Conformance Refinement for Online Planning

RSMG 6

Oliver Michael Kamperis

Marco Castellani

Yongjing Wang

What is planning?

- Formulate sequence of actions that achieve an objective:
  - Given an abstract world description (model),
  - An initial state and goal test.
- Inherently difficult, the agent must reason about and optimise series of decisions and their consequences into often large future horizons.
- Focusing on complex dynamic problems in/uncertain open worlds:
  - robot that can replace humans for completing tasks/doing jobs,
  - Combined navigation and manipulation type problems.

What do we want from a planner?

- **Expressive**: It should be able to express complex domain dynamics.
- **General**: It should be able to deal with a wide range of problems.
- **Fast**: It should find solutions quickly enough to be useful.
- **Reasonable solutions**: It is often intractable to find the optimal solution, so we can settle for something “close” but rational. A reasonable solution is one that a human might make or can at least understand.
- We don’t want to tell the planner how to find a plan:
  - Minimise reliance on domain-specific prescriptive knowledge where possible.

Classical Planning

Generates a single finite sequence of actions (a complete plan) through some specification of a discrete deterministic dynamic system.

- Usually these have a very large number of states and transitions:
  - Very large number of potential solutions,
  - Far too complex for naive exhaustive search methods.

- Typical solution: Add prescriptive knowledge, e.g. a heuristic.
  - Guide search, by exploiting specific nature/structural characteristics of the problem.

- Heuristics (under)estimate the cost or “distance” to the goal.
- Efficient and scales well if a “good” heuristic exists e.g. A* pathing.
- Path finding is “easy” as heuristic is based on literal distance to the goal.

Classical Planning

- Often hard to find good heuristics for complex problems:
  - Often have to reduce expressiveness to make up for this.
- Not always a fast and accurate way to measure “distance” to goal:
  - Problem may have an amorphous or elaborate structure,
  - May not be a way to obtain good cost estimates in reasonable time.
- Might need a lot of experience from past problems instances:
  - Planner could learn from solutions it has found previously,
  - Not very elaboration tolerant and usually doesn’t generalise well.
- Loss of generality if reliant on prescriptive knowledge to be tractable.

Refinement Planning

- Generate plans over an abstraction hierarchy.

- Successively refine a highly abstract classical plan down a hierarchy:
    - Problems are recursively divided and sub-divided into smaller subproblems,
    - Each action planned at abstract level becomes a subplan at bottom-level.

- Divide and conquer: Easy to solve a problem if it is broken into parts:
    - Finds abstract plan exponentially faster than original,
    - Refining this to the bottom-level then exponentially faster than directly solving the original problem via classical planning.

Examples of Refinement Planning by Humans

- **A milk-man problem:**
  - Decide on the order to visit the houses based on a simple Euclidean distance, a “relaxed” version of the problem, last place to visit should make it easy to get home.
  - Over time, the milk man gets used to the route and makes better choices.
- **A cooking problem:**
  - You find, or decide for yourself, some instructions, are relatively vague and general description of the sub-tasks to be completed, and the ingredients needed to complete them.
  - You consider the complexities later, e.g. how to crack the eggs, what pan to use, what hob to use.

Benefits and Problems of Refinement Planning

- Doesn’t need prescriptive knowledge for search or to divide problem:
    - Generally very easy to define the abstraction hierarchy,
    - Still possible to use (partial) heuristic in refinement planning if available.

- But generated plans can be arbitrarily worse:
    - **Ignorance problem**: Occurring from the lost knowledge in the abstractions.
    - **Dependency problem**: Occurring from the division of problems into subproblems that are assumed independent, this assumption almost always fails.
    - **Inflexible technique** makes this worse, each abstract action refined separately.

- Not very expressive in terms of defining abstractions:
    - Allows only “relaxing” the classical problem, a removal of action preconditions.

Answer Set Programming (ASP) based Planning

- Knowledge Representation and Reasoning language:
  - Extremely expressive in terms of modelled descriptive knowledge,
  - Can deal with very large number of facts and (recursive) relations,
  - Can directly combine planning and diagnostics,
  - Excellent for representing abstraction hierarchies.

- Has been limited to classical planning:
  - Planning is based on incremental search, similar to iterative deepening,
  - Becomes extremely poor when dealing with long plans,
  - Very difficult to represent prescriptive knowledge, heuristics non-applicable.

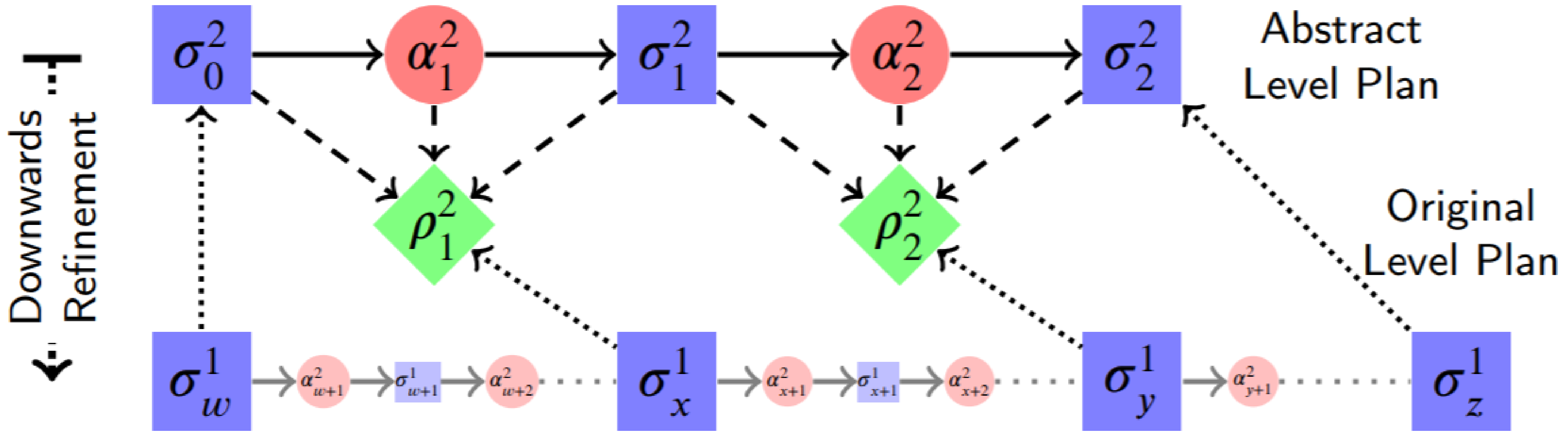
Conformance Refinement

Conformance refinement requires that a plan generated over an abstraction hierarchy remains structurally similar and achieve same effects at all levels.

- An attempt to make ASP based planners general, expressive and fast.
- Takes the human approach, only decide on the abstract stages of the plan initially, deals with the details later and as they emerge.
- Provides better integration of planning and diagnostics.
- Refines based on the effects of planned actions passed as subgoals:
  - More expressive abstractions, any abstract model with a state space mapping,
  - Allows refinement of any number of actions at once,
  - Allows flexible online plan generation and repair.

Conformance Refinement

- **Conformance Constraints**: Restrict the search space, by guiding the planner through a sequence of abstract intermediate subgoal stages.
- **Problem Division**: Divide the problem into smaller partial problems, defined by any contiguous subsequence of those subgoal stages.



Supports Different Abstract Models

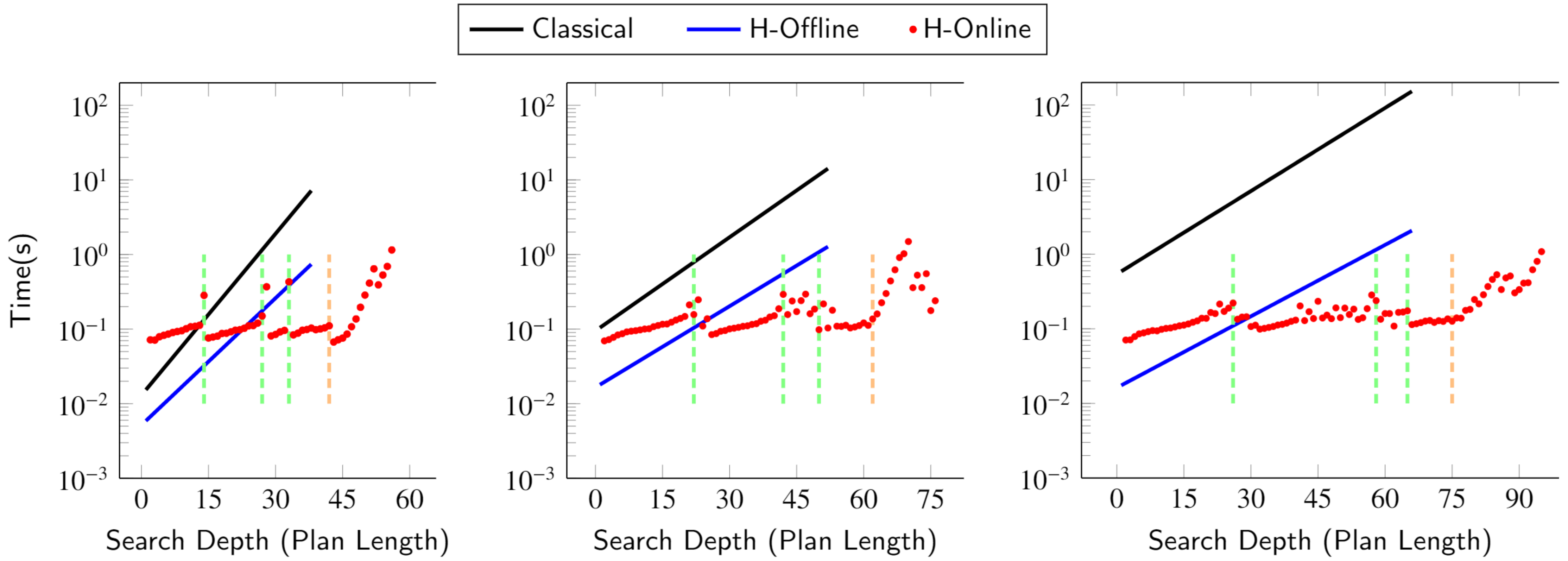
- **Relaxed Models:** Remove planning constraints, e.g. preconditions of actions, to shorten plans and reduce search spaces.
- **Condensed Models:** Remove the detailed parts/pieces of entities, combine them into abstract descriptors, reducing state space and plan length.
- **Discontinuous Models:** Ignore inertia, e.g. a robot is simultaneously in every location it has previously been, if we can get there once, we can get there again, reduces plan lengths.
- **Tasking Models:** Split the planning problem into a sequence of very high level tasks, allows more extensive problem division, but much more bespoke to each application domain.

Allows Hierarchical Plan Repair

- Deals with plan failure efficiently.
- Diagnostics and identification of the failed partial plan(s):
  - Hierarchical nature of plans (formed by sequences of partial plans) means that only the parts of plans that have failed need to be regenerated, this is called plan repair,
  - Repaired plans tend to get progressively worse over time,
  - Doesn’t need any additional knowledge to support this.
- May also be possible to determine if a goal can not longer be achieved and reject it, this kind of “goal reasoning” is usually difficult to support.

Current Results

- Offline finds complete plans exponentially faster than classical.
- Online yields partial plans exponentially faster than complete.



(a) Solution Checking Time: Problem 1 (b) Solution Checking Time: Problem 2 (c) Solution Checking Time: Problem 3

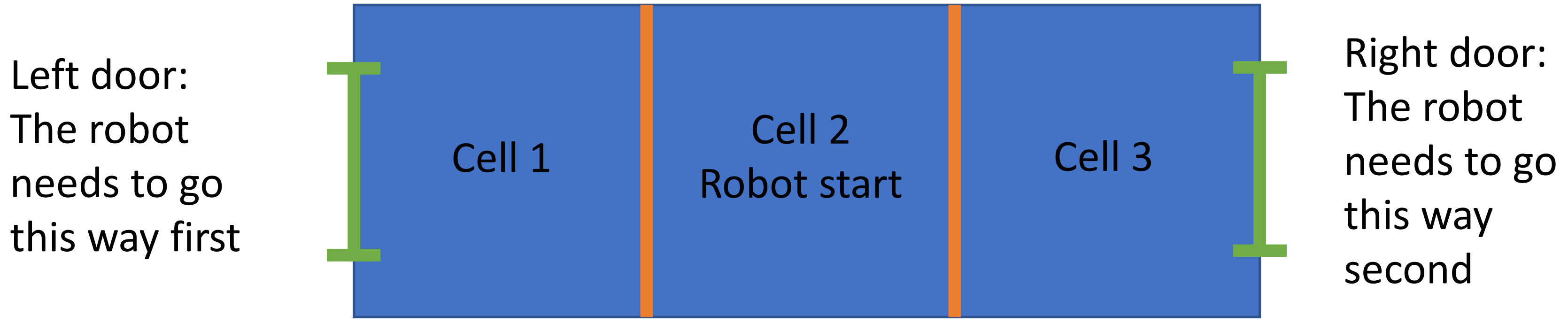
The trade-offs

- Offline planning achieves the **global refinement optimum**:
    - *Not subject to the dependency problem!*
    - *Still subject to ignorance problem*, since all decisions made (in ignorance) at the abstract levels must be maintained in the conformance refinement.

- Online planning only achieves the **local refinement optimums**:
    - *Still subject to the dependency problem*, hence the length of the concatenated partial plans are rarely same as the global refinement optimum,
    - If we had an oracle, we could find the admissible problem divisions (all partial problems are independent), but we don’t know this until it is solved.

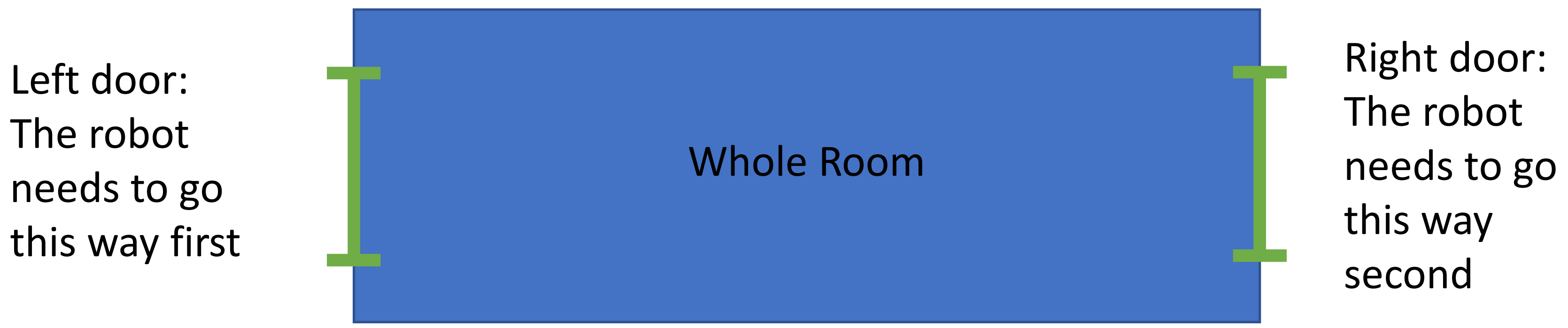
An example of these problems

- Room composed of a set of cells.
- Robot must move to correct cell to open a given door.
- To open a door the robot must extend an arm.
- Robot cannot move with its arms extended.
- Always optimal to open right hand side door first, then the left hand side door, as this requires less movement overall.


Left door: The robot needs to go this way first
Cell 1
Cell 2
Robot start
Cell 3
Right door: The robot needs to go this way second

An example of these problems

- Condensed abstract domain model used:
  - Cells are abstracted away in the abstract model,
  - Action enabling conditions are thus generalised.
- Robot is now able to open a door from anywhere in the room.
- Robot can open the doors in either order, without moving, because the resulting plan length is identical in both cases.

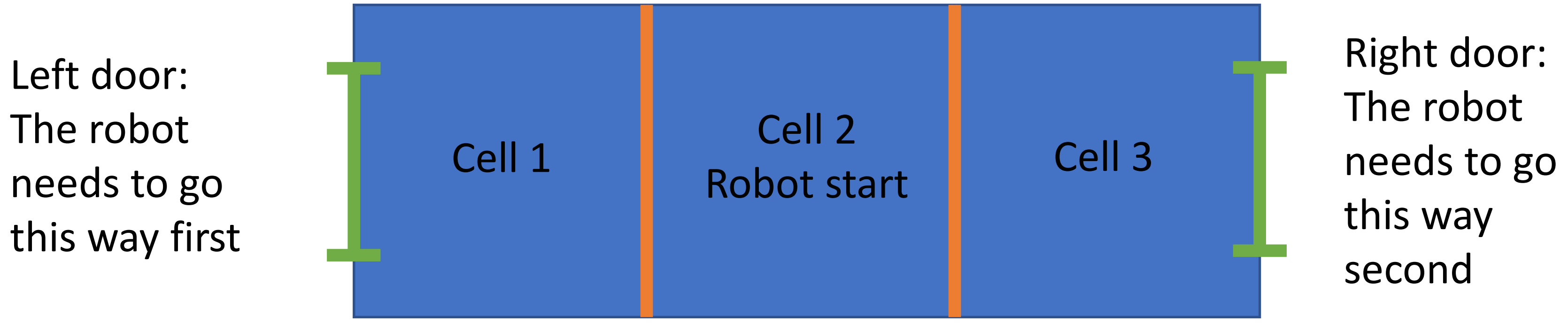

Left door:
The robot needs to go this way first
Whole Room
Right door:
The robot needs to go this way second

An example of these problems

- The decision made in the abstract version of the planning problem must be maintained in the refinement to achieve conformance.
- If the robot chose to open the left door first, then the plan is longer.

extend(arm) -> grasp(handle) -> actuate(door)

move_to(cell(x)) -> extend(arm) -> grasp(handle) -> actuate(door)

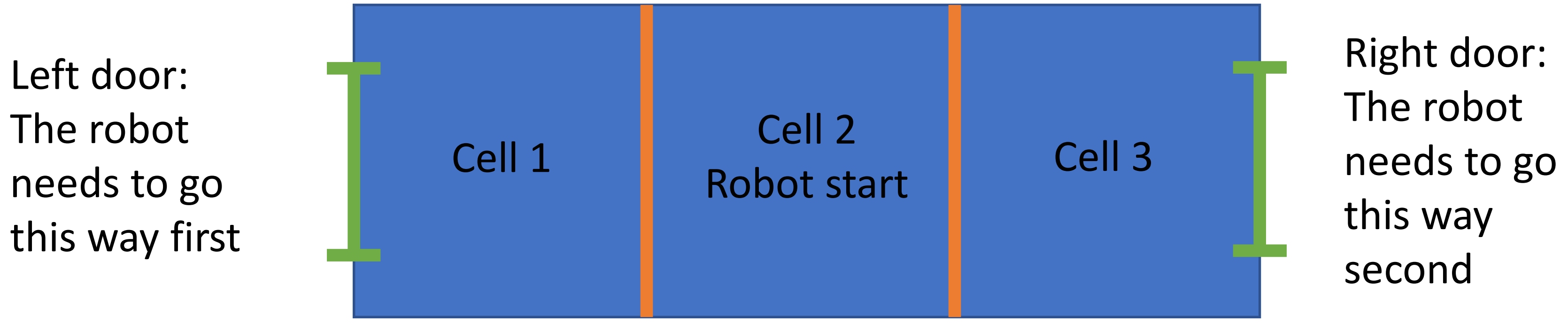

Left door: The robot needs to go this way first
Cell 1
Cell 2
Robot start
Cell 3
Right door: The robot needs to go this way second

An example of these problems

- If we refine the sub-goals separately, we get this plan, where the robot does not consider that it will need to grasp the handle as the immediately next subgoal and instantly extends its arm.

extend(arm) -> grasp(handle) -> actuate(door)

extend(arm) -> retract(arm) -> move_to(cell(x)) -> extend(arm) -> grasp(handle) -> actuate(door)



Current Progress and Future Work

- Planning systems currently work for a single robot and sequential actions.
- Old implementations need to be re-added:
  - Concurrency and multiple robots,
  - Discontinuous and Tasking models,
  - The diagnostics system.
- Work yet to do:
  - Various optimisations,
  - Plan repair algorithms,
  - Disassembly sequence planning case study.
- Hoping to test more domains to see how well the approach generalises.