a glance

Feature	Requirement	Version	Where
Soft goals, preferences in preconditions, `is-violated`	`:preferences`	3.0	Ch. 14
Trajectory constraints, in the domain and in the problem	`:constraints`	3.0	Ch. 15
Soft constraints (a preference around a constraint)	both of the above	3.0	Ch. 15
Functions with objects as values, `undefined`	`:object-fluents`	3.1	Ch. 16
Functions with numbers as values (the old `:fluents`)	`:numeric-fluents`	3.1	Ch. 8
Numeric and object fluents together	`:fluents`	3.1	
`total-cost` increased by actions	`:action-costs`	3.1	Ch. 9

- No new kind of action or world: a place to say directly what the modeler means. *This is wished, this is a rule, this has one value*
- The cost: fewer planners read it. Portability was the last word of each comparison

What to take away

- A **preference** may be violated at a price set in the metric: in a goal about the last state, in a precondition about each application. Always `minimize`
- Weights in the unit of the cost: wishes against effort (net benefit). With soft goals, let the anytime search run
- A **trajectory constraint** judges every state, S_0 included; ten operators, hard unless wrapped in a preference; in the domain or in the problem
- The domain says what *can* be done; the constraints, what *may* be done. Run the cheat test on every rule
- An **object fluent** has one value: no delete to forget; `undefined` can be assigned, never tested
- MODEL: step M separates required from wished, and rules by scope and strength; step O asks "one value or many"

Exercises to try

- **14.3.2**: weights for strict priority among three classes
- **14.4.1**: the exact weight at which the meal is delivered
- **14.6.3**: why a soft goal does not express the lobby rule
- **14.9.2**: step M from a student's evening
- **15.1.1**: goal, precondition, or whole plan?
- **15.3.3**: the sanitizing rule written with `always`
- **15.6.2**: change a weight in `courier-rules-p11`
- **15.9.2**: step M on a kitchen story
- **16.7.1**: predicate, object fluent or numeric fluent?
- **16.8.1**: which feature of Part V for each need