Chapter 2
Representations for Classical Planning

Dana S. Nau
CMSC 722, AI Planning
University of Maryland, Spring 2008
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 *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($l, l'$)
    - 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
Operators

- **Operator**: a triple $o=(\text{name}(o), \text{precond}(o), \text{effects}(o))$
  - $\text{name}(o)$ is a syntactic expression of the form $n(x_1, \ldots, x_k)$
    - $n$: *operator symbol* - must be unique for each operator
    - $x_1, \ldots, x_k$: 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

```plaintext
take(k, l, c, d, p)
    ;; crane k at location l takes c off of d 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)
```
Actions

- **Action**: ground instance (via substitution) of an operator

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

\[
\text{take}(\text{crane1}, \text{loc1}, c3, c1, p1) \\
\text{;; crane crane1 at location loc1 takes c3 off c1 in pile p1} \\
\text{precond: belong(\text{crane1}, \text{loc1}), attached(p1, \text{loc1}), empty(\text{crane1}), top(c3, p1), on(c3, c1)} \\
\text{effects: holding(\text{crane1}, c3), \neg empty(\text{crane1}), \neg in(c3, p1),} \\
\neg \text{top}(c3, p1), \neg \text{on}(c3, c1), \text{top}(c1, p1)
\]
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
  - $\text{precond}^+(a) = \{\text{atoms that appear positively in } a\text{'s preconditions}\}$
  - $\text{precond}^-(a) = \{\text{atoms that appear negatively in } a\text{'s preconditions}\}$
  - $\text{effects}^+(a) = \{\text{atoms that appear positively in } a\text{'s effects}\}$
  - $\text{effects}^-(a) = \{\text{atoms that appear negatively in } a\text{'s effects}\}$

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

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

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

- Here are an action and a state that it’s applicable to:

\[
\text{take(crane1,loc1,c3,c1,p1)} \\
\hspace{1cm} ;; \text{crane crane1 at location loc1 takes c3 off c1 in pile p1} \\
\text{precond: belong(crane1,loc1), attached(p1,loc1), empty(crane1), top(c3,p1), on(c3,c1)} \\
\text{effects: holding(crane1,c3), \neg empty(crane1), \neg in(c3,p1), \neg \text{top(c3,p1)}, \neg on(c3,c1), \text{top(c1,p1)}}
\]
Result of Performing an Action

- 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

```plaintext
take(crane1,loc1,c3,c1,p1)
  ;; crane crane1 at location loc1 takes c3 off c1 in pile p1
  precond: belong(crane1,loc1), attached(p1,loc1),
           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)
```

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

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
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)
  - The actual planning problem: \( \mathcal{P}=(\Sigma,s_0,S_g) \)
    - \( s_0 \) and \( S_g \) are as above
    - \( \Sigma=(S,A,\gamma) \) 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 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
    - $\gamma(s_0, a_1) = s_1$
    - $\gamma(s_1, a_2) = s_2$
    - $\ldots$
    - $\gamma(s_{n-1}, a_n) = s_n$
    - $s_n$ satisfies $g$
Example

- Let $P_1 = (O, s_1, g_1)$, where
  - $O$ is the set of operators given earlier
  - $g_1 = \{\text{loaded}(r1,c3), \text{at}(r1,\text{loc2})\}$
  - $s_1 = \{\text{attached}(p1,\text{loc1}), \text{in}(c1,p1), \text{in}(c3,p1), \text{top}(c3,p1), \text{on}(c3,c1), \text{on}(c1,\text{pallet}), \text{attached}(p2,\text{loc1}), \text{in}(c2,p2), \text{top}(c2,p2), \text{on}(c2,\text{pallet}), \text{belong}(\text{crane1},\text{loc1}), \text{empty}(\text{crane1}), \text{adjacent}(\text{loc2},\text{loc1}), \text{adjacent}(\text{loc2},\text{loc1}), \text{at}(r1,\text{loc2}), \text{occupied}(\text{loc2}), \text{unloaded}(r1)\}$. 
Example (continued)

Here are three solutions for $P_1$:

- $\langle$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)$\rangle$

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

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

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

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

\[
;; \text{crane } k \text{ at location } l \text{ takes } c \text{ off of } d \text{ in pile } p
\]

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

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

take all of the operator instances, e.g., this one,

\[
\text{take}(\text{crane}1, \text{loc}1, c3, c1, p1)
\]

\[
;; \text{crane } \text{crane}1 \text{ at location } \text{loc}1 \text{ takes } c3 \text{ off } c1 \text{ in pile } p1
\]

precond: \(\text{belong}(\text{crane}1, \text{loc}1), \text{attached}(p1, \text{loc}1), \text{empty}(\text{crane}1), \text{top}(c3, p1), \text{on}(c3, c1)\)

effects: \(\text{holding}(\text{crane}1, c3), \neg \text{empty}(\text{crane}1), \neg \text{in}(c3, p1), \neg \text{top}(c3, p1), \neg \text{on}(c3, c1), \text{top}(c1, p1)\)

and rewrite ground atoms as propositions

\[
\text{take-crate1-loc1-c3-c1-p1}
\]

precond: \(\text{belong-crate1-loc1}, \text{attached-p1-loc1}, \text{empty-crate1}, \text{top-c3-p1}, \text{on-c3-c1}\)

delete: \(\text{empty-crate1}, \text{in-c3-p1}, \text{top-c3-p1}, \text{on-c3-p1}\)

add: \(\text{holding-crate1-c3}, \text{top-c1-p1}\)
Comparison

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

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

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

move(r, l, m)

;; robot r at location l moves to an adjacent location m
precond: rloc(r) = l, adjacent(l, m)
effects: rloc(r) ← m

```plaintext
{top(p1)=c3,
cpos(c3)=c1,
cpos(c1)=pallet,
holding(crane1)=nil,
rloc(r1)=loc2,
loaded(r1)=nil, ...}
```
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
- Want to move blocks from one configuration to another
  - e.g.,

![Initial state diagram](image)

- Can be expressed as a special case of DWR
  - But the usual formulation is simpler
- 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: ~on(x,y), ~clear(x), ~handempty, holding(x), clear(y)

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

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

putdown(x)
Precond:  holding(x)
Effects: ~holding(x), ontable(x), clear(x), handempty
Set-Theoretic Representation: Symbols

- 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

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:** on-table-c, clear-c, handempty
- **Del:** on-table-c, clear-c, handempty
- **Add:** holding-c

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

- **Constant symbols:**
  - $a, b, c, d, e$ of type block
  - $0, 1, \text{table}, \text{nil}$ 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($x : \text{block}, y : \text{block}$)

Precond: $\text{pos}(x)=y$, $\text{clear}(y)=0$, $\text{clear}(x)=1$, $\text{holding}=\text{nil}$

Effects: $\text{pos}(x)=\text{nil}$, $\text{clear}(x)=0$, $\text{holding}=x$, $\text{clear}(y)=1$

stack($x : \text{block}, y : \text{block}$)

Precond: $\text{holding}=x$, $\text{clear}(x)=0$, $\text{clear}(y)=1$

Effects: $\text{holding}=\text{nil}$, $\text{clear}(y)=0$, $\text{pos}(x)=y$, $\text{clear}(x)=1$

pickup($x : \text{block}$)

Precond: $\text{pos}(x)=\text{table}$, $\text{clear}(x)=1$, $\text{holding}=\text{nil}$

Effects: $\text{pos}(x)=\text{nil}$, $\text{clear}(x)=0$, $\text{holding}=x$

putdown($x : \text{block}$)

Precond: $\text{holding}=x$

Effects: $\text{holding}=\text{nil}$, $\text{pos}(x)=\text{table}$, $\text{clear}(x)=1$
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, except when converting to set-theoretic (where we get an exponential blowup).
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