The method at work

Planning and PDDL, Part VII (Chapters 19–20)

Francisco Martín Rico

Intelligent Robotics Lab, URJC



Outline

1. The MODEL method, complete
2. Case study: a robot mission from story to execution
3. Summary

1 The MODEL method, complete

2 Case study: a robot mission from story to execution

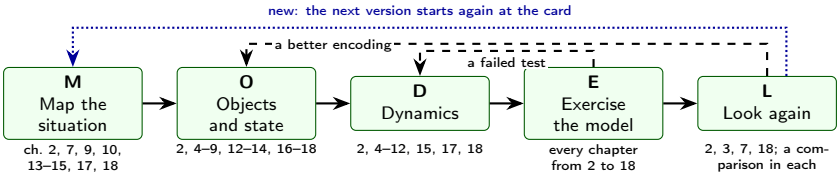
3 Summary

The method on one page

A laundromat, three loads, one washer, one dryer, a woolen sweater, a morning that is not endless. Where does the first line come from? Chapter 19 teaches no new PDDL: it puts the method in one place.

Step	Question	Artifact
M Map the situation	What is the problem, who acts, what counts as solved, what is left out?	**Model card**: Story, Agents, Success, Scope, Assumptions; later the metric, wishes, rules, schedule
O Objects and state	Which things exist, and what must the planner know about them to decide?	**State table**: each property, static or dynamic, how many values, how encoded
D Dynamics	What can be done or can happen, when, and what does it change?	**Action table**: each action, process and event; when, what it changes, how long
E Exercise the model	Does the planner do what the world would allow, and nothing else?	**Test set**: trivial, typical, no plan, cheat; every plan read and validated
L Look again	Which other encoding would do, and which is better here?	**Comparison**: alternatives on the five criteria, a decision and its reason

Five steps, three loops, five criteria



Criterion	How the book measures it
Fidelity	No impossible plans (cheat tests, hand plans rejected by `kairos validate`); no lost solutions (reasonable hand plans accepted)
Size	Atoms and ground actions of the `[ground]` line of `kairos -v`
Guidance	States to the first and last plan; first vs best; `[sas]` mutex groups; the engine
Clarity	File and plan read like the story; places to change when a rule changes
Portability	Requirements against the table of planners of Appendix B

The five checklists (1): M and O

A **checklist**: questions answered in writing, in order, every time. A quick "no" is a decision: it goes to *Scope*.

Checklist M: the model card

- **Agents**: who acts, who at the same time (7, 10)
- **Success**: required vs wished; price of a wish (14)
- **Quality**: one number, which unit? Does *when* matter (9, 10)?
- **Assumptions**: a calendar (13)? Change when nobody acts (17)?
- **Rules** apart from goals: one action or the whole plan; can it bend (15)?

Checklist O: the state table

- Static or dynamic (2)? A constant (4)?
- How many values: one, at most one (5, 16)?
- About absence? Twins in step (2, 5)?
- Deducible; a rule over a chain (6, 13)?
- A quantity: threshold, bounds, counter (8, 9)?
- Changes by itself: switch and measure (17)?

The five checklists (2): D, E and L

D: the action table

- **Who causes it?** Agent: action. World, while: process. World, at an instant: event. *Imagine the agent falls asleep* (17)
- Needs, or a rule (15)?
- Everything that changes (2)
- Duration, overlaps (10, 11)

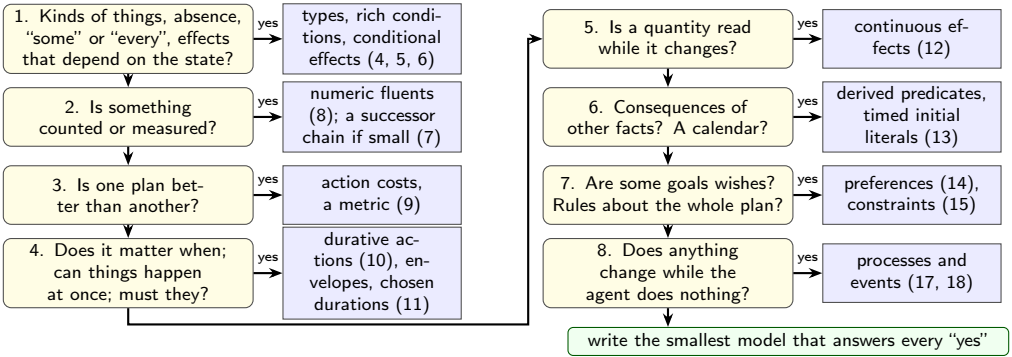
E: the test set

- Estimate first; pencil (3, 18)
- `check`, solve, read, `validate`
- Four tests, expected cost known
- Cheats as plan files (7)
- One test per promise
- No plan: a proof? (11, 18)

L: the comparison

- Every pattern has a rival
- The **lowest level** that does not lie (18)
- Two or three, run, with numbers
- Decision and reason in the card

Which feature for which need: the decision guide



A "yes" adds a feature and the reading goes on. Order of the book, and of cost: **stop at the lowest level that does not lie about what the problem turns on**.

From need to feature (1): state, conditions, effects

The problem needs	Feature	Requirement	Ch.
kinds of things, some treated alike	types, hierarchy, `either`	`:typing`	4
an object that the rules name	a constant	none	4
two parameters that must differ	`(not (= ?a ?b))`	`:equality`	4
a condition about absence	`not`	`:negative-preconditions`	5
several ways to satisfy a condition	`or`, `imply`	`:disjunctive-preconditions`	5
a condition about some object	`exists`	`:existential-preconditions`	5
a condition or goal about every object	`forall`	`:universal-preconditions`	5
outcome depends on the state; many objects	`when`, `forall` in effects	`:conditional-effects`	6
a quantity added, spent, compared	functions, `increase`, ...	`:numeric-fluents`	8
a better plan, a sum of fixed costs	`(total-cost)`, `:metric`	`:action-costs`	9
a better plan, a measure of the state	`:metric` over fluents	`:numeric-fluents`	9
a property with exactly one value	an object fluent	`:object-fluents`	16

`:adl` declares at once every requirement above the first line.

From need to feature (2): time, wishes, a world that moves

The problem needs	Feature	Requirement	Ch.
actions that last and overlap; finishing time	durative actions, `(total-time)`	`:durative-actions`	10
something that holds only while an action runs	an envelope	no flag of its own	11
a duration that is a decision	bounds on `?duration`	`:duration-inequalities`	11
a quantity read while it changes	`#t` in a durative action	`:continuous-effects`	12
a fact that follows from others, any depth	`:derived`	`:derived-predicates`	13
a calendar: hours, windows, deadlines	`(at t (fact))` in `:init`	`:timed-initial-literals`	13
numbers that change at known times	`(at t (= (f) v))`	`:timed-initial-fluents`	13
goals that are wishes, with a price	`preference`, `is-violated`	`:preferences`	14
a rule about the whole plan	`always`, `within`, ...	`:constraints`	15
change while a condition holds	`:process`	`:time`	17
change when a condition becomes true	`:event`	`:time`	17
a condition no instant may break	`:constraint` in the domain	`:time`	18

What each feature costs (1)

Feature	Buys (Fidelity)	Charges	Portability
Types, constants, =	mixed kinds become `check` errors	nothing: same ground problem	7; `either` 5
Negation, `or`	absence, alternatives said directly	`or` may duplicate actions	5
`forall`, `exists`	rules that follow the objects	expanded over the objects	5
Conditional effects	one action for one thing in the world	compiled into more actions by planners without them	3
Derived predicates	consequences cannot go out of step	evaluated in every state	3
Numeric fluents	amounts as data	the state space is no longer finite	5
Action costs; metrics	which plan is better	best plan found late, proved later	4; 4
Object fluents	one value cannot break	nothing	1

Portability: planners, of the eight compared in the documentation of Kairos (Appendix B), that returned a valid plan for one small problem with the feature, 10 s each.

What each feature costs (2)

Feature	Buys (Fidelity)	Charges	Portability
Durative actions	overlap, makespan	plans valid, not the shortest	5
Required concurrency	what exists only while an action runs	"no plan found" without proof	4
Chosen durations	"for as long as needed"	durations at their bounds	3
Continuous effects	a number read while it moves	instants become decisions	2
Timed literals	the calendar of the world	actions must fit in windows	3
Preferences	wishes with a price	optimization: first plan may be empty	4
Constraints	rules stated once, every state	a monitor in every state	2
Processes, events	a world that does not wait	steps of time; plans replayed	1

- The missing planners differ: POPF and OPTIC lack negation, `or`, conditional effects; LAMA lacks numbers
- A low-row feature is often worth trading for a high-row pattern: complementary predicates, a successor chain, blocks, an automaton

The catalog of patterns: forty names, each with a rival

Classical (Chapter 7, and 2–6)

Pattern	Rival
Static table	rule in an action
Mirrored facts	one fact and an `or`
At-most-one invariant	an object fluent
Successor chain	slots; a number
Phase token, turn token	preconditions
Resource lock	an envelope (time)
Inverse pair, one-way	each other
Merged action	separate actions
Complementary predicate	a `not`
One action per case	a `when`
Deduced, not stored	a stored fact

Numbers, time, PDDL 3, hybrid (8–18)

Pattern	Rival
Bounded quantity	a threshold fact
Counter fluent	action costs
Token	none: actions overlap
Envelope and contents	a lock
Slack	exact simultaneity
Chosen duration, blocks	each other
Window, deadline	constraints
Soft goal	a hard goal
Monitor	a constraint
Start-process-stop	a durative action
Global constraint	an event and a goal

The name says what a row of the state or action table is; the rival is a column of step L.

Anti-patterns: valid PDDL, caught by the symptom

They pass `kairos check`: `valid, plannable`. Four families of symptoms, and the family says where to look.

Too easy: the model allows too much

Too short, cheap or early. Forgotten delete (2, 7); consumption without precondition (8); an action that forgot its cost (9); a token forgotten (10); a limit checked only at the start (12)

Too big: something out of control

More ground actions or states than estimated. Idle parameter (3, 7); a horizon nobody bounded (18)

Too hard: the model allows too little

No plan, or odd detours. A forgotten initial value (8, 16); out-of-step twins (5, 7); a rule stronger than the world (15); equality on fractions (8) or on a flow (18)

Not what was asked

Right for the model, another question. Preferences without a metric (14); weights in the wrong unit (14); a constraint where an event was needed (18)

Debugging: read the verdict first

Verdict	Exit code	What it says, and an example
a plan	0	Valid for the model. `solved` is not `proven optimal` (9)
`unsolvable`	10	**A proof**: no plan exists (3). `laundry-v1-p03`
`no plan found`	14	Search ended without plan and without proof (11, 12, 17). `laundry-v5-p03`
`timeout`	11	Nothing: the search was still running (3)
`invalid`	2	The files do not say what was meant: read the first error (1)

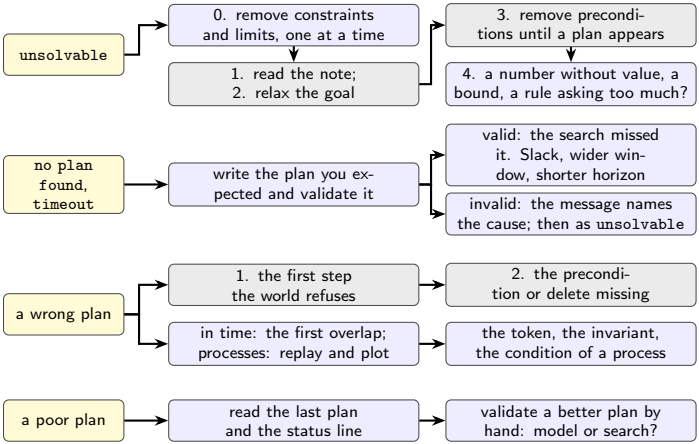
Only one verdict is a proof

The scripts that run a test set (`tests.txt`) state the exit code that each test expects.

No plan, without a proof

Write the plan you expected and run `kairos validate`. **Invalid**: the message names the cause. **Valid**: the search missed it.

The debugging procedure, extended



- Gray: the procedure of Chapter 7
- Blue: what numbers, time, constraints and processes add

A wrong "thing" is a **step** (sequential), an **overlap** (temporal), a **stretch of time** (processes)

What the planner of this book does not find

When the hand-written plan is valid, the model is right and the search missed it. With Kairos 1.0.6:

Four plans that are missed

- **A duration between its bounds**: tried only at the bounds. *Around*: make something need it, or blocks (11)
- **The same envelope used twice** (11). *Around*: write the plan by hand, validate it
- **Two starts at the same instant** (11). *Around*: slack
- **Between two steps of 0.1**: narrow windows walked past (17, 18). *Around*: windows $\geq$ rate $\times$ 0.1; inequalities

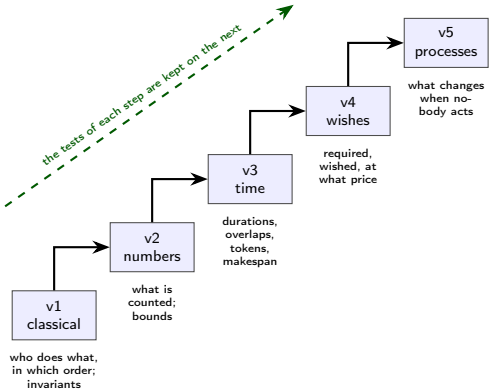
Two that concern good plans

- A temporal plan is a valid timetable, not the shortest (11)
- In PDDL+ the search minimizes the number of actions, whatever the metric: plans wait and charge more (18)

In every case

The plan that Kairos prints is valid. What may be missing is a plan, or a better one. A "no plan" test needs a **bounded horizon**.

Growing a model in versions



Rules for growing a model

1. **One dimension per version**
2. **Green before the jump**: all tests as expected
3. **The card changes first**, then the tables, then the PDDL
4. **Old tests translated and kept**; a changed verdict is explained by a line of the card
5. **New tests**: a "no plan" and a cheat the old version could not state
6. **Every version is a file**
7. **Stop** when the model no longer lies

The laundry: story and version 1

Saturday at the laundromat. Loads are washed, dried, folded. The label says which machines a garment may go in; wool must not be tumble-dried. One washer, one dryer.

Step M: the card, version 1

Agents	The person; machines are tools
Success	Every load of the goal folded
Scope	In: dirty, wet, folded; where; labels. Out: time, kilograms, coins, the clothesline
Assum.	Classical. One load per machine. A cycle never fails

Patterns: `allowed` static table; `in` at-most-one; `empty` resource lock; `put`/`take-out` inverse pair; the rest one-way

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

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

Version 1: the four tests

Trivial: one load, seven steps, written before running.

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

Typical: three loads, 21 steps, proved optimal. **No plan**: the sweater.

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

Cheats as plan files: two loads in one wash; folding wet towels.

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

Version 2: numbers

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

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

- Card: kilograms in; a drum holds up to its capacity; a cycle treats all inside
- `empty` disappears: `room`, a bounded quantity
- One jump pulled a feature of Chapter 6 along
- The old cheat (two loads) is now legal: rule 4

Loads of 3, 4, 5 kg, drums of 8 kg:

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

Duvet of 9 kg: `unsolvable`. Three loads in one wash:

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

Version 3: time

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

Card: *as early as possible*; one thing at a time; machines run alone; minutes.

Action	Dur.	Must hold
`put, take-out`	1	stopped, throughout
`wash, dry`	cycle	stopped at start; a load inside
`fold`	5	clean, dry at start

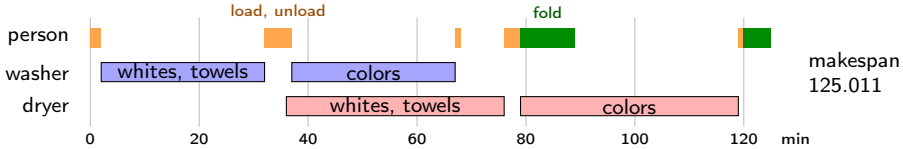
- Two **tokens**: `(hands-free)`, `(stopped ?m)`
- An **invariant**: nobody opens a running machine

Version 3: an empty cycle, then the honest timetable

The first version 3 (`laundry-v3-idle`):

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

- A cycle with nothing inside: it ends before the plan does, so it costs nothing
- The jump changed what the planner is indifferent to
- Wrong plan: a missing precondition, *something in the drum*



The dryer is the bottleneck: no timetable can be much shorter than 125. Know the honest makespan before trusting the planner.

Version 4: wishes

Wish	Condition	Weight
(required)	whites folded	
`colors-done`	colors folded	60
`towels-done`	towels folded	30

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

- Steps O, D unchanged; `:preferences`; the jump is in the problem
- Colors in the drum: 9 min for 60. Towels: a second batch, 37 min for 30
- Price test: colors worth 5 (p04): dropped, cost 93.010. Break-even 9

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

A poor plan (p03, towels worth 40):

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

Hand plan better: model right, search (M2) stopped short.

Version 5: processes

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

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

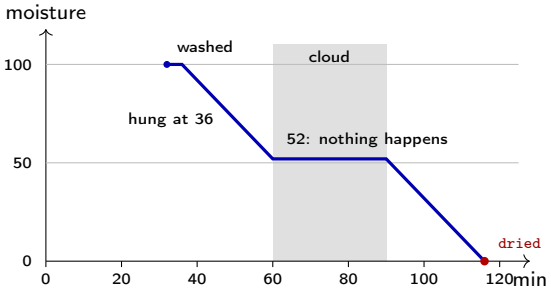
- Card: *the clothesline is in*
- Washing and dryer stay durative: length known, nothing interrupts them
- The wash sets the moisture to 100

Checklist D: who causes it?

- The person falls asleep after hanging: the sweater dries all the same. Not an action
- Dries while it hangs and the sun shines: a **process**
- Dry at one instant: an **event**
- The sun follows a calendar: **timed initial literals**

Switch: `(hanging ?l)`, `(sunny)`. Measure: `moisture`. The old "no plan" changes verdict: sun all morning (`p01`), folded at 90.

Version 5: a cloud, a replay, a verdict



p02: whites, wool, a cloud from 60 to 90. Plan of 169 min; wool collected at 116, when the event fired. The whites wait wet from 33 to 122.

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

- Cheat: dry in the shade (`p02-shade`), rejected: `(wet wool) is true`
- No plan: overcast (p03)

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

Exit 14: the pencil gives the proof; without sun, moisture stays at 100.

Five versions, side by side

	Adds to the card	Typical test (p02)	Search (`kairos -v`)
v1	order of the work; one load per machine	21 steps, optimal	808 states, 3 ms
v2	kilograms; a cycle treats the whole drum	19 steps, optimal	2 106 states, 12 ms
v3	durations; what may overlap; the makespan	125.011 min	101 696 states, 2.1 s
v4	required and wished; prices in minutes	cost 118.010	5 358 states, 20 ms
v5	the line, the sun, the cloud	169 min (122 by hand)	168 296 states, 2.0 s

- Each version says something true that the previous could not, costs more to search and runs on fewer planners (v2: 3 of 8; v5: 1)
- The versions are **not drafts**: which loads fit together, best in v2 with a proof in milliseconds; an ordinary morning, v3; v5 only on days with wool
- Different problems: the last column compares orders of magnitude

Step L: is the clothesline a process?

Three encodings on top of version 3, same morning (wool, whites, cloud from 60 to 90):

	A: fixed length (50 min)	B: inside a window	C: process and event
Fidelity	impossible plan: wool folded wet; right only without clouds	no impossible plan; loses solutions: drying cannot pause	exact: stops under the cloud, resumes
Guidance	10 407 states; makespan 91.005	9 535 states; 146.002	168 296 states; 169, 122 by hand
Clarity	one more action	the same, one condition	law of the line apart; plans replayed
Portability	durative, timed literals	the same	processes: Kairos alone

- A: where the sun can be trusted, or a first version. B: honest at the temporal level, when sunny spells are long. C: when drying stops and resumes
- The decision depends on a question nobody asked: *how reliable is the sun?* Back to the card

Documenting a model

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

- **Header** for the reader of the file. *Decides*: actions vs world. *Units*: the silent mistake. *Rules*: where each is enforced. *Problem*: the contract
- **Model card** for who owns the problem; it adds Quality, Wishes, Rules, **Level**, **Decisions**, **Known limits**, **Versions**
- **Test file** for whoever changes it next: `tests.txt`, one line per test with the expected exit code

Known limits of the laundry

v4 plans can improve when a second batch could overlap; v5 plans do not hurry.

Common mistakes in the way of working

A cheat that passes for the wrong reason

The v2 cheat run on v3 is rejected for `duration 0.000000 does not satisfy`, not for 12 kg. Green and tests nothing. *Fix*: translate plan files; read every message

A header of an older version

`laundry-v2-stale: warning: uses conditional-effects but :requirements does not include it.` *Fix*: copy requirements from the list `check` prints

A "no plan" test with an open horizon

Sun until 50 (p05): 3 s and 371 765 states, every second given. Overcast (p03): 8 093 states, under 0.1 s. *Fix*: bound the horizon

A verdict kept when the card changed

The sweater expected exit 10 for four versions; in v5 it has a plan of 90 min. *Fix*: a verdict changes only with a line of the card

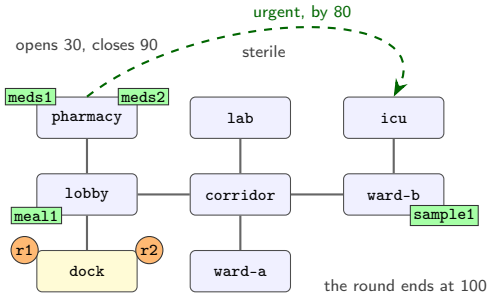
1 The MODEL method, complete

2 Case study: a robot mission from story to execution

3 Summary

The morning round, as the manager describes it

- Two couriers `r1`, `r2` in the dock, **one charger**; batteries 100 and 30; 1 unit per meter; +5 per s; *stays until full*
- Urgent: `meds1` to the ICU **by 80**. Routine: `meds2` to ward A; `sample1` from ward B to the lab; `meal1` from the lobby to ward B
- Pharmacy **opens at 30**, closes at 90
- Lab is sterile: samples **sealed**; hand over only at the destination
- Round ends at 100. Then medicines, sample, meal; robots home *if possible*



one charger; battery `r1` 100, `r2` 30

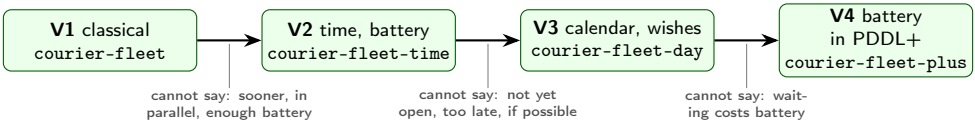
Step M is a conversation

- *Two things at once?* No: one compartment
- *Leave the charger before full?* No. **Decides an encoding**
- *Hand over, or leave on the way?* No: "no boxes in corridors"
- *An unsealed sample in the lab?* "It must not happen": not done, not lost
- *"If possible", compared to what?* Home is worth less than any delivery
- *Battery while waiting?* "A little": out at first

Every answer is a line of the model card.

A plan for the modeling

Grow in versions; jump only when the current one cannot say something that matters.



Step M: the model card of the morning round

Model card

Agents	`r1`, `r2`: at the same time, one thing each. Pharmacy, charger, clock: a schedule or a law
Success	Required: `meds1` in the ICU by 80. Wished: the other deliveries before 100; robots in the dock
Scope	In: rooms, distances, positions, carrying, sealing, durations, battery, hours, deadlines. Out: routes in a room, people, grasping, failures, uncertainty
Assumptions	One item per robot; hand over at destination; one robot at the charger, until full; 1 per meter, nothing standing (revised in v4); pharmacy 30–90, deadline 80, end 100

Wish	Condition	Weight
`meds2-done`	`meds2` in ward A	5
`sample-done`	`sample1` in the lab	4
`meal-done`	`meal1` in ward B	2
`r1-home, r2-home`	robot ends in the dock	1 each

The manager's order as numbers: $5 > 4 > 2 > 1$. Checked with her: no delivery is ever given up to bring the robots home.

Version 1: who carries what

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

- Types `medicine`, `sample`, `meal` under `item`
- `destination`: a **static table**, the requests are data
- No "contaminated": the bad state is unreachable, it needs no name
- "No boxes in corridors" removed `drop`: `deliver` is a **one-way action**, on purpose
- `(not (delivered ?i))` in `pick`
- The rule of the lab: a precondition of `move`; the `forall` does not rely on one compartment

Version 1: the test set

The round (p02), robots home:

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

- Sealed one move before the lab: correct
- The urgent box is the third delivery of four
- All 31 actions are of `r2`: `solved (proven optimal), cost 31`. The model counts actions

Test	File	Expected, and obtained
Trivial	`p01`	5 steps
Typical	`p02`	31 steps, proven optimal
No plan	`p03`	`unsolvable`: ICU door closed
Cheat	`p04, p05`	`unsolvable`; plan without `seal` rejected

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

Version 1, step L: three ways to write the rule of the lab

A, physics: a conditional effect marks `contaminated`. **B**, a precondition (the domain). **C**, a constraint:

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

All three: 31 steps, p04 unsolvable, the no-seal plan rejected, each in its own words.

Verdict

Size and Guidance do not decide. A out on Fidelity. Keep **B**, which travels to an executor; keep **C** as a test: validate any plan of B with the domain of C. B guards the door, C the room.

	A: conditional effect	B: precondition	C: constraint
Fidelity	samples get ruined	what she said; guards the door	every state
Size	54 ground actions, 40 facts	80 (the `or` splits `move`), 41 facts	54, 3 axioms
Guidance	10 749 states; a late dead end	10 432 states	10 749 states
Portability	`:conditional-effects`	ADL preconditions	`:constraints`: few

Version 2: time and battery

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

Scope adds durations, parallel work, battery, charging.

New in the state	Encoded as
doing nothing	`(idle ?r)`
a charger; free	`(has-charger ?l)`
	`(charger-free ?l)`
meters; speed	`(distance ?f ?t)`
	`(speed ?r)`
charge, capacity	`(battery ?r)`
per meter; per s	`(consumption ?r)`
	`(charge-rate ?r)`

- Token `(idle ?r)` taken at start, returned at end
- The battery of a move taken whole at the start: enough when nobody reads it meanwhile

Version 2: a charge fills the battery

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

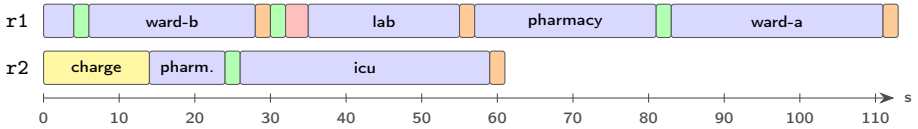
The charger is a **resource lock**: deleted at start, added at end.

Action	Duration	New in time
`move`	distance / speed	battery taken at start; nowhere while moving
`pick`	2 s	stays (`over all`); carried at end
`seal`	3 s	sealed at end
`deliver`	2 s	delivered, free at end
`charge`	missing / rate	charger at start; full at end

Version 2: the round in time

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

- Both robots work from the first second
- r2 charges 30 to 100 in $70/5 = 14$ s, takes the urgent box 43 m, arrives with 57
- r1: meal, sample sealed in ward B, routine box: 98 m on 100
- From "31 actions" to 113 s: the model can say *sooner*
- Blue: moves; green: pick; orange: deliver; red: seal; yellow: charge



Version 2: the cheat test becomes a family

No plan (p03): 20 of battery in ward B, dock 26 m away.

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

Exit 14, not 10: the temporal engines do not prove it; our arithmetic does.

Two cheats by hand: moving while picking; two robots on one charger.

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

Turns (p04): both robots with 10, one delivery each.

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

`r2` fetches the meal while `r1` charges, back the instant the charger is free. We had not thought of it, and it is right.

Version 2, step L: how long is a charge?

Chosen duration (1 s to full):

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

Four charges of 1 s: durations at their bounds. Valid, 6 s shorter. Continuous: the same plan.

	Full	Chosen	Continuous
Fid.	what the card says	partial charges, forbidden	as chosen
Size	82	84	84
Guid.	113.017	107.015, four charges	107.015
Port.	durative + numbers	Kairos, OPTIC, POPF	Kairos, POPF

Verdict: the full charge

"Stays until full" makes the shorter plan impossible: the encoding that lost solutions in Chapter 11 is the faithful one here. Had she said "whenever it wants", the planner would enter the decision (blocks).

Version 3: a calendar and wishes

V2 picks at 24 from a pharmacy that opens at 30, delivers after the round, has only goals. `(is-open ?l)`, `(in-time ?i)`: changed only by the world, at known times.

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

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

Version 3: the tests

The old plan is a test of the new version:

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

Trivial (p01): at the shutter at 10, waits 20 s.

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

No plan, deadline 30 (p03): proved.

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

No plan, opens 30, deadline 60 (p04):

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

Exit 11, ours to complete: $30 + 2 + 33 + 2 = 67 > 60$.
Prefer tests whose answer you can compute.

Version 3: the round, and a limit

The round (p02), 3 s:

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

The plan of the trivial test: one wish of five (r2-home). 25 s: cost 7. Relax: the round *without its calendar* (p07):

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

- Every wish, under 0.1 s: the model of the wishes is right
- Without a calendar: engine M2, almost classical. With timed literals: M3, over starts and ends; preferences multiply it
- A fact about Kairos 1.0.6, not about PDDL. We cannot remove the calendar; we can change what we ask in one call

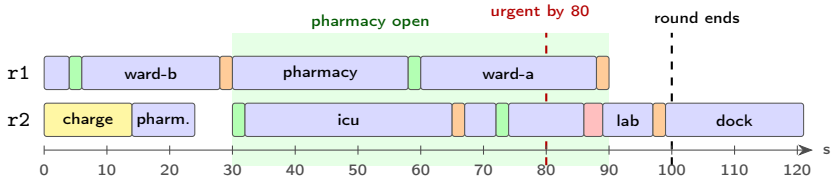
When the planner disappoints

Relax the problem until it does not: the feature whose removal fixes the answer is the one that costs.

Version 3, step L: three ways to plan the wishes

- **A** one problem (p02): cost 12 in 3 s
- **B** one robot at a time; merge; validate against A. r1 first: metric 2; r2 first: metric 1
- **C** a ladder of hard goals: third rung (p08) timeout, though it has a plan

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



The second merged plan: every delivery in time; only r1 does not get home (18 of battery, 26 m away).

Version 3, step L: the comparison

	A: one problem	B: one robot at a time	C: ladder of hard goals
Fidelity	exact: the card, line by line	loses plans where the other robot should choose first; its plans are valid for A	loses the weights: an order only
Size	76 durative actions, 5 preferences	half of it, twice	as A without preferences, several times
Guidance	poor with a calendar: 12 in 3 s, 7 in 25 s	good: under a second per call; 2, then 1	poor: the third rung times out
Clarity	one file is the round	two files and a merge; who goes first is a new decision	many files

Verdict: A is the model, B is how we plan with it

A stays as the specification, and the validator prices any plan against it. B is honest only because of that last step: merged plans can fail where robots meet (the charger). Who goes first: two orders for two robots, 120 for five.

Version 4: the battery belongs to the world

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

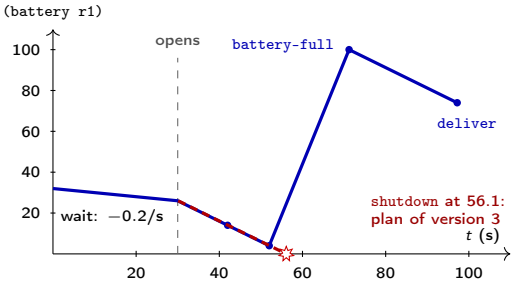
; The charger lets go by itself when the battery is full.
(:event battery-full
  :parameters (?r - robot ?l - location)
  :precondition (and (plugged ?r ?l)
                     (>= (battery ?r) (battery-capacity ?r)))
  :effect (and (not (plugged ?r ?l)) (charger-free ?l)
               (idle ?r)))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

No `unplug`, no duration to compute: "stays until full" finds its natural form.

Change	Caused by
next room	robot: `move`
plugged in	robot: `plug`
drains (0.2/s; +0.8 driving)	world: `drain`, `motor-drain`
fills (5.2/s)	world: `charging`
charger lets go	world: `battery-full`
switches off	world: `shutdown`
pharmacy opens	clock: timed literal

Left out, shaped by the planner: instantaneous handling; **no samples**. *Version 4 is about the battery; the lab is tested in versions 1 to 3.*

Version 4: one robot and a shutter



Cheat test (p04): 32 at the shutter, opens at 30, 28 m to go. V3: wait, pick at 30. Pencil: 30 s cost 6, leaves with 26, needs 28. Solid: pick at 42, plug at 52.

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

The planner: to the dock with the box, charge, deliver with 74. `battery-full` is not in the plan.

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

Idle until 30, a useless walk (fewest actions). Typical: 94; 84.1 by hand. p03: $5/0.2 = 25 < 30$.

The versions as models of the same page

	Version 2: time, battery	Version 3: calendar, wishes	Version 4: PDDL+ battery
Fidelity	closed pharmacy, late deliveries; no wishes	the card; waiting is free	the battery as it is; no wishes, no lab
Size	82 durative actions	76, and 5 preferences	17 for one robot
Guidance	both robots, 113.017, under 1 s	per robot: cost 1, four calls of 3 s	one robot, one goal: under 1 s
Clarity	timetables	the problem reads like the card	plans need a plot
Portability	most temporal planners	timed literals and preferences	Kairos

Verdict: no version replaces the others

V3 plans the round; **V4** audits it: robots still for 8 and 17 s cost 1.6 and 3.4 units; `r2` gets home with less than 7, not 10. Lesson carried back as a margin (10 in reserve). **V2** for a night with everything open. **V1** tests and proves the rules in milliseconds.

From plan to execution: what a plan is for a robot

For the planner a proof; for a robot instructions. Each line needs:

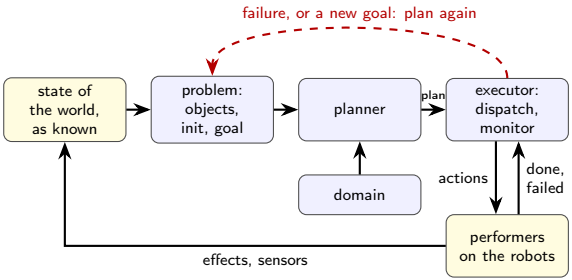
- **Somebody to carry it out**: `(move r1 lobby pharmacy)` is a navigation task; it reports done, failed, progress
- **A reason to start**: the times are one way of meeting **dependencies**
- **A model that is still true**

Dispatching

Sending each action to its performer at the right moment. **Time-triggered**: the plan as a timetable. **Event-triggered**: start when what it depends on ended and its conditions hold. Simple temporal networks keep the dependencies.

What happens	In the round	A reasonable reaction
An action takes longer	a move of 12 s takes 15	nothing while deadlines hold
An action fails	the box slips at the pick	try again; then plan again
A condition breaks while it runs	the pharmacy closes early	stop, plan again
The state differs	the charger gave 92, not 100	check what remains
The task changes	a new urgent request at 30	plan again with the new goal

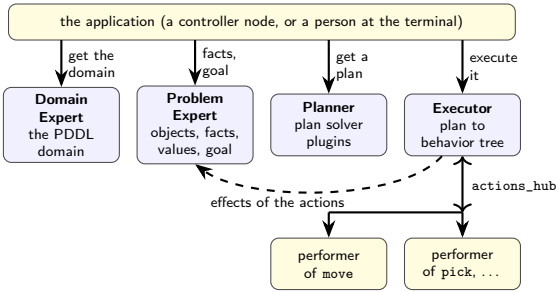
Planning again



Cheap because of the split of Chapter 1: the domain is fixed, the problem is new.

- **The state in the middle must be a state of the model**: `r1` carrying `meds2`, ward A blocked. No `drop`: a robot lost for the day. Keep **recovery actions**
- **Time has passed**: shift the calendar; the world ages while the planner thinks
- **Something is always running**: wait, cancel, or plan from where it ends

PlanSys2: four nodes



The ROS 2 Planning System, as in its sources at 4.0.0 (the fleet was not run on it). Four lifecycle nodes:

- **Domain Expert**: `model_file`; three checks, the last with the checker of Kairos as a library
- **Problem Expert**: the system's knowledge of the world; writes it as a PDDL problem
- **Planner**: plan solver plugins, Kairos by default, POPF too; `plan_solver_timeout` 15 s
- **Executor**: the ROS 2 action `execute_plan`

A plan becomes a behavior tree

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

The durative action of Chapter 10, executed: three places of a condition and two of an effect are five nodes. A condition in the wrong place is checked at the wrong moment.

- Default builder `STNBTBuilder`: a tree from a simple temporal network
- A failure cancels what runs: `FAILURE`. Replanning belongs to the application
- **Action performers**: `ActionExecutorClient`, `do_work`; `specialized_arguments`: each robot its own
- `actions_hub`: request, answer, confirm/reject, feedback, finish, cancel

What this asks of the model

Actions are what performers can do. `(over all (is-open ?l))` is checked while the arm moves: somebody must report it. The full charge is one action. The state is facts: a robot holding an undeliverable box had no action.

What PDDL leaves out

Determinism

An action has one outcome; a plan is one sequence, not a strategy. Practice: plan with the most likely outcome and a margin, watch, plan again. Fails when one must *prepare*: no deterministic plan takes a second bag in case the first tears.

Full knowledge

The closed world. **Sensing** actions change what the robot knows: plans branch, **partial observability**. Our models hide it in the performer.

Out for humbler reasons: geometry, other agents, learned quantities, new objects.

What is left out does not disappear

It lands in a performer, in the monitor or in the margin of a number. At the end of a project read *Scope* again: for each item under "Out", who takes care of it now?

Common mistakes of the case

Writing the last version first

A `timeout` then says nothing: a wrong rule, a wrong number, or just hard? *Fix*: versions

Taking the first answer with wishes

Cost 12, exit 0, one wish of five. *Fix*: know a good cost (at most 2) before running

Reading a timeout as "no plan"

`p08` has a plan. 10, 14, 11 are three answers. *Fix*: a hand plan or a computation

A valid plan with no slack

The sample delivered at 99.011, deadline 100. *Fix*: margins in the model (95 for 100); smallest slack as a test

A model that cannot start from the middle

No `drop`: a carrying robot with a closed destination is lost. *Fix*: list interruptible states, add recovery actions with a cost

The MODEL method: one story, four models

What the case adds

M	What did I ask the author? Which versions, in which order?
O, D	What is added? "Almost everything": too big a jump
E	Old plans validated by the new version? No-plan tests: proof, give up, timeout?
L	With my planner and my time limit? Which sentence would change the verdict?
After	Who performs each action? Smallest slack? A state it cannot start from?

- **The story decides**: "it must not happen", "stays until full", "a little". Fidelity is fidelity *to a card*
- **The planner**: twice a correct model got a poor answer. *Relax* to find the cost; keep the full model as *specification*; plan an easier question
- **The validator** is the fixed point: constraints, old plans, merged plans, continuous time

1 The MODEL method, complete

2 Case study: a robot mission from story to execution

3 Summary

What to take away

Chapter 19

- Five checklists hold every question; step M decides most
- The decision guide: need → feature → cost; stop at the lowest level that does not lie
- Forty patterns, each with a rival; anti-patterns by symptom
- `unsolvable` is a proof; hand plan and validator tell model from search
- Versions, card first, old tests kept; documentation in three places

Chapter 20

- One page, four tested models, each for a question
- 31 actions → 113.017 s → metric 1 → an audit of the battery
- The planner is part of the decision; the validator is the judge
- A plan is not yet behavior: performers, dependencies, monitor, replanning

The solver is somebody else's work and gets better every year. The model is yours.

Exercises to try

Chapter 19 (projects: card, tables, tests, comparison)

- 19.2.1 The library (classical, a basket of three)
- 19.3.1 The water truck: which questions were "no"?
- 19.3.3 The turnaround: makespan by hand first
- 19.3.6 The canal lock: who causes each change?
- 19.7.1 The laundry, with coins

Chapter 20 (three seconds per temporal problem)

- 20.3.2 Where the precondition and the constraint disagree
- 20.4.2 A charge in blocks of 5 s
- 20.5.3 The meal is worth 6
- 20.7.1 Replanning at 30 with `sample2`
- 20.10.1 The warehouse with two forklifts