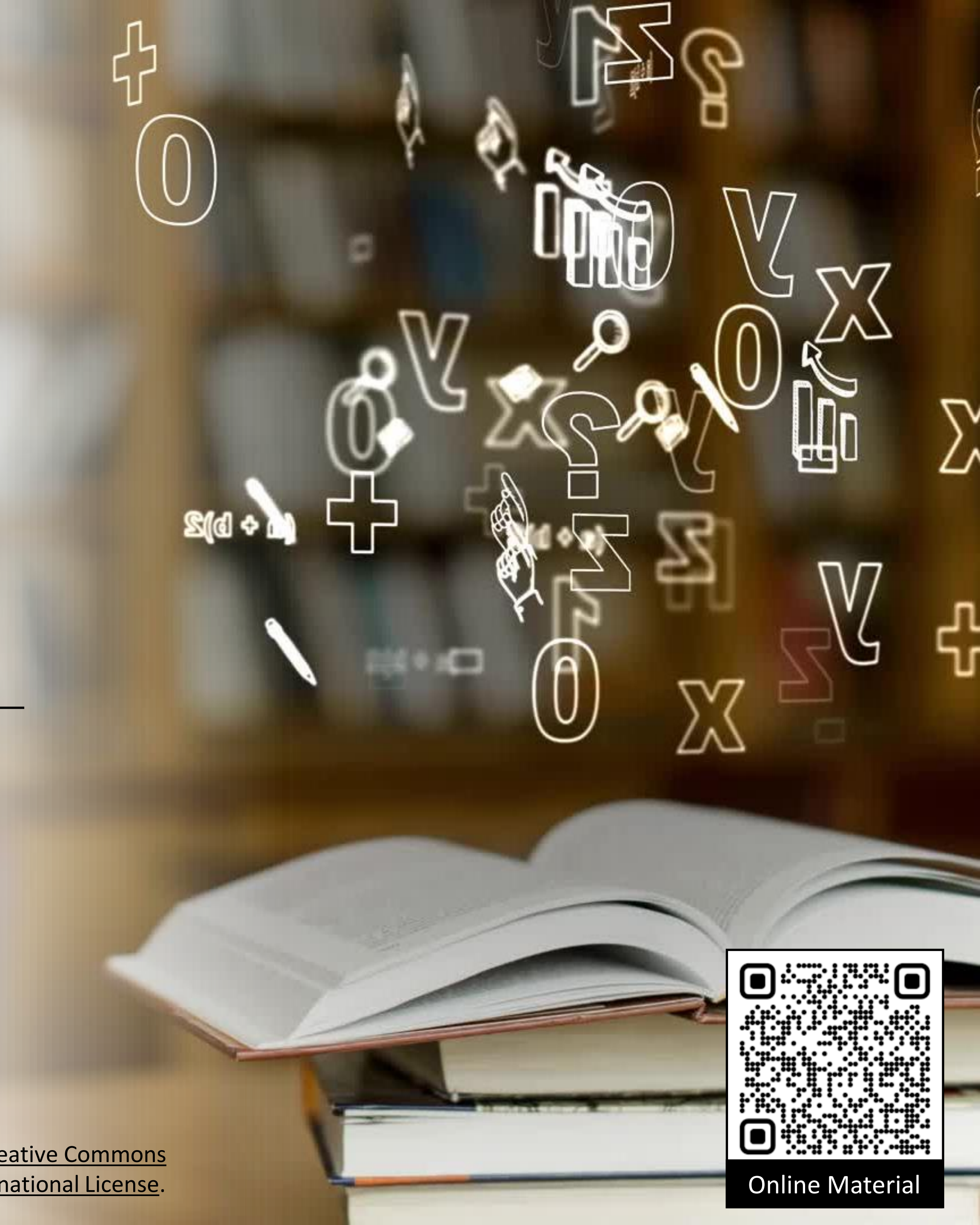
CS 5/7320
Artificial Intelligence

Automated Planning:
Hierarchical Planning and Monitoring

AIMA Chapter 11

Slides by Michael Hahsler
with figures from the AIMA textbook



Online Material

Contents

Classical Planning

Using Planning Domain Definition Languages

Classical Planning

- Find a sequence of actions to accomplish a goal in a discrete, deterministic, static, fully observable environment.

- Options we have already discussed:
  - Chapter 3: **Search** with a heuristic for informed search.
  - Chapter 7: Propositional **logic** with custom code.

- **Issue**: Large state space.

- **Solution**: Factored state representation using a Planning Domain Definition Language (PDDL) + Action schemas

Planning Domain Definition Language (PDDL)

an aspect of the world that can change over time

- **Factored state description**: a conjunction of ground atomic fluents (in 1-conjunctive normal form; 1-CNF).
- **Action Schema** (=precondition-effect description)

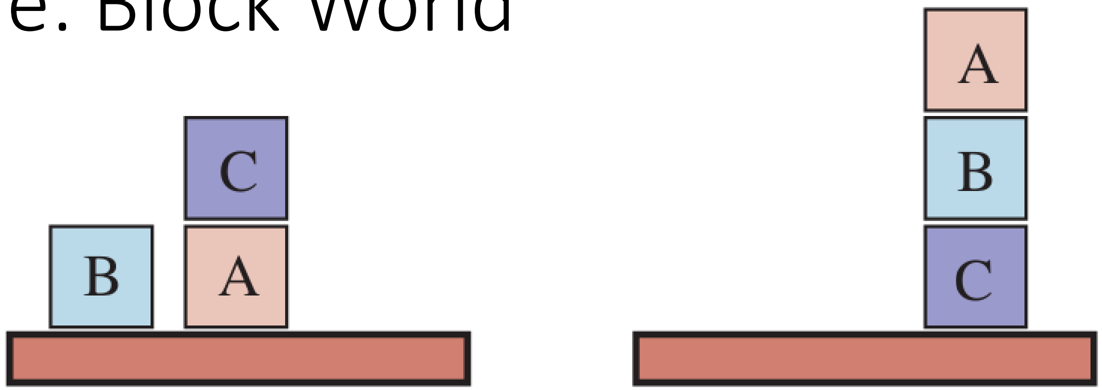
$$\begin{aligned}
&Action(Fly(p, from, to)),\\
&\quad \text{PRECOND: } Plane(p) \land Airport(from) \land\\
&\quad\quad\quad\quad\quad\quad Airport(to) \land At(p, from)\\
&\quad \text{EFFECT: } \underbrace{\neg At(p, from)}_{\text{DEL()}} \land \underbrace{At(p, to)}_{\text{ADD()}}
\end{aligned}$$

- Action a is applicable to state s if s entails the precondition of a.
- The effect of a on s is to remove the negated fluents and adds the positive fluents.

$$\mathrm{RESULT}(s, a) = \big(s - \mathrm{DEL}(a)\big) \cup \mathrm{ADD}(a))$$

- The **goal** is just like a precondition. E.g., $At(Plane_1, SFO) \land At(Plane_2, JFK)$

Example: Block World



Start State

Goal State

$Init(On(A, Table) \wedge On(B, Table) \wedge On(C, A)$
$\quad \wedge Block(A) \wedge Block(B) \wedge Block(C) \wedge Clear(B) \wedge Clear(C) \wedge Clear(Table))$
$Goal(On(A, B) \wedge On(B, C))$
$Action(Move(b, x, y),$
$\quad$ PRECOND: $On(b, x) \wedge Clear(b) \wedge Clear(y) \wedge Block(b) \wedge Block(y) \wedge$
$\qquad (b \neq x) \wedge (b \neq y) \wedge (x \neq y),$
$\quad$ EFFECT: $On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y))$
$Action(MoveToTable(b, x),$
$\quad$ PRECOND: $On(b, x) \wedge Clear(b) \wedge Block(b) \wedge Block(x),$
$\quad$ EFFECT: $On(b, Table) \wedge Clear(x) \wedge \neg On(b, x))$

Figure 11.4 A planning problem in the blocks world: building a three-block tower. One solution is the sequence $[MoveToTable(C, A), Move(B, Table, C), Move(A, Table, B)]$.

Algorithm Options

a) **Forward state-space search**: Action schema represents the transition model. Perform regular BFS/DFS search. Often A* with a heuristics is used to deal with the state space size.

b) **Backward search** (= regression search): keeps the branching factor low. Issue: How do we define heuristics?

c) Convert the PDDL description into propositional form and use an efficient solver for the **Boolean satisfiability problem (SAT).**

A* Heuristics for Planning

Use the factored state description to calculate a heuristic function $h(s)$ that estimates the distance from s to the goal. If it is admissible (does not overestimate the distance), then A* can be used.

Example relaxations to create a heuristic:

- Ignore preconditions: any action can be used in any state
- Ignore delete-list: no negative effects, problem progresses monotonically towards the goal.
- Serializable subgoals: subgoals can be achieved without undoing a previous subgoal.
- State abstraction to reduce the number of states. E.g., ignore some fluents.

Example: maze

State: $PosX(x) \wedge PosY(y)$

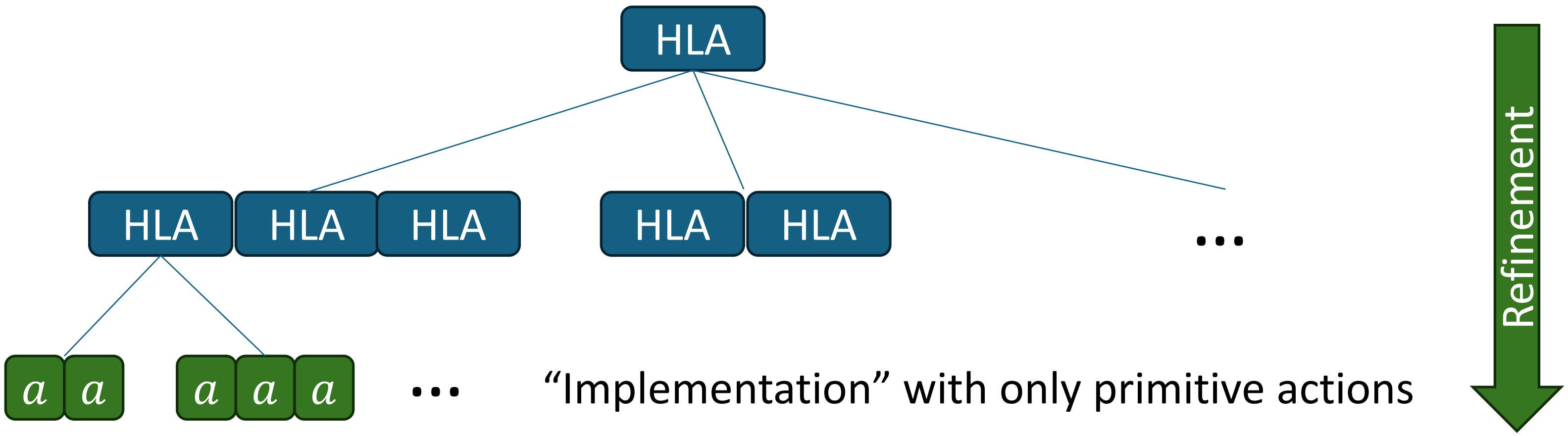
Heuristic: Ignore precondition that checks for walls

Hierarchical Planning

Manage complexity using high-level actions.

High-level Actions

- A **high-level action (HLA)** solves a problem or a subproblem in one step.
- An HLA has one or several refinements into a sequence of HLAs or primitive actions.



- Reasoning and search for HLAs reduces the search space.
- A top-level HLA achieves the goal if at least one implementation achieves the goal.

Example: Refinement

- Two refinements for the HLA $Go(Home, SFO)$ to go from home to the SFO airport:

$$
\begin{aligned}
&Refinement(Go(Home, SFO),\\
&\quad \textsc{Steps}: [Drive(Home, SFOLongTermParking),\\
&\qquad\qquad Shuttle(SFOLongTermParking, SFO)]\ )\\
&Refinement(Go(Home, SFO),\\
&\quad \textsc{Steps}: [Taxi(Home, SFO)]\ )
\end{aligned}
$$

- Since both refinements achieve the goal, the agent can choose which implementation of the HLA to use.

Option 1: Search for Primitive Solutions

- The top HLA is often just "Act" and the agent needs to find an implementation that achieves the goal.

- Classical Planning
  - For each primitive action, provide a refinement of Act with steps $[a_i, Act]$.
  - This can recursively build any sequence of actions.
  - To stop the recursion, define:

$Refinement(Act)$,
PRECOND: goal is reached
STEPS: []

- **Issue**: This approach must search through all possible sequences!
- **Improvement**:
  - Reduce the number of needed refinements + increase the number of steps in each refinement.

Option 1: Search for Primitive Solutions – BFS Implementation

```
function HIERARCHICAL-SEARCH(problem, hierarchy) returns a solution or failure
  frontier ← a FIFO queue with [Act] as the only element
  while true do
     if IS-EMPTY(frontier) then return failure
     plan ← POP(frontier)          // chooses the shallowest plan in frontier
     hla ← the first HLA in plan, or null if none
     prefix,suffix ← the action subsequences before and after hla in plan
     outcome ← RESULT(problem.INITIAL, prefix)
     if hla is null then          // so plan is primitive and outcome is its result
        if problem.IS-GOAL(outcome) then return plan
     else for each sequence in REFINEMENTS(hla, outcome, hierarchy) do
        add APPEND(prefix, sequence, suffix) to frontier
```

Figure 11.8 A breadth-first implementation of hierarchical forward planning search. The initial plan supplied to the algorithm is [*Act*]. The REFINEMENTS function returns a set of action sequences, one for each refinement of the HLA whose preconditions are satisfied by the specified state, *outcome*.

Option 2: Searching for Abstract Solutions

- **Issue**: Search for primitive solutions has to refine all HLAs all the way to primitive actions to determine if a plan is workable.

- **Idea: Determine what HLAs do.**
  - Write precondition-effect descriptions for HLAs (this is difficult because of neg. effects!)
  - This results in an exponential reduction of the search space.

- **Reachable set**: the set of states reachable with a sequence of HLAs $[h_1, h_2]$ in state s.

$$REACH(s, [h_1, h_2]) = \bigcup_{s' = REACH(s, h_1)} REACH(s', h_2)$$

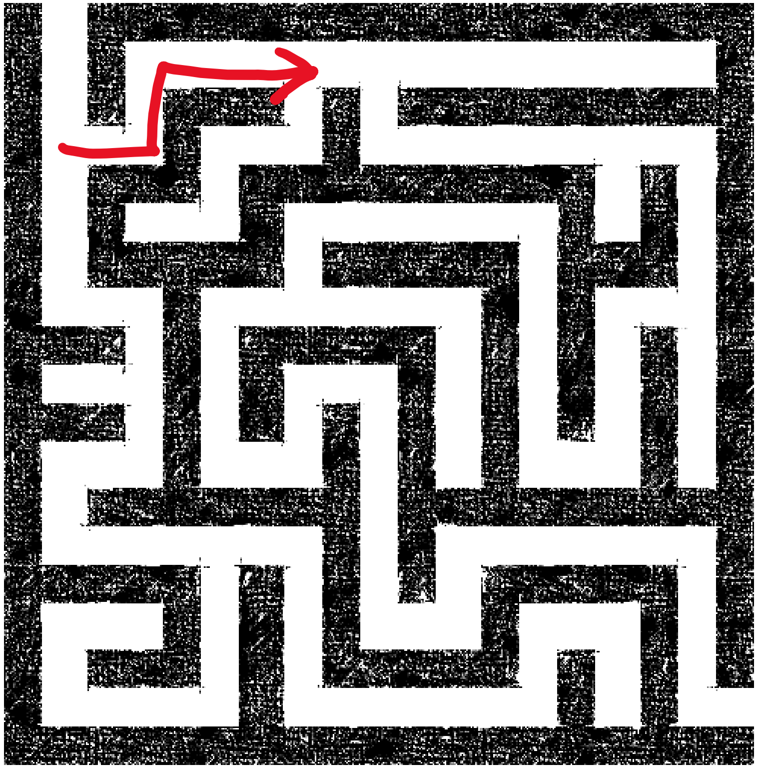
  A sequence of HLAs achieves the goal if its reachable set intersects the goal set.

- Typical implementation:
  1. Use a simplified (optimistic) version of precondition-effect descriptions to find a high-level plan that works.
  2. Check if a refinement of that plan that works really exists. If not, go back to 1.

Conclusion

- High-level actions are a powerful concept for dealing with large search spaces/search trees.
- Example:

Actions: {N, E, S, W}



Top high-level action:
A sequence of second-level HLAs.

Second-level HLA:
Go to the next intersection

Example Implementation:
[E, E, N, N, E, E, E]

This leads to a much smaller state space and search tree!

Monitoring and Replanning

Planning and Acting in Partially Observable, Nondeterministic, and Unknown Environments

Belief States

- For **nondeterministic** or **partially observable** environments we need belief states.

- A belief state is a set of possible physical states the agent might be in given its current knowledge.

- The belief state concept needs to be extended to the factored state representation.
  - A belief state becomes a logical formula of fluents.
  - Fluents that do not appear in the formula are unknow.

Technical note: If we manage to keep the belief state in 1-CNF (1-conjunctive normal form, i.e., fluents are combined with ANDs), then the complexity is reduced from being exponential in the number of fluents to linear!

Observability: Percept Schema

- For **partially observable** environments, we need to be able to define what percepts the agent can get when.
- The agent uses a percept schema to reason about percepts that it can obtain during executing a plan.
- Example: Whenever the agent sees an object, then it will perceive its color.

$$\boxed{\begin{array}{l} Percept(Color(x, c)), \\ \quad \text{PRECOND: } Object(x) \land inView(x) \end{array}}$$

  The agent can now reason that it needs to get an object inView to see the color.

- Percept schemata and observability
  - **Fully observable**: Percept schemas have no preconditions.
  - **Partially observable**: Some percepts have preconditions.
  - **Sensorless agent**: has no percept schemas.

Observability: Sensorless Planning

- We assume the underlying planning problem is deterministic.

- Similar to sensorless search in Chapter 4. Differences:
  - Transition model is a set of action schemata.
  - Belief state is represented as a logical formula where unknown fluents are missing.

  - Update:

$$\mathrm{b}' = \mathrm{RESULT}(b, a) = \{s' : s' = RESULT_P(s, a) \text{ and } s \in b\}$$

$RESULT\_P$ represents the physical transition model which adds positive and negative literals to the state description. The state description becomes more and more complete.

Determinism & Observability: Contingency Planning

- We can create a conditional plan for partially observable planning problems and non-deterministic problems.
- We already have introduced conditional plans in Chapter 4 and just need to augment it by:
  - Action schemata instead of a transition function.
  - Percept schemata to reason about how to get needed percepts.
  - The state has a factored representation as facts in 1-CNF.

- Use **AND-OR search** over belief states.

- **Issues**:
  - Contingency plans become very complicated with **non-deterministic effects** like failures in actions or percepts. E.g., moving north fails 1 out of 100 times.
  - Plan fails with **incorrect model of the world**. E.g., actions with missing preconditions or missing effects, missing fluents, exogenous effects.

→ Online Planning

Execution Monitoring and Replanning

- Perform regular planning, but **replan** when plan execution fails.

- Requires **execution monitoring** to determine the need for replanning. The agent can perform:
  - **Action monitoring**: Only execute the action if the preconditions are met.
  - **Plan monitoring**: Verify that the remaining plan will still succeed.
  - **Goal monitoring**: Check if a better set of goals has become available.

- Large contingency plans can often be made simpler by having unlikely branches just say “REPLAN.”
  E.g., Chess: don’t plan for very unlikely moves of the opponent.

Example: Plan Monitoring with Repair

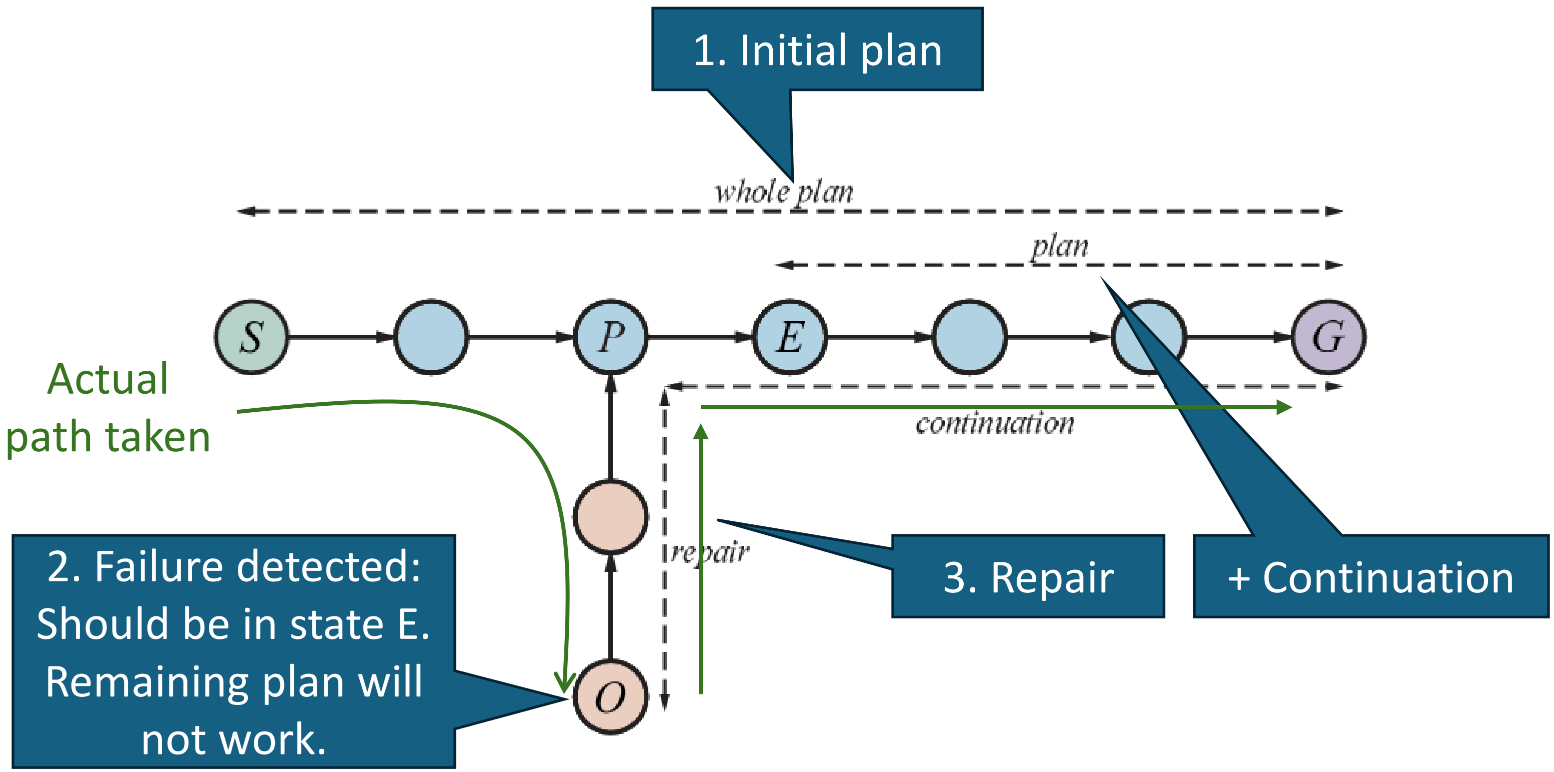
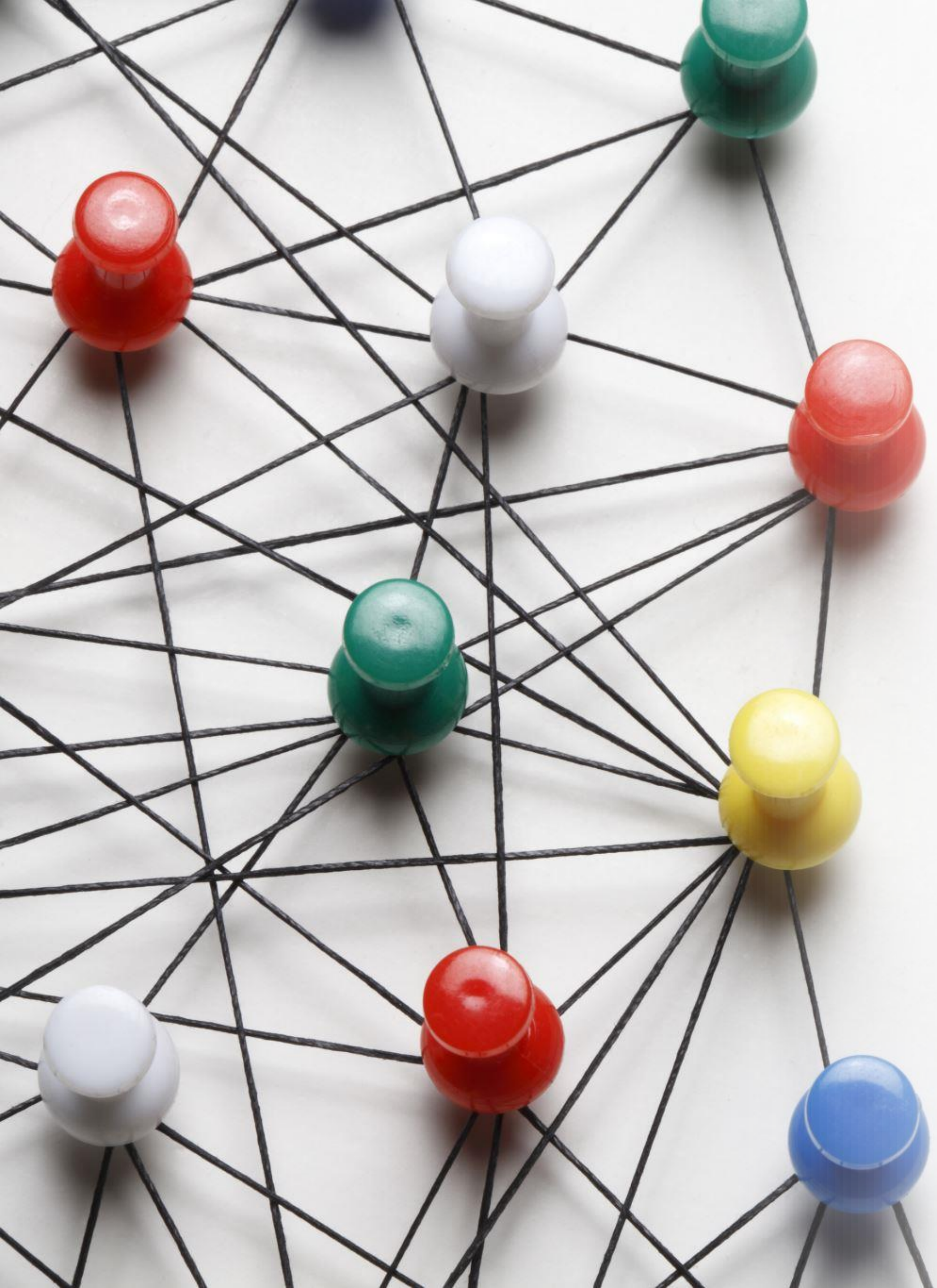


Figure 11.12 At first, the sequence "whole plan" is expected to get the agent from S to G. The agent executes steps of the plan until it expects to be in state E, but observes that it is actually in O. The agent then replans for the minimal *repair* plus *continuation* to reach G.



Summary

- **Action schemata** make specifying the transition function easier.

- **Hierarchical planning** lets us deal with the exponential size of the state space. The agent can reason at a more abstract level of high-level actions and the states are typically discrete.

- **Online planning with monitoring and replanning** is
  - very flexible
  - can deal with many types of issues (sensor/actuator failure, imperfect models of the environment)
  - Can make conditional plans smaller by omitting unlikely paths and leaving them for later replanning.