Representing Time

• Agent needs to reason about time as its own actions have to be sequenced over time.

- Discrete Time: Time can be modeled as jumping from one time point to another. Each action takes a number of discrete time steps.
- State Space: Instead of considering time explicitly, you can consider actions as mapping from one state to another.

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

Questions:
- How does the world change over time?
- So, need to reason about actions, and how they affect the world.
- But, other times it will take a sequence of actions to achieve the goal.
- After all, we want to build agents that can act purposely.
- Agents might also want to change the world and reason about it.

So far, we have considered how to represent the world.

Actions and Planning

Planning

- Shopping
- Modifying the World

≡ Why Plan
Time and Relations

Why is it useful to distinguish:

- **Static relations**: truth value does not depend on time
- **Dynamic relations**: truth values depend on time

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 relations**
  - Value can be derived from other relations
  - Value in the past and what actions have been performed

- **Dynamic relations**
  - Value depends on time
  - Derived from other relations

Why is it useful to distinguish between two basic types of relations:

Il helps to distinguish between two basic types of relations:

- Static
- Dynamic
**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.
- **open(Door):** true if Key opens door Door.
- **autonomous(Ag):** true if agent Ag can move autonomously.

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
- **Actions:**
  - Move from room to room
  - Pick up and put down keys and parcels
  - Unlock doors (with the appropriate key)
- **Relations:**
  - The position of packages and keys and locked doors
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
  - sitting at(rob, o109).
  - sitting at(parcel, storage).
  - sitting at(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 carry 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.

- at(Obj, Loc)
  true in a situation if Obj is at Loc.
  - at(Obj, Loc) ← sitting at(Obj, Loc).
  - at(Obj, Loc) ← carrying(Agt, Obj) ∧ sitting at(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 assumption: Primitive relations not mentioned in the description of the action stay unchanged.
- STRIPS: explicitly say what things have changed.

How can we represent the action?

- STRIPS Representation: 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.
- The actions are extended to the logic.

Overview

- Planning
- STRIPS
- Modelling the World
- Why Plan
### 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 \neq Obj, at(Ag,Pos), sitting at(Obj,Pos)`
  - **delete list**: `sitting at(Obj,Pos)`
  - **add list**: `carrying(Ag,Obj)`

### STRIPS Representation of an Action

- The STRIPS representation for an action consists of:
  - **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
  - **add list**: A list of atoms that need to be true after the action
  - **invariants**: A list of atoms that must be true forever

- The theorem proving used to find an instantiation that makes the preconditions true.
- That instantiation used to set variables in the delete and add lists.
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:

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

  What strategy should you use to find a solution?

Planning

- Given:

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

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

Overview
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
    - Take out top path/world from frontier
  - Loop: while frontier is not empty
    - Full path to <initial plan, initial world> in frontier

What would A∗ use as g and h?

Forward Planner

- Search in the state-space graph, where the nodes represent states in the state space, and the arcs represent actions.
- Search from initial state to a state that satisfies the goal.
- A complete search strategy (e.g., A*) or heuristic search 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.