Three Fleet Scenarios for ePlanSys

EPDDL survey missions, grounded with plank, solved with Aletheia, executed as behavior trees

Abstract

A fleet of survey robots shares a depot. The routes out of it may be blocked by debris; only a robot standing at a junction can see down a route; and a robot watching another one look learns *that* it looked, not *what* it saw. Before the fleet commits to a route, every robot has to know which routes are passable. This note records three scenarios built around that situation, at three sizes, together with their formalization in EPDDL, the policies Aletheia returns for them, what those policies cost to find, and how ePlanSys executes them. The scenarios are deliberately the same three mechanisms at every size — drive, look, and say what you saw — so that what grows between them is the search and not the modelling. Across the three, ground actions go from 8 to 48 and expanded nodes from 17 to 5.03 million, while peak resident memory moves from 4.34 MB to 4.85 MB. The two smaller scenarios are regression tests of the whole stack, planner through executor; the third is shipped as an example because a minute of search does not belong in a suite that runs on every push.

Contents

1 The situation

The scenarios are built around one situation that is common in fleet robotics and inexpressible in classical planning:

> A corridor may be blocked. Only a robot standing at the junction can see it. The rest of the fleet has to route around it, so the rest of the fleet has to know — and watching a robot look tells you that it looked, not what it saw.

Three properties of that paragraph put it outside PDDL, and each of them is a mechanism the scenarios use.

The initial state is not a state. Whether the corridor is blocked is not something the mission planner knows. A PDDL problem must assert it or deny it; either assertion produces a plan that is correct only if the assertion happens to hold. What is needed instead is an initial *model* in which both possibilities are live.

An action's effect can be on knowledge alone. Looking down the corridor changes nothing about the corridor. Broadcasting what was found changes nothing about the corridor either. Both change what the fleet can act on, and a formalism whose actions only add and delete facts has to encode that as a fiction — a `told` predicate, an `observed` flag — which then has to be maintained by hand and cannot express the difference between seeing something and being told it.

Who observes what is part of the action. A camera on one robot is neither private nor public. The fleet sees the behaviour and not the image, and that distinction is the entire reason the mission needs a radio.

The consequence is that a solution to any of these scenarios is not a sequence of actions. It is a policy: a tree that senses and then branches on what the sensing turned out to observe, with one continuation per outcome and a guarantee on every branch.

1.1 The pipeline

Figure 1 is the route from the two source files to a robot that moves. It is worth having in view because the three scenarios exercise different parts of it: the numbers in §7 are the middle of the diagram, and the claims in §8 are the right-hand end.

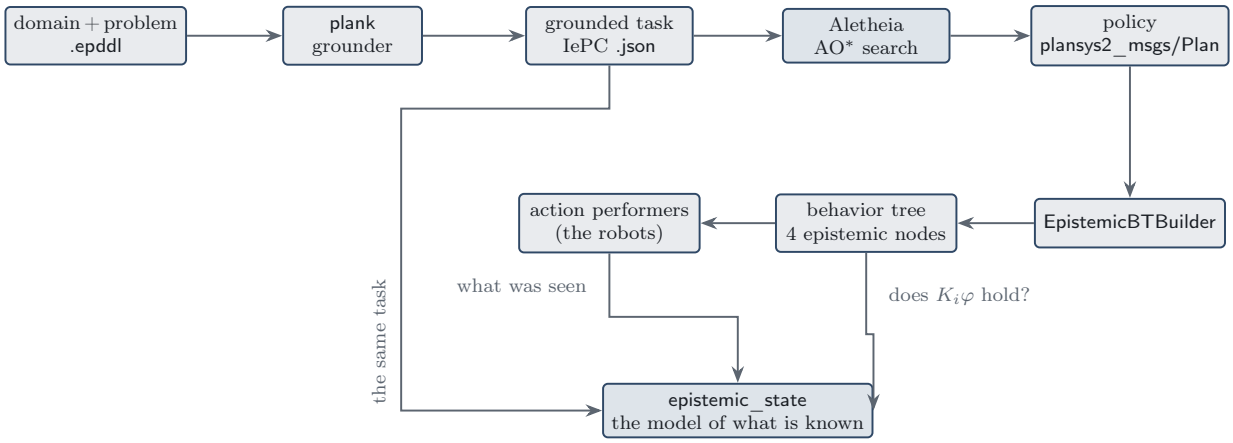


Figure 1: From two source files to a fleet that moves. The grounded task is read twice: once by the planner that searches it, and once by the node that holds the model the executed policy is checked against.

2 The three mechanisms

All three scenarios use the same three actions, and nothing else. This section states them once, semantically; the scenarios then differ only in how many routes and robots they apply to.

2.1 Models, and what an action does to one

An epistemic state is a multi-pointed S5 model $M = \langle W, \{\sim_i\}_{i\in\mathcal{A}}, V, W_d\rangle$: a set of worlds, an equivalence relation per agent, a valuation, and a non-empty $W_d \subseteq W$ of *designated* worlds — the ones still live as candidates for the actual world. Knowledge is truth in every world an agent cannot distinguish from the actual one, and the goals in this note are all of the form

$$Kw_i\varphi \;\equiv\; K_i\varphi \vee K_i\neg\varphi,$$

knowing *whether* rather than knowing *that*. That is not a stylistic choice. A goal of $K_i(\mathit{blocked})$ is unachievable in the worlds where the corridor is clear, so a plan for it would be unexecutable about half the time.

An action is an event model $E = \langle \mathcal{E}, \{\sim_i\}, \mathit{pre}, \mathit{post}\rangle$, and executing it is the product update $M \otimes E$, whose worlds are the pairs (w, e) with $M, w \models \mathit{pre}(e)$, whose valuation is $V(w)$ amended by $\mathit{post}(e)$, and whose relations are

$$(w, e) \sim_i (w', e') \quad \text{iff} \quad w \sim_i w' \ \text{ and } \ e \sim_i e'.$$

Everything the three mechanisms do is a choice of $\mathcal{E}$ and of the per-agent relation over it.

2.2 Drive, look, report

Driving. It is *public ontic*: one event, a precondition on where the robot is not, a postcondition putting it at the junction, and every agent distinguishing nothing — there is nothing to distinguish, since there is only one event. The world changes and the fleet sees it change.

Looking. It is *semi-private sensing*: two events with contradictory preconditions, *blocked* and $\neg$*blocked*, neither with a postcondition. The robot that looks distinguishes them; every other agent does not. The product update therefore splits the model into one successor per event whose precondition some designated world satisfies, and in the successors the looking robot's relation no longer joins a blocked world to a clear one.

Reporting. It is a *public announcement* guarded by the speaker's knowledge: the event's precondition is $K_i(\mathit{blocked})$, so a robot can broadcast only what it actually found. Because knowledge is factive, hearing it settles the matter for every listener. There are two report actions rather than one with a parameter, because a robot can only report what it found, and which one the mission uses is not known when the plan is built.

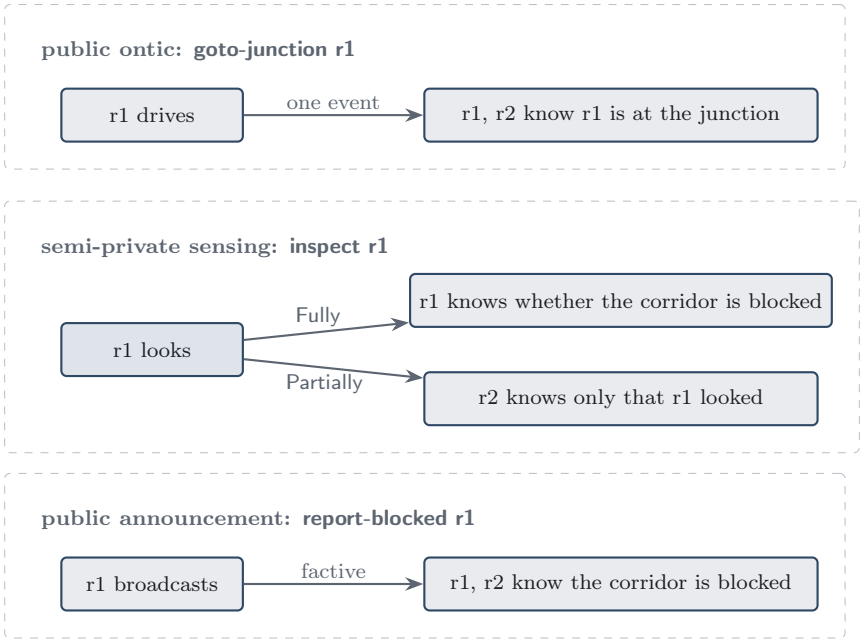


Figure 2: The three mechanisms, by what each does to the fleet's knowledge. Semi-private sensing is the one with no classical counterpart, and the one that makes the mission need a radio: if looking were public no announcement would be needed, and if it were private the others would not know a look had happened.

3 Scenario 1: the corridor

Two robots, one route, and the smallest instance that is recognisably a contingent plan rather than a sequence.

3.1 Mission and formalization

Two predicates carry the whole scenario:

```
(:predicates
  (blocked) ; the corridor is obstructed
  (at-junction ?i - agent)) ; ?i is where the corridor can be seen
```

and the three mechanisms are one action each. Sensing is where the observability conditions do the work:

```
(:event e-inspect-blocked
  :parameters (?i - agent)
  :precondition (and (at-junction ?i) (blocked) (<Kw. ?i> (blocked))))

(:event e-inspect-clear
  :parameters (?i - agent)
  :precondition (and (at-junction ?i) (not (blocked)) (<Kw. ?i> (blocked))))

(:action inspect
  :parameters (?i - agent)
  :action-type (semi-private-sensing (e-inspect-blocked ?i) (e-inspect-clear ?i))
  :observability-conditions (:and (?i Fully) (default Partially)))
```

The $\langle Kw_i \rangle$ conjunct in both preconditions — "i does not yet know whether" — keeps a robot from inspecting twice to no effect, which is a modelling detail with a search consequence: it prunes a whole family of useless successors before the heuristic ever sees them.

The problem file states what is common knowledge at the depot, and, just as importantly, what it leaves open:

```
(:agents r1 r2)
(:init
  (:and
    (:forall (?i - agent) ([C. All] (not (at-junction ?i))))
    (:forall (?i - agent) ([C. All] (<Kw. ?i> (blocked))))))
(:goal
  (and ([Kw. r1] (blocked)) ([Kw. r2] (blocked))))
```

It never says whether the corridor is blocked. The finitary S5 theory is therefore satisfied by two worlds, and plank designates both.

3.2 What the update does

Figure 3 is the whole scenario in three models. The sensing action removes r1's edge and the announcement removes r2's; the goal is exactly the absence of both.

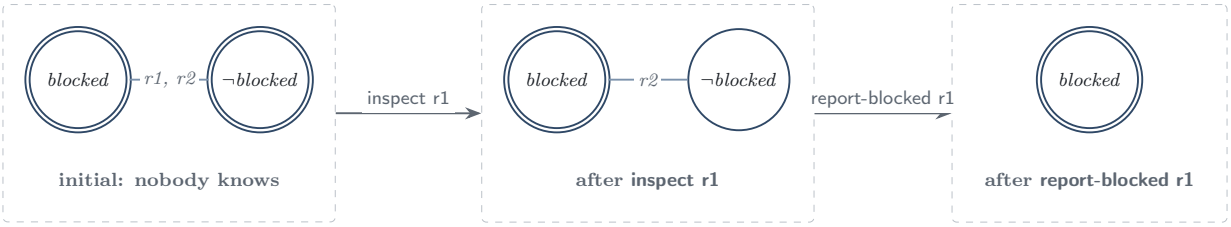


Figure 3: One execution of the corridor policy, as models. A double circle is a designated world; an edge labelled i is agent i being unable to tell the two apart. Sensing splits the model and drops the sensing agent's edge in each half — only the blocked half is drawn — and the announcement drops everyone else's. The mission is over when no edge is left, which is what Kw asks for.

3.3 The policy

AO* returns four nodes at depth three.

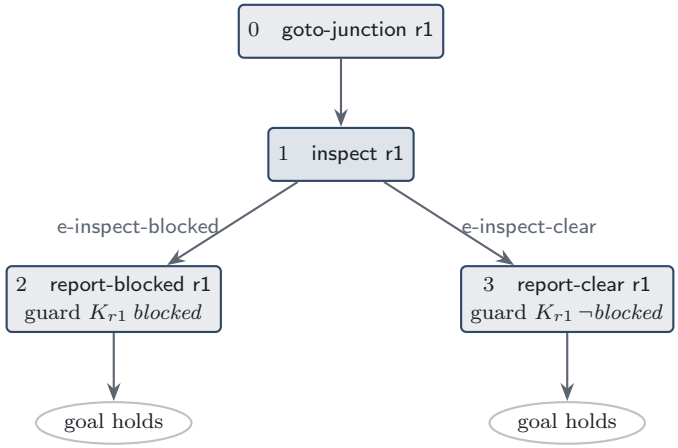


Figure 4: The corridor policy. Both continuations are in the tree; which one runs is decided while the robot is standing at the junction with its camera on. The guards are epistemic preconditions, checked against the model rather than against any predicate.

Two things in that tree are worth reading carefully. The guard on each announcement is not a PDDL precondition and cannot be checked against a problem expert, because no predicate records it; it is what stops the fleet from broadcasting a guess. And the branch is the planner declining to commit — a flattened version of this plan would be correct half the time.

Note also what the search worked out. Both robots could have driven out and looked, at four actions; one robot going, looking and broadcasting costs three. It found the second, which is to say it worked out that talking is cheaper than everybody going to see for themselves.

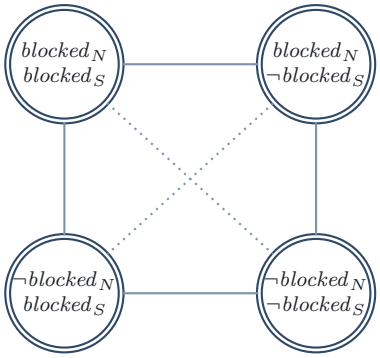
4 Scenario 2: the depot

Three robots, two routes, and one added rule: a robot dispatched down a route stays on it.

```
(:event e-goto-north
  :parameters (?i - agent)
  :precondition (and (not (at-north ?i)) (not (at-south ?i)))
  :effects (at-north ?i))
```

That precondition is what forbids driving from one junction to the other, and it is a depot rule rather than a modelling shortcut: it is what makes this a fleet problem instead of one robot's tour. No robot can survey both routes, so the work has to be divided, and whoever stays behind has to be told.

Two independent unknowns give four worlds, all designated (Figure 5), and a goal of six conjuncts — three robots × two routes — of which no robot can satisfy more than two by itself.



Every robot's relation is the whole square: at the depot nobody knows anything about either route, and each robot knows that of the others. Sensing the north route cuts the square into its two columns; sensing the south route then cuts a column into its two cells. Two sensings are therefore necessary as well as sufficient, whoever performs them — which is what makes the division of labour a search problem rather than a modelling one.

Figure 5: The four initial worlds of the depot scenario, and what the two sensings do to them.

4.1 The policy

AO* solves it at depth six and returns twelve nodes (Figure 6). Read down the left edge: r1 drives north, looks, and broadcasts; then r2 drives south, looks, and broadcasts. The whole of that second half appears twice, once inside each outcome of the first look — not because the plan repeats itself, but because those are different plans. They run against different epistemic states, and the announcement that ends each one differs.

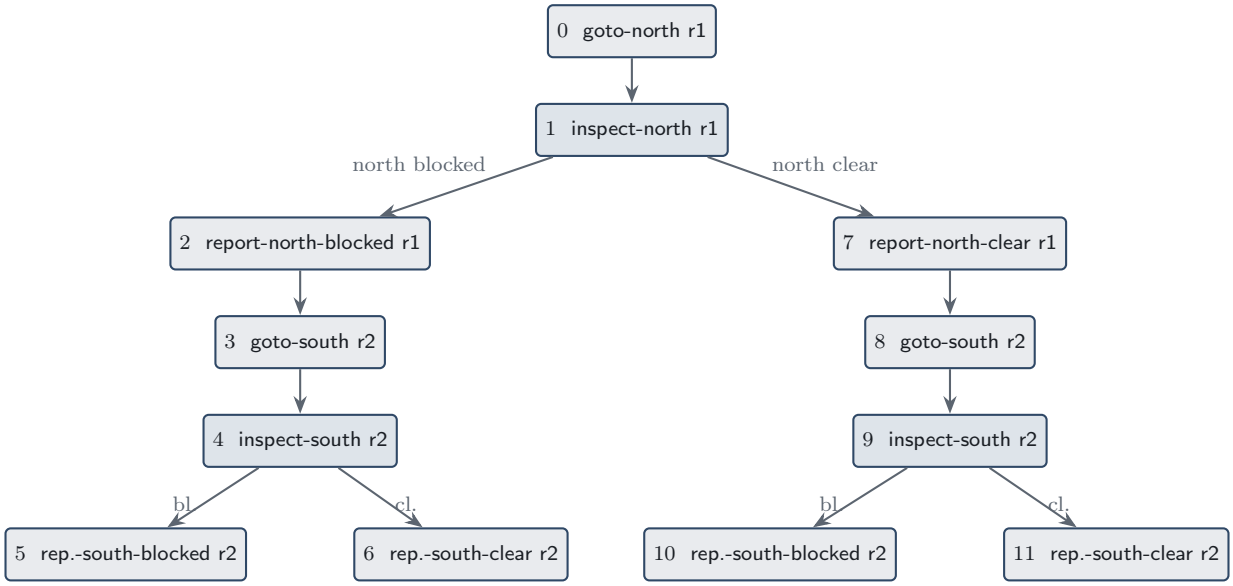


Figure 6: The depot policy: twelve nodes, four leaves, a branch inside a branch. r3 appears nowhere in it and finishes the mission knowing the state of both routes, purely by listening.

The division of labour is the planner's choice and not the domain's. One robot walking both routes is impossible here, but r1 and r3 splitting them would cost exactly what r1 and r2 splitting them costs; the search has to consider both allocations and settles on one. Six actions on every path, and never a robot going to see something it has already been told.

r3 finishing the mission informed without leaving the depot is the entire argument for modelling communication epistemically rather than as an ontic effect on a `told` predicate. Nothing in the domain says that hearing an announcement informs a listener; it follows from the announcement being public and knowledge being factive.

5 Scenario 3: the survey

Four robots, three routes, and nothing new. It is written to be big rather than clever, so that the comparison across the three sizes says something about the search and nothing about the

modelling: 15 atoms, 48 ground actions, eight worlds the fleet cannot tell apart, and a goal of twelve conjuncts.

AO* solves it at depth nine and returns 28 nodes with eight leaves. Each survey trip is three actions — drive, look, broadcast — and the look in the middle of each is where the policy branches; Figure 7 is the resulting skeleton.

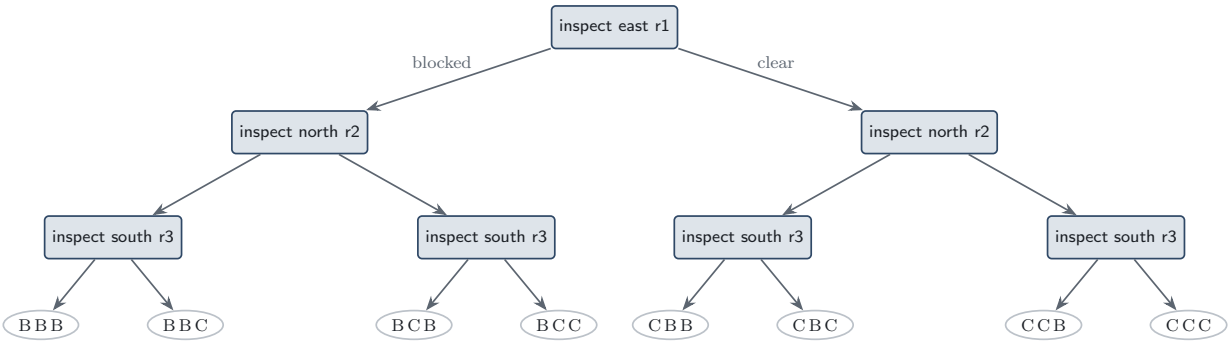


Figure 7: The sensing skeleton of the survey policy. Every edge also carries the drive that precedes the next look and the broadcast that follows the last one; each leaf is one of the eight ways the depot can turn out, written as the three route states in the order the policy senses them — east, north, south — with B for blocked and C for clear.

The order the search settles on is east, then north, then south, with r1, r2 and r3 taking one route each. r4 never acts. Four of the twelve goal conjuncts are about what r4 knows, and it satisfies all four by listening: the depot scenario's point, restated at a size where it is no longer a curiosity of a small instance.

Table 1: The three shipped fleet scenarios, as plank grounds them and as Aletheia solves them. Worlds is $|W|$ of the grounded initial model and all of them are designated, so every world is a live possibility at planning time. Goal conjuncts counts the Kw atoms of the goal. Times are single runs; see §6.

	Corridor	**Depot**	**Survey**
Robots	2	3	4
Routes	1	2	3
Atoms	3	8	15
Ground actions	8	24	48
Worlds (= designated)	2	4	8
Goal conjuncts	2	6	12
Solution depth	3	6	9
Policy leaves	2	4	8
Branches checked	4	12	28
Nodes expanded	17	1 670	5 029 063
Nodes generated	19	2 010	5 746 557
Search time (s)	0.00	0.01	58.82
Peak RSS (MB)	4.34	4.47	4.85

6 Method and environment

Every instance was ground with plank (`plank export -d ... -p ... -l intermediate.epddl`) and solved with the Aletheia binary (`epistemic_planner --task ... --plan ...`). Heuristic and strategy were left to Aletheia's selection policy in all three cases; it chose *knowledge-spread* under the rule `kw-only-goal` and *AO** under `sensing-small-designated` every time, which is the expected reading of a Kw-only goal over a small designated set. Every returned policy was accepted by Aletheia's own validator.

Timings are single runs of `/usr/bin/time` on an otherwise idle machine: Intel Core i7-10870H

at 2.20 GHz, 16 threads, 38 GB RAM, Ubuntu 22.04.5, kernel 6.8.0, GCC 11.4.0, Release build. They are not repeated measurements and should be read as orders of magnitude, not as benchmarks. The sub-10 ms figures in particular are at the resolution floor of the tool.

Versions. ePlanSys `feature/epddl-front-end` at `0050a87`; Aletheia at `d530d25`; plank at `a682480`.

7 Results

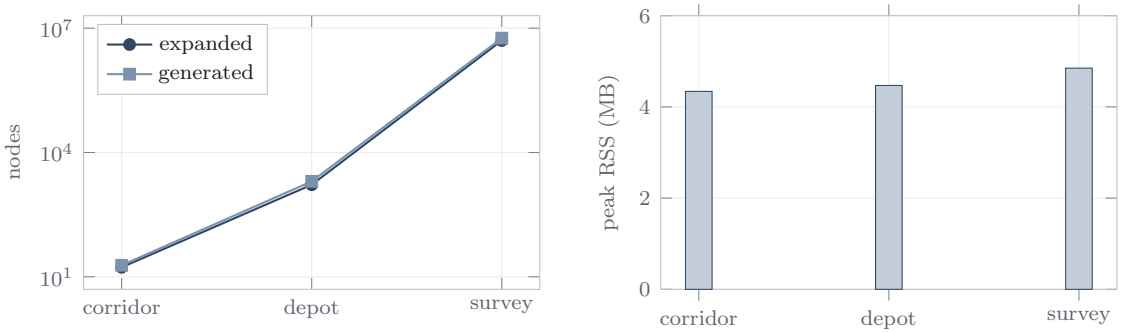


Figure 8: Five orders of magnitude of search on the left, a 12 % change in memory on the right, over the same three instances. Note the log scale on the left and the linear one on the right.

The search explodes and the representation does not. Expanded nodes grow by roughly two orders of magnitude per size step: 17, then 1 670, then 5.03 million, a factor of 98 and then 3011. Peak resident memory over the same range grows by 12 %, from 4.34 MB to 4.85 MB. This is the property Aletheia's design targets: models are three flat bit matrices and closed lists store 128-bit fingerprints rather than states, so the cost of a large search is time rather than memory. A planner storing states outright would not have finished the third instance on this machine.

Depth is what hurts. The three instances differ in every parameter at once, but the one that tracks the cost is solution depth, which AO* reaches by iterative deepening: 3, 6, 9. Each route added contributes three actions to the depth — a drive, a look and a broadcast — and doubles the number of worlds, and the product of those two is what the 3011× jump is made of.

The policy grows with the branching, not with the search. Four nodes, then twelve, then twenty-eight: 2^{k+1} minus something for the shared prefix, where k is the number of sensings. That is the number of subtrees the executor has to be handed, and it is also the number of distinct futures the fleet is committing to in advance. It grows fast enough to be worth noting that a policy is not a plan you can read at a glance, and slowly enough that rendering all of it into one behavior tree remains reasonable at these sizes.

The grounder produces multi-pointed models. Before these scenarios, every task plank produced from the available EPDDL instances was single-pointed, with one designated world, so sensing had exactly one possible outcome and AO* returned a chain. The only branching fixture in the ePlanSys tree was a file edited by hand. All three scenarios here ground to genuinely multi-pointed models, because their `:init` theories leave the corridors open rather than asserting a state for them. They are the first grounded branching instances in that tree, and the first that exercise the executor's branch node on a policy nobody hand-wrote.

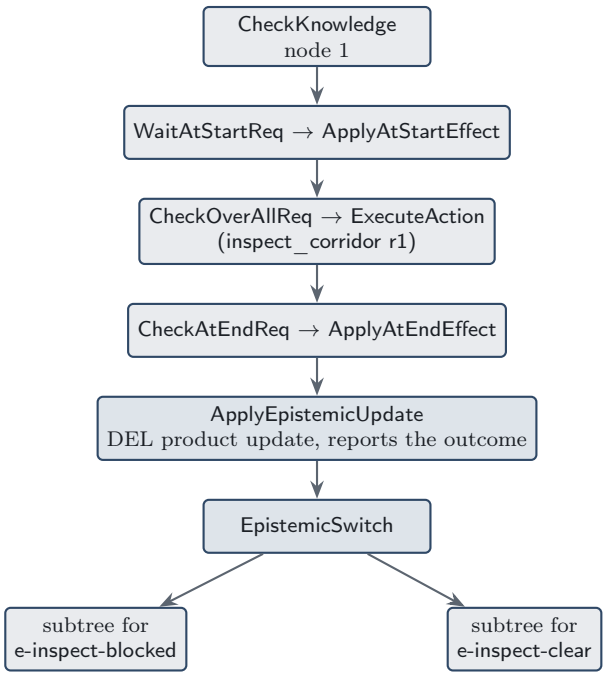
Both routes to the planner agree. ePlanSys reaches Aletheia two ways: `plansys2_epistemic_planner` links the search in, and `plansys2_aletheia_plan_solver` runs the binary as a subprocess and reads its plan file back. On the corridor and depot instances the in-process route reports node counts identical to the standalone binary, 17/19 and 1670/2010, and the subprocess route returns the same policy, translated through the action mapping into PlanSys2 action expressions. The two are the same planner and now have a shared test to say so.

8 Executing the policies

A policy that is only ever solved is half a result. The corridor and depot scenarios are executed end to end in the ePlanSys test suite, over a live ROS graph, with the planner, the executor, the epistemic state and one action performer per action all running as nodes.

8.1 How a policy becomes a behavior tree

`EpistemicBTBuilder` renders each policy node as the PlanSys2 action subtree with three additions, shown in Figure 9: a knowledge guard before the action, the DEL update after it, and — for a node that branches — a switch that runs the continuation planned for what was observed. The whole tree is followed by a goal check, because reaching a leaf of a policy is not the same as having achieved the goal it was built for.



The middle three rows are PlanSys2's own action subtree, unchanged: the same nodes drive the same performers against the same problem expert. What the epistemic builder adds is the guard above them, the update below them, and the switch that follows.

Figure 9: One policy node, as rendered. A node with a single continuation is rendered without a switch, so a classical plan comes out as the flat sequence PlanSys2 would have built.

8.2 Where the observation comes from

The epistemic state can name the outcome of a sensing action by itself only when its model already designates a single world. All three scenarios designate several — which is what makes them worth testing with, and what means the observation has to come from whoever did the sensing. `ApplyEpistemicUpdate` therefore takes it on an `observed` port that the packaged action template leaves unbound, and a deployment binds that port to whatever its performer reports. In the tests the binding is a small behavior tree node reading the corridor states from a topic; on a robot it would be the perception stack.

This is the one piece of wiring the stack does not supply, and it is deliberate: what a robot saw is domain-specific, and PlanSys2 carries nothing back from an action performer.

8.3 What the end-to-end tests assert

Each scenario is run twice with the world in different states, and the two runs must diverge. The corridor test asserts that one run executes `report_clear` and the other `report_blocked`, and that neither runs the branch it did not take. The depot test does the same for two routes in opposite states, which is the case that exercises a switch nested inside another switch's continuation — and, incidentally, the case a flattened plan gets wrong three times in four. The epistemic state's own log is the clearest record of a run:

```
applied goto-north_r1: 4 worlds, 4 designated
applied inspect-north_r1 -> e-inspect-north-blocked: 4 worlds, 2 designated
applied report-north-blocked_r1: 4 worlds, 2 designated
applied goto-south_r2: 4 worlds, 2 designated
applied inspect-south_r2 -> e-inspect-south-clear: 4 worlds, 1 designated
applied report-south-clear_r2: 4 worlds, 1 designated
[goal] (and (Kw r1 blocked-north) ... (Kw r3 blocked-south)) holds
```

Four candidate worlds at the start, two after the first sensing, one after the second, and the goal checked against the state that actually resulted rather than the one planning assumed.

9 Reproducing

```
# ground
plank export -d robot-fleet-depot-domain.epddl \
            -p robot-fleet-depot-problem.epddl \
            -l intermediate.epddl -o /tmp/out

# solve
epistemic_planner --task /tmp/out/robot-fleet-depot-problem.json \
                --plan /tmp/plan.json --explain
```

Inside a built ePlanSys workspace the grounding step is implicit: the plan solver is given the `.epddl` sources directly, and `policy` is what keeps the branches.

```
planner:
  ros__parameters:
    plan_solver_timeout: 120.0 # the survey needs it; 15 s is the default
    EPISTEMIC:
      epddl_domain: ".../robot-fleet-depot-domain.epddl"
      epddl_problem: ".../robot-fleet-depot-problem.epddl"
      action_mapping: ".../mappings/robot-fleet-depot.json"
      strategy: "aostar"
      conditional_plan: "policy"

executor:
  ros__parameters:
    bt_builder_plugin: "EpistemicBTBuilder"
    bt_node_plugins: ["libplansys2_epistemic_bt_nodes.so"]
```

The `plan_solver_timeout` line is the one worth copying. Its default of 15 seconds is four times short of what the survey scenario needs, and the symptom of leaving it is a planner that returns no plan rather than one that reports a timeout.

10 Limitations

The timings are single runs on one machine, and the three instances differ in several parameters simultaneously, so nothing here isolates the effect of any one of them; the scaling study in the

companion report does that properly.

The survey instance takes about a minute, which is why it is shipped as an example rather than as a regression test; the corridor and depot instances carry that role.

Execution has been demonstrated but not deployed. The policies are validated by Aletheia, rendered as behavior trees, and run end to end over a real ROS graph against real epistemic state — but the performers behind the action expressions are stubs that report success after a fixed time, and the observations they report are supplied by the test. The claim is therefore about planning and about the executor, and not about a fleet that drove anywhere.

One case remains untested: an outcome the policy does not plan for, where the branch node must fail rather than guess so that the executor can replan. It needs a performer that reports something the model cannot account for, which is straightforward to write and has not been written.