Module 9

Planning

9.1 Instructional Objective

- The students should understand the formulation of planning problems
- The student should understand the difference between problem solving and planning and the need for knowledge representation in large scale problem solving
- Students should understand the STRIPS planning language
- **Students should be able to represent a real life planning problem using STRIPS operators**
- Students should understand planning using situation calculus and the related frame problems
- Students should understand the formulation of planning as a search problem
- Students should learn the following planning algorithms
    - Situation space planning
    - Plan space planning
    - Progression planning
    - Regression planning
- The student should understand the difficulties of full commitment planning
- Students should understand the necessity of least commitment
- Students should learn partial order planning algorithms

At the end of this lesson the student should be able to do the following:

- **Represent a planning problem in STRIPS language**
- **Use a suitable planning algorithm to solve the problem.**

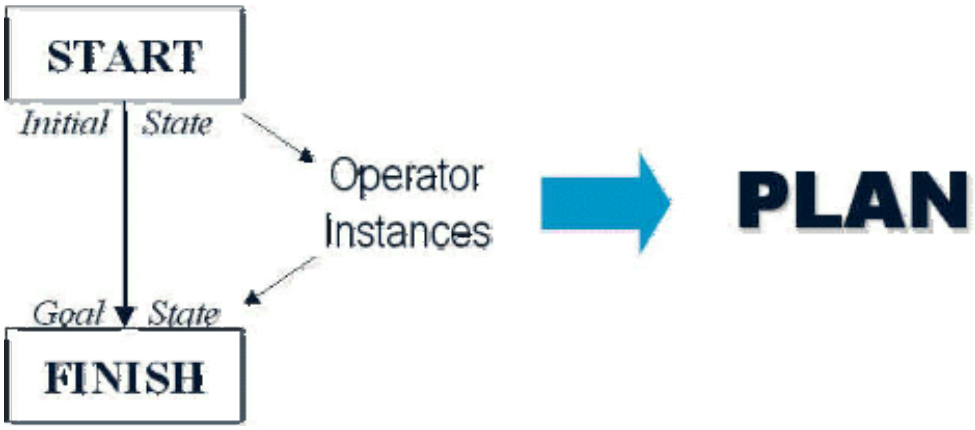
Lesson 22

Logic based planning

Version 2 CSE IIT, Kharagpur

9. 1 Introduction to Planning

The purpose of planning is to find a sequence of actions that achieves a given goal when performed starting in a given state. In other words, given a set of operator instances (defining the possible primitive actions by the agent), an initial state description, and a goal state description or predicate, the planning agent computes a plan.



What is a plan? A sequence of operator instances, such that "executing" them in the initial state will change the world to a state satisfying the goal state description. Goals are usually specified as a conjunction of goals to be achieved.

9.1.1 A Simple Planning Agent:

Earlier we saw that **problem-solving agents** are able to plan ahead - to consider the consequences of *sequences* of actions - before acting. We also saw that a **knowledge-based agents** can select actions based on explicit, logical representations of the current state and the effects of actions. This allows the agent to succeed in complex, inaccessible environments that are too difficult for a problem-solving agent

Problem Solving Agents + Knowledge-based Agents = Planning Agents

In this module, we put these two ideas together to build **planning agents.** At the most abstract level, the task of planning is the same as problem solving. Planning can be viewed as a type of problem solving in which the agent uses beliefs about actions and their consequences to search for a solution over the more abstract space of plans, rather than over the space of situations

6.1.2 Algorithm of a simple planning agent:

1. Generate a goal to achieve
2. Construct a plan to achieve goal from current state
3. Execute plan until finished
4. Begin again with new goal

The agent first generates a goal to achieve, and then constructs a plan to achieve it from the current state. Once it has a plan, it keeps executing it until the plan is finished, then begins again with a new goal.

This is illustrated in the following pseudocode:

```
function SIMPLE-PLANNING-AGENT (percept) returns an action
  static: KB, a knowledge base(includes action descriptions)
          p , a plan, initially NoPlan
          t, a counter, initially 0, indicating time
   local variables: G, a goal
                    current, a current state description

  TELL(KB, MAKE-PERCEPT-SENTENCE (percept,t))
  current <- STATE-DESCRIPTION(KB, t)
   if p = NoPlan then
      G <- ASK (KB, MAKE-GOAL-QUERY(t))
      p <- IDEAL-PLANNER(current, G, KB)
   if p = NoPlan or p is empty then action <- NoOp
   else
      action <- FIRST(p)
      p <- REST(p)
   TELL(KB, MAKE-ACTION-SENTENCE(action,t))
   t <- t+ 1
   return action
```

Functions:
STATE-DESCRIPTION: uses a percept as input and returns the description of the initial state in a format required for the planner.
IDEAL-PLANNER: is the planning algorithm and can be any planner described in these notes or chapter 11.
MAKE-GOAL-QUERY: asks the knowledge base what the next goal will be.

The agent in the above algorithm has to check if the goal is feasible or if the complete plan is empty. If the goal is not feasible, it ignores it and tries another goal. If the complete plan was empty then the initial state was also the goal state.

Assumptions:
A simple planning agent create and use plans based on the following assumptions:

- **Atomic time**: each action is indivisible
- **No concurrent actions** allowed
- **Deterministic actions**: result of each actions is completely determined by the definition of the action, and there is no uncertainty in performing it in the world.
- **Agent is the sole cause of change** in the world.
- **Agent is omniscient**: has complete knowledge of the state of the world

- **Closed world assumption**: everything known to be true in the world is included in a state description. Anything not listed is false

9.1.3 Problem Solving vs. Planning

A simple planning agent is very similar to problem-solving agents in that it constructs plans that achieve its goals, and then executes them. The limitations of the problem-solving approach motivates the design of planning systems.
To solve a planning problem using a state-space search approach we would let the:

- initial state = initial situation
- goal-test predicate = goal state description
- successor function computed from the set of operators
- once a goal is found, solution plan is the sequence of operators in the path from the start node to the goal node

In searches, operators are used simply to generate successor states and we can not look "inside" an operator to see how it's defined. The goal-test predicate also is used as a "black box" to test if a state is a goal or not. The search cannot use properties of how a goal is defined in order to reason about finding path to that goal. *Hence this approach is all algorithm and representation weak.*

Planning is considered different from problem solving because of the difference in the way they represent states, goals, actions, and the differences in the way they construct action sequences.

Remember the search-based problem solver had four basic elements:

- Representations of actions: programs that develop successor state descriptions which represent actions.

- Representation of state: every state description is complete. This is because a complete description of the initial state is given, and actions are represented by a program that creates complete state descriptions.

- Representation of goals: a problem solving agent has only information about it's goal, which is in terms of a goal test and the heuristic function.

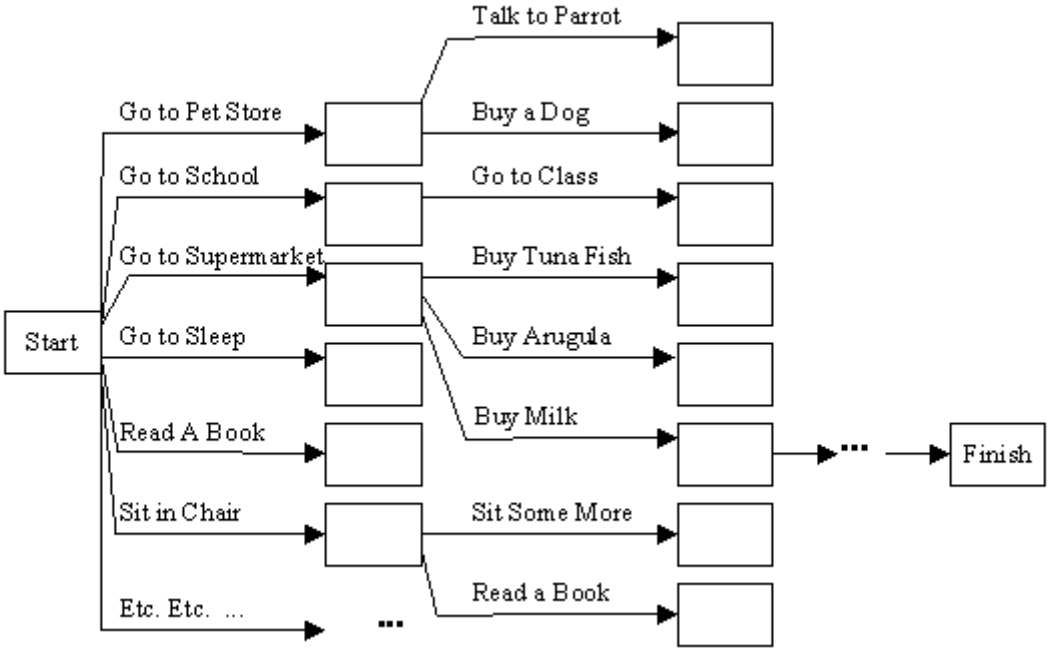
- Representation of plans: in problem solving, the solution is a sequence of actions.

In a simple problem: *"Get a quart of milk and a bunch of bananas and a variable speed cordless drill"* for a problem solving exercise we need to specify:

Initial State: the agent is at home without any objects that he is wanting.

Operator Set: everything the agent can do.

Heuristic function: the # of things that have not yet been acquired.



Problems with Problem solving agent:

- It is evident from the above figure that the actual branching factor would be in the thousands or millions. The heuristic evaluation function can only choose states to determine which one is closer to the goal. It cannot eliminate actions from consideration. The agent makes guesses by considering actions and the evaluation function ranks those guesses. The agent picks the best guess, but then has no idea what to try next and therefore starts guessing again.

- It considers sequences of actions beginning from the initial state. The agent is forced to decide what to do in the initial state first, where possible choices are to go to any of the next places. Until the agent decides how to acquire the objects, it can't decide where to go.

Planning emphasizes what is in operator and goal representations. There are three key ideas behind planning:

- *to "open up" the representations* of state, goals, and operators so that a reasoner can more intelligently select actions when they are needed
- *the planner is free to add actions to the plan* wherever they are needed, rather than in an incremental sequence starting at the initial state
- *most parts of the world are independent of most other parts* which makes it feasible to take a conjunctive goal and solve it with a divide-and-conquer strategy

9.2 Logic Based Planning

9.2.1 Situation Calculus

Situation calculus is a version of first-order-logic (FOL) that is augmented so that it can reason about actions in time.

- Add *situation variables* to specify time. A **situation** is a snapshot of the world at an interval of time when nothing changes
- Add a special predicate *holds(f,s)* that means "f is true in situation s"
- Add a function *result(a,s)* that maps the current situation *s* into a new situation as a result of performing action *a*.

Example:
The action *"agent-walks-to-location-y"* could be represented by:
(Ax)(Ay)(As)(at(Agent,x,s) -> at(Agent,y,result(walk(y),s))

9.2.1.1 Frame Problem

Actions in Situation Calculus describe what they *change*, not what they *don't change*. So, how do we know what's still true in the new situation? To fix this problem we add a set of frame axioms that explicitly state what doesn't change.

Example:
When the agent walks to a location, the locations of most other objects in the world do not change. So for each object (i.e. a bunch of bananas), add an axiom like:
(Ax)(Ay)(As)(at(Bananas,x,s) -> at(Bananas,x,result(walk(y),s))

9.2.2 Solving planning problems using situation calculus

Using situation calculus a planning algorithm is represented by logical sentences that describe the three main parts of a problem. This is illustrated using the problem described in AI: A Modern Approach by Russell & Norvig.

1. Initial State: a logical sentence of a situation So.

   For the shopping problem it could be:
   At(Home,So) /\ -Have(Milk,So) /\ -Have(Bananas,So) /\ -Have(Drill,So)

2. Goal State: a logical query asking for a suitable situation.

   For the shopping problem, a query is:
   $ s At(Home,s) /\ Have(Milk,s) /\ Have(Bananas,s) /\ Have(Drill,s)

3. Operators: a set of descriptions of actions.
   Ex. A successor-state action with the Buy(Milk) action

" a,s Have(Milk,Result(a,s)) <=> [(a = Buy(milk) /\ At(supermarket,s) \/ (Have(Milk,s) /\ a NOT= Drop(Milk))]

Result(a,s) names the situation resulting from being in s while executing action a.
It will be useful to handle action sequences than just a single action. We can use Result' (l,s) to mean the situation resulting from executing l (the sequence of actions) beginning in s. Result' is described by saying that an empty sequence of actions will not effect the situation, and a non-empty sequence of actions is the same as doing the first action and then finishing the rest of the actions from the resulting situation.

Empty sequence of actions is represented as:
" s Result' ([],s) = s

Non-empty sequence of actions is represented as:
" a,p,s Result' ([a|p],s) = Result' (p, Result(a,s))

p is a plan and when applied to start state So, develops a situation which satisfies the goal query.

p could be:
At(home, Result' (p,So)) /\ Have(Milk, Result' (p,So)) /\ Have(Bananas, Result' (p,So)) /\ Have(Drill,Result' (p,So))

The goal query could be:

G = [Go(Supermarket), Buy(Milk), Buy(Banana), Go(Hardware Store), Buy(Drill), Go(Home)]

In theory, there isn't much more to say. Doing planning this way doesn't guarantee a practical solution. Plan p guarantees to achieve the goal but not necessarily efficiently. This approach is representation strong, but reasoning weak because of the poor efficiency of resolution.

Practical Planning:

To make planning practical we need to do 2 things:

1. Restrict the language with which we define problems. With a restrictive language, there are fewer possible solutions to search through.

Use a special-purpose algorithm called a **planner** rather than a general-purpose theorem prover to search for a solution.