Planning and models

Planning and PDDL, Part I (Chapters 1–3)

Francisco Martín Rico

Intelligent Robotics Lab, URJC



Outline

1. Planning: deciding before acting
2. A first model: STRIPS
3. What a planner does with your model
4. Summary

1 Planning: deciding before acting

2 A first model: STRIPS

3 What a planner does with your model

4 Summary

Deciding before acting

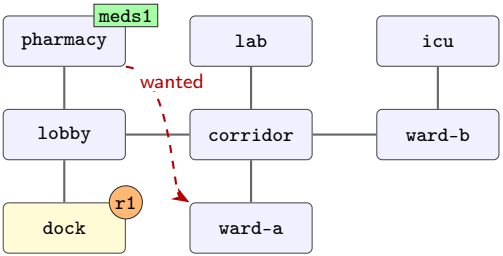
- Some decisions need no look ahead: an obstacle half a meter ahead, the robot stops
- Others depend on what comes later: to bring medicines to ward A, the courier must first move *away* from it, to the pharmacy
- **Planning**: given a situation, the things that can be done and a goal, find which actions to carry out, and in which order, to reach the goal
- The result is a **plan**, found *before* acting, on a description of the world

What makes a problem a planning problem

- The answer is a course of action, not a number or a class
- Actions interact: one makes another possible (pick only in the pharmacy) or impossible (holding a box, no second pick)
- Far too many sequences to try them all blindly

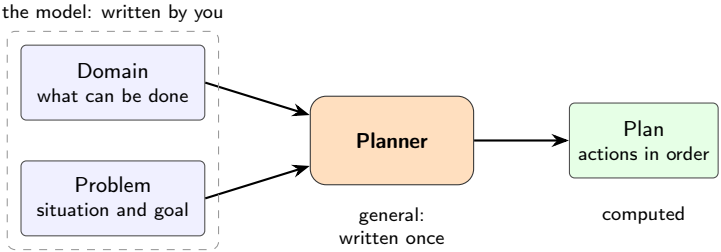
The courier world: the example of the whole course

- A hospital floor with **eight places**
- The ICU is reached only through ward B; every connection works both ways
- Robots `r1`, `r2`; items `meds1`, `meds2`, `sample1`, `meal1`
- First version: **move**, **pick**, **drop**; one item at a time
- Small on purpose, and room to grow: doors, battery, durations, opening hours...



The arrow is the wish, not the way: the way follows the gray connections.

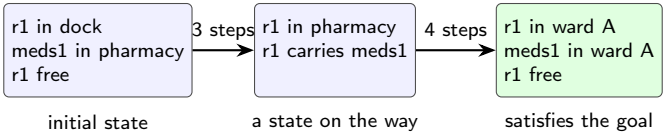
A general solver plus a model



- **Model-based autonomy**. **Model**: what exists, what can be true, what can be done, what is wanted; never *how*. **Planner**: knows only how to search
- To change the behavior, change the model: meds in the lab, a closed corridor, a new wing
- A planner is only as good as its model: "ten boxes per robot" gives plans that pile ten boxes on one tray

Three ingredients: states, actions, goal

- **State**: the world at one moment, all that matters for deciding and nothing else; the first one is the **initial state**
- **Action**: a **precondition** (when it can be done) and **effects** (what it changes)
- **Goal**: a condition on states; "meds1 in ward A" describes many states



States are nodes, actions edges, a plan a path. But **nobody draws the graph**: the model describes it in one page, and the planner builds the part it needs.

The classical assumptions

Finite in the lobby or not, never "2.37 m into it"

Deterministic one result per action

Fully known the initial state, completely

Static only the agent changes the world

Instantaneous actions have no duration

Each is false in a hospital; each makes planning much easier. A plan is then a plain sequence, guaranteed to work.

Assumption	Lifted by
Finite	numbers (Part III)
Instantaneous	durative actions (Part IV)
Static	timed facts (Ch. 13), processes and events (Part VI)
Deterministic	not in PDDL: execute, observe, replan (Ch. 20)
Fully known	not in PDDL (Ch. 20)

A wrong model that is useful

Ask of a model not "is it true?" but "is it good enough for the decisions we need from it?"

Planning and its alternatives

Approach	You provide	A new situation needs	Weak point
Hand-written behavior	The decisions	New code, if it was not foreseen	Variety
Hand-coded search	States, successors, heuristic, in code	New data; new code if the rules change	Reuse
Learning	Examples or a reward	More data, if it is far from what was seen	Guarantees
Planning	A model	A new problem file	The model must exist

- A planner is hand-coded search taken one step further: the state space is *declared*, and search and heuristics are reused for every problem
- Planning fits clear rules, many situations, changing goals
- In a robot they coexist: learned perception, hand-written control, path planning, and **task planning** for *what* to do and in which order

Where planning is used

Space

Remote Agent controlled Deep Space 1 in May 1999. A planning tool helped prepare the days of the Mars rovers Spirit and Opportunity (2004).

Logistics

Fleets, containers, cranes, factories. The *logistics* domain has been a standard test since the first competition (Ch. 4).

Robotics

Service robots with changing goals. PlanSys2 connects PDDL planners with ROS 2 robots (Ch. 20).

STRIPS and a robot called Shakey

In 1971 Fikes and Nilsson (Stanford Research Institute) published STRIPS, the planner of Shakey. Each action: the facts it needs, the facts it makes true, the facts it makes false. The planner was soon surpassed; that way of describing actions is still the core of PDDL.

Domain and problem

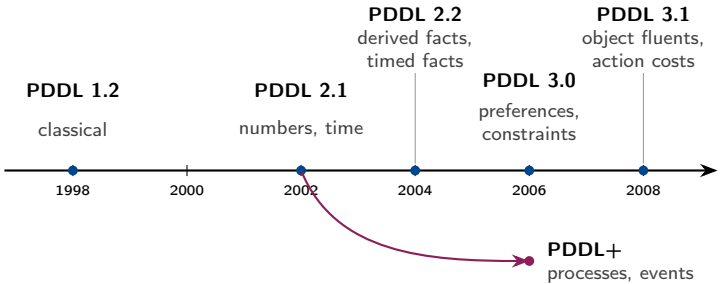
- **Domain**: a kind of world. What can be true, what can be done. No particular robot or room
- **Problem**: one situation and one request. Objects, map, start, goal
- **One domain, many problems**: every request of the day is a new problem; even the map belongs to the problem

Only for its flavor: the action `move` (Ch. 2 explains every symbol)

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

A name, parameters, a condition, an effect. Words with ? are variables: any robot, any pair of connected places.

PDDL and its versions

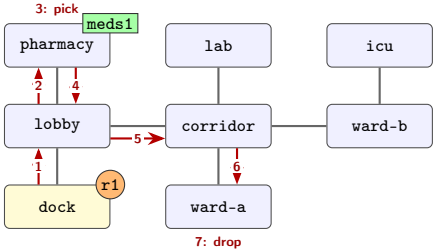


- 1998: McDermott and a committee, for the first planning competition (IPC)
- Each version adds what the classical assumptions leave out: our route
- A model declares its **requirements**; each planner accepts some: hence *portability*

Meeting Kairos

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

- Domain first, problem second
- One **step** per line: action, then objects in parameter order
- No costs in the model: cost = steps



Steps 1–2 go away from ward A: the box must be picked first. Nobody told the planner about pharmacies.

Validate, another problem, no plan

```
$ kairos validate courier-domain.pddl courier
      -p01.pddl \
    courier-p01.plan
plan valid, makespan 7
```

Validating: simulate the plan on the model, step by step.

Same domain, another day (p02):

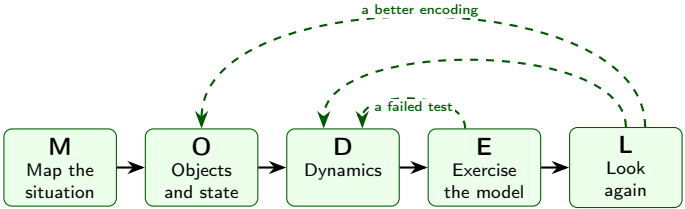
```
$ kairos courier-domain.pddl courier-p02.pddl
; cost = 8
(move r1 ward-a corridor)
...
(drop r1 sample1 lab)
```

p03: ward A closed, its connection not listed.

```
$ kairos courier-domain.pddl courier-p03.pddl
note: the goal needs atoms that no action can reach
(item-at meds1 ward-a) can never be reached:
  drop adds it, but needs (robot-at ?r - object ward-a), which
      never holds
    move adds it, but needs (connected ?from - object ward-a),
      which never holds
      no action adds it, and the initial state does not have it
       [orchestrator.unreachable_goal]
kairos: unsolvable (exhausted)
```

- **Unsolvable** is an answer: no sequence, however long, works
- The note explains it, in the vocabulary of the model

The MODEL method: modeling is choosing



Step	Question	Artifact
M	What is the problem, who acts, what counts as solved, what is left out?	model card
O	Which things exist, and what must the planner know about them?	state table
D	What can be done, when, and what does it change?	action table
E	Does the planner do what the world would allow, and nothing else?	test set
L	Which other encoding would do, and which is better here?	comparison

Step M on the courier world: the model card

Step M: a first model card	
Story	A robot carries items between the rooms of a hospital floor, on request.
Agents	One courier robot. It is the only one who acts.
Success	Each requested item lies in the room where it was asked for.
Scope	In: the rooms and their connections; where the robot and the items are; what the robot carries. Out: distances, time, battery, doors, people, how the arm grasps.
Assumptions	The five classical ones. The robot carries one item at a time.

- No PDDL in step M. Most bad models are bad here: they solve another problem
- Every line of the Scope is a decision: "Out: battery" allows tours that empty it (Ch. 8 changes the card)

The five criteria

Step L judges alternatives on five criteria, always in this order:

1. **Fidelity**: no impossible plans (ten boxes at once) and no lost solutions (never cross the lobby twice)
2. **Size**: how many concrete facts and actions result when the domain is applied to all the objects
3. **Guidance**: how easy the search is: dead ends, equivalent ways, detours, what the planner's estimates can see
4. **Clarity**: easy to read, change and explain, also to yourself in six months
5. **Portability**: which planners accept the features used

Fidelity first

A wrong model gives wrong answers quickly. The rest trade against one another, so step L ends with a decision *and its reason*.

Alternatives compared: where things are

Criterion	A: eight places	B: a thousand cells (40 by 25 m)
Fidelity	Plans say which rooms to cross, not how	Exact routes; wrong as soon as a chair is moved
Size	8 places, 14 possible moves	1000 cells, about 4000 possible moves
Guidance	A delivery is 7 steps	Dozens of steps, a huge number of equivalent routes
Clarity	Names that people use	Cell $(17, 4)$ means nothing to a nurse
Portability	Classical	Classical

Verdict

A is better on every criterion that differs when the planner decides *what* to do; B answers another question, how to cross a room, which navigation software already solves. The right level of detail is the lowest one that still tells apart the decisions you need.

Common mistakes of a first contact

Files in the wrong order

Domain first, problem second. Swapped: `error: expected 'domain', found name 'problem'`. Diagnostics give file, line, column, a code; read the first error first.

"Unsolvable" taken for a failure

Running longer changes nothing: the model has no plan. Either the world has none, or the model is wrong. Start from the note.

Expecting what the model does not say

The robot stays in ward A: the goal did not ask for the dock. What is not in the model does not exist.

Treating a plan as a program

A plan fits one initial state and one goal. The plan of `p01` on `p02`: `unknown object 'meds1', plan invalid`. Reuse the domain, never the plan.

1 Planning: deciding before acting

2 A first model: STRIPS

3 What a planner does with your model

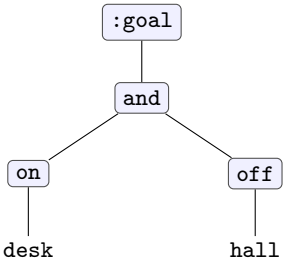
4 Summary

Lists inside lists: the shape of PDDL

Everything is a list in parentheses; its first word says what the list is (**s-expressions**, from Lisp).

```
(:goal (and (on desk) (off hall)))
```

- Layout is free: the planner counts parentheses
- Comments: from ; to the end of the line
- Names: letters, digits, -, _; no case
- Keywords start with :, variables with ?
- Each file is one `(define ...)` list



Each pair of parentheses is a node, named by its first word: how the planner sees it.

Predicates, facts and states

Three lamps, the **objects**: `hall`, `desk`, `porch`.

```
(:predicates
  (on ?l)        ; lamp ?l is lit
  (off ?l))      ; lamp ?l is dark
```

- **Predicate**: a property or relation
- **Fact**: a predicate applied to objects, `(on desk)` (Kairos: *atom*)
- **State**: the set of true facts, {`(on hall)`, `(off desk)`, `(off porch)`}
- **Closed world assumption**: what is not in the state is false

Names mean nothing to the planner

{`(on hall)`, `(off hall)`} is a valid state. Only the model, by its initial state and actions, keeps facts consistent.

Why `off` at all?

In STRIPS a condition asks only for true facts. "The lamp is dark" needs a fact that is true exactly then. Ch. 5 lifts this limit.

Actions: preconditions and effects

```
(define (domain switch)
  (:requirements :strips)

  (:predicates
    (on ?l)       ; lamp ?l is lit
    (off ?l))     ; lamp ?l is dark

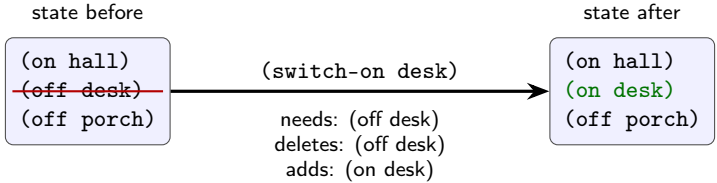
  (:action switch-on
    :parameters (?l)
    :precondition (off ?l)
    :effect (and (on ?l)
                 (not (off ?l))))

  (:action switch-off
    :parameters (?l)
    :precondition (on ?l)
    :effect (and (off ?l)
                 (not (on ?l)))))
```

- `:requirements :strips`: the features used; the basic level, every planner supports it
- `:parameters`: what the action acts on; the action is a pattern
- `:precondition`: facts that must be true, joined with `and`
- `:effect`: plain facts become true (**add**); `(not ...)` become false (**delete**)
- Objects for parameters: a **ground action**, `(switch-on desk)`

Applying an action

1. A ground action is **applicable** if every fact of its precondition is in the state
2. **Applying** it: remove the deletes, put in the adds. *Everything else stays*



- An action says only what changes: the other lamps need no mention
- But it must say *everything* that changes: without its delete, the desk lamp would be on and off at once

The problem: objects, initial state, goal

```
(define (problem switch-p01)
  (:domain switch)
  (:objects hall desk porch)
  (:init (on hall) (off desk) (off porch))
  (:goal (and (on desk) (off hall))))
```

- `:objects`: nothing else exists
- `:init` is **complete**: what it does not list is false
- `:goal` is **partial**: the porch lamp may end in any way

```
$ kairos check switch-domain.pddl switch-p01.pddl
valid, plannable
$ kairos switch-domain.pddl switch-p01.pddl
; cost = 2
(switch-off hall)
(switch-on desk)
```

- *Valid*: correct PDDL, files agree. *Plannable*: Kairos supports it
- **Plan**: each ground action applicable after the previous ones; the final state contains the goal

Step 0: objects and state

Property	About	Kind	Encoded as
It is a robot / a location / an item	an object	static	`(robot ?x)` etc.
One can go from a location to another	two locations	static	`(connected ?from ?to)`
Where a robot is	a robot, a location	dynamic	`(robot-at ?r ?l)`
Where an item lies	an item, a location	dynamic	`(item-at ?i ?l)`
Which item a robot carries	a robot, an item	dynamic	`(carrying ?r ?i)`
The robot carries nothing	a robot	dynamic	`(free ?r)`

- **Static**: no action changes it, data of the problem. **Dynamic**: some action changes it; every action must be checked against it
- **Type predicates**: STRIPS has no kinds; else nothing stops "move the lobby" (Ch. 4 brings types)
- `connected` has a direction: a two-way door is two facts. `free` is redundant for us, needed by STRIPS, kept in step with `carrying`

Step D: dynamics

Action	Parameters	Applies when	Changes
`move`	robot, from, to	the robot is in *from*; *from* is connected to *to*	the robot is in *to*; it is no longer in *from*
`pick`	robot, item, location	robot and item are both in the location; the robot is free	the robot carries the item; the item no longer lies in the location; the robot is no longer free
`drop`	robot, item, location	the robot is in the location and carries the item	the item lies in the location; the robot is free; it no longer carries the item

The routine of step D

- For every action and every dynamic row of the state table: does it make it true, false, or leave it?
- Parameters: every object that the condition and the changes mention (`pick` needs the location to delete `(item-at ?i ?l)`)

The courier domain in STRIPS: predicates and move

```
(define (domain courier)
  (:requirements :strips)

  (:predicates
    (robot ?x)                ; ?x is a robot
    (location ?x)             ; ?x is a room of the floor
    (item ?x)                 ; ?x is something to carry
    (connected ?from ?to)     ; the robot can go from ?from to ?to
    (robot-at ?r ?l)          ; robot ?r is in ?l
    (item-at ?i ?l)           ; item ?i lies in ?l
    (carrying ?r ?i)          ; robot ?r carries item ?i
    (free ?r))                ; robot ?r carries nothing

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

- State table → `:predicates`, one per row
- Action table → actions
- Preconditions: type predicates first, then the conditions of the table
- Effects: the deletes and adds of the column “Changes”

The courier domain in STRIPS: pick and drop

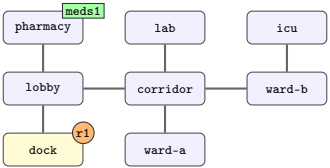
```
(:action pick
  :parameters (?r ?i ?l)
  :precondition (and (robot ?r) (item ?i) (location ?l)
                     (robot-at ?r ?l) (item-at ?i ?l) (free ?r))
  :effect (and (not (item-at ?i ?l)) (not (free ?r))
               (carrying ?r ?i)))

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

- `pick` and `drop` are mirror images: what one adds the other deletes
- Such symmetry is a good sign; its absence is worth a second look
- `free` and `carrying` change together, as step O demanded

A problem for the domain

```
(define (problem courier-p01)
  (:domain courier)
  (:objects r1 meds1
            lobby corridor pharmacy lab ward-a ward-b icu dock)
  (:init
    (robot r1) (item meds1)
    (location lobby) (location corridor) (location pharmacy)
    (location lab) (location ward-a) (location ward-b)
    (location icu) (location dock)
    ; the map: every connection in both directions
    (connected lobby corridor) (connected corridor lobby)
    (connected lobby pharmacy) (connected pharmacy lobby)
    (connected lobby dock) (connected dock lobby)
    (connected corridor lab) (connected lab corridor)
    (connected corridor ward-a) (connected ward-a corridor)
    (connected corridor ward-b) (connected ward-b corridor)
    (connected ward-b icu) (connected icu ward-b)
    ; where things are
    (robot-at r1 dock) (free r1)
    (item-at meds1 pharmacy))
  (:goal (item-at meds1 ward-a)))
```



- 10 objects, 27 facts in `:init`
- 24 static: 10 kinds, 14 for the map (7 connections, both directions)
- Only 3 dynamic: what a plan changes
- Goal: one fact; the robot may end anywhere

Step E: check, solve, read

```
$ kairos check courier-domain.pddl courier-
      p01.pddl
valid, plannable
```

Then solve (the 7-step plan we know) and read it on the state: could the real robot do each step?

After	Dynamic facts of the state
(start)	`(robot-at r1 dock) (free r1) (item-at meds1 pharmacy)`
`(move r1 dock lobby)`	`(robot-at r1 lobby) (free r1) (item-at meds1 pharmacy)`
`(move r1 lobby pharmacy)`	`(robot-at r1 pharmacy) (free r1) (item-at meds1 pharmacy)`
`(pick r1 meds1 pharmacy)`	`(robot-at r1 pharmacy) (carrying r1 meds1)`
`(move r1 pharmacy lobby)`	`(robot-at r1 lobby) (carrying r1 meds1)`
`(move r1 lobby corridor)`	`(robot-at r1 corridor) (carrying r1 meds1)`
`(move r1 corridor ward-a)`	`(robot-at r1 ward-a) (carrying r1 meds1)`
`(drop r1 meds1 ward-a)`	`(robot-at r1 ward-a) (free r1) (item-at meds1 ward-a)`

- A carried item is **nowhere**: no `item-at` between pick and drop, so nothing can pick it from the floor
- `kairos validate` does this reading on any plan file

The test set: trivial and typical

Test 1, trivial (p02): plan known before running

```
    ; where things are
    (robot-at r1 pharmacy) (free r1)
    (item-at meds1 pharmacy))
  (:goal (item-at meds1 lobby)))
```

```
$ kairos courier-domain.pddl courier-p02.pddl
; cost = 3
(pick r1 meds1 pharmacy)
(move r1 pharmacy lobby)
(drop r1 meds1 lobby)
```

Catches gross mistakes, easy to locate.

Test 2, typical (p03): two deliveries, back to the dock

```
    (item-at meds1 pharmacy) (item-at sample1 icu))
  (:goal (and (item-at meds1 ward-a)
              (item-at sample1 lab)
              (robot-at r1 dock))))
```

```
$ kairos courier-domain.pddl courier-p03.pddl
; cost = 18
(move r1 dock lobby)
...
(move r1 lobby dock)
```

Steps 1–7 meds, 8–15 sample, 16–18 home. Any step a person would not take?

Test 3: a problem that must have no plan

p04: the ICU is closed

```
    (connected corridor ward-b) (connected ward-b corridor)
    ; no (connected ward-b icu) and no (connected icu ward-b)
    ; where things are
    (robot-at r1 dock) (free r1)
    (item-at meds1 pharmacy))
  (:goal (item-at meds1 icu)))
```

- A model can be wrong by allowing too much; a test that expects a plan never notices
- Read the note backwards from the goal: drop needs the robot in the ICU, move needs a connection to it, and `connected` is static

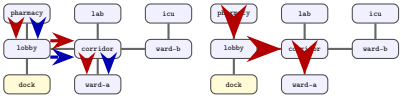
```
$ kairos courier-domain.pddl courier-p04.pddl
note: the goal needs atoms that no action can
       reach
(item-at meds1 icu) can never be reached:
  drop adds it, but needs (robot-at ?r -
      object icu), which never holds
    move adds it, but needs (connected ?from
      - object icu), which never holds
      no action adds it, and the initial
      state does not have it [orchestrator.
      unreachable_goal]
kairos: unsolvable (exhausted)
```

Atom = fact; `- object` is a type (Ch. 4), here "any object".

Test 4: a problem that tempts the planner to cheat

p05: two boxes, both for ward A

```
    ; where things are
    (robot-at r1 dock) (free r1)
    (item-at meds1 pharmacy) (item-at meds2 pharmacy))
  (:goal (and (item-at meds1 ward-a) (item-at meds2 ward-a))))
```



honest: two trips, 15 steps cheating: one trip, 9 steps

- The planner takes the shortest way the model permits
- Kairos: 15 steps, two trips. The model holds

The dishonest plan of 9 steps, typed by hand (the "redundant" `free` keeps the robot honest):

```
$ kairos validate courier-domain.pddl courier-p05.pddl \
    courier-p05-cheat.plan
courier-p05-cheat.plan:5: error: step 4 (pick r1 meds2 pharmacy) at 3.000: precondition false: (free r1) is false [
      validate.precondition_failed]
 5 | (pick r1 meds2 pharmacy)
plan invalid
```

The test set of the courier world

p06 asks for the forbidden state itself: `(and (carrying r1 meds1) (carrying r1 meds2))`

```
$ kairos courier-domain.pddl courier-p06.pddl
note: no plan: every state reachable from the initial one was searched (1220 states) and none satisfies the goal. [
      orchestrator.why]
kairos: unsolvable (exhausted)
```

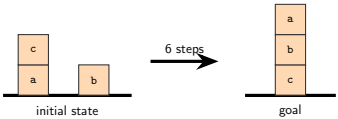
Another kind of unsolvable: each goal fact is reachable, never both together.

Step E: the test set

Test	File	Expected, and obtained
Trivial	`courier-p02`	3 steps: pick, move, drop
Typical	`courier-p01, p03`	7 and 18 steps; every step sensible
No plan	`courier-p04`	unsolvable: the ICU cannot be reached
Cheat	`courier-p05, p06`	15 steps (not 9); unsolvable

The test set stays with the model: it runs again after every change.

A classic: blocks world




```
(:action stack                    ; leave ?x on top of ?y
  :parameters (?x ?y)
  :precondition (and (holding ?x) (clear ?y))
  :effect (and (on ?x ?y) (clear ?x) (hand-empty)
               (not (holding ?x)) (not (clear ?y)))))
```

- Also `pick-up`, `put-down`, `unstack`; `(hand-empty)` has no parameters; `clear` is about absence
- The **Sussman anomaly**: the parts of the goal must be interleaved

```
$ kairos blocks-domain.pddl blocks-p01.pddl
; cost = 6
(unstack c a)
(put-down c)
(pick-up b)
(stack b c)
(pick-up a)
(stack a b)
```

`(and (on a b) (on b a))`: unsolvable, 40 states searched. No law is written: the actions never produce such a state.

The method after the first full pass

- **Static vs dynamic**: dynamic = appears in some effect. Courier: `robot-at`, `item-at`, `carrying`, `free`, as planned
- **The frame rule**: effects say *everything that changes, and nothing that does not*. The second half is free; the first is the modeler's duty
- Gripper (two grippers): a capacity is modeled by objects that hold things; the planner fills both

Questions this chapter adds

M	Can I write the card before any PDDL? Which rules of the world does it state?
O	Static or dynamic? About absence, so a fact of its own? Overlapping predicates, who keeps them in step?
D	Each action, each dynamic property: true, false, or left? Every new fact replaces one to delete?
E	`check`, solve, `validate` on trivial, typical, no-plan and cheat tests?
L	Which other encoding would do?

Alternatives compared: three STRIPS encodings

Criterion	A: `pick`, `drop`	B: `carry` (robot and item move together)	C: one `at` for robots and items
Fidelity	One item at a time, enforced by `free`. Plans say when to grasp and release	One item at a time, by construction. Grasping is invisible	As A, but only while every action keeps its type predicates
Size	8 predicates, 3 actions; 46 ground actions	6 predicates, 2 actions; 42 ground actions; grows with items times connections	7 predicates, 3 actions; 46 ground actions
Guidance	Plan of 15 steps	Plan of 11 steps; many equivalent relays	As A
Clarity	Reads like the story	Shortest file; plans need interpreting	One name less; kinds harder to see
Portability	`:strips`	`:strips`	`:strips`

- B relays the boxes through the corridor; C without type predicates: the medicines walk to the ward by themselves, `(move meds1 pharmacy lobby)`

Verdict

B when only which items go where matters; C with real types (Ch. 4). We keep A: its plans say everything, and `(carrying ?r ?i)` is needed later (contamination, load on the battery, a cart).

Common mistakes in STRIPS

A forgotten delete effect

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

`check` finds nothing; the cheat test gives 8 steps where 15 are needed: the robot leaves a copy of itself in every room. *Fix*: every add of a one-value property deletes the old value.

A goal already true

`; cost = 0`: know the expected length before running.

A missing precondition

No `(free ?r)` in `pick`: the 9-step cheat. Typical problems give the same plans as the right domain.

A misspelled predicate

`robot-in`: a warning, "never true", so `move` never applies. Read the warnings.

A problem that does not match

Wrong arity, unknown predicate: *the fact is ignored*, so false.

1 Planning: deciding before acting

2 A first model: STRIPS

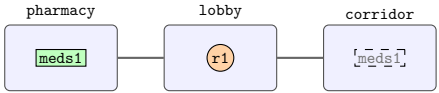
3 What a planner does with your model

4 Summary

Opening the box: a problem small enough to draw

Four deliveries, two robots: the first plan in milliseconds, then 3 seconds and half a million states to prove no better plan exists. A modeler must know why.

```
(define (problem courier-p01)
  (:domain courier)
  (:objects r1 meds1 pharmacy lobby corridor)
  (:init
    (robot r1) (item meds1)
    (location pharmacy) (location lobby) (location corridor)
    (connected pharmacy lobby) (connected lobby pharmacy)
    (connected lobby corridor) (connected corridor lobby)
    (robot-at r1 lobby) (free r1)
    (item-at meds1 pharmacy))
  (:goal (item-at meds1 corridor)))
```



- 16 facts in `:init`; 13 never change (types, map)
- Without them: $s_0 = \{$`(robot-at r1 lobby)`, `(item-at meds1 pharmacy)`, `(free r1)`$\}$

What a model means

- Ground action a: $pre(a)$, $add(a)$, $del(a)$
- **Applicable** in s: $pre(a) \subseteq s$
- **Successor**: $s' = (s \setminus del(a)) \cup add(a)$
- s **satisfies** G: $G \subseteq s$
- **Plan**: $a_1, \ldots, a_n$, each applicable in turn, the last state satisfies G. It may be long, it may walk in circles

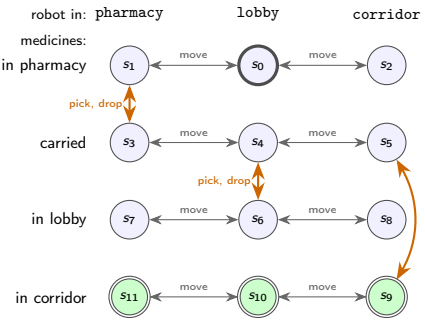
Three sets and one formula: the whole meaning of a STRIPS action.

`kairos validate` checks exactly this. Two ways to fail:

```
$ kairos validate courier-domain.pddl courier-p01.pddl
      courier-p01-shortcut.plan
courier-p01-shortcut.plan:4: error: step 3 (move r1 lobby
      corridor) at 2.000: precondition false: (robot-at
      r1 lobby) is false [validate.precondition_failed]
 4 | (move r1 lobby corridor)
plan invalid
```

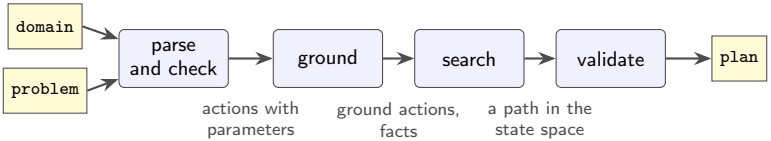
```
$ kairos validate courier-domain.pddl courier-p01.pddl
      courier-p01-halfway.plan
courier-p01-halfway.plan: error: the goal does not hold
      at the end of the plan: (item-at meds1 corridor)
      is false [validate.goal_not_reached]
plan invalid
```

The state space



- Apply every applicable action, again and again: the **reachable states** and their arrows form the **state space**
- Robot: 3 rooms; medicines: 3 rooms or carried. $3 \times 4 = 12$ states
- A plan is a path from s_0 to a goal state: here $s_0, s_1, s_3, s_4, s_5, s_9$
- Several goal states; nothing irreversible here
- The space is not in the files: it is *implied*, and the planner builds the part it needs

Grounding: from variables to objects



Counting by hand, tiny problem (5 objects):

- Every substitution: $3 \times 5^3 = 375$
- Static facts prune: `move` needs a robot and a `connected` fact: $1 \times 4 = 4$
- `pick`: $1 \times 1 \times 3 = 3$; `drop`: 3
- **10 ground actions**; 8 atoms that can change

- Optimistic analysis: a fact, once true, stays true. Never drops a needed action
- Whole floor (`p02`): $14 + 8 + 8 = 30$ actions, 18 atoms
- Towers of Hanoi, 3 discs: $25 + 16 + 9 = 50$, including `(move d1 peg2 peg2)`, never applicable

What the planner says

```
$ kairos courier-domain.pddl courier-p01.pddl -v --threads 1
...
[   0.001s] [ground] grounded in 0.1 ms: 8 atoms, 0 numeric fluents, 10 actions, 0 durative actions, 0 axioms (50
      Datalog facts, 17 rules, 0 instances pruned)
[   0.001s] [sas] SAS+ in 0.1 ms: 3 variables (2 from 3 mutex groups, 1 binary), 3 invariants from 13 candidates, 0
      axiom layers
...
kairos: solved (proven optimal), cost 5
```

```
$ cat out/ground.txt
...
(:action (move r1 lobby corridor)
  :precondition (robot-at r1 lobby)
  :effect
    (not (robot-at r1 lobby))
    (robot-at r1 corridor))
...
```

- 4 of 5 preconditions gone: **static** facts are checked once, in grounding
- **Mutex groups**: the robot is in one room, so one variable with three values: 3 variables

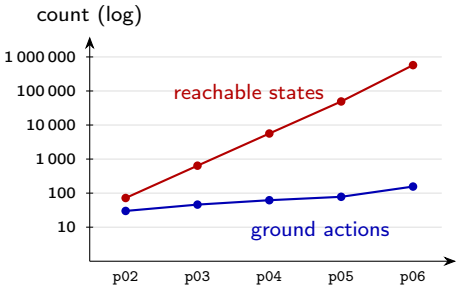
Why the space explodes

One robot, n items, 8 rooms:

$$\text{states} = 8 \cdot \left(8^n + n \cdot 8^{n-1}\right)$$

Problem	Atoms	Actions	States
p02: 1 item	18	30	72
p03: 2 items	27	46	640
p04: 3 items	36	62	5 632
p05: 4 items	45	78	49 152
p06: 4 items, 2 robots	58	156	573 440

Puzzle: 181 440 reachable boards. Hanoi: 3^n states.



The model grows by addition; the state space by multiplication. Deciding whether a STRIPS problem has a plan is **PSPACE-complete** (Bylander, 1994): no planner is fast on every model.

Forward search

Keep an **open list** and the set of states met. Repeat:

1. Take a state from the open list; empty: no plan
2. It satisfies the goal: the path to it is a plan
3. **Expand** it: applicable actions, successors
4. Add the successors never met before

Everything is in step 1: *which* state to take. Work is measured in states **expanded** and **evaluated**.

Searching blind: breadth-first

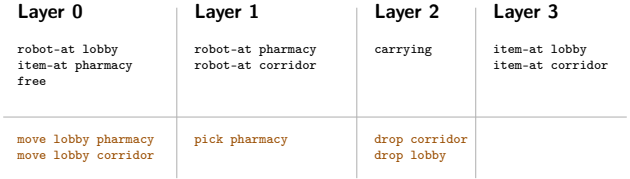
Take the oldest state. On the tiny problem: 7 expansions, a shortest plan of 5. It spreads like water.

The price

On `courier-p05` (shortest plan 28 actions) it would go through most of the 49 152 states. A search that knows nothing about the goal pays for the whole explosion.

Heuristics: ignoring the deletes

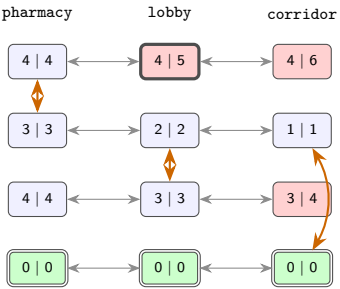
A **heuristic** h estimates the actions still needed. **Greedy best-first**: take the state with the smallest h. The estimate comes from a **relaxation**: ignore every delete effect.




```
(move r1 lobby pharmacy)
(pick r1 meds1 pharmacy)
(move r1 lobby corridor)     ; from the lobby: it never left
(drop r1 meds1 corridor)
```

Relaxed plan, backward from the goal: $h(s_0) = 4$, true distance 5. It forgets the walk back to the lobby. (HSP, FF; Kairos uses LAMA's FF and landmarks.)

The heuristic over the whole space



h | true distance. Exact in 9 states; too low in 3 (red). Along the plan: 4, 4, 3, 2, 1, 0, downhill. Two successors with 4: a **plateau**.

```
$ kairos courier-domain.pddl courier-p02.pddl --debug=search --
      threads 1 --no-anytime
...
[   0.001s] [search] initial h: ff 6.000000, lmcount 6.000000
[   0.001s] [search] new best h 5 (1 expanded, 2 evaluated)
[   0.001s] [search] new best h 4 (4 expanded, 5 evaluated)
[   0.001s] [search] new best h 3 (5 expanded, 6 evaluated)
[   0.001s] [search] new best h 2 (6 expanded, 7 evaluated)
[   0.001s] [search] new best h 1 (7 expanded, 8 evaluated)
[   0.001s] [search] new best h 0 (8 expanded, 9 evaluated)
[   0.001s] [search] plan 0 with 7 actions, cost 7
...
```

Whole floor: 8 expansions of 72 states. `courier-p05`: first plan after 79 of 49 152. That is why planners work.

When the relaxation sees little

- Good when the difficulty is *getting* things; poor when things *get in the way*, because getting in the way is done by deletes
- Towers of Hanoi, 3 discs: the relaxed plan moves each disc once, $h = 3$; the truth is 7. For n discs: n against $2^n - 1$

Discs	States (3^n)	Initial h	First plan	Shortest plan
3	27	3	14 actions, 23 expanded	7
4	81	4	18 actions, 28 expanded	15
6	729	6	77 actions, 284 expanded	63
8	6 561	8	357 actions, 3 170 expanded	255
10	59 049	10	1 322 actions, 22 333 expanded	1 023

Courier: 79 expansions in 49 152 states. Hanoi: 22 333 in 59 049. Same size; what differs is how much the heuristic sees, and that depends on how the problem is modeled.

A plan, a better plan, the best plan

Satisficing: a plan, fast. **Optimal**: with a guarantee. Kairos is **anytime**: the first plan at once, better ones later; the last printed is the best.

```
$ kairos puzzle-domain.pddl puzzle-p02.pddl --threads 1 -v
...
[   0.006s] [search] plan of cost 95 with 95 actions (440
      expanded, 441 evaluated)
[   0.010s] [search] plan of cost 51 with 51 actions (712
      expanded, 900 evaluated)
[   0.016s] [search] plan of cost 39 with 39 actions (1055
      expanded, 1477 evaluated)
[   0.050s] [search] plan of cost 31 with 31 actions (3239
      expanded, 5084 evaluated)
[   1.485s] [orchestrator] solved (proven optimal) in 1484.7
      ms
kairos: solved (proven optimal), cost 31
...
```

The proof met all 181 440 reachable boards.

	First plan		Proof	
Problem	Act.	Exp.	Act.	States
p02	7	8	7	33
p03	13	21	13	397
p04	19	47	19	5 054
p05	28	79	28	48 048
p06	31	393	27	544 039

- Find, improve, prove: each far dearer than the one before
- `--no-anytime`, `--timeout S` (30 by default), `--target-cost C`

No plan, or no plan found

- **Proved by grounding** (p07, no ward B–ICU link): even with deletes ignored the goal is unreachable; the note is a chain of reasons
- **Proved by search** (p08, one-way ICU door): the relaxation sees a plan; the open list empties after 8 states
- Puzzle with 7 and 8 swapped: 181 440 states. No heuristic shortens a proof
- Hanoi, 12 discs: a plan of 4 095 moves exists, but 2 seconds are not enough

```
$ kairos hanoi-domain.pddl hanoi-p07.pddl --threads 1
      --timeout 2
...
kairos: timeout (timeout)
$ echo $?
11
```

Status line	Exit	Meaning
`solved (...)`	0	a plan
`unsolvable (exhausted)`	10	proved: no plan
`timeout (timeout)`	11	nothing known

"Unsolvable" is a proof. "Timeout" only says the search was stopped.

For the modeler: parameters multiply, static preconditions prune

An idle parameter (`courier-extra-domain`)

```
(:action drop
  :parameters (?r ?i ?l ?to)     ; ?to is never used
  :precondition (and (robot ?r) (item ?i) (location ?l)
                     (robot-at ?r ?l) (carrying ?r ?i))
  :effect (and (not (carrying ?r ?i))
               (item-at ?i ?l) (free ?r))))
```

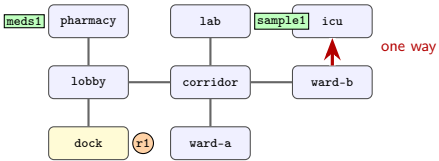
- Three deliveries: 62 ground actions become 326 ($24 \times 12 = 288$ drops)
- Plans still correct, `check` silent; symptom: `(drop r1 meds1 ward-a dock)`

No `connected` **in** `move` (`courier-anywhere`)

```
$ kairos courier-anywhere-domain.pddl courier-
      anywhere-p01.pddl --threads 1 -v
...
[   0.001s] [ground] grounded in 0.1 ms: 18 atoms,
       0 numeric fluents, 80 actions, 0 durative
      actions, 0 axioms (166 Datalog facts, 16
      rules, 0 instances pruned)
...
; cost = 4
(move r1 dock pharmacy)
(pick r1 meds1 pharmacy)
(move r1 pharmacy ward-a)
(drop r1 meds1 ward-a)
```

Moves: 14 become 64; actions 30 become 80. The robot walks through walls: **wrong and larger**.

For the modeler: dead ends and symmetry



- **Dead end**: the goal is unreachable from it; irreversible actions
- p09: the 16-step plan leaves the ICU for the end; the relaxation sees this dead end
- Unseen ones (p08) cost a full search

Symmetry: four identical boxes for ward A (p10)

Problem	First	Best	States (proof)
1 box	7	7	33
2 boxes	15	15	464
3 boxes	25	23	4 670
4 boxes	33	31	43 595
1 box, 3 robots	7	7	607

Same situation for the hospital, different states for the planner. It hurts improving and proving. Create no symmetry without need: no spare robots.

The MODEL method: size and guidance in numbers

- **Step E**: how large is the model, and is that what I expected? Estimate ground actions and states *before* running; read the `[ground]` line after. Far above the estimate: a bug until proved otherwise
- **Step L**: what do the numbers say? *Size*: atoms and ground actions, and how they grow. *Guidance*: expanded to the first plan, first vs best length, states met to prove

Step E on `courier-p05`: estimate, then run

Estimate. Moves $1 \times 14 = 14$; picks $1 \times 4 \times 8 = 32$; drops 32: 78 ground actions. Atoms $8 + 4 \times 8 + 4 + 1 = 45$. States $8 \cdot (8^4 + 4 \cdot 8^3) = 49\,152$: exhausted in well under a second, so we may ask for the proof.

Run. `-v --threads 1`: 45 atoms, 78 actions, first plan of 28 after 79 expansions, `proven optimal` after 48 048 states.

Compare. Everything matches. Had it said 326 actions, look for a stray parameter before any plan.

Alternatives compared: what the planner sees

Three deliveries (meds, sample, meal). B, **loaded moves**: a carried item is still somewhere, so `move-with` also moves its `item-at`.

B's plan: 18 actions, ends *without a drop*: entering the lab holding the sample makes `(item-at sample1 lab)` true. C, **go anywhere**: one go, 56 `reachable` pairs.

	A: base	B: loaded moves	C: go anywhere
Fidelity	plans can be carried out as they are	a carried item counts as delivered	correct only if any route will do
Size	36 atoms, 62 actions	36 atoms, 104 actions; 14 more moves per item	36 atoms, 104 actions; grows with the square of the rooms
Guidance	first: 19 actions, 47 expanded; proof: 5 054 states	first: 18, 34 expanded; proof: 4 567	first: 15, 26 expanded; best: 12; proof: 4 937
Clarity	one meaning for `item-at`	`item-at` means two things	the problem lists what the map implies
Portability	plain STRIPS	plain STRIPS	plain STRIPS

Verdict

Avoid B: larger, no easier, and its extra fact let a wrong plan through. C when navigation finds the route anyway; wrong when the route matters (doors, battery). We keep A.

Common mistakes with the planner

Reading "timeout" as "no plan exists"

Exit code 11: nothing proved. Only `unsolvable` (10) is a proof. Check the exit code; give more time before blaming the model.

Taking the first plan for the best

Puzzle: first plan 95 slides, best 31. Read the *last* plan and the status: `proven optimal` is a guarantee, `timeout` is not.

Waiting for a proof nobody needs

p06: best plan after 10 ms, proof after 3 s. Use `--no-anytime`, `--timeout`, `--target-cost`.

A parameter that nothing constrains

`(drop r1 meds1 ward-a dock)`, 326 actions for 62. Remove it (Ch. 7: the *idle parameter*).

A missing static precondition

`(move r1 dock pharmacy)`, 80 actions for 30: larger *and* wrong. The estimate catches it first.

Judging a model on a tiny problem

One item: a millisecond. Four items, two robots: 544 039 states. Test two or three times larger.

1 Planning: deciding before acting

2 A first model: STRIPS

3 What a planner does with your model

4 Summary

What to take away

- Planning: deciding before acting. A general **planner** plus a **model** (domain + problem); to change the behavior, change the model
- Classical planning: finite, deterministic, fully known, static, instantaneous; the course lifts them one by one, with the versions of **PDDL**
- STRIPS: facts, closed world, actions with precondition, adds and deletes; $s' = (s \setminus del(a)) \cup add(a)$; the **frame rule**
- The **MODEL method**: Map, Objects, Dynamics, Exercise, Look again; card, state table, action table, test set (trivial, typical, no plan, cheat), comparison on fidelity, size, guidance, clarity, portability
- The planner grounds (parameters multiply, static preconditions prune) and searches a space that explodes, guided by a heuristic that ignores deletes
- A plan, the best plan, a proof: very different costs. `unsolvable` is a proof, `timeout` is not

Exercises to try

- **1.1.2** Two deliveries on paper, in both orders: why is one so much better?
- **1.3.1** Which classical assumption breaks each hospital fact?
- **1.7.1–1.7.3** At the keyboard: meds to the ICU, delete (`free r1`), validate a hand-written plan
- **1.8.1** A model card for a robot vacuum cleaner
- **2.4.2** Remove (`off porch`): predict, run, read the note
- **2.7.3** The ferry: steps M to E, from a three-line story
- **3.2.1** Count atoms and ground actions of `courier-p03`; check with `kairos -v`
- **3.5.1** A relaxed plan by hand, from another state
- **3.9.2** Repair encoding B so that a carried item is not delivered