Chapter 2
Representations for Classical Planning

Dana S. Nau
University of Maryland
Fall 2009
Quick Review of Classical Planning

- Classical planning requires all eight of the restrictive assumptions:
  A0: Finite
  A1: Fully observable
  A2: Deterministic
  A3: Static
  A4: Attainment goals
  A5: Sequential plans
  A6: Implicit time
  A7: Offline planning
Representations: Motivation

- In most problems, far too many states to try to represent all of them explicitly as $s_0, s_1, s_2, \ldots$
- Represent each state as a set of features
  - e.g.,
    - a vector of values for a set of variables
    - a set of ground atoms in some first-order language $L$
- Define a set of operators that can be used to compute state-transitions
- Don’t give all of the states explicitly
  - Just give the initial state
  - Use the operators to generate the other states as needed
Outline

- Representation schemes
  - Classical representation
  - Set-theoretic representation
  - State-variable representation
  - Examples: DWR and the Blocks World
  - Comparisons
Classical Representation

Start with a first-order language

- Language of first-order logic
- Restrict it to be *function-free*
- Finitely many predicate symbols and constant symbols, but *no* function symbols

Example: the DWR domain

- Locations: \( l_1, l_2, \ldots \)
- Containers: \( c_1, c_2, \ldots \)
- Piles: \( p_1, p_2, \ldots \)
- Robot carts: \( r_1, r_2, \ldots \)
- Cranes: \( k_1, k_2, \ldots \)
Classical Representation

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

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

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

- **Substitution**: $\theta = \{x_1 \leftarrow v_1, \ x_2 \leftarrow v_2, \ldots, \ x_n \leftarrow v_n\}$
  - 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
**States**

- *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

\[ s_1 = \{ \text{attached}(p1,\text{loc1}), \text{in}(c1,p1), \text{in}(c3,p1), \text{top}(c3,p1), \text{on}(c3,c1), \text{on}(c1,\text{pallet}), \text{attached}(p2,\text{loc1}), \text{in}(c2,p2), \text{top}(c2,p2), \text{on}(c2,\text{pallet}), \text{belong}(\text{crane1,loc1}), \text{empty}(\text{crane1}), \text{adjacent}(\text{loc1,loc2}), \text{adjacent}(\text{loc2,loc1}), \text{at}(r1,\text{loc2}), \text{occupied}(\text{loc2}), \text{unloaded}(r1) \}. \]
Operators

- **Operator**: a triple $o=(\text{name}(o), \text{precond}(o), \text{effects}(o))$
  - **precond($o$): **preconditions**
    » literals that must be true in order to use the operator
  - **effects($o$): **effects**
    » literals the operator will make true
  - **name($o$): **a syntactic expression of the form $n(x_1, \ldots, x_k)$
    » $n$ is an *operator symbol* - must be unique for each operator
    » $(x_1, \ldots, x_k)$ is a list of every variable symbol (parameter) that appears in $o$
  - Purpose of name($o$) is so we can refer unambiguously to instances of $o$

- Rather than writing each operator as a triple, we’ll usually write it in the following format:

\[
\text{take}(k, l, c, d, p) \\
\begin{align*}
\text{;; crane } k \text{ at location } l \text{ takes } c \text{ off of } d \text{ in pile } p \\
\text{precond: } & \text{belong}(k, l), \text{attached}(p, l), \text{empty}(k), \text{top}(c, p), \text{on}(c, d) \\
\text{effects: } & \text{holding}(k, c), \neg \text{empty}(k), \neg \text{in}(c, p), \neg \text{top}(c, p), \neg \text{on}(c, d), \text{top}(d, p)
\end{align*}
\]
move($r, l, m$)
  ;; robot $r$ moves from location $l$ to location $m$
  precond: adjacent($l, m$), at($r, l$), $\neg$occupied($m$)
  effects: at($r, m$), occupied($m$), $\neg$occupied($l$), $\neg$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($k$), $\neg$holding($k, c$), loaded($r, c$), $\neg$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: $\neg$empty($k$), holding($k, c$), unloaded($r$), $\neg$loaded

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: $\neg$holding($k, c$), empty($k$), in($c, p$), top($c, p$), on($c, d$), $\neg$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($k, c$), $\neg$empty($k$), $\neg$in($c, p$), $\neg$top($c, p$), $\neg$on($c, d$), top($d, p$)
An action is a ground instance (via substitution) of an operator

Note that an action's name identifies it unambiguously

- take(crane1, loc1, c3, c1, p1)
Notation

- Let $S$ be a set of literals. Then
  - $S^+ = \{\text{atoms that appear positively in } S\}$
  - $S^- = \{\text{atoms that appear negatively in } S\}$

- 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)

\text{;; crane } k \text{ at location } l \text{ takes } c \text{ off of } d \text{ in pile } p
\text{precond: } \text{belong}(k, l), \text{attached}(p, l), \text{empty}(k), \text{top}(c, p), \text{on}(c, d)
\text{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)
```

- $\text{effects}^+(\text{take}(k, l, c, d, p)) = \{\text{holding}(k, c), \text{top}(d, p)\}$
- $\text{effects}^-(\text{take}(k, l, c, d, p)) = \{\text{empty}(k), \text{in}(c, p), \text{top}(c, p), \text{on}(c, d)\}$
Applicability

- An action $a$ is *applicable* to a state $s$ if $s$ satisfies $\text{precond}(a)$,
  - i.e., if $\text{precond}^+(a) \subseteq s$ and $\text{precond}^-(a) \cap s = \emptyset$

- An action:

  \begin{align*}
  \text{take}(\text{crane1, loc1, c3, c1, p1}) \\
  \text{precond: } & \text{belong(crane, loc1),} \\
  & \text{attached(p1, loc1),} \\
  & \text{empty(crane1), top(c3, p1),} \\
  & \text{on(c3, c1)} \\
  \text{effects: } & \text{holding(crane1, c3),} \\
  & \neg \text{empty(crane1),} \\
  & \neg \text{in(c3, p1),} \neg \text{top(c3, p1),} \\
  & \neg \text{on(c1, c1), top(c1, p1)}
  \end{align*}

- A state it’s applicable to

  \[ s_1 = \{\text{attached}(p1, \text{loc1}), \text{in}(c1, p1), \] 
  \[ \text{in}(c3, p1), \text{top}(c3, p1), \text{on}(c3, c1), \] 
  \[ \text{on}(c1, \text{pallet}), \text{attached}(p2, \text{loc1}), \] 
  \[ \text{in}(c2, p2), \text{top}(c2, p2), \text{on}(c2, \text{pallet}), \] 
  \[ \text{belong}(\text{crane1, loc1}), \] 
  \[ \text{empty}(\text{crane1}), \text{adjacent}(\text{loc1, loc2}), \] 
  \[ \text{adjacent}(\text{loc2, loc1}), \text{at}(r1, \text{loc2}), \] 
  \[ \text{occupied}(\text{loc2, unloaded}(r1)) \]
Executing an Applicable Action

- Remove a’s negative effects, and add a’s positive effects

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

\[
\text{take} \left( \text{crane1}, \text{loc1}, \text{c3}, \text{c1}, \text{p1} \right)
\]
- \text{precond: } \text{belong} \left( \text{crane}, \text{loc1} \right), \text{attached} \left( \text{p1}, \text{loc1} \right), \text{empty} \left( \text{crane1} \right), \text{top} \left( \text{c3}, \text{p1} \right), \text{on} \left( \text{c3}, \text{c1} \right)
- \text{effects: } \text{holding} \left( \text{crane1}, \text{c3} \right), \neg \text{empty} \left( \text{crane1} \right), \neg \text{in} \left( \text{c3}, \text{p1} \right), \neg \text{top} \left( \text{c3}, \text{p1} \right), \neg \text{on} \left( \text{c1}, \text{c1} \right), \text{top} \left( \text{c1}, \text{p1} \right)

\[ s_2 = \{ \text{attached} \left( \text{p1}, \text{loc1} \right), \text{in} \left( \text{c1}, \text{p1} \right), \text{in} \left( \text{c3}, \text{p1} \right), \text{top} \left( \text{c3}, \text{p1} \right), \text{on} \left( \text{c3}, \text{c1} \right), \text{on} \left( \text{c1}, \text{pallet} \right), \text{attached} \left( \text{p2}, \text{loc1} \right), \text{in} \left( \text{c2}, \text{p2} \right), \text{top} \left( \text{c2}, \text{p2} \right), \text{on} \left( \text{c2}, \text{pallet} \right), \text{belong} \left( \text{crane1}, \text{loc1} \right), \text{empty} \left( \text{crane1} \right), \text{adjacent} \left( \text{loc1}, \text{loc2} \right), \text{occupied} \left( \text{loc2}, \text{unloaded} \left( \text{r1} \right) \right), \text{holding} \left( \text{crane1}, \text{c3} \right), \text{top} \left( \text{c1}, \text{p1} \right) \} \]
Planning Problems

- Given a planning domain (language $L$, operators $O$)
  - **Statement** of a planning problem: a triple $P = (O, s_0, g)$
    - $O$ is the collection of operators
    - $s_0$ is a state (the initial state)
    - $g$ is a set of literals (the goal formula)
  - Planning problem: $\mathcal{P} = (\Sigma, s_0, S_g)$
    - $s_0 = \text{initial state}$
    - $S_g = \text{set of goal states}$
    - $\Sigma = (S, A, \gamma)$ is a state-transition system
    - $S = \{\text{all sets of ground atoms in } L\}$
    - $A = \{\text{all ground instances of operators in } O\}$
    - $\gamma = \text{the state-transition function determined by the operators}$
- I’ll often say “planning problem” when I mean the statement of the problem
Plans and Solutions

- **Plan**: any sequence of actions \( \sigma = \langle a_1, a_2, \ldots, a_n \rangle \) such that each \( a_i \) is an 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
    - \( \gamma(s_0, a_1) = s_1 \)
    - \( \gamma(s_1, a_2) = s_2 \)
    - \( \ldots \)
    - \( \gamma(s_{n-1}, a_n) = s_n \)
    - \( s_n \) satisfies \( g \)
Example

- Let $P_1 = (O, s_1, g_1)$, where
  - $O = \{\text{the four DWR operators given earlier}\}$

\[
\begin{align*}
  &O = \{\text{the four DWR operators given earlier}\} \\
  &g_1 = \{\text{loaded}(r1,c3), \text{at}(r1,\text{loc2})\} \\
  &s_1 = \{\text{attached}(p1,\text{loc1}), \text{in}(c1,p1), \text{in}(c3,p1), \text{top}(c3,p1), \text{on}(c3,c1), \text{on}(c1,\text{pallet}), \text{attached}(p2,\text{loc1}), \text{in}(c2,p2), \text{top}(c2,p2), \text{on}(c2,\text{pallet}), \text{belong}(\text{crane1,loc1}), \text{empty}(\text{crane1}), \text{adjacent}(\text{loc2,loc1}), \text{adjacent}(\text{loc2,loc1}), \text{at}(r1,\text{loc2}), \text{occupied}(\text{loc2}), \text{unloaded}(r1)\}.
\end{align*}
\]
Example, continued

- $P_1$ has infinitely many solutions
- Here are three of them:

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

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

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

- They each produce this state:
Example, continued

- The first one is *redundant*
  - Can remove actions and still have a solution

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

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

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

- The 2nd and 3rd are *irredundant*
- They also are *shortest*
  - No shorter solutions exist
Set-Theoretic Representation

- Like classical representation, but restricted to propositional logic
  - Equivalent to a classical representation in which all of the atoms are ground

**States:**
- Instead of ground atoms, use propositions (boolean variables):

\[
\{\text{on}(c1,\text{pallet}), \text{on}(c1,r1), \text{on}(c1,c2), \ldots, \text{at}(r1,l1), \text{at}(r1,l2), \ldots\}
\]

\[
\downarrow
\]

\[
\{\text{on-c1-pallet}, \text{on-c1-r1}, \text{on-c1-c2}, \ldots, \text{at-r1-l1}, \text{at-r1-l2}, \ldots\}
\]
Set-Theoretic Representation, continued

No operators, just actions:

- Instead of ground atoms, use propositions
- Instead of negative effects, use a delete list
- If you have negative preconditions, create new atoms to represent them
  - E.g., instead of using \( \neg \text{foo} \) as a precondition, use \( \text{not-foo} \)
- To use both \( \text{foo} \) and \( \text{not-foo} \):
  - Actions that delete one should add the other, and vice versa

\[
\begin{align*}
\text{take}(\text{crane1}, \text{loc1}, \text{c3}, \text{c1}, \text{p1}) \\
\text{precond:} & \quad \text{belong(\text{crane}, \text{loc1})}, \\
& \quad \text{attached(\text{p1}, \text{loc1})}, \text{empty(\text{crane1})}, \\
& \quad \text{top(\text{c3}, \text{p1})}, \text{on(\text{c3}, \text{c1})} \\
\text{effects:} & \quad \text{holding(\text{crane1}, \text{c3})}, \\
& \quad \neg \text{empty(\text{crane1})}, \\
& \quad \neg \text{in(\text{c3}, \text{p1})}, \neg \text{top(\text{c3}, \text{p1})}, \neg \text{on(\text{c1}, \text{c1})}, \\
& \quad \text{top(\text{c1}, \text{p1})} \\
\downarrow \\
\text{take-crane1-loc1-c3-c1-p1} \\
\text{precond:} & \quad \text{belong-\text{crane1-loc1}}, \\
& \quad \text{attached-p1-\text{loc1}}, \text{empty-\text{crane1}}, \\
& \quad \text{top-c3-p1}, \text{on-c3-c1} \\
\text{delete:} & \quad \text{empty-\text{crane1}}, \\
& \quad \text{in-c3-p1}, \text{top-c3-p1}, \text{on-c3-p1} \\
\text{add:} & \quad \text{holding-\text{crane1-c3}}, \text{top-c1-p1}
\end{align*}
\]
Exponential Blowup

- Suppose a classical operator contains $n$ atoms and each atom has arity $k$
- Suppose the language contains $c$ constant symbols
- Then there are $c^{nk}$ ground instances of the operator
  - Hence $c^{nk}$ set-theoretic actions
- Can reduce this by removing operator instances in which the arguments don’t make sense
  - $\text{take}(\text{crane1, crane1, crane1, crane1, crane1, crane1})$
  - Worst case is still exponential
State-Variable Representation

- Use ground atoms for properties that do not change, e.g., adjacent(loc1, loc2)
- For properties that can change, assign values to 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

\[ \text{move}(r, l, m) \]
\[ ;; \text{robot } r \text{ at location } l \text{ moves to an adjacent location } m \]
\[ \text{precond: } rloc(r) = l, \text{adjacent}(l, m) \]
\[ \text{effects: } rloc(r) \leftarrow m \]

\[ s_1 = \{ \text{top}(p1) = c3, \]
\[ \text{cpos}(c3) = c1, \]
\[ \text{cpos}(c1) = \text{pallet}, \]
\[ \text{holding}(\text{crane1}) = \text{nil}, \]
\[ rloc(r1) = \text{loc2}, \]
\[ \text{loaded}(r1) = \text{nil}, \text{...} \} \]
Example: The Blocks World

- Infinitely wide table, finite number of children’s blocks
- Ignore where a block is located on the table
- A block can sit on the table or on another block
- There’s a robot gripper that can hold at most one block

- Want to move blocks from one configuration to another
  - e.g.,

```
initial state  goal

  a  b  c
  a  b  e
  c  d
```

- Like a special case of DWR with one location, one crane, some containers, and many more piles than you need
- I’ll give classical, set-theoretic, and state-variable formulations
  - For the case where there are five blocks
Classical Representation: Symbols

- **Constant symbols:**
  - The blocks: a, b, c, d, e

- **Predicates:**
  - ontable\( (x) \) - block \( x \) is on the table
  - on\( (x,y) \) - block \( x \) is on block \( y \)
  - clear\( (x) \) - block \( x \) has nothing on it
  - holding\( (x) \) - the robot hand is holding block \( x \)
  - handempty - the robot hand isn’t holding anything
Classical Operators

unstack\((x,y)\)
Precond: \(on(x,y), clear(x), handempty\)
 Effects: \(\neg on(x,y), \neg clear(x), \neg handempty,\)
 \(holding(x), clear(y)\)

stack\((x,y)\)
Precond: \(holding(x), clear(y)\)
 Effects: \(\neg holding(x), \neg clear(y),\)
 \(on(x,y), clear(x), handempty\)

pickup\((x)\)
Precond: \(ontable(x), clear(x), handempty\)
 Effects: \(\neg ontable(x), \neg clear(x),\)
 \(\neg handempty, holding(x)\)

putdown\((x)\)
Precond: \(holding(x)\)
 Effects: \(\neg holding(x), ontable(x),\)
 \(clear(x), handempty\)
For five blocks, there are 36 propositions

Here are 5 of them:

- **ontable-a** - block a is on the table
- **on-c-a** - block c is on block a
- **clear-c** - block c has nothing on it
- **holding-d** - the robot hand is holding block d
- **handempty** - the robot hand isn’t holding anything
Set-Theoretic Actions

- There are fifty different actions
- Here are four of them:

  **unstack-c-a**
  - Pre: on-c-a, clear-c, handempty
  - Del: on-c-a, clear-c, handempty
  - Add: holding-c, clear-a

  **stack-c-a**
  - Pre: holding-c, clear-a
  - Del: holding-c, clear-a
  - Add: on-c-a, clear-c, handempty

  **pickup-c**
  - Pre: ontable-c, clear-c, handempty
  - Del: ontable-c, clear-c, handempty
  - Add: holding-c

  **putdown-c**
  - Pre: holding-c
  - Del: holding-c
  - Add: ontable-c, clear-c, handempty
State-Variable Representation: Symbols

- Constant symbols:
  - a, b, c, d, e of type block
  - 0, 1, table, nil of type other

- State variables:
  - \( \text{pos}(x) = y \) if block \( x \) is on block \( y \)
  - \( \text{pos}(x) = \text{table} \) if block \( x \) is on the table
  - \( \text{pos}(x) = \text{nil} \) if block \( x \) is being held
  - \( \text{clear}(x) = 1 \) if block \( x \) has nothing on it
  - \( \text{clear}(x) = 0 \) if block \( x \) is being held or has another block on it
  - \( \text{holding} = x \) if the robot hand is holding block \( x \)
  - \( \text{holding} = \text{nil} \) if the robot hand is holding nothing
State-Variable Operators

unstack(x : block, y : block)
  Precond: pos(x)=y, clear(y)=0, clear(x)=1, holding=nil
  Effects: pos(x)=nil, clear(x)=0, holding=x, clear(y)=1

stack(x : block, y : block)
  Precond: holding=x, clear(x)=0, clear(y)=1
  Effects: holding=nil, clear(y)=0, pos(x)=y, clear(x)=1

pickup(x : block)
  Precond: pos(x)=table, clear(x)=1, holding=nil
  Effects: pos(x)=nil, clear(x)=0, holding=x

putdown(x : block)
  Precond: holding=x
  Effects: holding=nil, pos(x)=table, clear(x)=1
Expressive Power

- Any problem that can be represented in one representation can also be represented in the other two.
- Can convert in linear time and space in all cases except one:
  - Exponential blowup when converting to set-theoretic.
Comparison

- Classical representation
  - The most popular for classical planning, partly for historical reasons

- Set-theoretic representation
  - Can take much more space than classical representation
  - Useful in algorithms that manipulate ground atoms directly
    » e.g., planning graphs (Chapter 6), satisfiability (Chapters 7)
  - Useful for certain kinds of theoretical studies

- State-variable representation
  - Equivalent to classical representation in expressive power
  - Less natural for logicians, more natural for engineers
  - Useful in non-classical planning problems as a way to handle numbers, functions, time