Review - Prolog

• Prolog can be looked at as both a theorem prover and an interpreter.

• Programmer controls Prolog’s search for a proof by choosing
  – Order of clauses
  – Order of subgoals within a clause
  – Overall formulation of the logic.

• Programmer also adds explicit control primitives such as cut.
Cut

- ! Always succeeds, but if you fail back to it, then goal that matched head of the clause fails. E.g., in
  a(X):-b(x), !, c(x).
  If c(x) fails the a(X) goal fails
  – Ignore other ways for b(X) to succeed
  – Ignore other clauses for a(X).

Negation by failure

- The goal \+(G) succeeds whenever the goal G fails.
- Variables in G either
  – Must be fully instantiated (they must have a value and, after doing all substitutions, that value must not have any variables in it), or
  – Must appear only in G and not anywhere else in the clause where \+(G) appears.
Iterative deepening

• Add a depth-bound to all predicates that are recursive (even indirectly, as in A calls B calls A).
  – Check bound > 0 in each rule
  – In recursive subgoals use a bound 1 less
• Write a search predicate that calls your top-level goal with a bound of 1, then 2, etc.

Planning

• We have seen that it is possible to express (much of) what an agent needed to know about the world in FOPC and answer (most) questions the agent needed answers to by theorem proving.
• We also saw that the representation and inference processes are long and slow.
• It is often better to use less general but more tractable representation and inference methods.
• One important application of this idea is the problem of planning
Planning

- Given: start and goal states, and a set of operations
- Find: a sequence of operations that gets from start to goal

Why Not Just Search the Graph of States?

- Consider the following problem:
  - Get a quart of milk, a bunch of bananas, and a variable-speed cordless drill.
Problems

• Branching factor is too large. Agent must effectively guess at each possible action and then evaluate it in terms of the goal.
  – It cannot eliminate actions from consideration.
• Must start building action sequences from initial state.
  – Since actions in the sequence interact, planning is like constraint satisfaction, and execution-order is often not the best ordering of decisions.

Why Not Just Use Prolog?

• Problems:
  – logical inference is semidecidable.
  – unguided logical inference is inefficient.
With a Planner:

• We use first-order logic to represent states, goals, and actions
  – This allows the planner access to the internals of these representations, e.g., the preconditions of an action
• We restrict the form of FOL used to define the problems, resulting in a smaller search space.
• Actions can be added at any point in the sequence of the plan.
• We rely on divide and conquer, assuming that most plan parts are independent.
• We use a special-purpose planner, rather than a general theorem-prover.

The Strips Language for Planning

• developed at SRI in 1970 (STanford Research Institute Problem Solver).
• states are represented as conjunctions of function-free ground literals, e.g.,
  At(Home) ^ ~ Have(Milk) ^ ~Have(Bananas) ^ ...
  – states may be incomplete - positive literals not mentioned are assumed to be false.
• goals are conjunctions of literals, and can contain variables.
Representing Actions in Strips

- action description
- precondition
- effect

Go(there)

At(here), Path(here, there)
At(there), ~At(here)

Situation Space vs. Plan Space

- Situation Space Planner
  - searches through the space of possible situations.
  - if it searches forward, it is called a progression planner.
  - if it searches backward, it is called a regression planner.
- Plan Space Planner
  - starting with an incomplete, partial plan, we examine ways of expanding the plan until we arrive at a solution.
  - operators can add a step, impose an ordering on steps, or instantiate a variable.
How do We Represent Plans?

Example problem: putting on your shoes
- goal: RightShoeOn LeftShoeOn
- operators:
  - Op(ACTION: RightShoe, PRECOND: RightSockOn, EFFECT: RightShoeOn)
  - Op(ACTION: RightSock, EFFECT: RightSockOn)
  - Op(ACTION: LeftShoe, PRECOND: LeftSockOn, EFFECT: LeftShoeOn)
  - Op(ACTION: LeftSock, EFFECT: LeftSockOn)

Least commitment
- The solution plan consists of: RightShoe and LeftShoe, but in which order should they be specified?
  - principal of least commitment: only make choices that are essential and leave the rest unspecified.
  - Application here: allow specification of some but not all order relationships, i.e. a partial order

Diagram:
```
  LeftShoe
   /     \
  /       \
RightShoe
```
Components of a Plan

- a set of steps.
- a set of step ordering constraints, $S_i < S_j$
  - read "Si sometime before Sj".
- a set of variable binding constraints.
- a set of causal links, written $S_i \longrightarrow^c S_j$
  - read "Si achieves c for Sj".
- Initial plan consists of two steps: Start and Finish,
  Start $<$ Finish
  - Start has no preconditions and its effect is to add the propositions that are true in the initial state.
  - Finish has the goal state as its precondition and no effects.

Initial Plan

- Shoes and Socks Problem - Initial Plan

(a) Start
   \[ Initial State \]
   \[ Goal State \]
   \[ Finish \]

(b) Start
   \[ LeftShoeOn, RightShoeOn \]
   \[ Finish \]
Solutions

- A solution must be complete and consistent.
- A plan is complete if:
  - every precondition of every step is achieved by some other step.
    - formally, step $S_i$ achieves a precondition $c$ of step $S_j$ if
      1. $S_i$ precedes $S_j$ and $c$ is in $\text{EFFECTS}(S_i)$, and
      2. there is no step $S_k$ such that $\neg c$ is in $\text{EFFECTS}(S_k)$, and that $S_i, \ldots, S_k, \ldots, S_j$ is a possible sequence of steps
- A plan is consistent if:
  - there are no conflicts in the ordering or binding constraints.
  - I.e. never have $S_i$ precedes $S_k$ and $S_k$ precedes $S_i$ or $v=A$ and $v=B$
A Partial-Order Planning

Example

• initial state:
  − Op(ACTION:Start,
    EFFECT:At(Home) ^ Sells(HWS,Drill)
    ^ Sells(SM,Milk),Sells(SM,Banana))

• goal state:
  − Op(ACTION:Finish
    PRECOND:Have(Drill) ^ Have(Milk) ^ Have(Banana)
    ^ At(Home))

• actions:
  − Op(ACTION:Go(there),PRECOND:At(here),
    EFFECT:At(there) ^ ~ At(here))
  − Op(ACTION:Buy(x),PRECOND:At(Store) ^ Sells(Store,x),
    EFFECT:Have(x))

Initial Plan
How do we elaborate on it?

- Partial plan achieves 3 of 4 preconditions for Finish.
  - Heavy arrows indicate causal links, while light arrows indicate ordering.

Next step

- Refining partial plan by adding causal links to achieve the Sells preconditions of the Buy steps.
Extending the Plan

- Choose Go actions that achieve the At preconditions of the Buy actions.
  - We still haven't performed any search!!

- Achieving the preconditions of the two Go actions by linking them to the At(Home) condition in the initial state.
  - What's wrong with this plan?
Protecting Causal Links

- The Go(HWS) step adds the AT(HWS) condition, but deletes the At(Home). This clobbers the precondition of Go(SM)
- We can protect causal links by insisting that steps that might delete the protected condition are ordered before (demotion) or after (promotion) the protected link.

Protection added

Order constraint
STRIPS Domain: Robot Problem Solving

- state description
  - CLEAR(x) means block x has a clear top.
  - ON(x,y) means block x is on top of block y.
  - HANDEMPLOYEE means that the robot's hand is empty.
  - ONTABLE(x) means that block x is resting on the table.
  - HOLDING(x) means that the robot is holding block x.

- goal description
  - e.g., ON(B,C) ∧ ON(A,B), describes a family of world states that satisfy the goal.
Restrictions to FOL

- goal (and subgoal) expressions must consist of a conjunction of literals (atomic sentences).
- any variables in goal expressions must be existentially quantified.
- initial and intermediate state descriptions must consist of a conjunction of ground literals.

Operators:

- pickup(x)
  - PRECONDITIONS: ONTABLE(x), CLEAR(x), HANDEMPty
  - DELETIONS: ONTABLE(x), CLEAR(x), HANDEMPty
  - ADDITIONS: HOLDING(x)

- putdown(x)
  - PRECONDITIONS: HOLDING(x)
  - DELETIONS: HOLDING(x)
  - ADDITIONS: ONTABLE(x), CLEAR(x), HANDEMPty
Operators:

- **stack**(x,y)
  - PRECONDITIONS: HOLDING(x), CLEAR(y)
  - DELETIONS: HOLDING(x), CLEAR(y)
  - ADDITIONS: HANDEMPY, ON(x,y), CLEAR(x)

- **unstack**(x,y)
  - PRECONDITIONS: HANDEMPY, CLEAR(x), ON(x,y)
  - DELETIONS: HANDEMPY, CLEAR(x), ON(x,y)
  - ADDITIONS: HOLDING(x), CLEAR(y)

Regression Planning

- progression search not feasible for more complex problems due to prohibitive size of search space.
- Regression planning uses a process of goal regression:
  - To regress an assertion through an operator we ask, “What needs to be true before the operation in order to ensure that the assertion will be true after the operation?”
Regression Planning

- in regression planning, we apply the rules backwards to produce subgoal expressions.
  - unify one of the literals in the (sub)goal expression with one of the literals in the add list of the rule.
  - subgoal expression is created by regressing the other (nonmatched) literals in the goal expression through the instantiated rule,
  - and conjoining these with the preconditions of the instantiated rule.

Examples

- Goal: \([\text{ON}(A, B) \land \text{ON}(B, C)]\)
  - Unifies with \(\text{ON}(A, B)\) unifies with \(\text{ON}(X, Y)\) effect of stack, with \(A/X, B/y\)
  - Regress rest of goal, i.e., \(\text{ON}(B, C)\) through \(\text{stack}(A, B)\), giving \(\text{ON}(B, C)\).
  - Add preconditions \(\text{HOLDING}(A), \text{CLEAR}(B)\)
  - to yield the subgoal:
    - \([\text{ON}(B, C) \land \text{HOLDING}(A) \land \text{CLEAR}(B)]\)

- Goal: \(\text{CLEAR}(A)\)
  - Unifies with \(\text{CLEAR}(y)\) effect of \(\text{unstack}(x, y)\)
  - Add preconditions to get subgoal \(\text{HANDEMPHY} \land \text{CLEAR}(x) \land \text{ON}(x, A)\).
Examples

• Goal: \([\text{CLEAR}(A) \land \text{HANDEMPNT}]\)
  – Unifies with \(\text{CLEAR}(y)\) effect of \(\text{unstack}(x, y)\)
  – But when we try to regress \(\text{HANDEMPNT}\) through \(\text{clear}(A)\) we fail, because \(\text{HANDEMPNT}\) is the delete list of \(\text{clear}\).

But What About Interacting Subgoals?

• Goal stack planning
  – maintain a stack of goals and focus on solving the top goal of the stack.
  – when top goal is eliminated, pop it and apply substitutions to expressions underneath it on the stack.
  – if goal is compound goal, add each subgoal in some order to the top of the stack \(\text{ABOVE}\) the compound goal.
  – when top item on stack is a operation, it means that the preconditions of the operation have been satisfied. Popping the operation consists of applying the operation to the current state description.