Informatics 2D – Reasoning and Agents

Semester 2, 2019–2020

Alex Lascarides
alex@inf.ed.ac.uk

Lecture 19 – Planning and Acting in the Real World II
3rd March 2020

Where are we?

Last time ...

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

Today ...

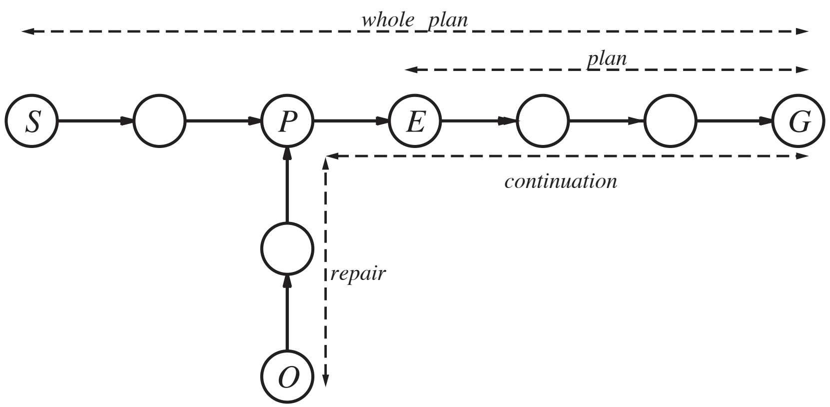
- **Planning and Acting in the Real World II**

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 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

$$\begin{aligned}&\mathit{Refinment}(\mathit{Go}(\mathit{Home}, \mathit{SFO}),\\ &\quad \textsc{Precond:} \mathit{At}(\mathit{Car}, \mathit{Home})\\ &\quad \textsc{Steps:}[\mathit{Drive}(\mathit{Home}, \mathit{SFOLongTermParking})\\ &\qquad\qquad \mathit{Shuttle}(\mathit{SFOLongTermParking}, \mathit{SFO})])\end{aligned}$$

$$\begin{aligned}&\mathit{Refinment}(\mathit{Go}(\mathit{Home}, \mathit{SFO}),\\ &\quad \textsc{Precond:} \mathit{Cash}, \mathit{At}(\mathit{Home})\\ &\quad \textsc{Steps:}[\mathit{Taxi}(\mathit{Home}, \mathit{SFO})])\end{aligned}$$

Refinements can be Recursive

$Refinment(Navigate([a, b], [x, y]),$
PRECOND:$a = x, b = y$
STEPS:[])

$Refinment(Navigate([a, b], [x, y]),$
PRECOND:$Connected([a, b], [a - 1, b])$
STEPS:$[Left, Navigate([a - 1, b], [x, y])])$

$Refinment(Navigate([a, b], [x, y]),$
PRECOND:$Connected([a, b], [a + 1, b])$
STEPS:$[Right, Navigate([a + 1, b], [x, y])])$

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:

$$[Drive(Home, SFOParking), Shuttle(SFOParking, SFO)]$$

- We can capture this if we add to HLAs *themselves* a set of preconditions and effects.

Adding Preconditions and Effects to HLAs

- 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

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

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

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

$$s' \models g \text{ and } s' \in \textsc{Reach}(s, p)$$

- 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 $\textsc{Reach}(s, h)$ 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:
  - $\tilde{+}A$: you can possibly add A (or leave A unchanged)
  - $\tilde{-}A$: you can possibly delete A (or leave A unchanged)
  - $\tilde{\pm}A$: 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

$Refinment(Go(Home, SFO),$
$\quad \textsc{Precond:} At(Car, Home)$
$\quad \textsc{Steps:} [Drive(Home, SFOLongTermParking)$
$\qquad\qquad Shuttle(SFOLongTermParking, SFO)])$

$Refinment(Go(Home, SFO),$
$\quad \textsc{Precond:} Cash, At(Home)$
$\quad \textsc{Steps:} [Taxi(Home, SFO)])$

The 'Primitive' Actions

$Action(Taxi(a, b),$
$\quad \textsc{Precond:} Cash, At(Taxi, a)$
$\quad \textsc{Effect:} \neg Cash, \neg At(Taxi, a), At(Taxi, b))$

$Action(Drive(a, b),$
$\quad \textsc{Precond:} At(Car, a)$
$\quad \textsc{Effect:} \neg At(Car, a), At(Car, b))$

$Action(Shuttle(a, b),$
$\quad \textsc{Precond:} At(Shuttle, a)$
$\quad \textsc{Effect:} \neg At(Shuttle, a), At(Shuttle, b))$

Deriving the PRECONDS and EFFECTS of the HLA

- $\neg Cash$ is EFFECT of one HLA refinement, but not the other.
- So $\widetilde{\neg} Cash$ in HLA EFFECT!

Not so Simple!

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

Approximate Descriptions

Optimistic Description: $\textsc{Reach}^+(s, h)$

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

Pessimistic Description: $\textsc{Reach}^-(s, h)$

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

$$\textsc{Reach}^-(s, h) \subseteq \textsc{Reach}(s, h) \subseteq \textsc{Reach}^+(s, h)$$

Algorithm for Finding a Plan

Two Important Facts:

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