Where are we?

Last time . . .
- Looked at methods for real-world planning
- Sensorless planning and contingent planning
- Fully and partially observable environments

Today . . .
- Planning and Acting in the Real World II
Execution monitoring and replanning

- **Execution monitoring** = checking whether things are going according to plan (necessitated by unbounded indeterminacy in realistic environments)
  - Action monitoring = checking whether next action is feasible
  - Plan monitoring = checking whether remainder of plan is feasible
- **Replanning** = ability to find new plan when things go wrong (usually repairing the old plan)
- Taken together these methods yield powerful planning abilities
Action monitoring and replanning

- While attempting to get from $S$ to $G$, a problem is encountered in $E$, agent discovers actual state is $O$ and plans to get to $P$ and execute the rest of the original plan.
Plan monitoring

- Action monitoring often results in suboptimal behaviour, executes everything until actual failure
- **Plan monitoring** checks preconditions for entire remaining plan
- Can also take advantage of **serendipity** (unexpected circumstances might make remaining plan easier)
- In partially observable environments things are more complex (sensing actions have to be planned for, they can fail in turn, etc.)
Hierarchical decomposition in planning

- **Hierarchical decomposition** seems a natural idea to improve planning capabilities.

- **Key idea**: at each level of the hierarchy, activity involves only a small number of steps (i.e. small computational cost)

- **Hierarchical task network** (HTN) planning: initial plan provides only high-level description, refined by **action refinements**

- Refinement process continued until plan consists only of **primitive actions**
Representing action decompositions

- Each high level action (HLA) has (at least) one refinement into a sequence of actions.
- The actions in the sequence may be HLAs or primitive.
  - So HLAs form a hierarchy!
- If they’re all primitive, then that’s an implementation of the HLA.
Example: Go to SF Airport

\[
\text{Refinement}(\text{Go}(\text{Home}, \text{SFO}), \\
\text{Precond}: \text{At}(\text{Car}, \text{Home}) \\
\text{Steps}: [
\text{Drive}(\text{Home}, \text{SFOLongTermParking}) \\
\text{Shuttle}(\text{SFOLongTermParking}, \text{SFO})]
]\]

\[
\text{Refinement}(\text{Go}(\text{Home}, \text{SFO}), \\
\text{Precond}: \text{Cash}, \text{At}(\text{Home}) \\
\text{Steps}: [
\text{Taxi}(\text{Home}, \text{SFO})]
]\]
Refinements can be Recursive

\[
\text{Refinement}(\text{Navigate}([a, b], [x, y])), \\
\text{PRECOND}: a = x, b = y \\
\text{STEPS}:[[]]
\]

\[
\text{Refinement}(\text{Navigate}([a, b], [x, y])), \\
\text{PRECOND}: \text{Connected}([a, b], [a - 1, b]) \\
\text{STEPS}:[\text{Left}, \text{Navigate}([a - 1, b], [x, y])])
\]

\[
\text{Refinement}(\text{Navigate}([a, b], [x, y])), \\
\text{PRECOND}: \text{Connected}([a, b], [a + 1, b]) \\
\text{STEPS}:[\text{Right}, \text{Navigate}([a + 1, b], [x, y])])
\]
High-Level Plans

- High-Level Plans (HLP) are a sequence of HLAs.
- An implementation of a High Level Plan is the concatenation of an implementation of each of its HLAs.
- An HLP achieves the goal from an initial state if at least one of its implementations does this.
- Not all implementations of an HLP have to reach the goal state!
- The agent gets to decide which implementation of which HLAs to execute.
Searching for Primitive Solutions

- The HLA plan library is a hierarchy:
  - (Ordered) Daughters to an HLA are the sequences of actions provided by one of its refinements;
  - Because a given HLA can have more than one refinement, there can be more than one node for a given HLA in the hierarchy.

- This hierarchy is essentially a search space of action sequences that conform to knowledge about how high-level actions can be broken down.

- So you can search this space for a plan!
Searching for Primitive Solutions: Breadth First

- Start your plan $P$ with the HLA [Act],
- Take the first HLA $A$ in $P$ (recall that $P$ is an action sequence).
- Do a breadth-first search in your hierarchical plan library, to find a refinement of $A$ whose preconditions are satisfied by the outcome of the action in $P$ that is prior to $A$.
- Replace $A$ in $P$ with this refinement.
- Keep going until your plan $P$ has no HLAs and either:
  1. Your plan $P$’s outcome is the goal, in which case return $P$; or
  2. Your plan $P$’s outcome is not the goal, in which case return failure.
Problems!

- Like forward search, you consider lots of irrelevant actions.
- The algorithm essentially refines HLAs right down to primitive actions so as to determine if a plan will succeed.
- This contradicts common sense!
- Sometimes you know an HLA will work regardless of how it’s broken down!
- We don’t need to know which route to take to SFOParking to know this plan works:
  
  \[\text{Drive(Home, SFOParking)}, \text{Shuttle(SFOParking, SFO)}\]
  
- We can capture this if we add to HLAs themselves a set of preconditions and effects.
One challenge in specifying preconditions and effects of an HLA is that the HLA may have more than one refinement, each one with slightly different preconditions and effects!

- If you refine `Go(Home, SFO)` with `Taxi` action: you need `Cash`.
- If you refine it with `Drive`, you don’t!
- This difference may affect your choice on how to refine the HLA!

Recall that an HLA achieves a goal if one of its refinements does this.

And you can choose the refinement!
Getting Formal

- \( s' \in \text{Reach}(s, h) \) iff \( s' \) is reachable from \textit{at least one} of HLA \( h \)'s refinements, given (initial) state \( s \).

\[
\text{Reach}(s, [h_1, h_2]) = \bigcup_{s' \in \text{Reach}(s, h_1)} \text{Reach}(s', h_2)
\]

- HLP \( p \) achieves goal \( g \) given initial state \( s \) iff \( \exists s' \) st

\[
s' \models g \text{ and } s' \in \text{Reach}(s, p)
\]

- So we should search HLPs to find a \( p \) with this relation to \( g \), and then focus on refining it.

- But a pre-requisite to this algorithm is to define \( \text{Reach}(s, h) \) for each \( h \) and \( s \).

- In other words, we still need to determine how to represent effects (and preconditions) of HLAs...
Defining **REACH**

- A primitive action makes a fluent true, false, or leaves it unchanged.
- But with HLAs you sometimes get to *choose*, by choosing a particular refinement!
- We add new notation to reflect this:
  - $\vdash A$: you can possibly add $A$ (or leave $A$ unchanged)
  - $\neg A$: you can possibly delete $A$ (or leave $A$ unchanged)
  - $\pm A$: you can possibly add $A$, or possibly delete $A$ (or leave $A$ unchanged)
- You should now *derive* the correct preconditions and effects from its refinements!
Our SFO Example

\[
\text{Refinement}(\text{Go(Home, SFO)}, \\
\text{Precond:At(Car, Home)} \\
\text{Steps:}[\text{Drive(Home, SFO\text{LongTermParking})} \\
\hspace{1cm}\text{Shuttle(SFO\text{LongTermParking, SFO})}])
\]

\[
\text{Refinement}(\text{Go(Home, SFO)}, \\
\text{Precond:Cash, At(Home)} \\
\text{Steps:[Taxi(Home, SFO)]})
\]
The ‘Primitive’ Actions

\(\text{Action}(\text{Taxi}(a, b))\),

**Precond:** \(\text{Cash}, \text{At}(\text{Taxi}, a)\)

**Effect:** \(\neg \text{Cash}, \neg \text{At}(\text{Taxi}, a), \text{At}(\text{Taxi}, b)\)

\(\text{Action}(\text{Drive}(a, b))\),

**Precond:** \(\text{At}(\text{Car}, a)\)

**Effect:** \(\neg \text{At}(\text{Car}, a), \text{At}(\text{Car}, b)\)

\(\text{Action}(\text{Shuttle}(a, b))\),

**Precond:** \(\text{At}(\text{Shuttle}, a)\)

**Effect:** \(\neg \text{At}(\text{Shuttle}, a), \text{At}(\text{Shuttle}, b)\)
Deriving the Preconds and Effects of the HLA

- $\neg Cash$ is Effect of one HLA refinement, but not the other.
- So $\sim Cash$ in HLA Effect!

Not so Simple!

- Similar argument for $At(Car, SFOParking)$
- But you can't choose the combination: $\neg Cash \land At(Car, SFOParking)$
- Solution is to write approximate descriptions.
Approximate Descriptions

Optimistic Description: $\text{REACH}^+(s, h)$

- Take union of all possible outcomes from all refinements.
- So this includes $\neg\text{Cash} \text{ and } \neg\text{At}(\text{Car, SFOParking})$.
- This overgenerates reachable states.

Pessimistic Description: $\text{REACH}^-(s, h)$

- Only states that satisfy effects from all refinements survive.
- So this does not include $\neg\text{Cash}$ or $\neg\text{At}(\text{Car, SFOParking})$.
- This undergenerates reachable states.

$\text{REACH}^-(s, h) \subseteq \text{REACH}(s, h) \subseteq \text{REACH}^+(s, h)$
Algorithm for Finding a Plan

Two Important Facts:

1. If $\exists s' \in \text{Reach}^-(s, h) \text{ st } s' \models g$, you know $h$ can succeed.
2. If $\neg \exists s' \in \text{Reach}^+(s, h) \text{ st } s' \models g$, you know $h$ will fail!

The Algorithm:

- Do breadth first search as before.
- But now you can stop searching and implement instead when you reach an $h$ where 1. is true.
- And you can drop $h$ (and all its refinements) when 2. is true.
- If 1. and 2. are both false for the current $h$, then you don’t know if $h$ will succeed or fail, but you can find out by refining it.
Summary

- Execution monitoring: checking success of execution
- Replanning: repairing plans in case of failure
- HLAs and HLPs
- Using refinements and preconditions and effects of primitive actions to approximate which states are reachable.
- Such approximate descriptions of HLAs help to inform search and when to refine an HLP so as to reach a goal.
- Next time: Acting under Uncertainty