Overview

⇒ Why Plan
• Modeling the World
• Reasoning about Sequences of Actions
• Strips
• Planning
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.
  - After all, we want to build agents that can act purposely.
- But, often times it will take a sequence of actions to achieve a goal.
- So, need to reason about actions, and how they affect the world.
  - How does the world change over time?
- We will assume that Agent is the only one changing the world.
Overview

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

• Two basic types of relations:
  - **Static relations**: truth value do not change
  - **Dynamic relations**: actions can change truth values

• Dynamic relations can be further classified as:
  - **Primitive relations**: truth value can be determined by considering its value in the past and what actions have been performed
  - **Derived relations**: truth value can be derived from other relations

• Why is it useful to distinguish?
  - Static always the same: no need to recompute them at each time point
  - Derived need to be re-derived at each time point (usually from primitives)
  - Only need to reason about how the primitive relations change over time
Modeling the Delivery Robot World

• Individuals: rooms, doors, keys, parcels, and the robot

• Relations:
  + Robot’s position
  + Position of packages, keys, locked doors
  + What robot is holding

• Actions:
  + Move around
  + Pick up and put down keys and packages
  + Unlock doors (with the appropriate keys)
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.
Primitive Relations

- **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)
  - Robot can be carrying any number of objects
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.

  \[
  \text{adjacent}(\text{Pos1}, \text{Pos2}) \leftarrow \text{nodoorbetween}(\text{Pos1}, \text{Pos2}).
  \]

  \[
  \text{adjacent}(\text{Pos1}, \text{Pos2}) \leftarrow \text{doorbetween}(\text{Door}, \text{Pos1}, \text{Pos2}) \land \text{unlocked}(\text{Door})
  \]

• **at(Obj,Loc):** true in a situation if Obj is at Loc

  - How **at(Obj,Loc)** is derived depends on if object being carried or not

    \[
    \text{at}(\text{Obj}, \text{Loc}) \leftarrow \text{sitting	extunderscore at}(\text{Obj}, \text{Loc})
    \]

    \[
    \text{at}(\text{Obj}, \text{Loc}) \leftarrow \text{carrying}(\text{Agt}, \text{Obj}) \land \text{sitting	extunderscore at}(\text{Agt}, \text{Loc})
    \]

  - predicate ‘at’ let’s us reason about where an object is regardless of whether it is being carried or not
## 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.
Initial World

• Static Facts
  between(door1,o103,lab2).
  opens(k1,door1).
  robot(rob).
  nodoorbetween(o109,o103).
  nodoorbetween(o103,o109).
  ...
  nodoorbetween(lab2,o109).

• Primative Relations of Initial World
  sitting_at(rob,o109).
  sitting_at(parcel,storage).
  sitting_at(k1,mail).
Overview

- Why Plan
- Modeling the World
  ⇒ Reasoning about Sequences of Actions
- Strips
- Planning
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 a goal

• Goal: \( \text{carrying(rob,k1)} \land \text{at(rob,o103)} \)

• How do we know that this plan accomplished the goal?
  
  \[
  \text{move(rob,o109,o103)} \\
  \text{move(rob,o103,mail)} \\
  \text{pickup(rob,k1)} \\
  \text{move(rob,mail,o103)}
  \]
Key Insight

• 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 as outside of the logic
  - A sequence of actions induces a sequence of worlds
    + first one is the initial world
    + last one has the goal true in it
• 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
• Strategy 2: Situation Calculus
  - View domain as having world objects in it that can be referred to
    + constant init refers to the initial world
    + static relations are true of
    + primitive relations are true of just some
    + actions cause a to be mapped to another
Overview

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

• STRIPS Representation:
  - The actions are external to the logic
  - Applying an action to a world creates a new world
    + Each world is separate KB: usually Datalog + UNA + negation as failure

• Each action has:
  - **preconditions**: A list of atoms that need to be true for the action to occur
  - **delete list**: A list of primitive relations no longer true after the action
    - Just delete the primitive, do not assert it is not true
  - **add list**: A list of the primitive relations made true by the action

• STRIPS assumption: Primitive relations not mentioned in the description of the action stay unchanged
STRIPS and Theorem Proving

• Atoms in preconditions might have variables in them
  - Those variables might also be in delete and add list

• Theorem proving
  - used to find substitution that makes preconditions true in current world
    - world usually represented as Datalog + UNA + negation as failure
  - Substitution used to set variables in delete and add list!

• Types of relations:
  - Precondition can use static, derived or primitive relations
  - Add/delete list can only use primitive relations
STRIPS Representation of “pickup”

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

  | preconditions | robot(Ag), Ag≠Obj, at(Ag,Pos), sitting at(Obj,Pos) |
  | preconditions | robot(Ag) | Ag≠Obj | at(Ag,Pos) | sitting at(Obj,Pos) | adding list | carrying(Ag,Obj) |
  | delete list | sitting at(Obj,Pos) |
  | add list | sitting at(Ag,Pos) |

- **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):**
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)?
Overview

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

⇒ Planning
Planning

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

• A plan is a sequence of actions that will achieve the goal
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
  
  Goal: carrying(rob, k1, S) $\land$ at(rob, o_{103}, S)

• This has an answer
  
  $move(rob, o_{109}, o_{103})$
  $move(rob, o_{103}, mail)$
  $pickup(rob, k1)$
  $move(rob, mail, o_{103})$

• What strategy should you use to find a solution?
Forward Planner

• Search in the state-space graph, where the nodes represent worlds and the arcs represent actions
  - Nodes will have a complete listing of what is true in world
• 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 goal
• You usually can’t do backward planning in the state space, as the goal doesn’t uniquely specify a state.
STRIPS Forward Planner

• Put pair of <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
  - 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?