Title: Planning

AIMA: Chapter 11 (Sections 11.1–11.4)

Introduction to Artificial Intelligence

CSCE 476-876, Spring 2000

www.cse.unl.edu/~cse476

Berthe Y. Choueiry

choueiry@cse.unl.edu, (402)472-5444

Outline:

A Planning agent, like a problem-solving agent:

— builds plans to achieve goals, then

— executes them

Planning differ from problem-solving:

— uses logical representations for $\left\{\begin{array}{l}\text{goals}\\\text{states}\\\text{actions}\end{array}\right.$

— the way solutions are represented

— the way search is conducted

Partial-order planning (POP) algorithm:

— finds one solutions

— is complete and sound

POP is flexible enough to handle complicated domains

Problem solving: representation/design decisions

Actions : represented by programs that generate successor-state descriptions

States : (1) all state descriptions are complete

— a complete description of the initial state

— Programs of actions generate complete descriptions of new states

(2) typically, a state is a simple data structure (*e.g.*, 8-puzzle, n-queens, etc.)

(3) used to generate a successor, evaluate a heuristic function, and test goal

Goals : big picture only (*i.e.* test goal + heuristic function) black-box, no detail

Plans : a solution is a sequence of actions, typically "linearly" constructed

Shortcomings?

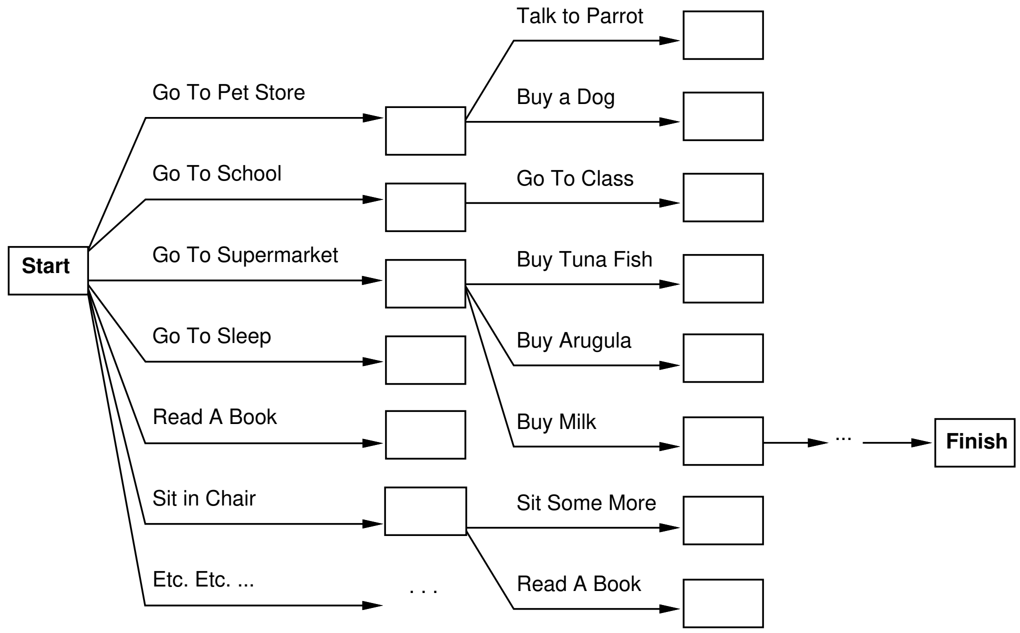
Example: Shopping

"Get a quart of milk, a bunch of bananas and a cordless drill."

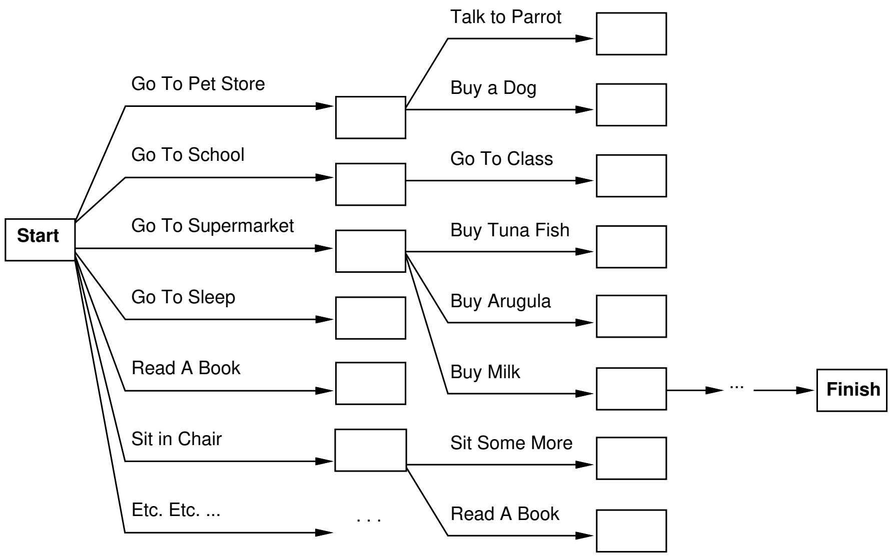
Initial state: "at home without any of these objects"

Operator set: all things the agent can do.

Heuristic function: # things not yet acquired



Branching factor? depth of solution?



Heuristic function: { chooses the states closer to the goal; cannot <u>eliminate</u> actions from consideration }

Starting from initial state:

(1) Go places then (2) acquire objects vs.

(1) Want to acquire, so (2) target the places where things can be acquired (don't go to school, or sleep)

Planning vs. problem solving (I)

1. "Open up the representation of states, goals and actions"
2. Add actions as they are needed, relax requirement for sequential construction of solutions
3. Decomposability: develop subplans independently, then merge them

Planning vs. problem solving (II)

1– "Open up the representation of states, goals and actions"

States and goals: sets of FOL sentences

Actions: logical descriptions of preconditions and effects

Example: Goal includes $Have(Milk)$, then an action $Buy(x)$ is relevant but not $ReadaBook$

2– Add actions as they are needed, do not concentrate on completing a partial solution

Consider $Buy(Milk)$ even before choosing from where to buy it, how to get there, etc.

State description need not be complete: $At(Supermarket)$ with or without bananas, drill, etc..

Planning vs. problem solving (III)

3– Decomposability: most parts of the world are independent of most other parts

Example

- Subplan 1: "get milk and bananas" (*e.g.*, go to Hy-Vee)
- Subplan 2 "get a drill" (*e.g.*, ask ur neighbor)

then put subplans together

— Works as long as subplans do not interfere (*e.g.*, out of money money/time, shops closed)

— How about the 8-puzzle problem?

	Search	**Planning**
States	Lisp data structures	Logical sentences
Actions	Lisp code	Preconditions/outcomes
Goal	Lisp code	Logical sentence (conjunction)
Plan	Sequence from s_0	Constraints on actions

Planning in Situation Calculus (I)

Planning as a logical inference problem, using situation calculus

Initial state : a sentence about S_0

$$At(Home, S_0) \wedge \neg Have(Milk, S_0) \wedge \neg Have(Bananas, S_0) \wedge \neg Have(Drill, S_0)$$

Goal state : a logical query asking for "suitable situations"

$$\exists s At(Home, s) \wedge Have(Milk, s) \wedge Have(Bananas, s) \wedge Have(Drill, s)$$

Operators : action descriptions, such as a successor-state axiom

$$\forall a, s Have(Milk, Result(a, s)) \Leftrightarrow \quad [(a = Buy(Milk) \wedge At(Supermarket, s) \vee (Have(Milk, s) \wedge (a \neq Drop(Milk))]$$

Reminder, in situation calculus: $Result(action, s_i) = s_{i+1}$

Planning in Situation Calculus (II)

$\longrightarrow$ Situation calculus, logical inference is good,
but have problems: complexity, semidecidable, etc..

1. Restrict the language. Always based on logic, with a well-defined syntax, semantics, and proof theory.
   Example: STRIPS
2. Special-purpose algorithm: **planner**

Basic representations for Planning

STRIPS: expressive and 'efficient' [*argh*] language

1. Representations for states and goals
2. Representations for actions
3. Situation space and plan space
4. Representations for plans

STRIPS: States and goals are inputs

States: function-free ground literals

i.e., predicates applied to constant symbols, possibly negated

$$At(Home) \wedge \neg Have(Milk) \wedge \neg Have(Bananas) \wedge \neg Have(Drill) \ldots$$

$\longrightarrow$ **State description need not be complete**

Typically, planner keeps track of only small parts of states representation, keeps track of changes only

$\longrightarrow$ **"Negation-as-failure"-like principle**

if state description does not mention a given positive literal, then literal assumed to be false

Goals: conjunction of literals

$$At(Home) \wedge Have(Milk) \wedge Have(Bananas) \wedge Have(Drill)$$

may contain variables: $At(x) \wedge Sells(x, Milk)$

$\rightarrow$ variables existentially quantified

Planner: asking for a sequence of actions to the goal

Theorem prover: asking whether the sentence is entailed by KB

STRIPS: actions (I)

Operator, 3 components: { action description, precondition, effect}

Action description : returned to the environment to act upon.
In planner: a name.

Precondition : conjunction of atoms (*i.e.*, positive literals)
must be true so that the operator can be applied

Effect : conjunction of literals (*i.e.*, positive or negative)
describing change as a result of applying the operator
(traditionally, `add-list` & `delete-list`)

$$\begin{aligned} Op \quad & (Action : Go(there), \\ & Precondition : At(here) \wedge Path(here, there), \\ & Effect : At(there) \wedge \neg At(here)) \end{aligned}$$

→ Conditions on precondition and effect are relaxed in more advanced planners

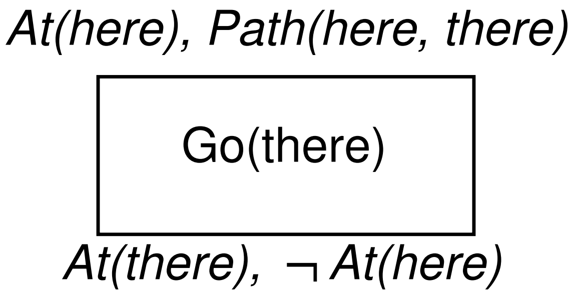
STRIPS: actions(II)

$Op \quad (Action : Go(there),$

$\qquad Precondition : At(here) \wedge Path(here, there),$

$\qquad Effect : At(there) \wedge \neg At(here))$

Graphical notation:



Note: STRIPS does not have an explicit situation variable
($e.g.$, s_i, before, and s_{i+1}, after)

STRIPS: actions(III)

Operator **schema** (family of operators): operator with variables

$$
\begin{aligned}
Op \quad & (Action : Go(x), \\
& Precondition : At(y) \land Path(y, x), \\
& Effect : At(x) \land \neg At(y))
\end{aligned}
$$

Usually, only fully instantiated operators are executable
→ Planner has to make sure all variables are instantiated

STRIPS: actions(IV)

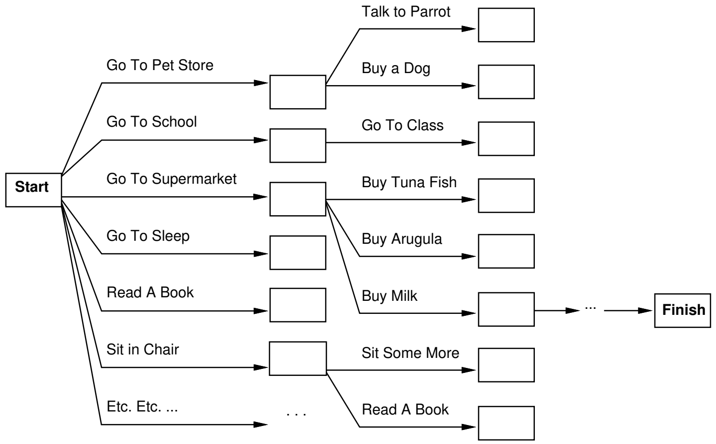
An operator o is applicable in a state s
if there is a way to instantiate variables in o
so that all literals in the precondition are true in s

$$Precond(o) \subseteq s$$

In $Result(o, s)$:
— All positive literals in $\mathit{Effect}(o)$
— All literals that held in s, except those that are negative in $\mathit{Effect}(o)$

Progression/regression planners: Situation space

Situation space planner: search in the space of situations of the world:



Generate a path in the situation space using any search algorithm
→ yields a solution to the planning problem

⟶ **progression** planner
⟶ alternative, **regression** planner: possible and desirable
Start from goal state (few literals) instead of initial state (too many literals)

Plan space

Planning search: search in the space of possible plans

Start with a partial plan, gradually move from incomplete/vague plans to complete, correct plans

Operators on partial plans:
- add a step
- impose an ordering between steps
- instantiate a previously unbound variable

√ **Refinement operators**: add constraints to a partial plan
gradually eliminate plans

× **Modification operators**: start with incorrect plans, then
debug them

Partial order plans (I)

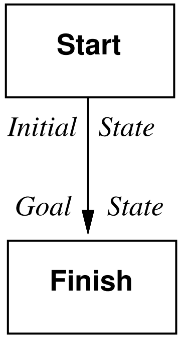
Example: put on your shoes

Actions: RightSock, RightShoe, LeftSock, LeftShoe

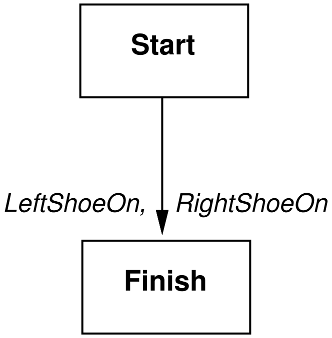
Initial plan: two steps, Start $\prec$ Finish

Start: Precondition: none
Effect: add the literals of initial state

Finish: Precondition: goal state
Effect: none



(a)

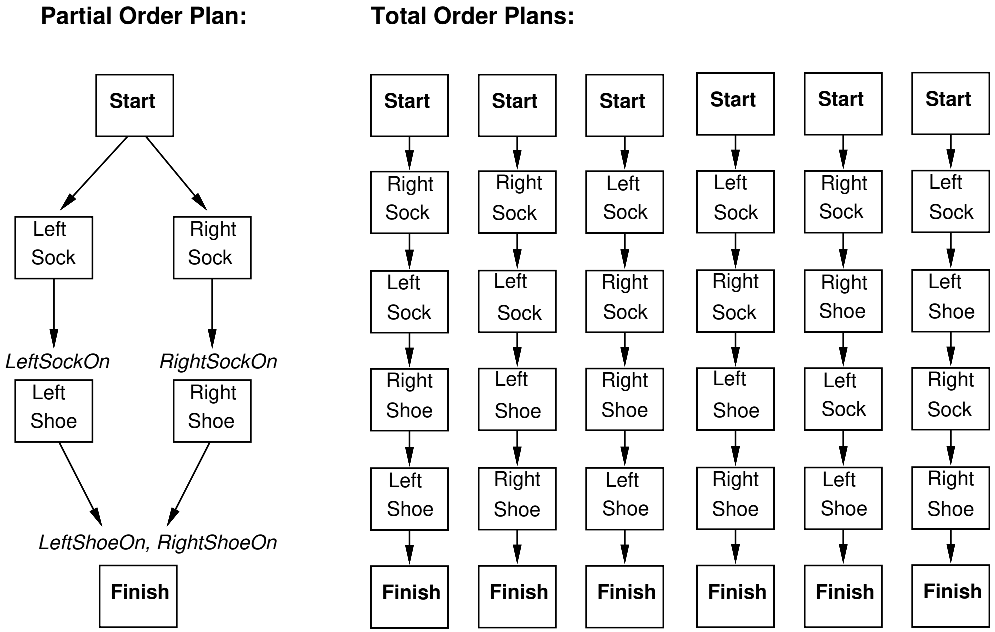


(b)

Partial order plans (II)

Least commitment principle

Partial order plan, total order plan, linearization of a plan



A plan is <u>complete</u> iff every precondition is achieved

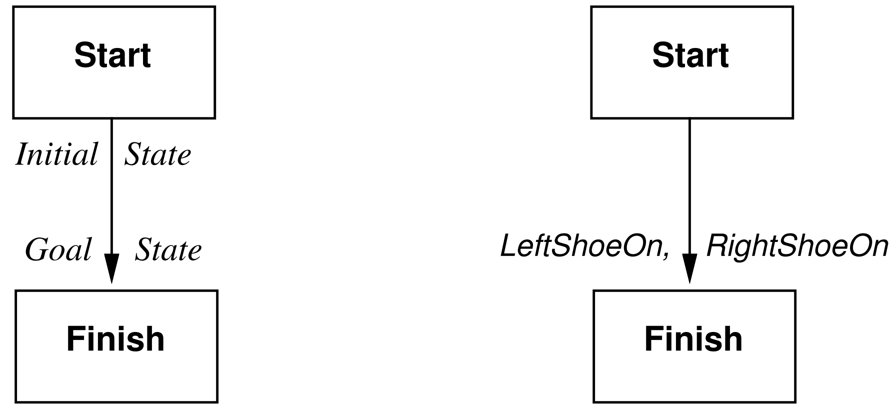
A precondition is <u>achieved</u> iff it is the effect of an earlier step and no <u>possibly intervening</u> step undoes it

Representations for plans

A plan is a data structure:

- A set of plan steps (a step is an operator)
- A set of step ordering constraints: *e.g.*, $S_i \prec S_j$
- A set of variable binding constraints: *e.g.*, $v = x$ where v is a variable and x is a constant.
- A set of **causal links**: record the purpose of a step
  A causal link $S_i \xrightarrow{c} S_j$: S_i achieves c for S_j

$$\begin{aligned}
&\text{Plan (Steps: } \{\, S_1 : Op(Action : Start), \\
&\qquad\qquad S_2 : Op(\ Action : Finish, \\
&\qquad\qquad\qquad Precondition : RightShoeOn \wedge LeftShoeOn)\}, \\
&\quad \text{Orderings: } \{S_1 \prec S_2\}, \\
&\quad \text{Bindings: } \{\}, \\
&\quad \text{Links: } \{\})
\end{aligned}$$



(a) (b)

Representation of solutions

A solution is a plan the agent can execute $\begin{cases} \text{fully instantiated} \\ \text{totally ordered} \end{cases}$

Unsatisfactory:

- An ordering can be arbitrary, partial order is more natural
- Some agents can do things in parallel
- Need to retain flexibility for, later, combining plans

A solution is a **complete**, **consistent** plan

A plan is complete iff every precondition is achieved

A plan is consistent iff there is no contradictions
in the ordering ($S_i \prec S_j$ and $S_j \prec S_i$)
or binding constraints ($v = A, V = B,\ A \neq B$)

List of terms for glossary:

Due date: Mon, March 27, 2000

Words in margins in AIMA + ...

- Negation-as-failure
- Theorem proving