• Embedded in an environment (sensing and acting)
• Inference mechanism
  • based on internal representation of the perceived information
  • usage of knowledge (facts, rules)
  • maybe goal-directed
• Examples for inference mechanisms:
  • theorem proving
  • planning

slides partially taken from Dana Nau’s lecture slides, http://creativecommons.org/licenses/by-nc-sa/2.0/
Intuitions on Planning

• Intelligent action is rational action, that is, the best possible action in a given situation

• **Planning is the reasoning side of acting**

• Abstract, explicit deliberation process that chooses and organizes actions by anticipating their expected outcomes

• Some actions require planning, many do not
  • we act more frequently than we explicitly plan
  • performing well-trained behaviors based on pre-stored plans
  • acting and adapting in flexible settings

• Planning is a complicated, time consuming, and costly process

• Planning is needed when
  • new situations, unfamiliar actions are involved
  • complex tasks, complex objectives are addressed
  • actions are constrained by high risks, high costs, joint activities, need for synchronization

• Typically we seek feasible, good plans, not optimal plans (cf. Simon’s ‘bounded rationality’)

Space Exploration

- Autonomous planning, schedule, control
  - NASA: JPL and AMES
- Remote Agent Experient (RAX)
  - Deep Space 1
- Mars Exploration Rover (MER)
Manufacturing

- Sheet-metal bending machines - Amada Corporation
- Software to plan the sequence of bends
  [Gupta and Bourne, *J. Manufacturing Sci. and Engr.*, 1999]
• **Bridge Baron** - Great Game Products
  • 1997 world champion of computer bridge
    [Smith, Nau, and Throop, *AI Magazine*, 1998]
  • 2004: 2nd place
Outlook

- Conceptual model of planning
- Example: Dock Worker Robots Domain
- Restrictive Assumptions for Classical Planning
- Classical representation
  - relies on closed-world assumption
  - compare to set-theoretical and
  - state-variable representation
  - PDDL as language for planners
- STRIPS
- Heuristic Search Planning
- Graphplan

*see textbook: Ghallab, Nau, Traveso, Automated Planning*
Conceptual Model of Planning

Environment

Planner

Controller

System $\Sigma$

State transition system

$\Sigma = (S, A, E, \gamma)$

$S =$ states

$A =$ actions

$E =$ exogenous events

$\gamma =$ state-transition function
**State Transition System**

\( \Sigma = (S, A, E, \gamma) \)

- \( S = \{ \text{states} \} \)
- \( A = \{ \text{actions} \} \)
- \( E = \{ \text{exogenous events} \} \)
- State-transition function \( \gamma : S \times (A \cup E) \rightarrow 2^S \)
  - \( S = \{ s_0, ..., s_5 \} \)
  - \( A = \{ \text{move1, move2, put, take, load, unload} \} \)
  - \( E = \{ \} \)
  - \( \gamma : \text{see the arrows} \)

The Dock Worker Robots (DWR) domain
Conceptual Model of Planning

Controller

Planner

Controller

System $\Sigma$

Initial state

Objectives

Execution status

Description of $\Sigma$

Plans

Observations

Actions

Events

Given observation $o$ in $O$, produces action $a$ in $A$
Conceptual Model of Planning

**Planner's Input**

- Planning problem
- Initial state
- Objectives
- *Execution status*
- Description of $\Sigma$

**Controller**

- Plans
- Actions
- Observations

**System $\Sigma$**

- Events
Planning Problem

Description of $\Sigma$
Initial state or set of states
Initial state $= s_0$

Objective
Goal state, set of goal states, set of tasks, “trajectory“ of states, objective function, ...
Goal state $= s_5$

The Dock Worker Robots (DWR) domain
Conceptual Model of Planning

Planner’s Output

- Description of $\Sigma$
- Initial state
- Objectives
- Execution status
- Plans
- Instructions to the controller

- Observations
- Actions

- System $\Sigma$

- Events
Conceptual Model of Planning

**Plans**

**Classical plan:**
a sequence of actions

\[ \langle \text{take}, \text{move}_1, \text{load}, \text{move}_2 \rangle \]

**Policy:**
partial function from \( S \) into \( A \)

\[ \{(s_0, \text{take}),
(s_1, \text{move}_1),
(s_3, \text{load}),
(s_4, \text{move}_2)\} \]

The Dock Worker Robots (DWR) domain
Restrictive Assumptions

- **A0: Finite system:**
  - finitely many states, actions, events

- **A1: Fully observable:**
  - the controller always $\Sigma$’s current state

- **A2: Deterministic:**
  - each action has only one outcome

- **A3: Static (no exogenous events):**
  - no changes but the controller’s actions

- **A4: Attainment goals:**
  - a set of goal states $S_g$

- **A5: Sequential plans:**
  - a plan is a linearly ordered sequence of actions $(a_1, a_2, \ldots, a_n)$

- **A6: Implicit time:**
  - no time durations; linear sequence of instantaneous states

- **A7: Off-line planning:**
  - planner doesn’t know the execution status
• Start with a *function-free* first-order language
  • Finitely many predicate symbols and constant symbols, but no function symbols

• Example: the DWR domain
  • Locations: l1, l2, ...
  • Containers: c1, c2, ...
  • Piles: p1, p2, ...
  • Robot carts: r1, r2, ...
  • Cranes: k1, k2, ...
Classical Representation

- **Atom**: predicate symbol and args
  - Use these to represent both fixed and dynamic relations
    - adjacent($l,l'$)
    - occupied($l$)
    - loaded($r,c$)
    - holding($k,c$)
    - in($c,p$)
    - top($c,p$)
    - attached($p,l$)
    - at($r,l$)
    - unloaded($r$)
    - empty($k$)
    - on($c,c'$)
    - belong($k,l$)

- **Ground** expression: contains no variable symbols - e.g.,
  - in($c1,p3$)

- **Unground** expression: at least one variable symbol - e.g.,
  - in($c1,x$)

- **Substitution**: $\theta = \{x_1 \leftarrow v_1, x_2 \leftarrow, ..., x_n \leftarrow v_n\}$
  - Each $x_i$ is a variable symbol; each $v_i$ is a term

- **Instance** of $e$: result of applying a substitution $\theta$ to $e$
  - Replace variables of $e$ simultaneously, not sequentially
• **State**: a set $s$ of ground atoms
  
  • The atoms represent the things that are true in one of $\Sigma$’s states
  
  • Only finitely many ground atoms, so only finitely many possible states

$s_1 = \{\text{attached}(p1, loc1), \text{in}(c1, p1), \text{in}(c3, p1), \text{top}(c3, p1), \text{on}(c3, c1),
\text{on}(c1, \text{pallet}), \text{attached}(p2, loc1), \text{in}(c2, p2), \text{top}(c2, p2), \text{on}(c2, \text{pallet}),
\text{belong}(\text{crane1}, loc1), \text{empty}(\text{crane1}), \text{adjacent}(\text{loc1}, \text{loc2}),
\text{adjacent}(\text{loc2}, \text{loc1}), \text{at}(r1, \text{loc2}), \text{occupied}(\text{loc2}), \text{unloaded}(r1)\}$
• **Operator**: a triple \( o = ( \text{name}(o), \text{precond}(o), \text{effects}(o)) \)
  - \( \text{name}(o) \) is a syntactic expression of the form \( n(x_1, ..., x_k) \)
    - \( n: \) operator symbol - must be unique for each operator
    - \( x_1, ..., x_k: \) variable symbols (parameters)
      must include every variable symbol in \( o \)
  - \( \text{precond}(o): \) **preconditions**
    - \( \) literals that must be true in order to use the operator
  - \( \text{effects}(o): \) **effects**
    - \( \) literals the operator will make true

\textit{take}(k, l, c, d, p)

;; crane \( k \) at location \( l \) takes \( c \) off of \( d \) in pile \( p \)
precond: \( \) belong\( (k, l) \), attached\( (p, l) \), empty\( (k) \), top\( (c, p) \), on\( (c, d) \)
effects: \( \) holding\( (k, c) \), \( \neg \)empty\( (k) \), \( \neg \)in\( (c, p) \), \( \neg \)top\( (c, p) \), \( \neg \)on\( (c, d) \), top\( (d, p) \)
• **Action:**
ground instance (via substitution) of an operator

**Operator** (i.e., an abstract schema of actions, containing variable)

\[
take(k, l, c, d, p)
\]

;; crane \(k\) at location \(l\) takes \(c\) off of \(d\) in pile \(p\)

precond: belong\((k, l)\), attached\((p, l)\), empty\((k)\), top\((c, p)\), on\((c, d)\)

effects: holding\((k, c)\), \(\neg\)empty\((k)\), \(\neg\)in\((c, p)\), \(\neg\)top\((c, p)\), \(\neg\)on\((c, d)\), top\((d, p)\)

**Action** (i.e., an instantiated operator which can be applied to some specific states)

\[
take(crane1, loc1, c3, c1, p1)
\]

;; crane \(crane1\) at location \(loc1\) takes \(c3\) off of \(c1\) in pile \(p1\)

precond: belong\((crane1, loc1)\), attached\((p1, loc1)\),

\hspace{1cm} empty\((crane1)\), top\((c3, p1)\), on\((c3, c1)\)

effects: holding\((crane1, c3)\), \(\neg\)empty\((crane1)\), \(\neg\)in\((c3, p1)\), \(\neg\)top\((c3, p1)\), \(\neg\)on\((c3, c1)\), top\((c1, p1)\)
Notation

Let $S$ be a set of literals. Then

- $S^+ = \{ \text{atoms that appear positively in } S \}$
- $S^- = \{ \text{atoms that appear negatively in } S \}$

More specifically, let $a$ be an operator or action. Then

- $\text{precond}^+(a) = \{ \text{atoms that appear positively in } a's \text{ preconditions} \}$
- $\text{precond}^-(a) = \{ \text{atoms that appear negatively in } a's \text{ preconditions} \}$
- $\text{effects}^+(a) = \{ \text{atoms that appear positively in } a's \text{ effects} \}$
- $\text{effects}^-(a) = \{ \text{atoms that appear negatively in } a's \text{ effects} \}$

$\text{take}(k, l, c, d, p)$

;; crane $k$ at location $l$ takes $c$ off of $d$ in pile $p$

precond: $\text{belong}(k, l)$, $\text{attached}(p, l)$, $\text{empty}(k)$, $\text{top}(c, p)$, $\text{on}(c, d)$
effects: $\text{holding}(k, c)$, $\neg \text{empty}(k)$, $\neg \text{in}(c, p)$, $\neg \text{top}(c, p)$, $\neg \text{on}(c, d)$, $\text{top}(d, p)$

Let $S$ be a set of literals. Then

- $\text{effects}^+(\text{take}(k, l, c, d, p)) = \{ \text{holding}(k,c), \text{top}(d,p) \}$
- $\text{effects}^-(\text{take}(k, l, c, d, p)) = \{ \text{empty}(k), \text{in}(c,p), \text{top}(c,p), \text{on}(c,d) \}$
• An action $a$ is **applicable** to a state $s$ if $s$ satisfies $\text{precond}(a)$,
  
i.e., if $\text{precond}^+(a) \subseteq s$ and $\text{precond}^-(a) \cap s = \emptyset$
• Here are an action and a state that it’s applicable to:

\[
s_1 = \{ \text{attached}(p_1, \text{loc}_1), \text{in}(c_1, p_1), \text{in}(c_3, p_1), \text{top}(c_3, p_1), \text{on}(c_3, c_1), \text{on}(c_1, \text{pallet}), \text{attached}(p_2, \text{loc}_1), \text{in}(c_2, p_2), \text{top}(c_2, p_2), \text{on}(c_2, \text{pallet}), \text{belong}(\text{crane}_1, \text{loc}_1), \text{empty}(\text{crane}_1), \text{adjacent}(\text{loc}_1, \text{loc}_2), \text{adjacent}(\text{loc}_2, \text{loc}_1), \text{at}(r_1, \text{loc}_2), \text{occupied}(\text{loc}_2), \text{unloaded}(r_1) \}
\]

\text{take(\text{crane}_1, \text{loc}_1, c_3, c_1, p_1)}

;;; crane \text{crane}_1 at location \text{loc}_1 takes \text{c}_3 off of \text{c}_1 in pile \text{p}_1
\begin{align*}
\text{precond: } & \text{belong}(\text{crane}_1, \text{loc}_1), \text{attached}(p_1, \text{loc}_1), \\
& \text{empty}(\text{crane}_1), \text{top}(c_3, p_1), \text{on}(c_3, c_1) \\
\text{effects: } & \text{holding}(\text{crane}_1, c_3), \neg \text{empty}(\text{crane}_1), \neg \text{in}(c_3, p_1), \\
& \neg \text{top}(c_3, p_1), \neg \text{on}(c_3, c_1), \text{top}(c_1, p_1)
\end{align*}
• If $a$ is applicable to $s$, the result of performing it is

$$\gamma(s, a) = s - \text{effects}^- (a) \cup \text{effects}^+ (a)$$

• Delete the negative effects, and add the positive ones

$\text{take}(\text{crane1, loc1, c3, c1, p1})$

;;; crane crane1 at location loc1 takes c3 off of c1 in pile p1

precond: belong(crane1,loc1), attached(p1,loc1),
empty(crane1), top(c3,p1), on(c3,c1)
effects: holding(crane1, c3), ¬empty(crane1), ¬in(c3,p1),
¬top(c3,p1), ¬on(c3,c1), top(c1,p1)
s₁ = \{attached(p₁, loc₁), in(c₁, p₁), in(c₃, p₁), top(c₃, p₁), on(c₃, c₁),
    on(c₁, pallet), attached(p₂, loc₁), in(c₂, p₂), top(c₂, p₂), on(c₂, pallet),
    belong(crane₁, loc₁), empty(crane₁), adjacent(loc₁, loc₂),
    adjacent(loc₂, loc₁), at(r₁, loc₂), occupied(loc₂), unloaded(r₁)\}

take(crane₁, loc₁, c₃, c₁, p₁) with
precond: belong(crane₁,loc₁), attached(p₁,loc₁), empty(crane₁), top(c₃,p₁), on(c₃,c₁)
is applicable to s₁:
\{belong(crane₁, loc₁), attached(p₁, loc₁), empty(crane₁), top(c₃, p₁), on(c₃, c₁) ⊆ s₁\} =
\{attached(p₁, loc₁), in(c₁, p₁), in(c₃, p₁), top(c₃, p₁), on(c₃, c₁),
    on(c₁, pallet), attached(p₂, loc₁), in(c₂, p₂), top(c₂, p₂), on(c₂, pallet),
    belong(crane₁, loc₁), empty(crane₁), adjacent(loc₁, loc₂),
    adjacent(loc₂, loc₁), at(r₁, loc₂), occupied(loc₂), unloaded(r₁)\}

Application:

s₂ = \{attached(p₁, loc₁), in(c₁, p₁), in(c₃, p₁), top(c₃, p₁), on(c₃, c₁),
    on(c₁, pallet), attached(p₂, loc₁), in(c₂, p₂), top(c₂, p₂), on(c₂, pallet),
    belong(crane₁, loc₁), empty(crane₁), adjacent(loc₁, loc₂),
    adjacent(loc₂, loc₁), at(r₁, loc₂), occupied(loc₂), unloaded(r₁)
    \cup \text{holding(crane₁, c₃), top(c₁,p₁)}\}
move(r, l, m)
;; robot r moves from location l to location m
precond: adjacent(l,m), at(r,l), ¬occupied(m)
effects: at(r,m), occupied(m), ¬occupied(l), ¬at(r,l)

load(k, l, c, r)
;; crane k at location l loads container c onto robot r
precond: belong(k,l), holding(k,c), at(r,l), unloaded(r)
effects: empty(k), ¬holding(k,c), loaded(r,c), ¬unloaded(r)

unload(k, l, c, r)
;; crane k at location l takes container c from robot r
precond: belong(k,l), at(r,l), loaded(r,c), empty(k)
effects: ¬empty(k), holding(k,c), unloaded(r), ¬loaded(r)

put(k, l, c, d, p)
;; crane k at location l puts c onto d in pile p
precond: belong(k,l), attached(p,l), holding(k,c), top(d,p)
effects: ¬holding(k,c),empty(k), in(c,p), top(c,p),on(c,d),¬top(d,p)

take(k, l, c, d, p)
;; crane k at location l takes c off of d in pile p
precond: belong(k,l), attached(p,l), empty(k), top(c,p), on(c,d)
effects: holding(k,c),¬empty(k), ¬in(c,p),
¬top(c,p),¬on(c,d),¬top(d,p)

• Planning domain: language plus operators
  • Corresponds to a set of state-transition systems
  • Example: operators for the DWR domain
• Given a planning domain (language $L$, operators $O$)
  • Statement of a planning problem: a triple $P = (O, s_0, g)$
    $\Rightarrow$ $O$ is the collection of operators
    $\Rightarrow$ $s_0$ is a state (the initial state)
    $\Rightarrow$ $g$ is a set of literals (the goal formula)
  • The actual planning problem: $\mathcal{P} = (\Sigma, s_0, S_g)$
    $\Rightarrow$ $s_0$ and $S_g$ are as above
    $\Rightarrow$ $\Sigma = (S, A, \gamma)$ is a state-transition system
    $\Rightarrow$ $S = \{\text{all sets of ground atoms in } L\}$
    $\Rightarrow$ $A = \{\text{all ground instances of operators in } O\}$
    $\Rightarrow$ $\gamma$ = the state-transition function determined by the operators

• ’planning problem’ often used sloppily for ‘statement of the problem’
• **Plan**: any sequence of actions $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$ such that each $a_i$ is a ground instance of an operator in $O$

• The plan is a *solution* for $P = (O, s_0, g)$ if it is executable and achieves $g$
  • i.e., if there are states $s_0, s_1, \ldots, s_n$ such that
    $\Rightarrow \gamma(s_0, a_1) = s_1$
    $\Rightarrow \gamma(s_1, a_2) = s_2$
    $\Rightarrow \ldots$
    $\Rightarrow \gamma(s_{n-1}, a_n) = s_n$
    $\Rightarrow s_n$ satisfies $g$
• Let $P_1 = (O, s_1, g_1)$, where
  • $O$ is the set of operators given earlier
  • $s_1 = \{ \text{attached}(p1,loc1), \text{in}(c1,p1), \text{in}(c3,p1), \text{top}(c3,p1), \text{on}(c3,c1),\]
  \hspace{1cm} \text{on}(c1,pallet), \text{attached}(p2,loc1), \text{in}(c2,p2), \text{top}(c2,p2), \text{on}(c2,pallet),\]
  \hspace{1cm} \text{belong}(\text{crane1},loc1), \text{empty}(\text{crane1}), \text{adjacent}(\text{loc1,loc2}),\]
  \hspace{1cm} \text{adjacent}(\text{loc2,loc1}), \text{at}(r1,\text{loc2}), \text{occupied}(\text{loc2}), \text{unloaded}(r1)\}$

• $g_1 = \{\text{loaded}(r1, c3), \text{at}(r1, \text{loc2})\}$
Here are three solutions for $P_1$:

- $\langle \text{take(crane1,loc1,c3,c1,p1)}, \text{move(r1,loc2,loc1)}, \text{move(r1,loc1,loc2)}, \text{move(r1,loc2,loc1)}, \text{load(crane1,loc1,c3,r1)}, \text{move(r1,loc1,loc2)} \rangle$

- $\langle \text{take(crane1,loc1,c3,c1,p1)}, \text{move(r1,loc2,loc1)}, \text{load(crane1,loc1,c3,r1)}, \text{move(r1,loc1,loc2)} \rangle$

- $\langle \text{move(r1,loc2,loc1)}, \text{take(crane1,loc1,c3,c1,p1)}, \text{load(crane1,loc1,c3,r1)}, \text{move(r1,loc1,loc2)} \rangle$

Each of them produces the state shown here:
Example (continued)

- The first is *redundant*: can remove actions and still have a solution
  - ⟨take(crane1,loc1,c3,c1,p1), move(r1,loc2,loc1),
    move(r1,loc1,loc2), move(r1,loc2,loc1), load(crane1,loc1,c3,r1), move(r1,loc1,loc2)⟩
  - ⟨take(crane1,loc1,c3,c1,p1), move(r1,loc2,loc1), load(crane1,loc1,c3,r1),
    move(r1,loc1,loc2)⟩
  - ⟨move(r1,loc2,loc1), take(crane1,loc1,c3,c1,p1), load(crane1,loc1,c3,r1),
    move(r1,loc1,loc2)⟩
- The 2nd and 3rd are *irredundant* and *shortest*
Set-Theoretic Representation

- Like classical representation, but restricted to propositional logic

- States:
  - Instead of a collection of ground atoms...
    
    \{on(c1,\text{pallet}), \text{on}(c1,r1), \text{on}(c1,c2), \ldots, \text{at}(r1,l1), \text{at}(r1,l2), \ldots\}

    ... use a collection of propositions (boolean variables):
    
    \{on-c1-pallet, on-c1-r1, on-c1-c2, \ldots, at-r1-l1, at-r1-l2, \ldots\}
• Instead of operators like this one,

\[\text{take}(k, l, c, d, p)\]

;; crane \(k\) at location \(l\) takes \(c\) off of \(d\) in pile \(p\)

precond: \(\text{belong}(k, l), \text{attached}(p, l), \text{empty}(k), \text{top}(c, p), \text{on}(c, d)\)

effects: \(\text{holding}(k, c), \neg \text{empty}(k), \neg \text{in}(c, p), \neg \text{top}(c, p), \neg \text{on}(c, d), \neg \text{top}(d, p)\)

---

\[\text{take}(\text{crane1}, \text{loc1}, c_3, c_1, p_1)\]

;; crane \(\text{crane1}\) at location \(\text{loc1}\) takes \(c_3\) off of \(c_1\) in pile \(p_1\)

precond: \(\text{belong}(\text{crane1}, \text{loc1}), \text{attached}(p_1, \text{loc1}), \text{empty}(\text{crane1}), \text{top}(c_3, p_1), \text{on}(c_3, c_1)\)

effects: \(\text{holding}(\text{crane1}, c_3), \neg \text{empty}(\text{crane1}), \neg \text{in}(c_3, p_1), \neg \text{top}(c_3, p_1), \neg \text{on}(c_3, c_1), \text{top}(c_1, p_1)\)

---

\[\text{take} - \text{crane1} - \text{loc1} - c_3 - c_1 - p_1\]

precond: \(\text{belong-crane1-loc1, attached-p1-loc1, empty-crane1, top-c3-p1, on-c3-c1}\)

delete: \(\text{empty-crane1, in-c3-p1, top-c3-p1, on-c3-p1}\)

add: \(\text{holding-crane1-c3, top-c1-p1}\)
Remarks on Set-Theoretic Representation

- A set-theoretic representation is equivalent to a classical representation in which all of the atoms are ground.

- Exponential blow-up
  - If a classical operator contains $n$ atoms and each atom has arity $k$, then it corresponds to $c^{nk}$ actions where $c = |\{\text{constant symbols}\}|$.

- Set-theoretic representation is applied in practice for SAT-solvers and model-checking. For these applications, typically, a set-theoretical representation is algorithmically generated from a classical representation.

- Set-theoretic representation is a very useful concept for theoretical analyses of planning (using an explicit representation of all states of a problem).
State-Variable Representation

• Use ground atoms for properties that do not change, e.g., \text{adjacent(loc1,loc2)}
• For properties that can change, assign values to \textit{state variables}
  • Like fields in a record structure
• Classical and state-variable representations take similar amounts of space
  • Each can be translated into the other in low-order polynomial time

\text{move}(r,l,m)

\begin{align*}
\text{;; robot } r \text{ at location } l \text{ moves to an adjacent location } m \\
\text{precond: } & \text{rloc}(r)=l, \text{ adjacent}(l,m) \\
\text{effects: } & \text{rloc}(r) \leftarrow m
\end{align*}

\begin{align*}
\{\text{top}(p1)=c3, \text{cpos}(c3)=c1, \\
\text{cpos}(c1)=\text{pallet}, \text{holding}(\text{crane1})=\text{nil}, \\
\text{rloc}(r1)=\text{loc2}, \text{loaded}(r1)=\text{nil}, \ldots\}\end{align*}
Remarks on State-Variable Representation

- The engineering view rather than the logic view
- Especially useful when a new value which is assigned to a variable has to be calculated by some procedure
- Example: Depots Domain with Fluents in PDDL
  - Predefined functions assign and increase
- In general, arbitrary user-defined methods can be used:
  fuel(r) ← calcDecrease(r,l,m)
• Problem Domain Definition Language as common language for most modern planners (see PDDL Specification)
• Example: Equality constraints and conditioned effects

(define (domain blocks-world-domain)
  (:requirements :strips :equality :conditional-effects)
  (:constants Table)
  (:predicates (on ?x ?y)
    (clear ?x)
    (block ?b)
  )
  ;; Define step for placing one block on another.
  (:action puton
    :parameters (?X ?Y ?Z)
      (not (= ?Y ?Z)) (not (= ?X ?Z))
      (not (= ?X ?Y)) (not (= ?X Table)))
    :effect
    (and (on ?X ?Y) (not (on ?X ?Z))
      (when (not (= ?Z Table)) (clear ?Z))
      (when (not (= ?Y Table)) (not (clear ?Y)))))))
(define (domain Depot-object-fluents)
  (:requirements :typing :equality :fluents)
  (:types place locatable - object
depot distributor - place
truck hoist surface - locatable
pallet crate - surface)

  (:constants no-crate - crate)
  (:predicates (clear ?s - surface))
  (:functions
    (load-limit ?t - truck)
    (current-load ?t - truck)
    (weight ?c - crate)
    (fuel-cost) - number
    (position-of ?l - locatable) - place
    (crate-held ?h - hoist) - crate
    (thing-below ?c - crate) - (either surface truck))

  (:action drive
    :parameters (?t - truck ?p - place)
    :effect (and (assign (position-of ?t) ?p)
      (increase (fuel-cost) 10)))
by Fikes & Nilsson (1971),
“Stanford Research Institute Problem Solver”

classical example:
moving boxes between rooms (“Strips World”)

Originally:
representation formalism (relying on closed-world assumption)
and planning algorithm
today: STRIPS planning refers to classical representation
without extensions and not to a specific algorithm

STRIPS algorithm:
a linear (and therefore incomplete!) approach

compare to:
General Problem Solver (GPS), a cognitively motivated
problem solving algorithm relying on means-end analysis which
is also linear and therefore incomplete (Newell & Simon)
STRIPS Algorithm

- Backward-search with a kind of hill climbing strategy
- In each recursive call only such sub-goals are relevant which are preconditions of the last operator added
- Consequence: considerable reduction of branching, but resulting in incompleteness
- Linear planning: organizing sub-goals in a stack
- in contrast to non-linear planning: organizing sub-goals in a set, interleaving of goals
STRIPS Algorithm

STRIPS \((O, s, g)\)

\[
\pi \leftarrow \text{empty plan}
\]

\[
\text{loop}
\]

\[
\text{if } s \text{ satisfies } g \text{ then}
\]

\[
\text{return } \pi
\]

\[
\text{end if}
\]

\[
A \leftarrow \{a | a \text{ is a ground instance of an operator in } O, \text{ and } a \text{ is relevant for } g\}
\]

\[
\text{if } A = \emptyset \text{ then}
\]

\[
\text{return failure}
\]

\[
\text{end if}
\]

\[
\text{non-deterministically choose any action } a \in A
\]

\[
\pi' \leftarrow \text{STRIPS}(O, s, \text{precond}(a))
\]

\[
\text{if } \pi' = \text{failure then}
\]

\[
\text{return failure} \quad ;;\text{if we get here, then } \pi' \text{ achieves precond}(a) \text{ from } s
\]

\[
\text{end if}
\]

\[
s \leftarrow \gamma(s, \pi') \quad ;;s \text{ now satisfies precond}(a)
\]

\[
s \leftarrow \gamma(s, a)
\]

\[
\pi \leftarrow \pi.\pi'.a
\]

\[
\text{end loop}
\]
Incompleteness of Linear Planning

The Sussman Anomaly

Initial State

Goal:

on(A, B) and
on(B, C)

on(B, C)

on(A, B)
Interleaving of Goals

- Non-linear planning allows that a sequence of planning steps dealing with one goal is interrupted to deal with another goal.
- For the Sussman Anomaly, that means that after block C is put on the table pursuing goal \( \text{on}(A, B) \), the planner switches to the goal \( \text{on}(B, C) \).
- Non-linear planning corresponds to dealing with goals organized in a set.
- The correct sequence of goals might not be found immediately without backtracking.
(from Manuela Veloso)

• Objects:
  $n$ boxes, Positions (Earth, Moon), one Rocket

• Operators:
  load/unload a box, move the Rocket
  (oneway: only from earth to moon, no way back!)

• Linear planning:
  to reach the goal, that Box1 is on the Moon, load it, shoot
  the Rocket, unload it, now no other Box can be transported!
Current Approaches to Planning

- End of the 1990s many new and significantly more efficient planning approaches have been proposed.
- Planning has been reformulated as:
  - network flow problem: Graphplan
  - heuristic search with automatically derived cost function
  - satisfiability problem: SAT-planning
Propositional satisfiability: given a boolean formula, does there exist an assignment of truth values that makes the formula true (e.g., a model)?

Very first problem shown to be NP-complete

Many algorithms exist which work on average case polynomial time (e.g., Davis-Putnam, GSAT)

Encode a planning problem $P$ with a fixed length solution path $n$ as satisfiability problem $\Psi$ (see lecture Formal Characteristics)

That is based on set-theoretical representation

Frame axioms are needed to describe what does not change

Introduction of additional argument to represent plan-length
Example

- Planning domain:
  - one robot $r1$
  - two adjacent locations $l1, l2$
  - one planning operator
    (to move the robot from one location to another)

- Encode $(P,n)$ where $n = 1$

  1. Initial state: $\{at(r1, l1)\}$
     Encoding: $at(r1, l1, 0) \land \neg at(r1, l2, 0)$

  2. Goal: $\{at(r1, l2)\}$
     Encoding: $at(r1, l2, 1) \land \neg at(r1, l1, 1)$

  3. Operator: see next slide
• **Operator:** move\((r,l,l')\)
  
  precond: at\((r,l)\)
  
  effects: at\((r,l'), \neg\text{at}(r,l)\)

**Encoding:**

\[
\begin{align*}
\text{move}(r_1, l_1, l_2, 0) & \Rightarrow \text{at}(r_1, l_1, 0) \land \text{at}(r_1, l_2, 1) \land \neg\text{at}(r_1, l_1, 1) \\
\text{move}(r_1, l_2, l_1, 0) & \Rightarrow \text{at}(r_1, l_2, 0) \land \text{at}(r_1, l_1, 1) \land \neg\text{at}(r_1, l_2, 1) \\
\text{move}(r_1, l_1, l_1, 0) & \Rightarrow \text{at}(r_1, l_1, 0) \land \text{at}(r_1, l_1, 1) \land \neg\text{at}(r_1, l_1, 1) \quad \{\text{contradictions}\} \\
\text{move}(r_1, l_2, l_2, 0) & \Rightarrow \text{at}(r_1, l_2, 0) \land \text{at}(r_1, l_2, 1) \land \neg\text{at}(r_1, l_2, 1) \quad \{\text{easy to detect}\} \\
\text{move}(l_1, r_1, l_2, 0) & \Rightarrow \ldots \\
\text{move}(l_2, r_1, l_1, 0) & \Rightarrow \ldots \\
\text{move}(l_1, r_2, l_1, 0) & \Rightarrow \ldots \\
\text{move}(l_2, r_1, l_1, 0) & \Rightarrow \ldots \\
\end{align*}
\]

nonsensical, and we can avoid
generating them if we use data types
like we did for state-variable
representation

• **Operator:** move\((r: \text{robot}, l: \text{location}, l': \text{location})\)
  
  precond: at\((r,l)\)
  
  effects: at\((r,l'), \neg\text{at}(r,l)\)
4. Complete-exclusion axiom:
\( \neg \text{move}(r1, l1, l2, 0) \land \neg \text{move}(r1, l2, l1, 0) \)

5. Explanatory frame axioms:
\( \neg \text{at}(r1, l1, 0) \land \text{at}(r1, l1, 1) \Rightarrow \text{move}(r1, l2, l1, 0) \)
\( \neg \text{at}(r1, l2, 0) \land \text{at}(r1, l2, 1) \Rightarrow \text{move}(r1, l1, l2, 0) \)
\( \text{at}(r1, l1, 0) \land \neg \text{at}(r1, l1, 1) \Rightarrow \text{move}(r1, l1, l2, 0) \)
\( \text{at}(r1, l2, 0) \land \neg \text{at}(r1, l2, 1) \Rightarrow \text{move}(r1, l2, l1, 0) \)

• \( \Phi \) is the conjunct of all these
Summary of the Example

• $P$ is a planning problem with one robot and two locations
  ◦ initial state \{at(r1,l1)\}
  ◦ goal \{at(r1,l2)\}

• Encoding of $(P,1)$
  ◦ $\Phi =$
    
    \[
    \begin{align*}
    &\text{[at}(r1,l1,0) \land \neg \text{at}(r1,l2,0)] \quad \text{(initial state)} \\
    &\land [\text{at}(r1,l2,0) \land \neg \text{at}(r1,l1,1)] \quad \text{(goal)} \\
    &\land [\text{move}(r1,l1,l2,0) \Rightarrow \text{at}(r1,l1,0) \land \text{at}(r1,l2,1) \land \neg \text{at}(r1,l1,1)] \quad \text{(action)} \\
    &\land [\text{move}(r1,l2,l1,0) \Rightarrow \text{at}(r1,l2,0) \land \text{at}(r1,l1,1) \land \neg \text{at}(r1,l2,1)] \quad \text{(action)} \\
    &\land [\neg \text{move}(r1,l1,l2,0) \lor \neg \text{move}(r1,l2,l1,0)] \quad \text{(complete exclusion)} \\
    &\land [\neg \text{at}(r1,l1,0) \land \text{at}(r1,l1,1) \Rightarrow \text{move}(r1,l2,l1,0)] \quad \text{(frame axiom)} \\
    &\land [\neg \text{at}(r1,l2,0) \land \text{at}(r1,l2,1) \Rightarrow \text{move}(r1,l1,l2,0)] \quad \text{(frame axiom)} \\
    &\land [\text{at}(r1,l1,0) \land \neg \text{at}(r1,l1,1) \Rightarrow \text{move}(r1,l1,l2,0)] \quad \text{(frame axiom)} \\
    &\land [\text{at}(r1,l2,0) \land \neg \text{at}(r1,l2,1) \Rightarrow \text{move}(r1,l2,l1,0)] \quad \text{(frame axiom)}
    \end{align*}
    \]
Extracting a Plan

• Let Φ be an encoding of \((P,n)\)

• Suppose we find an assignment of truth values that satisfies Φ.
  ◦ This means \(P\) has a solution of length \(n\)

• For \(i = 1, \ldots, n\), there will be exactly one action \(a\) such that \(a_i = true\)
  ◦ This is the \(i\)'th action of the plan

• Example:
  • The formula on the previous side
    ◦ \(\Phi\) can be satisfied with \(move(r1,l1,l2,0) = true\)
      ⇒ Thus \(\langle move(r1,l1,l2,0)\rangle\) is a solution for \((P,1)\)
    ◦ It’s the only solution - no other way to satisfy \(\Phi\)
Heuristic Functions for Planning

- $\Delta^*(s, p)$: minimum distance from state $s$ to a state that contains $p$
- $\Delta^*(s, s')$: minimum distance from state $s$ to a state that contains all of the literals in $s'$
  - Hence $h^*(s) = \Delta^*(s, g)$ is the minimum distance from $s$ to the goal
- For $i = 0, 1, 2, \cdots$ we will define the following functions:
  - $\Delta_i(s, p)$: an estimate of $\Delta^*(s, p)$
  - $\Delta_i(s, s')$: an estimate of $\Delta^*(s, s')$
  - $h_i(s) = \Delta_i(s, g)$, where $g$ is the goal
- Estimating the heuristics is based on relaxation of the problem
- Ignoring negative preconditions and effects allows for very fast progression from initial state to goals
Heuristic Functions for Planning

- $\Delta_0(s, s') = \text{what we get if we pretend that}$
  - Negative preconditions and effects don’t exist
  - The cost of achieving a set of preconditions $\{p_1, \cdots, p_n\}$ is the sum of the costs of achieving each $p_i$ separately

\[
\Delta_0(s, p) = \begin{cases} 
0, & \text{if } p \in s \\
\infty, & \text{if } p \notin s \text{ and } \forall a \in A, p \notin \text{effects}^+(a) \\
\min_a \{1 + \Delta_0(s, \text{precond}^+(a)) | p \in \text{effects}^+(a)\}, & \text{otherwise}
\end{cases}
\]

\[
\Delta_0(s, g) = \begin{cases} 
0, & \text{if } g \subseteq s \\
\sum_{p \in g} \Delta_0(s, p), & \text{otherwise}
\end{cases}
\]

- $\Delta_0(s, s')$ is not admissible, but we don’t necessarily care
- Usually we’ll want to do a depth-first search, not an $A^*$ search
  - This already sacrifices admissibility (because DFS does not guarantee optimal solutions)
Computing $\Delta_0$

Delta(s)

foreach $p$ do
  if $p \in s$ then
    $\Delta_0(s, p) \leftarrow 0$
  else
    $\Delta_0(s, p) \leftarrow \infty$
  end
end

$U \leftarrow s$;

repeat
  $A \leftarrow \{a | \text{precond}(a) \subseteq U\}$;
  foreach $a \in A$ do
    $U \leftarrow U \cup \text{effects}^+(a)$;
    foreach $p \in \text{effects}^+(a)$ do
      $\Delta_0(s, p) \leftarrow \min\{\Delta_0(s, p), 1 + \sum_{q \in \text{precond}(a)} \Delta_0(s, q)\}$;
    end
  end
until no change occurs in the above updates;

Slightly modified from Dana Nau
Heuristic-forward-search($\pi, s, g, A$)

if $s$ satisfies $g$ then return $\pi$

$\text{options} \leftarrow \{ a \in A \mid a \text{ applicable to } s \}$
for each $a \in \text{options}$ do $\Delta_0(\gamma(s, a))$
while $\text{options} \neq \emptyset$ do
  $a \leftarrow \text{argmin}\{\Delta_0(\gamma(s, a), g) \mid a \in \text{options}\}$
  $\text{options} \leftarrow \text{options} - \{a\}$
  $\pi' \leftarrow \text{Heuristic-forward-search}(\pi.a, \gamma(s, a), g, A)$
  if $\pi' \neq \text{failure}$ then return($\pi'$)
return(failure)

• This is depth-first search, so admissibility is irrelevant
• This is roughly how the HSP planner works
  ◦ First successful use of an A*-style heuristic in classical planning
procedure Graphplan:

- for \( k = 0, 1, 2, \ldots \)

- **Graph expansion:**
  - create a planning graph that contains \( k \) levels

- Check whether the planning graph satisfies a necessary (but insufficient) condition for plan existence

- If it does, then
  - do solution extraction:
    - backward search, modified to consider only the actions in the planning graph
    - if we find a solution, then return it
• Search space for a relaxed version of the planning problem

• Alternating layers of ground literals and actions
  • Nodes at action-level $i$: actions that might be possible to execute at time $i$
  • Nodes at state-level $i$: literals that might possibly be true at time $i$
  • Edges: preconditions and effects

A maintenance action for a literal $l$. It represents what happens if we don’t change $l$. 
• Due to Dan Weld (U. of Washington)

• Suppose you want to prepare dinner as a surprise for your sweetheart (who is asleep)

\[ s_0 = \{ \text{garbage}, \text{cleanHands}, \text{quiet} \} \]
\[ g = \{ \text{dinner}, \text{present}, \neg \text{garbage} \} \]

<table>
<thead>
<tr>
<th>Action</th>
<th>Preconditions</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>cook()</td>
<td>cleanHands</td>
<td>dinner</td>
</tr>
<tr>
<td>wrap()</td>
<td>quiet</td>
<td>present</td>
</tr>
<tr>
<td>carry()</td>
<td>none</td>
<td>( \neg \text{garbage}, \neg \text{cleanHands} )</td>
</tr>
<tr>
<td>dolly()</td>
<td>none</td>
<td>( \neg \text{garbage}, \neg \text{quiet} )</td>
</tr>
</tbody>
</table>

Also have the maintenance action: one for each literal
• state-level 0:
  \[ \{ \text{all atoms in } s_0 \} \cup \{ \text{negations of all atoms not in } s_0 \} \]

• action-level 1:
  \[ \{ \text{all actions whose preconditions are satisfied and non-mutex in } s_0 \} \]

• state-level 1:
  \[ \{ \text{all effects of all of the actions in action-level 1} \} \]

<table>
<thead>
<tr>
<th>Action</th>
<th>Preconditions</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>cook()</td>
<td>cleanHands</td>
<td>dinner</td>
</tr>
<tr>
<td>wrap()</td>
<td>quiet</td>
<td>present</td>
</tr>
<tr>
<td>carry()</td>
<td>none</td>
<td>¬garbage, ¬cleanHands</td>
</tr>
<tr>
<td>dolly()</td>
<td>none</td>
<td>¬garbage, ¬quiet</td>
</tr>
</tbody>
</table>

Also have the maintenance action
Mutual Exclusion

- Two actions at the same action-level are mutex if
  - Inconsistent effects: an effect of one negates an effect of the other
  - Interference: one deletes a precondition of the other
  - Competing needs: they have mutually exclusive preconditions

- Otherwise they don’t interfere with each other
  - Both may appear in a solution plan

- Two literals at the same state-level are mutex if
  - Inconsistent support: one is the negation of the other, or all ways of achieving them are pairwise mutex
Planning is concerned with providing efficient algorithms to generate feasible action sequences.

Problem spaces are typically very large:

- DWR with 5 locations, 3 robot carts, 100 containers, 3 piles: $10^{277}$ states.
  Number of particles in the universe is about $10^{87}$

Most research is on classical planning with many different algorithms.

Planning Competition (starting at AIPS 1998, AIPS 2000, IPC 2002, now at ICAPS) shows the progress over the years.