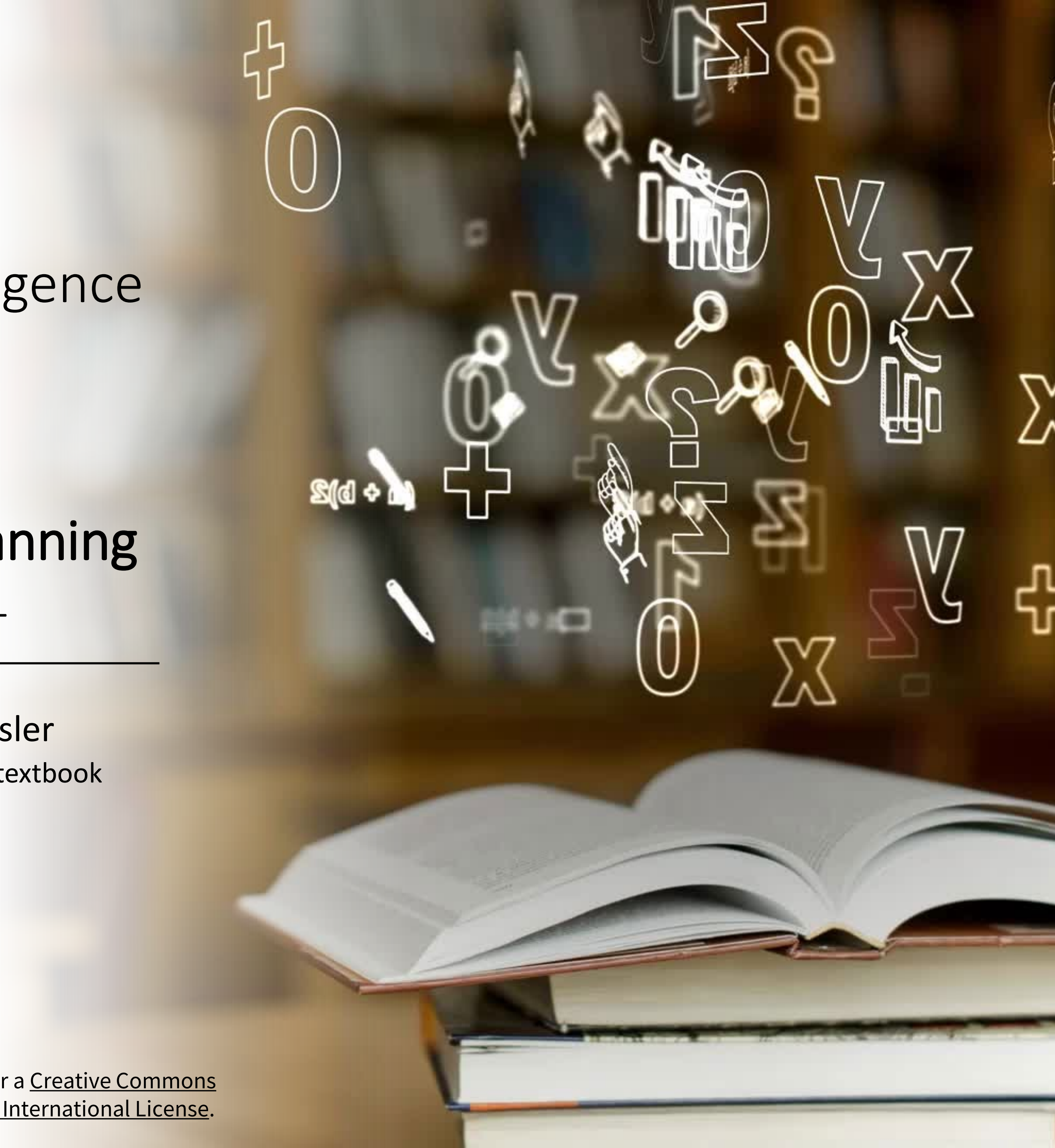
CS 5/7320
Artificial Intelligence

Automated Planning
AIMA Chapter 11



Slides by Michael Hahsler
with figures from the AIMA textbook

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:
  - Search with a custom heuristic evaluation function (Chapter 3)
  - Propositional logic with custom code (Chapter 7)

- Issue: state space is exponentially large state

- Solution: factored state representation using a Planning Domain Definition Language (PDDL)

Planning Domain Definition Language (PDDL)

- State: a conjunction of ground atomic fluents (in Conjunctive Normal Form; CNF).
- Fluent: an aspect of the world that changes over time.
- Action Schema (=precondition-effect description)

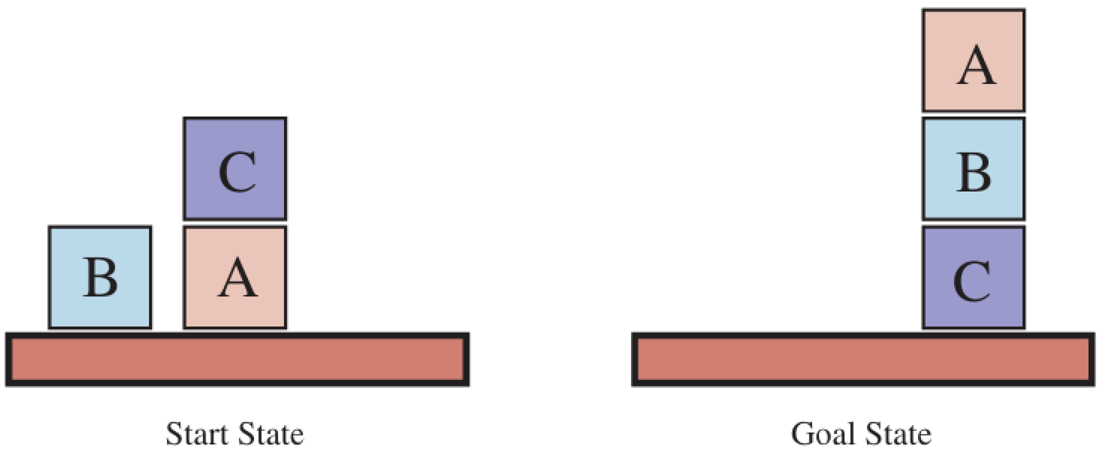
$$\boxed{\begin{array}{l} Action(Fly(p, from, to)), \\ \quad \text{PRECOND: } At(f, from) \land Plane(p) \land \\ \qquad\qquad Airport(from) \land Airport(to) \\ \quad \text{EFFECT: } \underbrace{\neg\, At(p, from)}_{\mathrm{DEL}(a)} \land \underbrace{At(p, to)}_{\mathrm{ADD}(a)} \end{array}}$$

- Action a is applicable to state s if s entails the precondition of a.
- The result of a on s is s' which removes the negated fluents and adds the positive fluents.

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

- Goal is just like a precondition.

Example



$$
\begin{aligned}
&\mathit{Init}(\mathit{On}(A, \mathit{Table}) \wedge \mathit{On}(B, \mathit{Table}) \wedge \mathit{On}(C, A) \\
&\quad \wedge \mathit{Block}(A) \wedge \mathit{Block}(B) \wedge \mathit{Block}(C) \wedge \mathit{Clear}(B) \wedge \mathit{Clear}(C) \wedge \mathit{Clear}(\mathit{Table})) \\
&\mathit{Goal}(\mathit{On}(A, B) \wedge \mathit{On}(B, C)) \\
&\mathit{Action}(\mathit{Move}(b, x, y), \\
&\quad \text{PRECOND: } \mathit{On}(b, x) \wedge \mathit{Clear}(b) \wedge \mathit{Clear}(y) \wedge \mathit{Block}(b) \wedge \mathit{Block}(y) \wedge \\
&\qquad\qquad (b \neq x) \wedge (b \neq y) \wedge (x \neq y), \\
&\quad \text{EFFECT: } \mathit{On}(b, y) \wedge \mathit{Clear}(x) \wedge \neg \mathit{On}(b, x) \wedge \neg \mathit{Clear}(y)) \\
&\mathit{Action}(\mathit{MoveToTable}(b, x), \\
&\quad \text{PRECOND: } \mathit{On}(b, x) \wedge \mathit{Clear}(b) \wedge \mathit{Block}(b) \wedge \mathit{Block}(x), \\
&\quad \text{EFFECT: } \mathit{On}(b, \mathit{Table}) \wedge \mathit{Clear}(x) \wedge \neg \mathit{On}(b, x))
\end{aligned}
$$

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

Algorithms

- Forward state-space search: Needs heuristics to deal with the state space.
- Backward search (= regression search): keeps the branching factor low. But how do we define heuristics?
- Convert the PDDL description into propositional form and use one of th4 efficient solvers for the Boolean satisfiability problem (SAT).

- Heuristics for Planning
  Use the factored state space 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.
- Apply relaxations:
  - Ignore-preconditions: any action can be used in any state
  - Ignore delete-list: no negative effects, problem progresses monotonic 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.

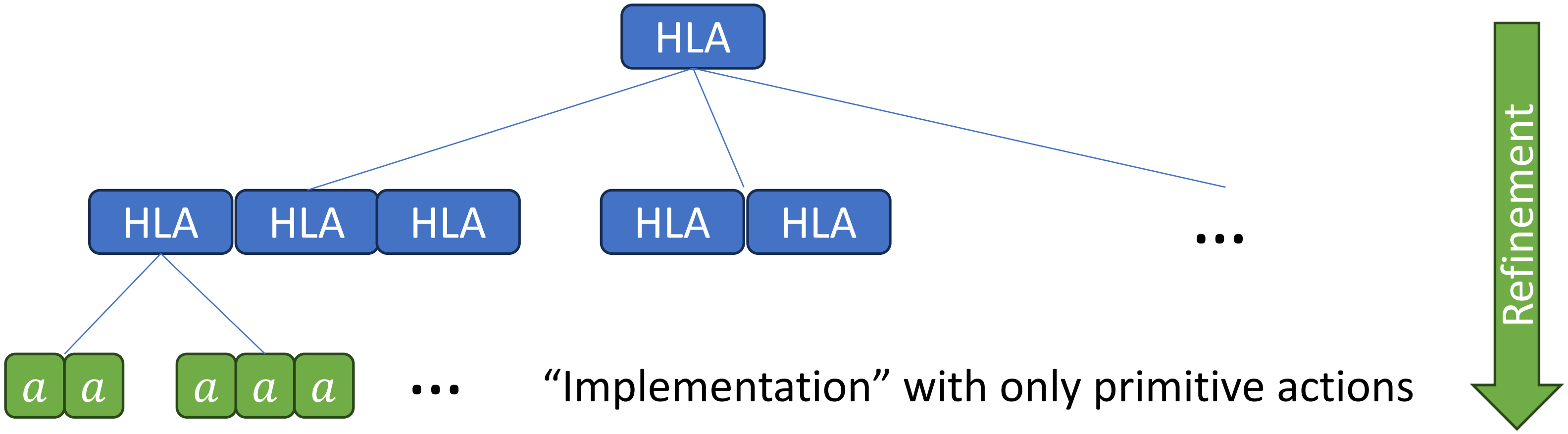
Remember the maze: We used x and y coordinates to calculate the distance to the goal.

Hierarchical Planning

Manage complexity using high-level actions.

High-level Actions

- A high-level action (HLA) have one or several refinements into a sequence of HLAs or primitive actions.



- An HLA achieves the goal if at least one implementation achieves the goal.
- The agent can reason with HLAs.

Example

- Two refinements for the HLA to go to the SFO airport:

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

- The agent can choose which implementation of an HLA to use.

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 on refinement of Act with steps $[a_i, Act]$.
  - Recursively builds any sequence of actions.
  - To stop the recursion, define:

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

- Improvements
  - Manually reduce the number of refinements.
  - Manually increase the number of steps in each refinement.

Implementation: Search for Primitive Solutions

```
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*.

Searching for Abstract Solutions

- 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 would result 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. Use that plan to search if a refinement that works really exists.

Monitoring and Replanning

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

Belief States

- A belief state is a set of possible physical states the agent might be in.
- Belief states need to be extended for factored state representation.
  - The belief state becomes a set of logical formula of fluents.
  - Fluents that do not appear in the formula are unknow.

Percept Schema

- The agent uses a percept schema to reason about percepts that it can obtains 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 the 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.

Sensorless Planning (Conformant 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 an be represented as a logical formula.
    - 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 literals and negative literals to the state description.

Contingency Planning

- For partially observable planning 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 CNF.

- Use AND-OR search over belief states.

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

→ Online Planning

Online Planning

- Online planning **replans** when necessary.
- Requires **execution monitoring** to determine the need for replanning. The agent can perform
  - Action monitoring (verify that the preconditions are met)
  - Plan monitoring (verify that the remaining plan will still succeed)
  - Goal monitoring (check if a better set of goals becomes available)
- Often contingency plans are made simpler by having some branches that just say “REPLAN”
- Process:

Plan Monitoring Example with Repair

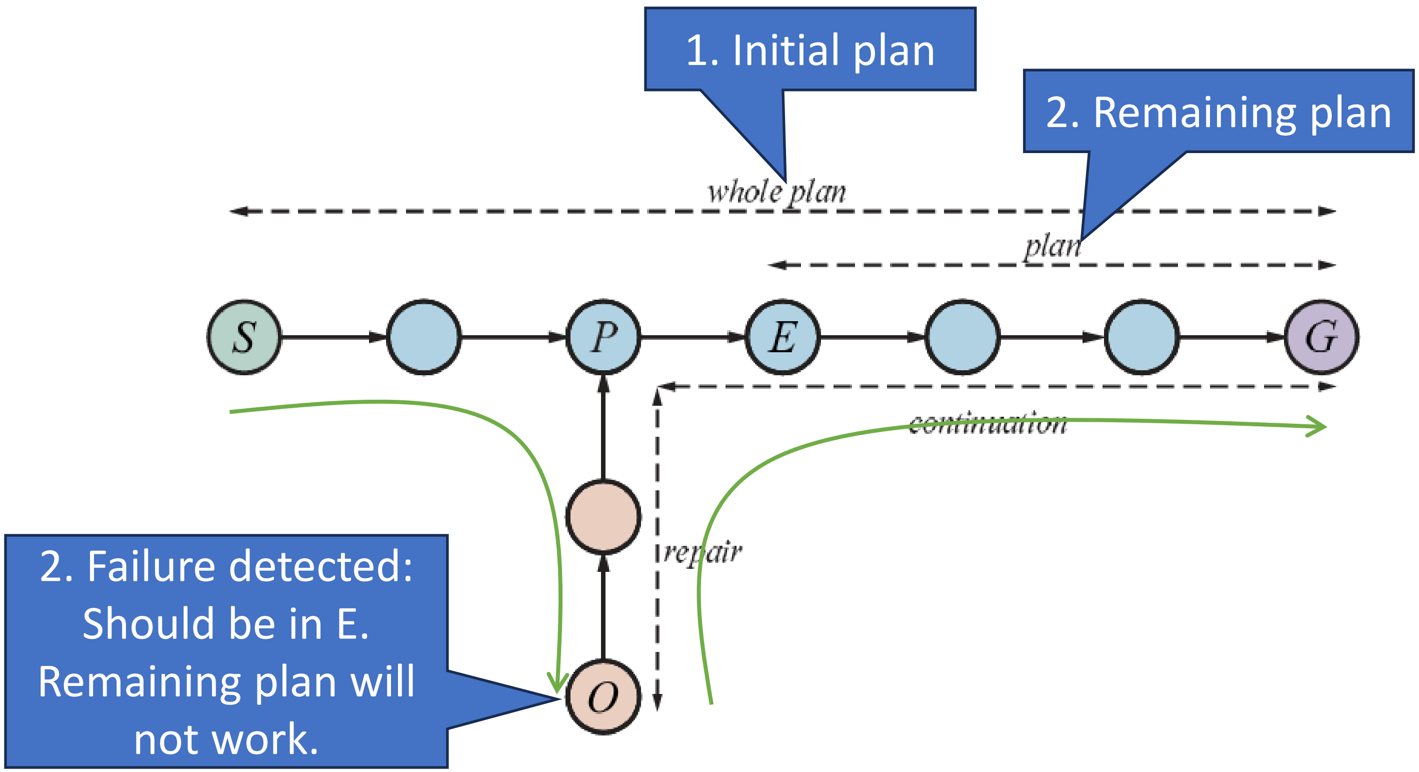
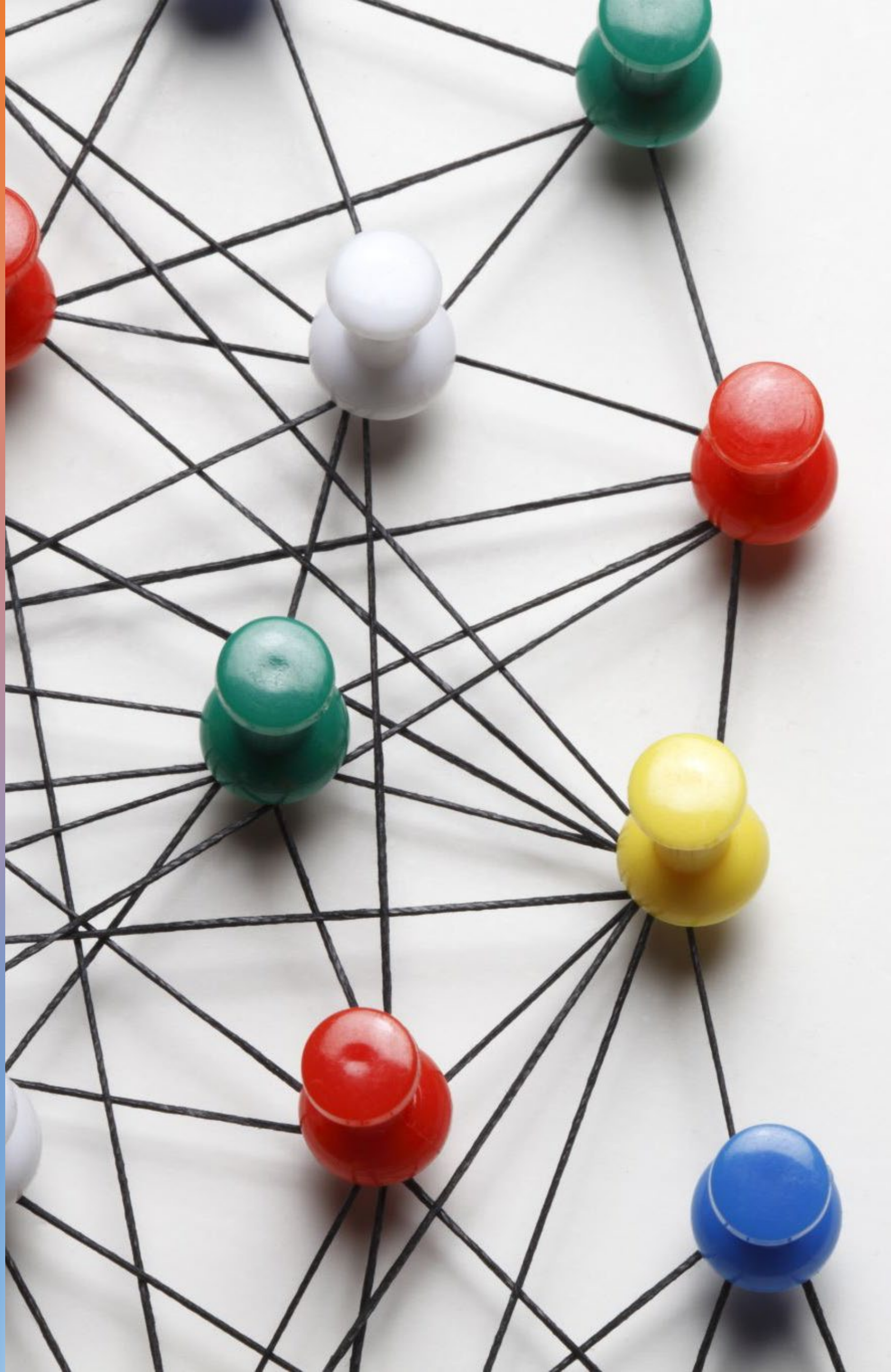


Figure 11.12 At first, the sequence "whole plan" is expected to get the agent from S to G. The agent executes steps of the plan until it expects to be in state E, but observes that it is actually in O. The agent then replans for the minimal *repair* plus *continuation* to reach G.



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 and can deal with many types of issues (sensor/actuator failure, imperfect models of the environment)