Representing Time

- Agent needs to reason about time as its own actions have to be sequenced over time.
- Time can be modeled in a number of ways:
  - Discrete time: Time can be modeled as jumping from one time point to another.
  - State space: Instead of considering time explicitly, actions can be modeled as mapping from one state to another.

Actions and Planning

- So far, we have considered how to represent the world and reason about it.
- Agents might also want to change the world.
- But, often times it will take a sequence of actions to achieve the goal.
- How does the world change over time?
- So, need to reason about actions, and how they affect the world.
- After all, we want to build agents that can adapt.
- Agents might also want to change the world and reason about it.
- We will assume that Agent is the only one changing the world.
- So, need to consider how actions to achieve the goal.
- But, other times it will take a sequence of actions to achieve the goal.
- We will assume that Agent is the only one changing the world.
- After all, we want to build agents that can adapt.

Overview

- Why Plan?
- Snapshots
- Modifying the World
- Why Plan
Time and Relations

- Why Plan
  ⇒ Modeling the World
  • Strips
  • Planning

Delivery Robot World

- Mail
- Storage
- Parcel
- Key K1
- Door1
- Stairs
- r101 r103 r105 r107 r109 r111 r113 r115 r117 r119 r121 r123 r125 r127 r129 r131

Plan
• Shirts

Why Plan
• Modeling the World

Overview
Primitive Relations

- unlocked(Door): true in a situation if Door is unlocked in that situation.
- sitting at(Obj,Loc): true in a situation if Obj is sitting on the ground (not being carried) at location Loc in the situation. Also used to describe robot's location, where Obj is the robot.
- carrying(Ag,Obj): true in a situation if agent Ag is carrying Obj in that situation. Either carrying or sitting will be true for any object (including robot).

Static Relations

- autonomous(Ag): true if agent Ag can move autonomously.
- opens(Key,Door): true if key Key opens door Door.
- nodoorbetween(Pos1,Pos2): true if position Pos1 is adjacent to position Pos2 and there is no door between them.
- between(Door,Pos1,Pos2): true if Door is between position Pos1 and position Pos2.

This is a slightly different conceptualization of the world than in the textbook. Textbook version has adjacent as being a static & derived relation.

Modeling the Delivery Robot World

- Individuals: rooms, doors, keys, parcels, and the robot.
- Relations:
  - The robot's position
  - The position of packages and keys and locked doors
  - What the robot is holding
  - The position of packages and keys and locked doors

- Actions:
  - Move from room to room
  - Pick up and put down keys and packages
  - Unlock doors (with the appropriate keys)
Initial Situation

- Static Facts:
  - between(door1,o103,lab2).
  - opens(k1,door1).
  - autonomous(rob).
  - nodoorbetween(o109,o103).
  - nodoorbetween(o103,o109).
  - ... nodoorbetween(lab2,o109).

Primative Relations of Initial Situation:
- sittingat(rob,o109).
- sittingat(parcel,storage).
- sittingat(k1,mail).

Actions:
- move(Ag,From,To) => agent Ag moves from location From to adjacent location To. The agent must be sitting at location From.
- pickup(Ag,Obj) => agent Ag picks up Obj. The agent must be at the location that Obj is sitting.
- putdown(Ag,Obj) => the agent Ag puts down Obj. It must be holding Obj.
- unlock(Ag,Door) => agent Ag unlocks Door. It must be outside the door and carrying the key to the door.

Derived Relations:
- adjacent(Pos1,Pos2): true if the robot can move from Pos1 to Pos2 in one step. So, Pos1 and Pos2 must have no door between them or have an unlocked door between them.
  - adjacent(Pos1,Pos2) ← nodoorbetween(Pos1,Pos2).
  - adjacent(Pos1,Pos2) ← doorbetween(Door,Pos1,Pos2) ∧ unlocked(Door).

- at(Obj,Loc): true in a situation if Obj is at Loc
  - at(Obj,Loc) ← sittingat(Obj,Loc).
  - at(Obj,Loc) ← carrying(Agt,Obj) ∧ sittingat(Agt,Loc).

- How at(Obj,Loc) is derived depends on if object is being carried or not.
STRIPS Representation: Idea

- Predicates are static, primitive or derived
- Use normal rules for derived & static predicates
- Must specify how to determine truth values of primitive predicates based on the previous state and the action
- STRIPS: explicitly say what things have changed
- STRIPS assumption: Primitive relations not mentioned in the description of the action stay unchanged

STRIPS Representation

- How can we represent the action?
- State-based view of time
- The actions are external to the logic
- Given a state and an action, the STRIPS representation is used to determine
  - whether the action can be carried out in the state
  - what is true in the resulting state

Overview

- Why Plan
- Modeling the World
  - STRIPS
Example Transitions

- Initial State
  - between(door1,o103,lab2).
  - opens(k1,door1).
  - autonomous(rob).
  - nodoorbetween(o109,o103).
  - nodoorbetween(o103,o109).
  - nodoorbetween(lab2,o109).
  - sitting at(rob,o109).
  - sitting at(parcel,storage).
  - sitting at(k1,mail).

- After move(rob,o109,storage)?

- After pickup(rob,parcel)?

STRIPS Representation of "pickup"

- The action pickup(Ag,Obj) can be defined by:
  - preconditions: autonomous(Ag), Ag ≠ Obj, at(Ag,Pos), sitting at(Obj,Pos)
  - delete list: sitting at(Obj,Pos)
  - add list: carrying(Ag,Obj)

- move(Ag,Pos1,Pos2)
  - preconditions: autonomous(Ag), adjacent(Pos1,Pos2), sitting at(Ag,Pos1)
  - delete list: sitting at(Obj,Pos1)
  - add list: sitting at(Ag,Pos2)

- putdown(Ag,Obj)

- unlock(Ag,Door)
Example Planning

If you want a plan to achieve Rob holding the key \( k_1 \) and being at \( o_{103} \), you can issue the query

\[
\text{Goal: } \text{carrying}(\text{rob}, k_1, S) \land \text{at}(\text{rob}, o_{103}, S)
\]

This has an answer

\[
\begin{align*}
\text{move(rob, o_{109}, o_{103})} \\
\text{move(rob, o_{103}, mail)} \\
\text{pickup(rob, k_1)} \\
\text{move(rob, mail, o_{103})}
\end{align*}
\]

What strategy should you use to find a solution?

Planning

A plan is a sequence of actions that will achieve the goal.

- A goal
- A description of available actions
- An initial world description
- Given

Overview

Why Plan

Modeling the World

Strips

Planning
STRIPS Forward Planner

- Put pair of \(<\text{initialplan, initialworld}>\) in frontier.
- Initialworld could just include primitives (not static nor derived).
- Initialplan is just an empty list.
- Loop: while frontier is not empty:
  - Take out top path/world from frontier.
  - If goal true in top world then stop.
  - Otherwise figure out all possible actions using precondition lists.
  - For each possible action:
    - Apply add-list and delete-list.
    - Append resulting world (and list of actions to get there) to frontier.

What would A* use as g and h?

STRIPS Forward Planner

- Search in the state-space graph, where the nodes represent states and the arcs represent actions.
- Search from initial state to a state that satisfies the goal.
- A complete search strategy (e.g., A* or iterative deepening) is guaranteed to find a solution.
- Branching factor is the number of legal actions. Path length is the number of actions to achieve the goal.
- You usually can't do backward planning in the state space as the goal doesn't uniquely specify a state.