Chapter 2
Representations for Classical Planning
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
  - *Atom*: predicate symbol and args - e.g., on(c1,c3), on(c1,x)
  - *Ground* expression: contains no variable symbols - e.g., on(c1,c3)
  - *Unground* expression: at least one variable symbol - e.g., on(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

- *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
Example of a State

\{\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 $o=(\text{name}(o), \text{precond}(o), \text{effects}(o))$
  - **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 $o$
  - **precond**($o$): *preconditions*
    - literals that must be true in order to use the operator
  - **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 $p$
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)
```
Actions

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

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

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

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

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

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

effects: \(\text{holding}(\text{crane1}, c3), \neg \text{empty}(\text{crane1}), \neg \text{in}(c3, p1), \neg \text{top}(c3, p1), \neg \text{on}(c3, c1), \text{top}(c1, p1)\)
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
  - $\text{precond}^+(a) = \{\text{atoms that appear positively in } a\text{'s preconditions}\}$
  - $\text{precond}^-(a) = \{\text{atoms that appear negatively in } a\text{'s preconditions}\}$
  - $\text{effects}^+(a) = \{\text{atoms that appear positively in } a\text{'s effects}\}$
  - $\text{effects}^-(a) = \{\text{atoms that appear negatively in } a\text{'s effects}\}$

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

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

- An action \( a \) is \textit{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:

```
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)
```
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

```
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*: $P = (\Sigma, s_0, S_g)$
    - $s_0$ and $S_g$ are as above
    - $\Sigma = (S, A, \gamma)$ is a state-transition system
      - $S = \{\text{all sets of ground atoms in } L\}$
      - $A = \{\text{all ground instances of operators in } O\}$
      - $\gamma = \text{the state-transition function determined by the operators}$
- “planning problem” often means 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

- $g_1 = \{ \text{loaded}(r1, c3), \text{at}(r1, \text{loc2}) \}$

- $s_1 = \{ \text{attached}(p1, \text{loc1}), \text{in}(c1, p1), \text{in}(c3, p1), \text{top}(c3, p1), \text{on}(c3, c1), \text{on}(c1, \text{pallet}), \text{attached}(p2, \text{loc1}), \text{in}(c2, p2), \text{top}(c2, p2), \text{on}(c2, \text{pallet}), \text{belong}(\text{crane1, loc1}), \text{empty}(\text{crane1}), \text{adjacent}(\text{loc1, loc2}), \text{adjacent}(\text{loc2, loc1}), \text{at}(r1, \text{loc2}), \text{occupied}(\text{loc2}), \text{unloaded}(r1) \}$. 
Here are three solutions for $P_1$:

- $\langle \text{take}(\text{crane}1, \text{loc1}, c3, c1, p1), \text{move}(r1, \text{loc2}, \text{loc1}), \text{move}(r1, \text{loc1}, \text{loc2}), \text{move}(r1, \text{loc2}, \text{loc1}), \text{load}(\text{crane}1, \text{loc1}, c3, r1), \text{move}(r1, \text{loc1}, \text{loc2}) \rangle$
- $\langle \text{take}(\text{crane}1, \text{loc1}, c3, c1, p1), \text{move}(r1, \text{loc2}, \text{loc1}), \text{load}(\text{crane}1, \text{loc1}, c3, r1), \text{move}(r1, \text{loc1}, \text{loc2}) \rangle$
- $\langle \text{move}(r1, \text{loc2}, \text{loc1}), \text{take}(\text{crane}1, \text{loc1}, c3, c1, p1), \text{load}(\text{crane}1, \text{loc1}, c3, r1), \text{move}(r1, \text{loc1}, \text{loc2}) \rangle$

Each of them produces the state shown here:
The first is redundant: can remove actions and still have a solution

- \langle \text{take}(\text{crane}1, \text{loc}1, c3, c1, p1), \text{move}(r1, \text{loc}2, \text{loc}1), \text{move}(r1, \text{loc}1, \text{loc}2), \text{move}(r1, \text{loc}2, \text{loc}1), \text{load}(\text{crane}1, \text{loc}1, c3, r1), \text{move}(r1, \text{loc}1, \text{loc}2) \rangle

- \langle \text{take}(\text{crane}1, \text{loc}1, c3, c1, p1), \text{move}(r1, \text{loc}2, \text{loc}1), \text{load}(\text{crane}1, \text{loc}1, c3, r1), \text{move}(r1, \text{loc}1, \text{loc}2) \rangle

- \langle \text{move}(r1, \text{loc}2, \text{loc}1), \text{take}(\text{crane}1, \text{loc}1, c3, c1, p1), \text{load}(\text{crane}1, \text{loc}1, c3, r1), \text{move}(r1, \text{loc}1, \text{loc}2) \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 …
    \{on(c1,pallet), on(c1,r1), on(c1,c2), ..., at(r1,l1), at(r1,l2), ...\}
  - ... use a collection of propositions (boolean variables):
    \{on-c1-pallet, on-c1-r1, on-c1-c2, ..., at-r1-l1, at-r1-l2, ...\}
Instead of operators like this one,

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

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

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

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

\[
\begin{align*}
\text{;; crane crane1 at location loc1 takes c3 off c1 in pile p1} \\
\text{precond: belong}(\text{crane1}, \text{loc1}), \text{attached}(p1, \text{loc1}), \\
\text{empty}(\text{crane1}), \text{top}(c3, p1), \text{on}(c3, c1) \\
\text{effects: holding}(\text{crane1}, c3), \neg \text{empty}(\text{crane1}), \neg \text{in}(c3, p1), \\
\neg \text{top}(c3, p1), \neg \text{on}(c3, c1), \text{top}(c1, p1)
\end{align*}
\]

and rewrite ground atoms as propositions

\[
\text{take-crane1-loc1-c3-c1-p1}
\]

\[
\begin{align*}
\text{precond: belong-crane1-loc1, attached-p1-loc1,} \\
\text{empty-crane1, top-c3-p1, on-c3-c1} \\
\text{delete: empty-crane1, in-c3-p1, top-c3-p1, on-c3-p1} \\
\text{add: holding-crane1-c3, top-c1-p1}
\end{align*}
\]
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 $k$ atoms and each atom has arity $n$, then it corresponds to $c^{nk}$ actions where $c = |\{\text{constant symbols}\}|$. 

An example

- Suppose a computer has a n-bit register $r$ and a single operator $inc$ that assigns $r \leftarrow r + 1 \text{ mod } m$, where $m = 2^n$
- Let $L = \{ \text{val}_0, \ldots, \text{val}_{m-1} \}$, where val$_i$ means “$r$ contains value $i$”
- $\Sigma = (S, A, \gamma)$, where $s \in L$, $A = \{inc\}$, $\gamma(\text{val}_i, \text{inc}) = \text{val}_{i+1} \text{ mod } m$
- Suppose $P = (\Sigma, s_0, g)$, where $s_0 = \text{val}_c$, $S_g = \{\text{val}_i| i \text{ is prime}\}$
  - There is no set-theoretic action representation for inc,
  - nor any set of proposition $g \subseteq 2^L$ that represents the set of goal states $S_g$
An Example (Cont.)

- However, we can define $\Sigma'$ and $P'$ as follows:
  - $L' = L \cup \{\text{prime}\}$, $S' = 2^{L'}$, $A' = \{\text{inc}_0, \ldots, \text{inc}_{m-1}\}$
  - $\gamma'(\text{val}_i, \text{inc}) = \{\text{prime}, \text{val}_{i+1 \mod m}\}$ if $i + 1 \mod m$ is prime, or
  - $\gamma' = \{\text{val}_{i+1 \mod m}\}$, otherwise
  - $\Sigma' = (S', A', \gamma')$, $S'_g = \{s \subset 2^{L'} | \text{prime} \in s\}$, $P' = (\Sigma', s_0, S'_g)$

- $P'$ has the following set-theoretic representation:
  - $g' = \{\text{prime}\}$
  - $\text{precond}(\text{inc}_i) = \{\text{val}_i\}$, $i = 1, \ldots, m$
  - $\text{effect}^{-}(\text{inc}_i) = \{\text{val}_i, \neg\text{prime}\}$, if $i$ is prime, or
  - $\text{effect}^{-} = \{\text{val}_i\}$, otherwise
  - $\text{effect}^{+}(\text{inc}_i) = \{\text{val}_{i+1 \mod m}, \text{prime}\}$, if $i + 1 \mod m$ is prime, or
  - $\text{effect}^{+} = \{\text{val}_{i+1 \mod m}\}$, otherwise

- There are $2^n$ different actions, and to write them, we must compute all prime numbers between 1 and $2^n$
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 \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

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

\{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.,

```
initital state: c  a  b  e
goal: a  b  c
d```

- 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:** $\text{on}(x,y), \text{clear}(x), \text{handempty}$
- **Effects:** $\neg\text{on}(x,y), \neg\text{clear}(x), \neg\text{handempty}, \text{holding}(x), \text{clear}(y)$

### stack($x,y$)
- **Precond:** $\text{holding}(x), \text{clear}(y)$
- **Effects:** $\neg\text{holding}(x), \neg\text{clear}(y), \text{on}(x,y), \text{clear}(x), \text{handempty}$

### pickup($x$)
- **Precond:** $\text{ontable}(x), \text{clear}(x), \text{handempty}$
- **Effects:** $\neg\text{ontable}(x), \neg\text{clear}(x), \neg\text{handempty}, \text{holding}(x)$

### putdown($x$)
- **Precond:** $\text{holding}(x)$
- **Effects:** $\neg\text{holding}(x), \text{ontable}(x), \text{clear}(x), \text{handempty}$
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
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

\textbf{unstack}(x : \text{block}, y : \text{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

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

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

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

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

\textbf{putdown}(x : \text{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, except when converting to set-theoretic (where we get an exponential blowup).

\[ P(x_1, \ldots, x_n) \]

becomes

\[ f_P(x_1, \ldots, x_n) = 1 \]

\[ f(x_1, \ldots, x_n) = y \]

becomes

\[ P_f(x_1, \ldots, x_n, y) \]

Set-theoretic representation

Classical representation

State-variable representation
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)

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