Classical Planning

Manuela M. Veloso

Carnegie Mellon University
Computer Science Department

15-780 Graduate AI – Spring 2013

Readings:
• Chapter 10, Russell & Norvig
• Integrating planning and learning: The Prodigy architecture, (Sections 1 and 2)
  M M. Veloso, J. Carbonell, M. Perez, D. Borrajo, E. Fink, and J. Blythe.
  Journal of Experimental and Theoretical Artificial Intelligence, 7(1):81--120,
  1995 (see pdf file off course website or www.cs.cmu.edu/~mmv/)

Planning – Problem Solving

Newell and Simon 1956

• Given the actions available in a task domain.

• Given a problem specified as:
  – an initial state of the world,
  – goal statement - a set of goals to be achieved.

• Find a solution to the problem, i.e., a way to transform
  the initial state into a new state of the world where the
  goal statement is true.

• Planning is “thinking…”
Intelligent Agents: Planning, Execution, and Learning

Outline (2 lectures)

- Introduction: Search and Planning
- State, Actions, and Goal Representation
- Planning Algorithms
  - State-space Planning – GPS, Prodigy
  - Plan-space Planning – SNLP
  - GraphPlan
  - SATPlan
- Heuristics for Planning Algorithms
- Planning and Execution
  - Conditional Planning
    - Representation and algorithms
  - Information gathering actions
  - Replanning

Manuela Veloso, Carnegie Mellon

15-780, Spring 2013
Planning - Search

• Search agent - issues
  – Atomic state representations
  – Instantiated actions
  – Domain-specific heuristics
  – Number of actions and states

• Planning agent
  – Factored state representation
    • Collection of variables
  – Actions schemas
    • Changes to the state

Models of World State

• “Information-less” numerical identification (s1, s2,...)
• Symbolic – factored
  – Features
  – Predicates
• Conjunctive, enumerative, observable
• Complete, correct, deterministic
• Probabilistic, approximate, incremental, on-demand
Classical Deterministic Planning

- Action Model
  - complete, deterministic, correct, STRIPS/PDDL representation, typed variables, CWA
- Single initial state, fully known
- Goal statement – set of goals

Several different planning algorithms

STRIPS Representation

- Implicit Solution to Frame Problem
  - $\textit{State}$ is database of ground literals
  - If literal is not in database, assumed to be false
  - Effects of actions represented using add and delete lists (insert and remove literals from database)
  - No explicit representation of time
  - No logical inference rules

- Action representation
  - Conjunctive preconditions
  - Effects as add and delete lists
The Blocks World - States

- Objects
  - Blocks: A, B, C
  - Table: Table

- Predicates
  - On(A, B), On(C, Table), Clear(B), Handempty, Holding(C)
  - On-table(A), On(A,B), Top(B), …
  - Tower(A,B,C,…)

- States – Conjunctive
  - On(A,B) and On(B,C) and Clear(A) and Handempty

The Blocks World Definition – Actions

- Blocks are picked up and put down by the arm
- Blocks can be picked up only if they are clear, i.e., without any block on top
- The arm can pick up a block only if the arm is empty, i.e., if it is not holding another block, i.e., the arm can be pick up only one block at a time
- The arm can put down blocks on blocks or on the table
**Action Schema**

- **Action Name**
- **All variables used in the schema**
  - Universally quantified
  - Can choose values to instantiate the variables
- **Precondition**
- **Effect**
  - Positive effects – adds effect to state
  - Negative effects – deletes effects from state (if in state)

---

**STRIPS – The Blocks World**

Pickup_from_table(b)
- **Pre**: Block(b), Handempty
- Add: Holding(b)
- Delete: Handempty, On(b, Table)

Putdown_on_table(b)
- **Pre**: Block(b), Holding(b)
- Add: Handempty, On(b, Table)
- Delete: Holding(b)

Pickup_from_block(b, c)
- **Pre**: Block(b), Handempty
- Add: Holding(b)
- Delete: Handempty, On(b, c)

Putdown_on_block(b, c)
- **Pre**: Block(b), Holding(b)
- Add: Handempty, On(b, c)
- Delete: Holding(b, Clear(c))

Manuela Veloso, Carnegie Mellon
Actions

• An action $a$ is **applicable** in $s$ if all the preconditions of action $a$ are satisfied by $s$.

• $\text{RESULT}(s,a) = (s – \text{Del} (a)) \cup \text{Add} (a)$

• No explicit mention of *time*
  – The precondition always refers to time $t$
  – The effect always refers to time $t+1$

---

Example – Action Model

![Diagram showing action model with examples of actions like drill-<part>, drill-bit, put-drill-bit, and part-<part>.]
Example – Problem and Plan

Initial State

<table>
<thead>
<tr>
<th>part-holder-empty</th>
</tr>
</thead>
<tbody>
<tr>
<td>drill-holder-empty</td>
</tr>
</tbody>
</table>

Set of Objects

- part-1, part-2 : type PART
- drill-1, drill-2 : type SPOT-DRILL
- drill-3 : type TWIST-DRILL

Goal Statement

has-hole (part-1)

- put-part(part-1)
- put-drill-bit(drill-1)
- drill-spot(part-1, drill-1)
- remove-drill-bit(drill-1)
- put-drill-bit(drill-2)
- drill-hole(part-1, drill-2)

---

Initial State, Goal, Actions Example-1

\[
\text{Init}(\text{At}(C_1, \text{SFO}) \land \text{At}(C_2, \text{JFK}) \land \text{At}(P_1, \text{SFO}) \land \text{At}(P_2, \text{JFK}) \\
\land \text{Cargo}(C_1) \land \text{Cargo}(C_2) \land \text{Plane}(P_1) \land \text{Plane}(P_2) \\
\land \text{Airport}(\text{JFK}) \land \text{Airport}(\text{SFO}))
\]

\[
\text{Goal}(\text{At}(C_1, \text{JFK}) \land \text{At}(C_2, \text{SFO}))
\]

\[
\text{Action}(\text{Load}(c, p, a), \\
\text{PRECOND}: \text{At}(c, a) \land \text{At}(p, a) \land \text{Cargo}(c) \land \text{Plane}(p) \land \text{Airport}(a) \\
\text{EFFECT}: \neg \text{At}(c, a) \land \text{In}(c, p))
\]

\[
\text{Action}(\text{Unload}(c, p, a), \\
\text{PRECOND}: \text{In}(c, p) \land \text{At}(p, a) \land \text{Cargo}(c) \land \text{Plane}(p) \land \text{Airport}(a) \\
\text{EFFECT}: \text{At}(c, a) \land \neg \text{In}(c, p))
\]

\[
\text{Action}(\text{Fly}(p, \text{from}, \text{to}), \\
\text{PRECOND}: \text{At}(p, \text{from}) \land \text{Plan}(p) \land \text{Airport}(\text{from}) \land \text{Airport}(\text{to}) \\
\text{EFFECT}: \neg \text{At}(p, \text{from}) \land \text{At}(p, \text{to}))
\]

**Figure 10.1** A PDDL description of an air cargo transportation planning problem.
Questions

• Instantiated actions?

• Applicable actions?

• Result of applying an action?

Domain and Actions

• A domain can be represented by many possible choices of literals, variables, actions, preconditions, effects.

• Choice of domain
  – Granularity of representation
  – Detail of reasoning
  – Effectiveness of search
Initial State, Goal, Actions Example-2

\[
\begin{aligned}
\text{Init}(\text{On}(A, \text{Table}) & \land \text{On}(B, \text{Table}) \land \text{On}(C, A) \\
& \land \text{Block}(A) \land \text{Block}(B) \land \text{Block}(C) \land \text{Clear}(B) \land \text{Clear}(C)) \\
\text{Goal}(\text{On}(A, B) & \land \text{On}(B, C)) \\
\text{Action}(\text{Move}(b, x, y), \\
\text{PRECOND: } \text{On}(b, x) \land \text{Clear}(b) \land \text{Clear}(y) \land \text{Block}(b) \land \text{Block}(y) \land \\
(b \neq x) \land (b \neq y) \land (x \neq y), \\
\text{EFFECT: } \text{On}(b, y) \land \text{Clear}(x) \land \neg \text{On}(b, x) \land \neg \text{Clear}(y)) \\
\text{Action}(\text{MoveToTable}(b, x), \\
\text{PRECOND: } \text{On}(b, x) \land \text{Clear}(b) \land \text{Block}(b) \land (b \neq x), \\
\text{EFFECT: } \text{On}(b, \text{Table}) \land \text{Clear}(x) \land \neg \text{On}(b, x))
\end{aligned}
\]

**Figure 10.3** A planning problem in the blocks world: building a three-block tower. One solution is the sequence \([\text{MoveToTable}(C, A), \text{Move}(B, \text{Table}, C), \text{Move}(A, \text{Table}, B)]\).

Domain Representation – Blocksworld

\{(OPERATOR MOVE \\
:preconds \\
?block BLOCK \\
?from OBJECT \\
?to OBJECT \\
(and (clear ?block) \\
\quad (clear ?to) \\
\quad (on ?block ?from)) \\
:effects \\
add (on ?block ?to) \\
del (on ?block ?from) \\
(if (block-p ?from) \\
\quad add (clear ?from)) \\
(if (block-p ?to) \\
\quad del (clear ?to))\}
Planning Algorithms

- Progression: Forward state-space search

- Regression: Backward state-space search

Search Transitions Complete State
Finding a Plan – Plan Generation

• **Backtracking Search** Through a Search Space
  – How to conduct the search
  – How to represent the search space
  – How to evaluate the solutions

• Non-Deterministic **Choice Points** Determine Backtracking
  – Choice of actions
  – Choice of variable bindings
  – Choice of temporal orderings
  – Choice of subgoals to work on

Properties of Planning Algorithms

• **Soundness**
  – A planning algorithm is *sound* if all solutions are *legal* plans
    • All preconditions, goals, and any additional constraints are satisfied

• **Completeness**
  – A planning algorithm is *complete* if a solution can be found whenever one actually exists
  – A planning algorithm is *strictly complete* if all solutions are included in the search space

• **Optimality**
  – A planning algorithm is *optimal* if it maximizes a predefined measure of plan quality
Linear Planning

• Basic Idea – Goal stack
  – Work on one goal until completely solved before moving on to the next goal

Means-Ends Analysis

• Basic Idea
  – Search by reducing the difference between the state and the goals
  – What *means* (operators) are available to achieve the desired *ends* (goal)
Means-ends Analysis in Linear Planning

(Newell and Simon 60s)

GPS Algorithm \((state, goals, plan)\)

- If \(goals \subseteq state\), then return \((state, plan)\)
- **Choose** a difference \(d \in goals\) between \(state\) and \(goals\)
- **Choose** an operator \(o\) to reduce the difference \(d\)
- If no applicable operators, then return \(False\)
- \((state, plan) = GPS (state, \text{preconditions}(o), plan)\)
- If \(state\), then return \(GPS (apply (o, state), goals, [plan,o])\)

Initial call: \(GPS (initial-state, initial-goals, [])\)

---

GPS Blocks-World Example

1. Search Stack
\[
\begin{align*}
\text{State} & : \text{Clear(B)} \\
& : \text{Clear(C)} \\
& : \text{On(C, A)} \\
& : \text{On(A, Table)} \\
& : \text{On(B, Table)} \\
& : \text{Handempty} \\
\end{align*}
\]

Goal
\[
\begin{align*}
A & \quad C \\
& \quad B
\end{align*}
\]

Initial State

2. Search Stack
\[
\begin{align*}
\text{State} & : \text{Clear(B)} \\
& : \text{Clear(C)} \\
& : \text{On(C, A)} \\
& : \text{On(A, Table)} \\
& : \text{On(B, Table)} \\
& : \text{Handempty} \\
\end{align*}
\]

3. Search Stack
\[
\begin{align*}
\text{State} & : \text{Clear(B)} \\
& : \text{Clear(C)} \\
& : \text{On(C, A)} \\
& : \text{On(A, Table)} \\
& : \text{On(B, Table)} \\
& : \text{Handempty} \\
\end{align*}
\]

4. Search Stack
\[
\begin{align*}
\text{State} & : \text{Clear(B)} \\
& : \text{Clear(C)} \\
& : \text{On(C, A)} \\
& : \text{On(A, Table)} \\
& : \text{On(B, Table)} \\
& : \text{Handempty} \\
\end{align*}
\]
### GPS Blocks-World Example

#### 5. Search Stack

**State**
- Clear(B)
- Clear(C)
- On(C, A)
- On(A, Table)
- On(B, Table)
- Handempty

**Actions**
- On(A, C) On(C, B)
- Put_Block(C, B)
- Holding(C) Clear(B)
- Holding(C)

#### 6. Search Stack

**State**
- Clear(B)
- Clear(C)
- On(C, A)
- On(A, Table)
- On(B, Table)
- Handempty

**Actions**
- On(A, C) On(C, B)
- Put_Block(C, B)
- Holding(C) Clear(B)
- Pick_Block(C)
- Handempty Clear(C) On(C, ?b)

#### 7. Search Stack

**State**
- Clear(B)
- Clear(C)
- On(C, A)
- On(A, Table)
- On(B, Table)
- Handempty

**Actions**
- On(A, C) On(C, B)
- Put_Block(C, B)
- Holding(C) Clear(B)
- Pick_Block(C)

#### 8. Search Stack

**State**
- Clear(B)
- Clear(C)
- On(C, A)
- On(A, Table)
- On(B, Table)
- Holding(C)
- Clear(A)

**Actions**
- On(A, C) On(C, B)
- Put_Block(C, B)
- Holding(C) Clear(B)
- [Pick_Block(C)]

#### 9. Search Stack

**State**
- Clear(B)
- Clear(C)
- On(C, A)
- On(A, Table)
- On(B, Table)
- Holding(C)

**Actions**
- On(A, C) On(C, B)
- Put_Block(C, B)
- [Pick_Block(C)]

#### 10. Search Stack

**State**
- Clear(B)
- Clear(C)
- On(C, A)
- On(A, Table)
- On(B, Table)
- Holding(C)

**Actions**
- On(A, C) On(C, B)
- Put_Block(C, B)
- Holding(C) Clear(B)
- [Pick_Block(C); Put_Block(C, B)]

#### 11. Search Stack

**State**
- Clear(B)
- Clear(C)
- On(C, A)
- On(A, Table)
- On(B, Table)
- Handempty

**Actions**
- On(A, C) On(C, B)
- Put_Block(A, C)
- Holding(A) Clear(C)
- [Pick_Block(C)]

#### 12. Search Stack

**State**
- Clear(B)
- Clear(C)
- On(C, A)
- On(A, Table)
- On(B, Table)
- Holding(A)

**Actions**
- On(A, C) On(C, B)
- Put_Block(A, C)
- Holding(A) Clear(C)
- [Pick_Block(C)]

---

Manuela Veloso, Carnegie Mellon

15-780, Spring 2013
### GPS Blocks-World Example

**13. Search Stack**

<table>
<thead>
<tr>
<th>State</th>
<th>On(A, C) On(C, B)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Put_Block(A, C)</td>
</tr>
<tr>
<td></td>
<td>Holding(A) Clear(C)</td>
</tr>
<tr>
<td></td>
<td>Holding(A)</td>
</tr>
</tbody>
</table>

**14. Search Stack**

<table>
<thead>
<tr>
<th>State</th>
<th>On(A, C) On(C, B)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Put_Block(A, C)</td>
</tr>
<tr>
<td></td>
<td>Holding(A) Clear(C)</td>
</tr>
<tr>
<td></td>
<td>Pick_Table(A)</td>
</tr>
<tr>
<td></td>
<td>Handempty Clear(A)</td>
</tr>
<tr>
<td></td>
<td>On(A, Table)</td>
</tr>
</tbody>
</table>

**15. Search Stack**

<table>
<thead>
<tr>
<th>State</th>
<th>On(A, C) On(C, B)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Put_Block(A, C)</td>
</tr>
<tr>
<td></td>
<td>Holding(A) Clear(C)</td>
</tr>
<tr>
<td></td>
<td>Pick_Table(A)</td>
</tr>
</tbody>
</table>

**16. Search Stack**

<table>
<thead>
<tr>
<th>State</th>
<th>On(A, C) On(C, B)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Put_Block(A, C)</td>
</tr>
<tr>
<td></td>
<td>Holding(A) Clear(C)</td>
</tr>
<tr>
<td></td>
<td>Pick_Table(A)</td>
</tr>
</tbody>
</table>

**17. Search Stack**

<table>
<thead>
<tr>
<th>State</th>
<th>On(A, C) On(C, B)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Put_Block(A, C)</td>
</tr>
</tbody>
</table>

**18. Search Stack**

<table>
<thead>
<tr>
<th>State</th>
<th>On(A, C) On(C, B)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[Pick_Block(C); Put_Block(C, B)]; Pick_Table(A); Put_Block(A, C)</td>
</tr>
</tbody>
</table>

**19. Search Stack**

<table>
<thead>
<tr>
<th>State</th>
<th>On(B, Table) Clear(A) On(C, B) Handempty On(A, C)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[Pick_Block(C); Put_Block(C, B); Pick_Table(A); Put_Block(A, C)]</td>
</tr>
</tbody>
</table>
Linear Planning with MEA

- Sound?
- Optimal?
- Complete?

The Sussman Anomaly

\[ \text{A} \quad \text{B} \quad \text{C} \quad \text{A} \quad \text{B} \quad \text{C} \]
4-Action Blocks World Domain

**Pickup (?b)**
Pre: (handempty)  
( (clear ?b)  
(on-table ?b)
Add: (holding ?b)
Delete: (handempty)  
( (on-table ?b)  
( (clear ?b)

**Putdown (?b)**
Pre: (holding ?b)
Add: (handempty)  
(on-table ?b)
Delete: (holding ?b)

**Unstack (?a, ?b)**
Pre: (handempty)  
( (clear ?a) (on ?a ?b)
Add: (holding ?a)  
( (clear ?b)
Delete: (handempty)  
( (on ?a ?b) (clear ?a)

**Stack (?a, ?b)**
Pre: (holding ?a)  
( (clear ?b)
Add: (handempty)  
( (on ?a ?b)
Delete: (holding ?a)  
( (clear ?b)

The Sussman Anomaly

Linear Solution:
- (on B C)
  - Pickup (B)
  - Stack (B, C)
- (on A B)
  - Unstack (B, C)
  - Putdown (B)
  - Unstack (C, A)
  - Putdown (C)
  - Stack (A, B)
- (on B C)
  - Unstack (A, B)
  - Putdown (A)
  - Pickup (B)
  - Stack (B, C)
- (on A B)
  - Pickup (A)
  - Stack (A, B)

Linear Solution:
- (on A B)
  - Unstack (C, A)
  - Pardown (C)
  - Stack (A, B)
- (on B C)
  - Unstack (A, B)
  - Pardown (A)
  - Pickup (B)
  - Stack (B, C)
- (on A B)
  - Pickup (A)
  - Stack (A,B)
NonLinear Solution – Optimal

NonLinear Solution:
- (on A B)
- Unstack (C, A)
- Putdown (C)
- (on B C)
- Pickup (B)
- Stack (B, C)
- (on A B)
- Pickup (A)
- Stack (A, B)

Linear Planning – Goal Stack

- Advantages
  - Reduced search space, since goals are solved one at a time, and not all possible goal orderings are considered
  - Advantageous if goals are (mainly) independent
  - Linear planning is sound

- Disadvantages
  - Linear planning may produce suboptimal solutions (based on the number of operators in the plan)
  - Planner's efficiency is sensitive to goal orderings
    - Control knowledge for the "right" ordering
    - Random restarts
    - Iterative deepening

- Completeness?
Plan-Space, Partial-Order Planning

- General Approach
  - Find unachieved precondition
    - Add new action or connect to existing action
  - Determine if conflicts occur
    - Previously achieved precondition is "clobbered"
    - Fix conflicts (reorder, bind, …)
  - Partial-order planning can easily (and optimally) solve blocks world problems that involve goal interactions (e.g., the "Sussman Anomaly" problem)

POP and Sussman’s Anomaly

1. Start
   On(C, A) On(A, Table) On(B, Table)
   Clear(C), Clear(B)
   On(A, B) On(B, C)
   Finish

2. Start
   On(C, A) On(A, Table) On(B, Table)
   Clear(C) Clear(B)
   Clear(C)
   Move(B, C)
   Finish

3. Start
   On(C, A) On(A, Table) On(B, Table)
   Clear(C) Clear(B)
   Clear(A) Clear(B)
   Move(C, Table)
   Move(A, B)
   On(A, B) On(B, C)
   Finish
POP and Sussman’s Anomaly

4. \[ \begin{align*}
\text{Start} & & \text{Start} \\
\text{On}(C, A) & & \text{On}(C, A) \\
\text{On}(A, \text{Table}) & & \text{On}(A, \text{Table}) \\
\text{Clear}(C) & & \text{Clear}(C) \\
\text{Clear}(B) & & \text{Clear}(B) \\
\text{Move}(A, B) & & \text{Move}(C, \text{Table}) \\
\text{On}(A, B) & & \text{Clear}(C) \\
\text{On}(B, C) & & \text{Clear}(B) \\
\text{Finish} & & \text{Clear}(C) \\
\end{align*} \]

5. \[ \begin{align*}
\text{Start} & & \text{Start} \\
\text{On}(C, A) & & \text{On}(C, A) \\
\text{On}(A, \text{Table}) & & \text{On}(A, \text{Table}) \\
\text{Clear}(C) & & \text{Clear}(C) \\
\text{Clear}(B) & & \text{Clear}(C) \\
\text{Clear}(C) & & \text{Clear}(B) \\
\text{Move}(C, \text{Table}) & & \text{Move}(C, \text{Table}) \\
\text{Clear}(C) & & \text{Clear}(C) \\
\text{Move}(B, C) & & \text{Clear}(B) \\
\text{On}(A, B) & & \text{Clear}(B) \\
\text{On}(B, C) & & \text{Clear}(C) \\
\text{Finish} & & \text{Clear}(C) \\
\text{Clear}(B) & & \text{Clear}(C) \\
\end{align*} \]

Least Commitment

- **Basic Idea**
  - *Make choices that are relevant only to solving the current part of the problem*

- **Least Commitment Choices**
  - **Orderings**: Leave actions unordered, unless they must be sequential
  - **Bindings**: Leave variables unbound, unless needed to unify with conditions being achieved
  - **Actions**: Usually not subject to “least commitment”
POP – The Details

• **Causal Links**
  – *Adding new actions or connecting to existing actions*
  – The "purpose" of an action (which condition it supports)
  – \( a_i \rightarrow^c a_j \), where \( a_i, a_j \) are actions and \( c \) is an effect of \( a_i \)
  – Plan = \(< A, O, B, L >\)

• **Threats**
  – *Finding and fixing conflicts*
  – Action \( a_k \) with an effect \( c' \) that might "clobber" a causal link
  – *Promotion*: Order \( a_k \) after \( a_j \)
  – *Demotion*: Order \( a_k \) before \( a_i \)
  – *Separation*: Constrain \( c' \) so that it does not unify with \( c \)
    (non-codesignation constraint)

Partial Order Planning: Discussion

• **Advantages**
  – Partial order planning is **sound** and **complete**
  – Typically produces better solutions (shorter), but not guaranteed **optimal**
  – Least commitment may lead to shorter search times

• **Disadvantages**
  – Significantly more complex algorithms (higher *per-node* cost)
  – Hard to determine what is true in a state
  – Larger search space (**infinite**)!
Plan Terminology

• **Totally Ordered** Plan
  – There exists sufficient orderings $O$ such that all actions in $A$ are ordered with respect to each other

• **Fully Instantiated** Plan
  – There exists sufficient constraints in $B$ such that all variables are constrained to be equal to some constant

• **Consistent** Plan
  – There are no contradictions in $O$ or $B$

• **Complete** Plan
  – Every precondition $p$ of every action $a_i$ in $A$ is achieved:
    There exists an effect of an action $a_j$ that comes before $a_i$ and unifies with $p$, and no action $a_k$ that deletes $p$ comes between $a_j$ and $a_i$

• Partial plans are typically not executable nor consistent

---

Example: One-Way Rocket (Veloso 89)

<table>
<thead>
<tr>
<th>(OPERATOR LOAD-ROCKET)</th>
<th>(OPERATOR UNLOAD-ROCKET)</th>
<th>(OPERATOR MOVE-ROCKET)</th>
</tr>
</thead>
<tbody>
<tr>
<td>:preconds</td>
<td>:preconds</td>
<td>:preconds</td>
</tr>
<tr>
<td>?roc ROCKET</td>
<td>?roc ROCKET</td>
<td>?roc ROCKET</td>
</tr>
<tr>
<td>?obj OBJECT</td>
<td>?obj OBJECT</td>
<td>?obj OBJECT</td>
</tr>
<tr>
<td>?loc LOCATION</td>
<td>?loc LOCATION</td>
<td>?loc LOCATION</td>
</tr>
<tr>
<td>(and (at ?obj ?loc))</td>
<td>(and (inside ?obj ?roc))</td>
<td>(and (at ?roc ?from-l))</td>
</tr>
<tr>
<td>(at ?roc ?loc))</td>
<td>(at ?roc ?loc))</td>
<td>(has-fuel ?roc))</td>
</tr>
<tr>
<td>:effects</td>
<td>:effects</td>
<td>:effects</td>
</tr>
</tbody>
</table>
Incompleteness of Linear Planning

Initial state:
(at obj1 locA)
(at obj2 locA)
(at ROCKET locA)
(has-fuel ROCKET)

Goal statement:
(and
(at obj1 locB)
(at obj2 locB))

<table>
<thead>
<tr>
<th>Goal</th>
<th>Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>(at obj1 locB)</td>
<td>(LOAD-ROCKET obj1 locA)</td>
</tr>
<tr>
<td></td>
<td>(MOVE-ROCKET)</td>
</tr>
<tr>
<td></td>
<td>(UNLOAD-ROCKET obj1 locB)</td>
</tr>
<tr>
<td>(at obj2 locB)</td>
<td>failure</td>
</tr>
</tbody>
</table>

State-Space Nonlinear Planning

Extend linear planning:
- From stack to set of goals.
- Include in the search space all possible interleaving of goals.

State-space nonlinear planning is complete.

<table>
<thead>
<tr>
<th>Goal</th>
<th>Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>(at obj1 locB)</td>
<td>(LOAD-ROCKET obj1 locA)</td>
</tr>
<tr>
<td>(at obj2 locB)</td>
<td>(LOAD-ROCKET obj2 locA)</td>
</tr>
<tr>
<td>(at obj1 locB)</td>
<td>(MOVE-ROCKET)</td>
</tr>
<tr>
<td>(at obj2 locB)</td>
<td>(UNLOAD-ROCKET obj1 locB)</td>
</tr>
</tbody>
</table>
Prodigy Planner

• Extension to GPS
  – Set of goals, instead of stack of goals
  – Means-ends analysis for selection of “pending goals”
  – Choice point for applying an operator when applicable and continue backward-chaining (subgoaling)

Prodigy4.0 (Veloso et al. 90)

1. Terminate if the goal statement is satisfied in the current state.
2. Compute the SET of pending goals $G$, and the set of applicable operators $A$.
   • A goal is pending if it is a precondition, not satisfied in the current state, of an operator already in the plan.
   • An operator is applicable when all its preconditions are satisfied in the state.
1. Choose a goal $G$ in $G$ or choose an operator $A$ in $A$. 
Prodigy4.0 Planning Algorithm

4. If $G$ has been chosen, then
   - Expand goal $G$,
     i.e., get the set $O$ of relevant instantiated operators that could achieve $G$,
   - Choose an operator $O$ from $O$,
   - Go to step 1.

5. If an operator $A$ has been selected as directly applicable, then
   - Apply $A$,
   - Go to step 1.

Prodigy4.0 – Search Representation

Applying an operator (moving it to the head)  Adding an operator to the tail-plan
Why is Planning Hard?

Planning involves a complex search:
• Alternative operators to achieve a goal
• Multiple goals that interact
• Solution optimality, quality
• Planning efficiency, soundness, completeness

Many Issues in Planning

• State representation
  – The frame problem
  – The “choice” of predicates
    • On-table (x), On (x, table), On-table-A, On-table-B,…
• Action representation
  – Many alternative definitions
  – Reduce to “needed” definition
  – Conditional effects
  – Uncertainty
  – Quantification
  – Functions
• Generation – planning algorithm(S)
Many Planning “Domains”

- Web management agents
- Robot planning
- Manufacturing planning
- Image processing management
- Logistics transportation
- Crisis management
- Bank risk management
- Blocksworld
- Puzzles
- Artificial domains

Summary

- **Planning**: selecting one sequence of actions (operators) that transform (apply to) an initial state to a final state where the goal statement is true.

- **Means-ends analysis**: identify and reduce, as soon as possible, differences between state and goals.

- **Linear planning**: backward chaining with means-ends analysis using a stack of goals - potentially efficient, possibly unoptimal, incomplete; GPS, STRIPS.

- **Plan-space planning**: least commitment search; causal links and threat resolution; effective management of goal interactions

- **Nonlinear planning with means-ends analysis**: backward chaining using a set of goals; reason about when “to reduce the differences;” Prodigy4.0.