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
Problem Space and Search Tree

- problem spaces are for most planning problems too large to be represented completely
- remember: number of states for DWR
- Therefore, standard graph search algorithms (such as the Dijkstra algorithm) for finding the shortest path between two nodes (i.e. initial state and a state fulfilling the goals) cannot be applied
- During search for a plan, a part of the problem space is explored (search tree)
Problem Space and Search Tree

- Example: problem space
Problem Space and Search Tree

Example: search tree
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: $l_1, l_2, \ldots$
- Containers: $c_1, c_2, \ldots$
- Piles: $p_1, p_2, \ldots$
- Robot carts: $r_1, r_2, \ldots$
- Cranes: $k_1, k_2, \ldots$
More Formal Treatment of Representation

First-order language, function-free and without equality:
Signature $\Sigma = (S, OP, REL)$

- Do not confuse signature “Sigma“ with state-transition system (also denoted with $\Sigma$)
- a signature defines the syntactical structure of a language
- It is defined by three sets: sorts, operators, and relations
- Operators have different arities, constants are operators with arity zero (range represents “type“)
- Relations have range “prop“ which can be true or false
- Never confuse operators (with a range of some sort) and relations which result in a truth value!

as introduced by Michael Mender
More Formal Treatment of Representation

Sorts $S$:

$$S = \{ \text{location, container, pile, } \ldots \}$$

Operators $OP$:

$$OP = \{ l_1, l_2, \ldots : \text{location},$$
$$c_1, c_2, \ldots : \text{container},$$
$$p_1, p_2, \ldots : \text{piles},$$
$$r_1, r_2, \ldots : \text{robot carts},$$
$$k_1, k_2, \ldots : \text{cranes} \}$$

Relations $REL$:

$$REL = \{ \text{adjacent: location } \times \text{ location } \rightarrow \text{ Prop},$$
$$\text{occupied: location } \rightarrow \text{ Prop},$$
$$\text{loaded: robot\_cart } \times \text{ container } \rightarrow \text{ Prop},$$
$$\ldots \}$$
More Formal Treatment of Representation

- **Ground** (or closed) expression: contains no variable symbols
  - e.g., \( \text{in}(c_1, p_3) \)  
    - formally, we write \( \emptyset \vdash \text{in}(c_1, p_3) : \text{Prop} \)

- **Unground** (or open) expression: at least one variable symbol
  - e.g., \( \text{in}(c_1, x) \)  
    - formally, we write \( x : \text{pile} \vdash \text{in}(c_1, x) : \text{Prop} \)

- **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 (of the same type)
  - \( \theta \) is ground if all \( v_i \) are ground

- **Instance** \( e\theta \) of expression \( e \):
  - \( \text{in}(c_1, x)\{x \leftarrow p_3\} = \text{in}(c_1, p_3) \)
  - result of applying a substitution \( \theta \) to \( e \)
  - Replace variables of \( e \) simultaneously, not sequentially
  - if \( e\theta \) is ground, called ground instance.

\[
\emptyset \vdash p_3 : \text{pile} \quad x : \text{pile} \vdash \text{in}(c_1, x) : \text{Prop} \\
\emptyset \vdash \text{in}(c_1, x)\{x \leftarrow p_3\} : \text{Prop}
\]
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')`
    - `top(pallet,p)`
    - `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**: \( \theta = \{ x_1 \leftarrow v_1, x_2 \leftarrow, \ldots, x_n \leftarrow v_n \} \)
  - 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

Dana Nau: Lecture slides for Automated Planning
Licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License: http://creativecommons.org/licenses/by-nc-sa/2.0/
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

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

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


**Actions**

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

  **Operator** (i.e., an abstract schema of actions, containing variable)
  
  \[ \text{take}(k, l, c, d, p) \]
  
  ;; 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: \( \text{holding}(k, c), \neg \text{empty}(k), \neg \text{in}(c, p), \neg \text{top}(c, p), \neg \text{on}(c, d), \text{top}(d, p) \)

- **Action** (i.e., an instantiated operator which can be applied to some specific states)
  
  \[ \text{take}(\text{crane1}, \text{loc1}, c3, c1, p1) \]
  
  ;; crane \text{crane1} at location \text{loc1} takes \text{c3} off of \text{c1} in pile \text{p1}
  
  precond: \( \text{belong(\text{crane1}, \text{loc1}), attached(p1, \text{loc1}), empty(\text{crane1}), top(c3, \text{p1}), on(\text{c3}, \text{c1})} \)
  
  effects: \( \text{holding(\text{crane1}, \text{c3}), \neg empty(\text{crane1}), \neg in(c3, p1)}, \neg \text{top(c3, p1)}, \neg \text{on(\text{c3}, \text{c1})}, \text{top(c1, p1)} \)
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: $\text{holding}(k, c)$, $\neg\text{empty}(k)$, $\neg\text{in}(c, p)$, $\neg\text{top}(c, p)$, $\neg\text{on}(c, d)$, $\text{top}(d, p)$

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 \textit{applicable} to a state $s$ if $s$ satisfies 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:

$$s_1 = \{\text{attached}(p1, \text{loc}1), \text{in}(c1, p1), \text{in}(c3, p1), \text{top}(c3, p1), \text{on}(c3, c1), \text{on}(c1, \text{pallet}), \text{attached}(p2, \text{loc}1), \text{in}(c2, p2), \text{top}(c2, p2), \text{on}(c2, \text{pallet}), \text{belong}(\text{crane}1, \text{loc}1), \text{empty}(\text{crane}1), \text{adjacent}(\text{loc}1, \text{loc}2), \text{adjacent}(\text{loc}2, \text{loc}1), \text{at}(r1, \text{loc}2), \text{occupied}(\text{loc}2), \text{unloaded}(r1)\}$$

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

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

precond:  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)
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

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

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

\text{take}(\text{crane1}, loc1, c3, c1, p1) \text{ is applicable to } s_1 : \\
\{ \text{belong}(\text{crane1}, loc1), \text{attached}(p1, loc1), \text{empty}(\text{crane1}), \text{top}(c3, p1), \text{on}(c3, c1) \subseteq 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}(loc1, loc2), \text{adjacent}(loc2, loc1), \text{at}(r1, loc2), \text{occupied}(loc2), \text{unloaded}(r1) \} \]

Application:
\[ s_1 = \{ \text{attached}(p1, loc1), \text{in}(c1, p1), \text{in}(c3, p1), \text{top}(c3, p1), \text{on}(c3, c1), \text{on}(c1, \text{pallet}), \text{attached}(p2, loc1), \text{in}(c2, p2), \text{top}(c2, p2), \text{on}(c2, \text{pallet}), \text{belong}(\text{crane1}, loc1), \text{empty}(\text{crane1}), \text{adjacent}(loc1, loc2), \text{adjacent}(loc2, loc1), \text{at}(r1, loc2), \text{occupied}(loc2), \text{unloaded}(r1) \} \]
\cup \text{holding}(\text{crane1}, c3), \text{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 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
    \[
    \begin{align*}
    \Rightarrow & \quad \gamma(s_0, a_1) = s_1 \\
    \Rightarrow & \quad \gamma(s_1, a_2) = s_2 \\
    \Rightarrow & \quad \ldots \\
    \Rightarrow & \quad \gamma(s_{n-1}, a_n) = s_n \\
    \Rightarrow & \quad s_n \text{ satisfies } g
    \end{align*}
    \]
Example

Let $P_1 = (O, s_1, g_1)$, where
- $O$ is the set of operators given earlier

$g_1 = \{\text{loaded}(r1, c3), \text{at}(r1, \text{loc}2)\}$

$s_1 = \{\text{attached}(p1, \text{loc}1), \text{in}(c1, p1), \text{in}(c3, p1), \text{top}(c3, p1), \text{on}(c3, c1), \text{on}(c1, \text{pallet}), \text{attached}(p2, \text{loc}1), \text{in}(c2, p2), \text{top}(c2, p2), \text{on}(c2, \text{pallet}), \text{belong}(\text{crane}1, \text{loc}1), \text{empty}(\text{crane}1), \text{adjacent}(\text{loc}1, \text{loc}2), \text{adjacent}(\text{loc}2, \text{loc}1), \text{at}(r1, \text{loc}2), \text{occupied}(\text{loc}2), \text{unloaded}(r1)\}$
Here are three solutions for $P_1$:

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

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

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

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

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

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

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

The 2nd and 3rd are *irredundant* and *shortest*.
Set-Theoretic Representation

- Like classical representation, but restricted to propositional logic

**States:**
- Instead of a collection of ground atoms . . .
  \{\text{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):
  \{\text{on-c1-pallet}, \text{on-c1-r1}, \text{on-c1-c2}, \ldots, \text{at-r1-l1}, \text{at-r1-l2}, \ldots\}
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: \(\text{holding}(k, c), \neg\text{empty}(k), \neg\text{in}(c, p), \neg\text{top}(c, p), \neg\text{on}(c, d), \neg\text{top}(d, p)\)

---

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

\[
\text{take}(\text{crane1}, \text{loc1}, c3, c1, p1) \\
;; \text{crane crane1 at location loc1 takes c3 off of c1 in pile p1}
\]

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

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

---

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

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

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

effect: \(\text{holding-crane1-c3, top-c1-p1}\)
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
```

{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.,

![Diagram of initial and goal states]

- 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:**
  - $\text{ontable}(x)$ - block $x$ is on the table
  - $\text{on}(x,y)$ - block $x$ is on block $y$
  - $\text{clear}(x)$ - block $x$ has nothing on it
  - $\text{holding}(x)$ - the robot hand is holding block $x$
  - $\text{handempty}$ - 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
For five blocks, there are 36 propositions

Here are 5 of them:

- \textit{ontable-a} - block \( a \) is on the table
- \textit{on-c-a} - block \( c \) is on block \( a \)
- \textit{clear-c} - block \( c \) has nothing on it
- \textit{holding-d} - the robot hand is holding block \( d \)
- \textit{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: 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*
  - $0, 1, \textit{table}, \textit{nil}$ of type *other*

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

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).
A function $f : D^n \Rightarrow W$

is a unique mapping from ($n$) values of domains to a value of a range

Examples:

- $\text{succ} : \text{Nat} \rightarrow \text{Nat}$
- $\text{succ}(0) = 1, \text{succ}(1) = 1, \ldots$

- $\text{plus} : \text{Nat} \times \text{Nat} \rightarrow \text{Nat}$
- $\text{plus}(0, 0) = 0, \text{plus}(0, 1) = 1, \ldots$
Transformations

Functions and Relations

- A relation $R$ is a relationship between sets
- Examples:
  - $R_{\text{succ}} \subseteq \mathbb{Nat} \times \mathbb{Nat} = \{(0, 1), (0, 2), (0, 3), \ldots\}$
  - $R_{\text{plus}} \subseteq \mathbb{Nat} \times \mathbb{Nat} \times \mathbb{Nat} = \{(0, 0, 0), (0, 1, 1), \ldots\}$
- Each function can be transformed into a relation but not each relation is a function!

Append in the functional language Lisp:

```
(defun append (L1, L2)
  (cond ((null L1) L2)
        (T (cons (car L1) (append (cdr L1) L2))))
)
```

Append in the logical language Prolog:

```
append([], L, L).
append([X|Xs], L2, [X|L]) :- append(Xs, L2, L).
```
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
Summary

- In classical representation a state is a set of ground literals
- The problem domain is defined by a set of operators representing the possible actions
- An action is a fully instantiated operator
- An action can be applied to a state if all preconditions hold (e.g. are included in the state)
- The effect of an action is calculated by deleting literals which no longer hold after the execution of the action and by inserting literals which hold after the execution
Summary

- This simple way to model state-transitions can be realized because of the closed-world-assumption.
- Set-theoretic representations can be generated by writing all ground instances (exponential blow-up).
- This representation is useful because there exist efficient algorithms for manipulating ground atoms (SAT-solvers, model checking).
- The state-variable representation works with assignments of values to state-variables. This is useful when dealing with numbers (time, functions, ...).