Overview

Why Plan

Modeling the World

Reasoning about Sequences of Actions

Strips

Planning

Actions and Planning

We will assume that Agent is the only one changing the world.

- How does the world change over time?
- So, need to reason about actions and how they affect the world.
- But, often times it will take a sequence of actions to achieve goal.
- After all we want to build systems that can act proactively.
- Agents might also want to change the world and reason about it.
- So far, we have considered how to represent the world.

Planning

Strips

Reasoning about Sequences of Actions

Modeling the World

Why Plan
Static Relations

- robot(Ag): true if Ag is a robot
- opens(Key,Door): true if Key opens 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.
**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.
  
  \[
  \text{adjacent}(\text{Pos1,Pos2}) \leftarrow \neg \text{doorbetween}(\text{Door}, \text{Pos1}, \text{Pos2}).
  \]
  \[
  \text{adjacent}(\text{Pos1,Pos2}) \leftarrow \text{doorbetween}(\text{Door}, \text{Pos1}, \text{Pos2}) \land \text{unlocked}(\text{Door}).
  \]

- **at(Obj,Loc)**: true in a situation if Obj is at Loc. How \( \text{at}(\text{Obj,Loc}) \) is derived depends on if object is being carried or not.
  
  \[
  \text{at}(\text{Obj,Loc}) \leftarrow \text{sitting}.
  \]
  \[
  \text{at}(\text{Obj,Loc}) \leftarrow \text{carrying}(\text{Agt,Obj}) \land \text{sitting}.
  \]

- **unlocked(Door)**: true in a situation if Door is unlocked in the 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 at will be true for any object (including robot).

**Primitive Relations**

- **Door** can be carrying any number of objects.
- **Object** can be carrying any number of objects.
- **Object** in that situation.

\[
\text{carrying}(\text{Ag,Obj}) \leftarrow \text{in a situation if agent Ag is carrying Obj}.
\]

\[
\text{Object}(\text{Obj,Loc}) \leftarrow \text{in a situation if Object is sitting at Loc}.
\]

\[
\text{unlock}(\text{Door}) \leftarrow \text{in a situation if Door is unlocked in the situation}.
\]

\[
\text{before}(\text{Agt,Obj}) \leftarrow \text{in a situation if Agt is before Obj in the situation}.
\]
How do we reason about sequences of actions?

Before determining how we plan, let's focus on just being able to verify a plan accomplished the goal.

Goal: carrying(rob, k1) ∧ at(rob, o103)

How do we know that this plan accomplished the goal?

move(rob, o109, o103)

...
Overview

• Why Plan
• Modeling the World
• Reasoning about Sequences of Actions

⇒ Strips

Planning

Continued

• Actions: move, pickup, unlock
- They are not predicates, that are true or false
- They are things that transform the current world

+ Need to specify if they can be applied to the current world
+ Need to specify how the current world is transformed

• Strategy 1: Strips
- View actions are outside of the logic
- A sequence of actions induces a sequence of worlds

- They are things that transform the current world
- They are not predicates that are true or false

• Strategy 2: Situation Calculus
- View actions are inside of the logic
- They are things that transform the current world
- They are not predicates that are true or false

Key Insight

- They are things that transform the current world
- They are not predicates that are true or false

- A sequence of actions induces a sequence of worlds
- They are things that transform the current world
- They are not predicates that are true or false

Actions: move, pickup, unlock
STRIPS Representation of "pickup"

• The action pickup(Ag,Obj) can be defined by:

  preconditions
  robot(Ag), Ag \neq Obj, at(Ag,Pos), sitting at(Obj,Pos)

  delete list
  sitting at(Obj,Pos)

  add list
  carrying(Ag,Obj)

• move(Ag,Pos1,Pos2)

  preconditions
  robot(Ag), adjacent(Pos1,Pos2), at(Ag,Pos1)

  delete list
  sitting at(Obj,Pos1)

  add list
  sitting at(Ag,Pos2)

• putdown(Ag,Obj):

• unlock(Ag,Door):

Atoms in preconditions might have variables in them.

STRIPS and Theorem Proving

• Types of relations:
  - Preconditions can use static or primitive relations
  - Add/deletes can only use primitive relations

• Theorem proving:
  - Used to find a substitution that makes preconditions true in current world
  - World usually represented as a database with LNA
  - Used to find a substitution that makes preconditions true in current world

• STRIPS assumption:
  - Primitive relations not mentioned in the description of the action stay unchanged
  - Add is a list of primitive relations added to the action
  - Delete is a list of primitive relations no longer true after the action
  - Preconditions: A list of formulas that need to be true for the action to occur
  - Each action has:
    - Preconditions:
      - Each world is represented as a list of formulas
      - Adding an action to a world creates a new world
    - Add is a list of facts to be asserted in the new world
    - Delete is a list of facts to be deasserted in the new world
    - The actions are executed in the order

STRIPS Representation
Planning

• Given
  - a goal
  - a description of available actions
  - an initial world description

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

Overview

• Why Plan
• Modeling the World
• Reasoning about Sequences of Actions
• Strips
  ⇒ Planning

Example Transitions

Initial State

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

After move(rob,o109,storage)?

- sitting at(parcel,storage).
- sitting at(k1,lab2).
STRIPS Forward Planner

- Put pair of `<initial_plan, initial_world>` in frontier.
- Initial_world could just include primitives (not static nor derived).
- Initial_plan 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.
  - Figure out all possible actions using precondition lists.
    + This will tell you the neighboring worlds.
  - 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?

Forward Planner

- The goal does not uniquely specify a state.
- You usually can't do backward planning in the state space as
  the number of actions to achieve goal
  Branching factor is the number of legal actions.
  Path length is
  Guaranteed to find a solution
  A complete search strategy (e.g., A∗ or iterative deepening) is needed.
  Search from initial state to a state that satisfies the goal.
- Nodes will have a complete listing of what is true in world.
- Nodes will have a complete listing of what is true in each world and the arcs represent actions.
- Search in the state-space graph, where the nodes represent
  Forward Planning.

Example Planning

- If you want a plan to achieve Rob holding the key and being at o103, you can issue the query.
  Goal: carrying(rob, k1, S) ∧ at(rob, o103, S).
- This has an answer:
  move(rob, o109, o103)
  move(rob, o103, mail)
  pickup(rob, k1)
  move(rob, mail, o103)