Lecture slides for

Automated Planning: Theory and Practice

Chapter 2
Representations for Classical Planning

Dana S. Nau

University of Maryland

4:56 PM   January 30, 2012

Quick Review of Classical Planning

- Classical planning requires all eight of the restrictive assumptions:

  A0: Finite

  A1: Fully observable

  A2: Deterministic

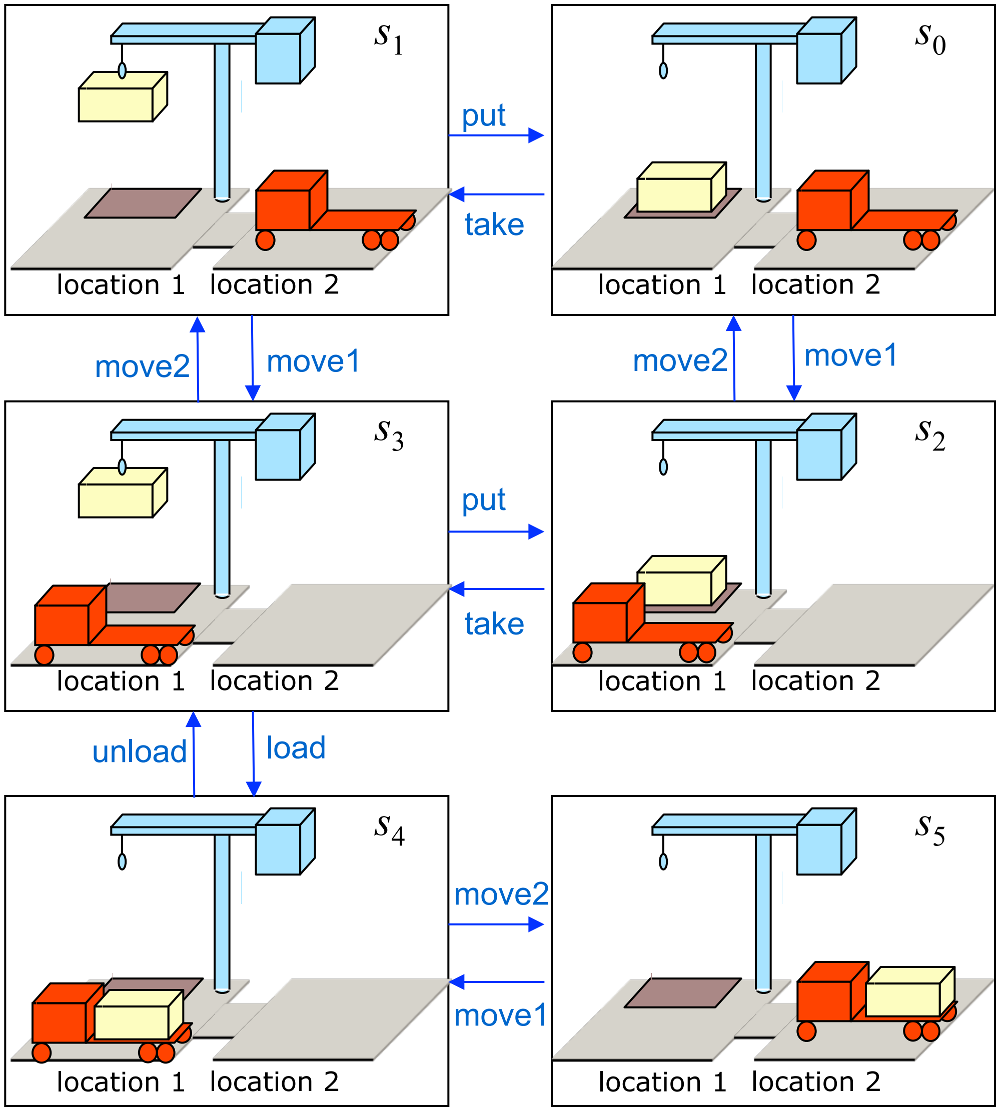
  A3: Static

  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

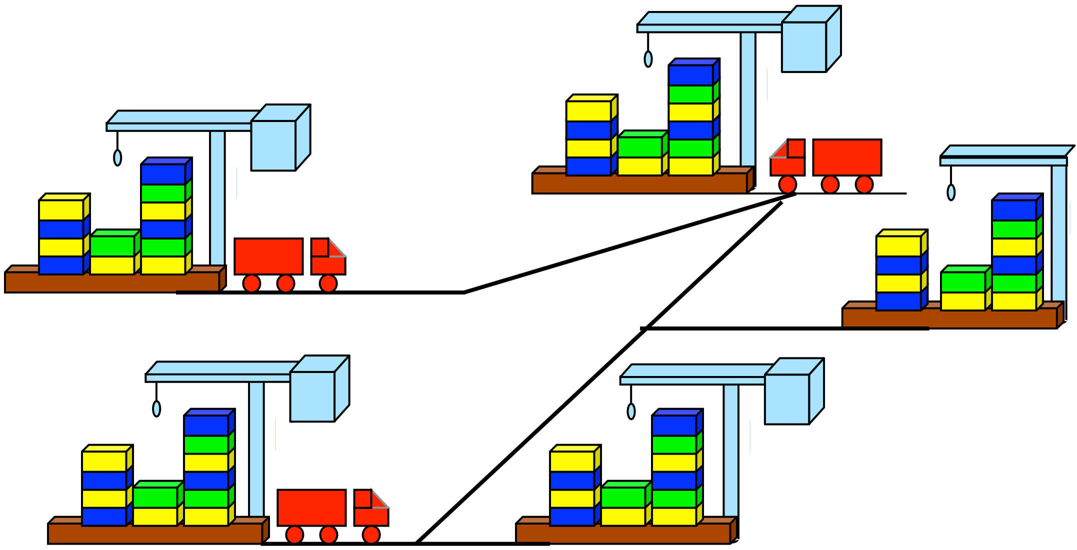
Outline

- 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: 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)

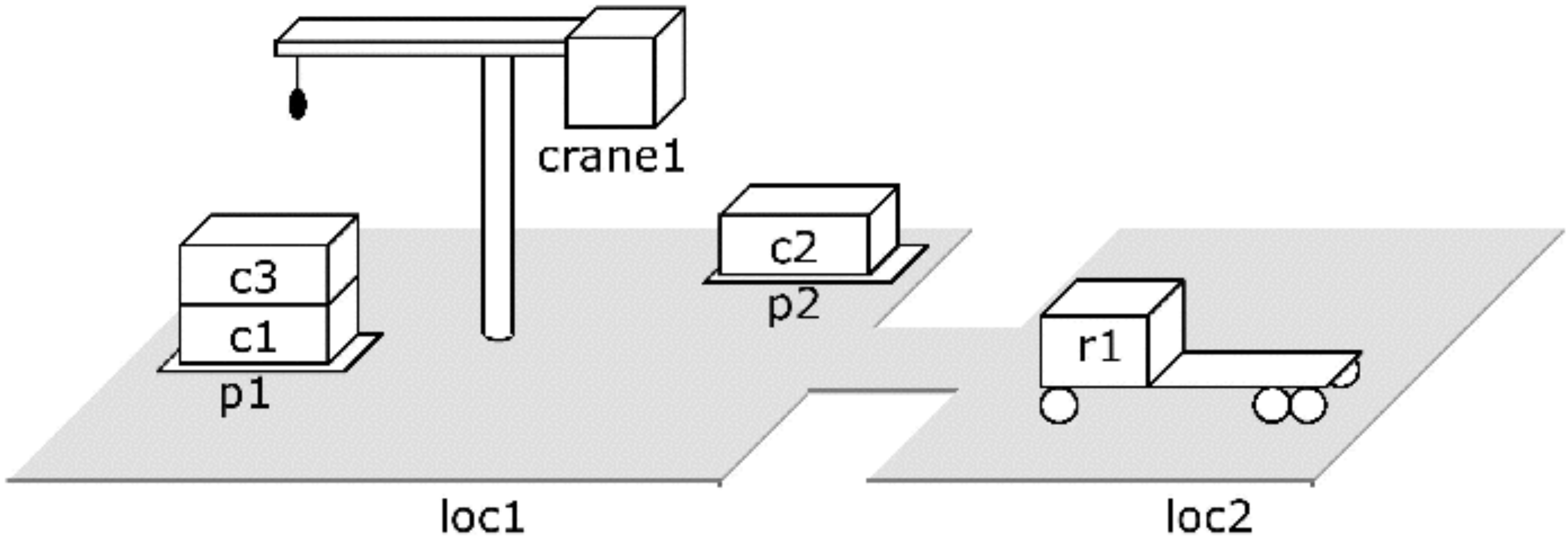
    in(c,p) on(c,c')

    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 θ 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 Σ's states
  - Only finitely many ground atoms, so only finitely many possible states



s_1 = {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,palet),
belong(crane1,loc1), empty(crane1), adjacent(loc1,loc2),
adjacent(loc2,loc1), at(r1,loc2), occupied(loc2, unloaded(r1)}

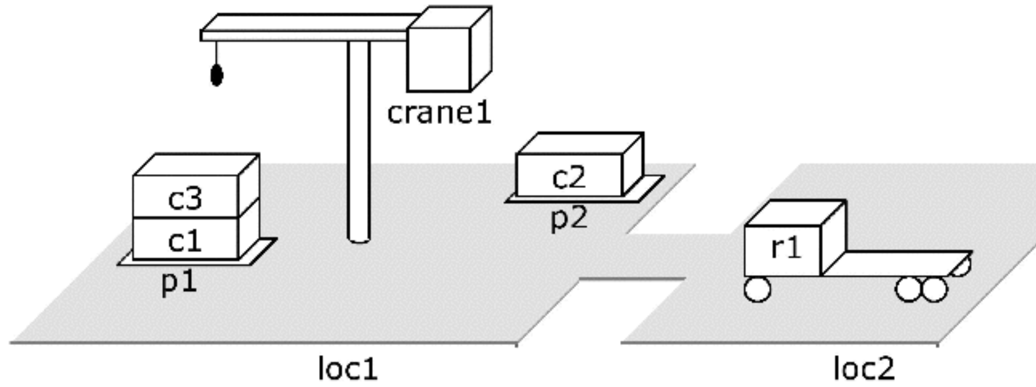
Operators

- *Operator*: a triple o=(name(o), precond(o), 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 like this:

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

Actions



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
  - Let $\theta = \{k \leftarrow$ crane1, $l \leftarrow$ loc1, $c \leftarrow$ c3, $d \leftarrow$ c1, $p \leftarrow$ p1$\}$
  - Then (take(k,l,c,d,p))θ is the following action:

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

    precond: belong(crane,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)

  - i.e., crane crane1 at location loc1 takes c3 off of c1 in pile p1

Notation

- Let S be a set of literals. Then
  - S^+ = {atoms that appear positively in S}
  - S^- = {atoms that appear negatively in S}
- Let a be an operator or action. Then
  - precond$^+(a)$ = {atoms that appear positively in a's preconditions}
  - precond$^-(a)$ = {atoms that appear negatively in a's preconditions}
  - effects$^+(a)$ = {atoms that appear positively in a's effects}
  - effects$^-(a)$ = {atoms that appear negatively in a's effects}

- Example: take(crane1,loc1,c3,c1,p1)

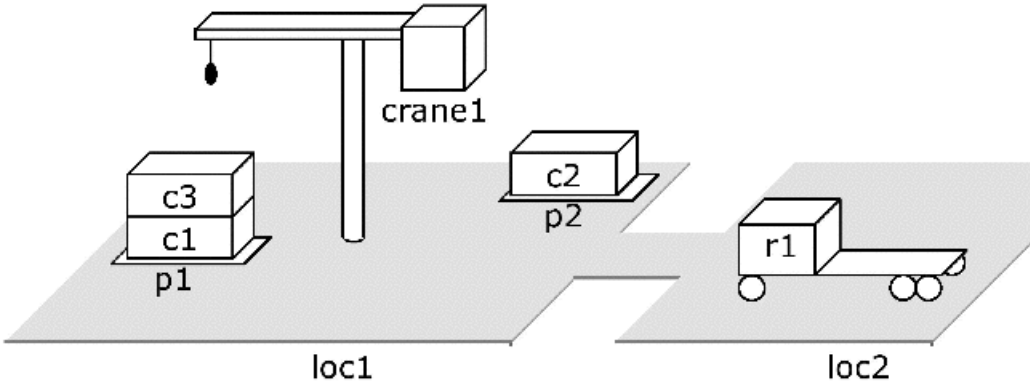
  precond: belong(crane,loc1), attached(p1,loc1),
  empty(crane1), top(c3,p1), on(c3,c1)

  effects: holding(crane1,c3), ¬empty(crane1), ¬in(c3,p1),
  ¬top(c3,p1), ¬on(c3,c1), top(c1,p1)

  - effects$^+$(take(crane1,loc1,c3,c1,p1)) = {holding(crane1,c3), top(c1,p1)}
  - effects$^-$(take(crane1,loc1,c3,c1,p1))
    = {empty(crane1), in(c3,p1), top(c3,p1), on(c3,c1)}

Applicability

- Let s be a state and a be an action
- a is *applicable* to (or *executable* in) if s satisfies precond(a)
  - precond$^+(a) \subseteq s$
  - precond$^-(a) \cap s = \varnothing$



- An action:

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

  precond: belong(crane,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)

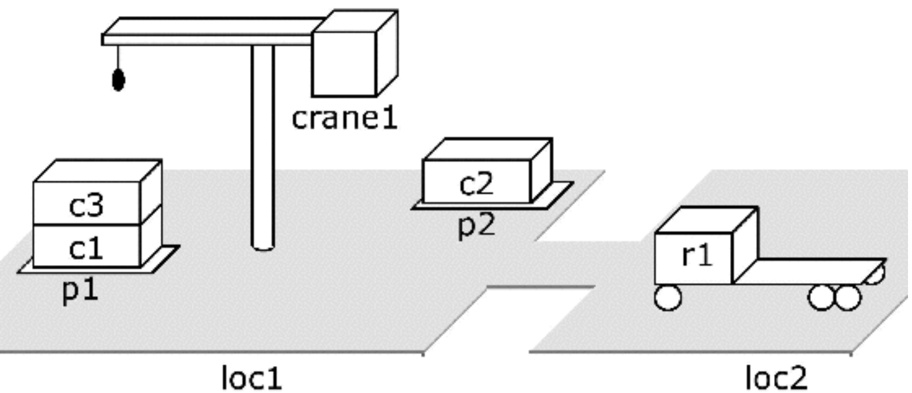
- A state it's applicable to

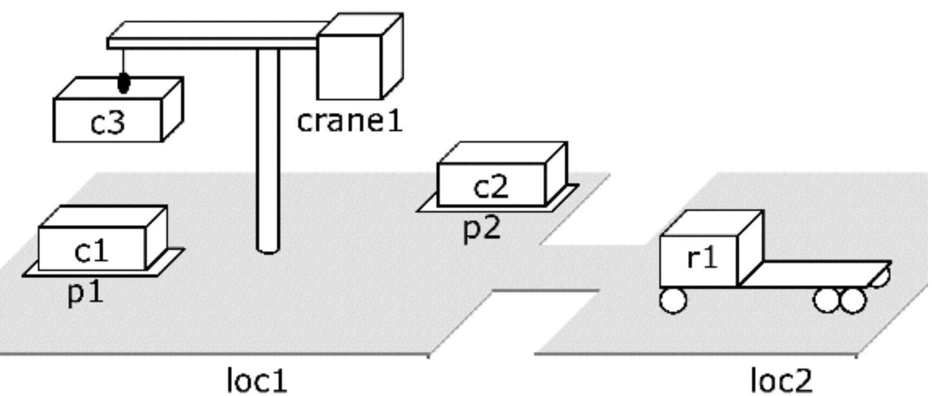
  s_1 = {**attached(p1,loc1)**, in(c1,p1), in(c3,p1), **top(c3,p1)**, **on(c3,c1)**, on(c1,pallet), attached(p2,loc1), in(c2,p2), top(c2,p2), on(c2,palet), **belong(crane1,loc1)**, **empty(crane1)**, adjacent(loc1,loc2), adjacent(loc2,loc1), at(r1,loc2), occupied(loc2, unloaded(r1)}

Executing an Applicable Action

- Remove *a*'s negative effects, and add *a*'s positive effects

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





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

precond: belong(crane,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)

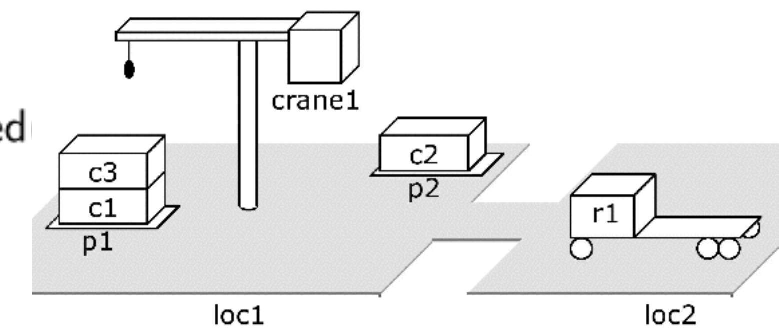
s_2 = {attached(p1,loc1), in(c1,p1), ~~in(c3,p1)~~, ~~top(c3,p1)~~, ~~on(c3,c1)~~, on(c1,pallet), attached(p2,loc1), in(c2,p2), top(c2,p2), on(c2,palet), belong(crane1,loc1), ~~empty(crane1)~~, adjacent(loc1,loc2), adjacent(loc2,loc1), at(r1,loc2), occupied(loc2, unloaded(r1), **holding(crane1,c3), top(c1,p1)**}

- **Planning domain**: language plus operators
  - Corresponds to a set of state-transition systems
  - Example: operators for the DWR domain

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)

Planning Problems

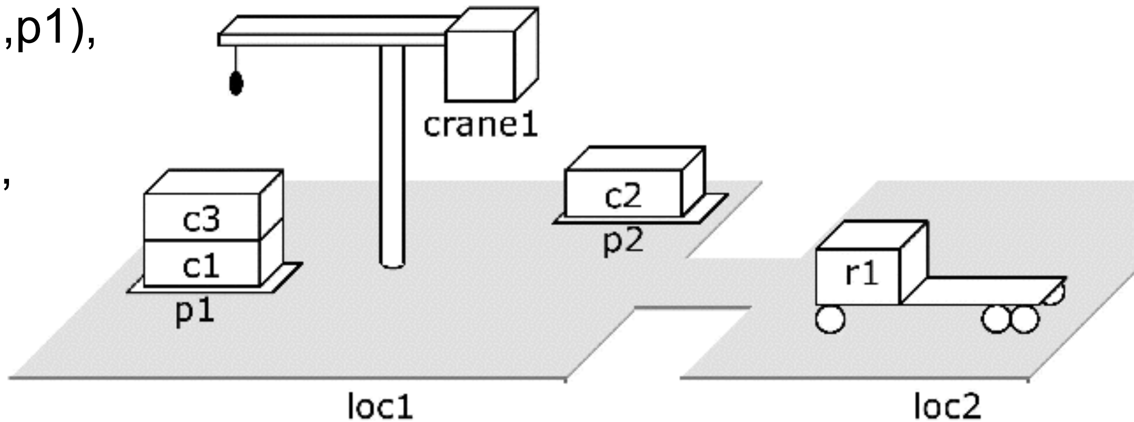
- Given a planning domain (language L, operators O)
  - *Statement* of a planning problem: a triple $P=(O,s_0,g)$
    - » O is the collection of operators
    - » s_0 is a state (the initial state)
    - » g is a set of literals (the goal formula)
  - Planning problem: $\mathcal{P} = (\Sigma,s_0,S_g)$
    - » s_0 = initial state
    - » S_g = set of goal states
    - » $\Sigma = (S,A,\gamma)$ is a state-transition system that satisfies all of the restrictive assumptions in Chapter 1
    - » S = {all sets of ground atoms in L}
    - » A = {all ground instances of operators in O}
    - » γ = the state-transition function determined by the operators
- I'll often say "planning problem" to mean the statement of the problem

Plans and Solutions

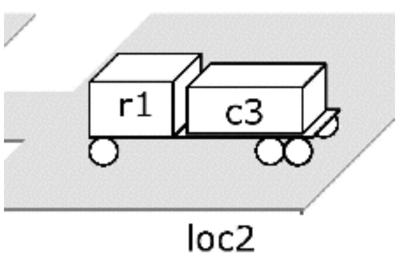
- Let $P=(O,s_0,g)$ be a planning problem
- *Plan*: any sequence of actions $\pi = \langle a_1, a_2, \ldots, a_n \rangle$ such that each a_i is an instance of an operator in O
- π is a *solution* for $P=(O,s_0,g)$ if it is executable and achieves g
  - i.e., if there are states $s_0, s_1, \ldots, s_n$ such that
    - $\gamma(s_0,a_1) = s_1$
    - $\gamma(s_1,a_2) = s_2$
    - …
    - $\gamma(s_{n-1},a_n) = s_n$
    - s_n satisfies g

Example

- Let $P_1 = (O, s_1, g_1)$, where
  - O = {the four DWR operators given earlier}
  - s_1 = {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,palet),
    belong(crane1,loc1), empty(crane1),
    adjacent(loc1,loc2), adjacent(loc2,loc1),
    at(r1,loc2), occupied(loc2), unloaded(r1)}



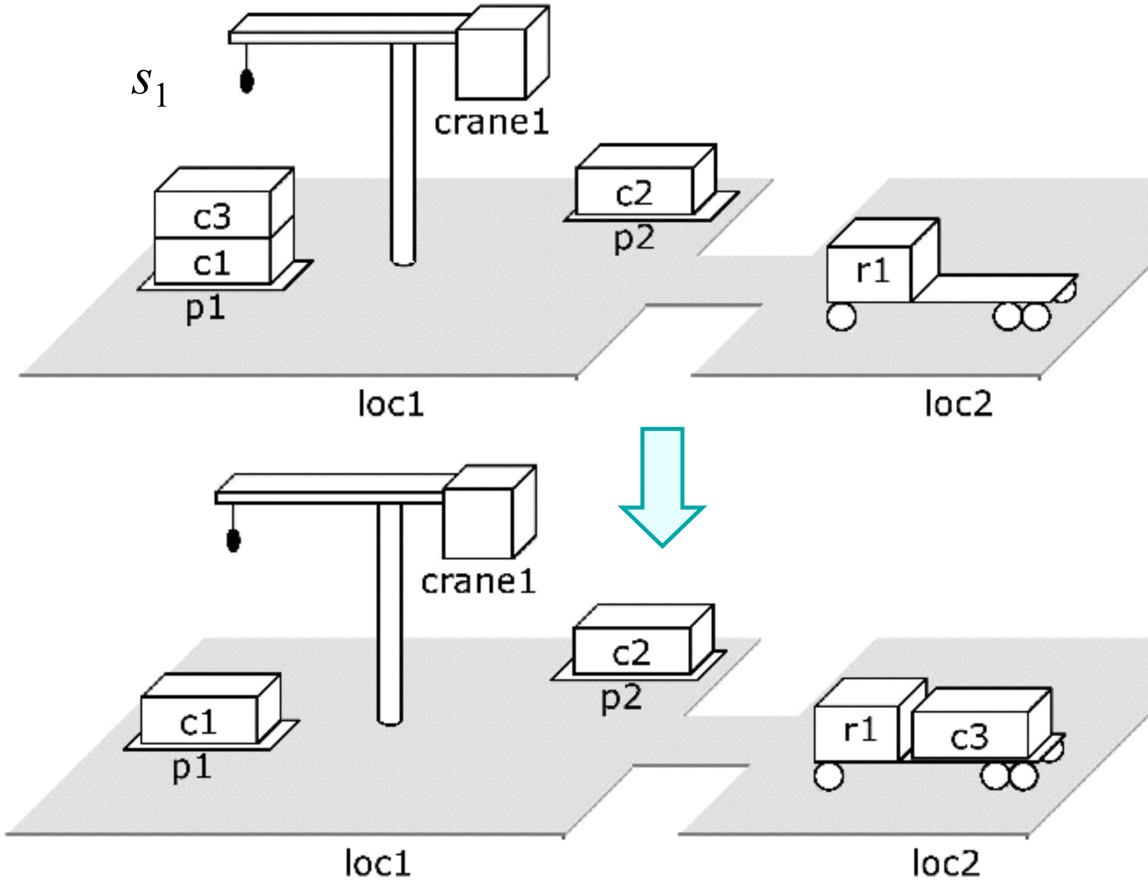
  - g_1 = {loaded(r1,c3), at(r1,loc2)}



- Two *redundant* solutions (can remove actions and still have a solution):

  ⟨move(r1,loc2,loc1),
  take(crane1,loc1,c3,c1,p1),
  ~~move(r1,loc1,loc2),~~
  ~~move(r1,loc2,loc1),~~
  load(crane1,loc1,c3,r1),
  move(r1,loc1,loc2)⟩

  ⟨take(crane1,loc1,c3,c1,p1),
  ~~put(crane1,loc1,c3,c2,p2),~~
  move(r1,loc2,loc1),
  ~~take(crane1,loc1,c3,c2,p2),~~
  load(crane1,loc1,c3,r1),
  move(r1,loc1,loc2)⟩

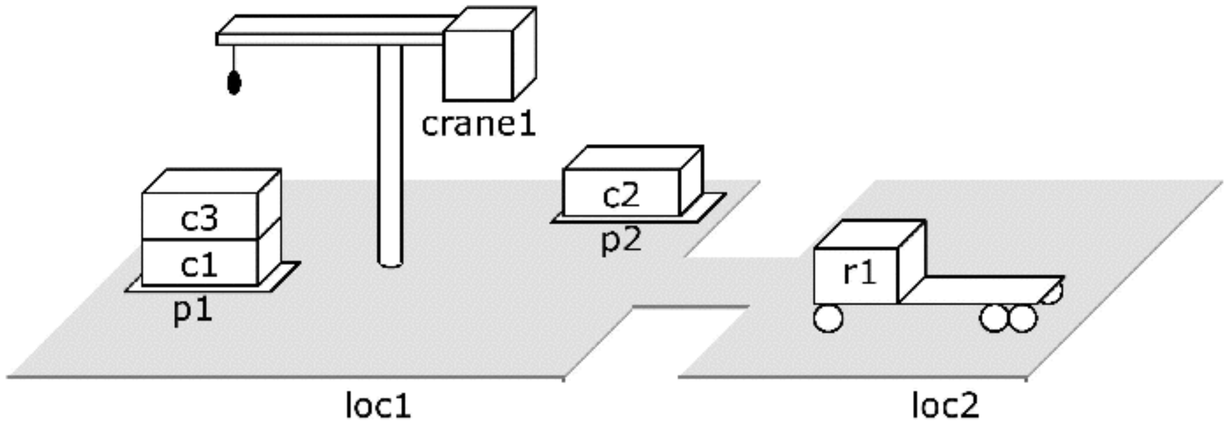


- A solution that is both *irredundant* and *shortest*:
  ⟨move(r1,loc2,loc1), take(crane1,loc1,c3,c1,p1),
  load(crane1,loc1,c3,r1), move(r1,loc1,loc2)⟩

- Are there any other shortest solutions? Are irredundant solutions always shortest?

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):

{on(c1,pallet), on(c1,r1), on(c1,c2), …, at(r1,l1), at(r1,l2), …}

⇩

{on-c1-pallet, on-c1-r1, on-c1-c2, …, at-r1-l1, at-r1-l2, …}

Set-Theoretic Representation, continued

No operators, just actions:

- Instead of ground atoms, use propositions
- Instead of negative effects, use a delete list
- If there are any negative preconditions, create new atoms to represent them
- E.g., instead of using ¬foo as a precondition, use not-foo
  - Delete foo iff you add not-foo
  - Delete not-foo iff you add foo

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

precond: belong(crane,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)

take-crane1-loc1-c3-c1-p1

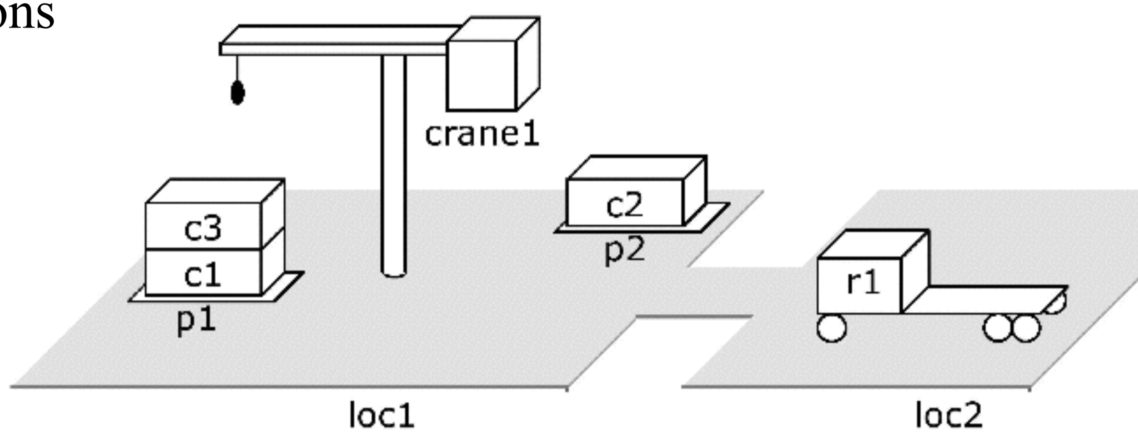
precond: belong-crane1-loc1,
attached-p1-loc1, empty-crane1,
top-c3-p1, on-c3-c1

delete: empty-crane1,
in-c3-p1, top-c3-p1, on-c3-p1

add: holding-crane1-c3, top-c1-p1

Exponential Blowup

- Suppose the language contains *c* constant symbols
- Let *o* be a classical operator with *k* parameters
- Then there are c^k ground instances of *o*
  - Hence c^k set-theoretic actions
- Example:
  take(crane1,loc1,c3,c1,p1)
  - $k = 5$
  - 1 crane, 2 locations, 3 containers, 2 piles
    - » 8 constant symbols
  - $8^5 = 32768$ ground instances

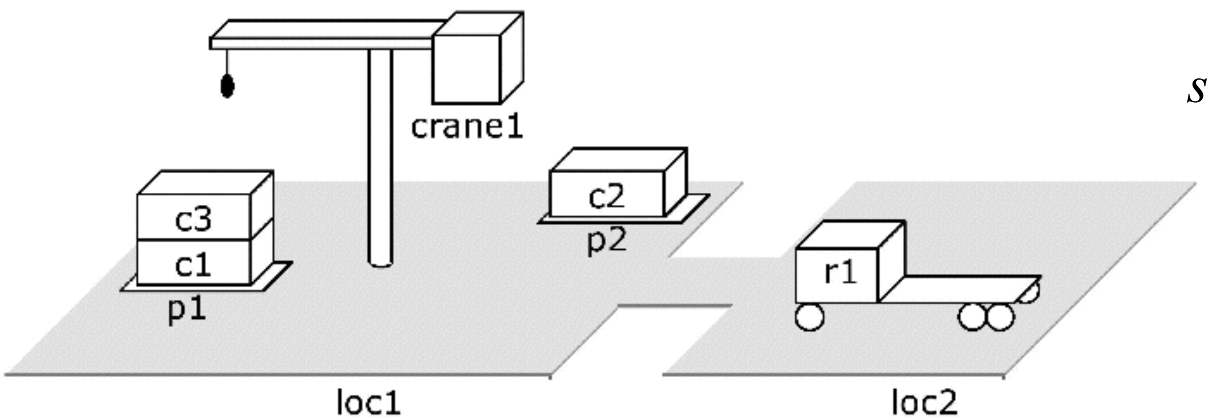


- Can reduce this by assigning data types to the parameters
    - » e.g., first arg must be a crane, second must be a location, etc.
    - » Number of ground instances is now 1 * 2 * 3 * 3 * 2 = 36
  - Worst case is still exponential

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

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) $\leftarrow m$



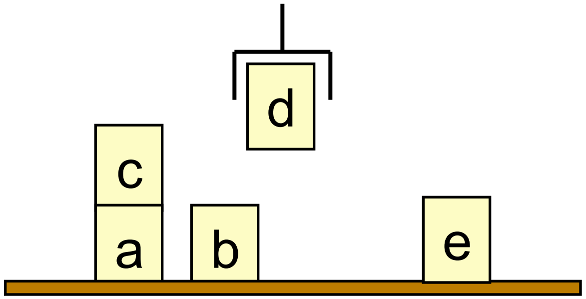
$s_1 = \{$top(p1)=c3,
cpos(c3)=c1,
cpos(c1)=pallet,
holding(crane1)=nil,
rloc(r1)=loc2,
loaded(r1)=nil, ...$\}$

Example: The Blocks World

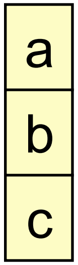
- Infinitely wide table, finite number of children's blocks
- Ignore where a block is located on the table
- A block can sit on the table or on another block
- There's a robot gripper that can hold at most one block

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

initial state



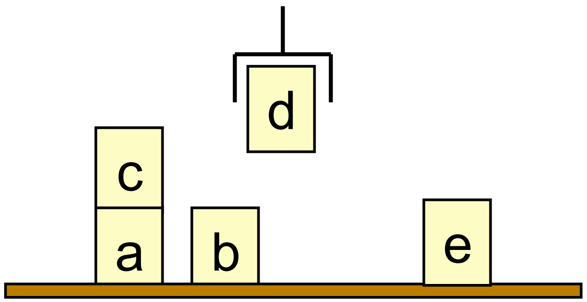
goal



- Like a special case of DWR with one location, one crane, some containers, and many more piles than you need

Classical Representation: Symbols

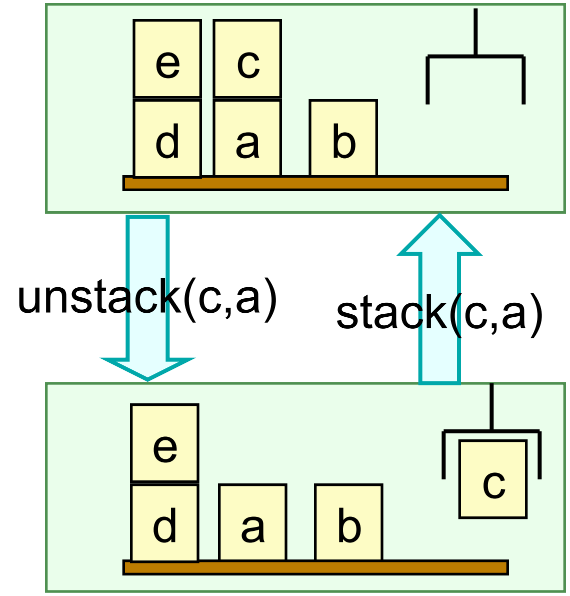
- Constant symbols:
  - The blocks: a, b, c, d, e
- Predicates:
  - ontable(x) - block x is on the table
  - on(x,y) - block x is on block y
  - clear(x) - block x has nothing on it
  - holding(x) - the robot hand is holding block x
  - handempty - the robot hand isn't holding anything



Classical Operators

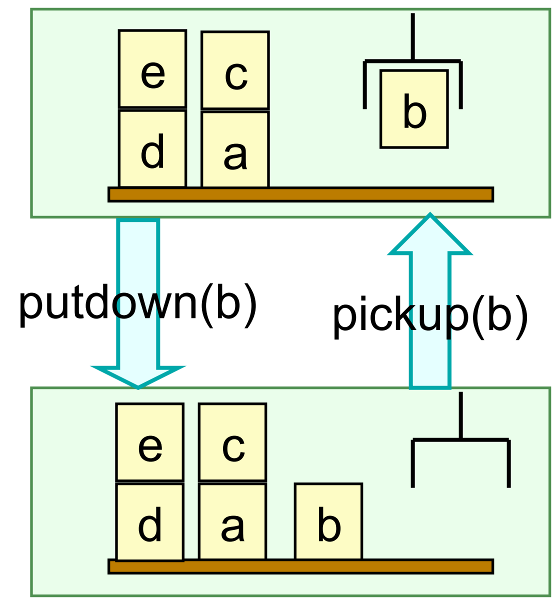
unstack(x,y)
Precond: on(x,y), clear(x), handempty
Effects: ¬on(x,y), ¬clear(x), ¬handempty, holding(x), clear(y)

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



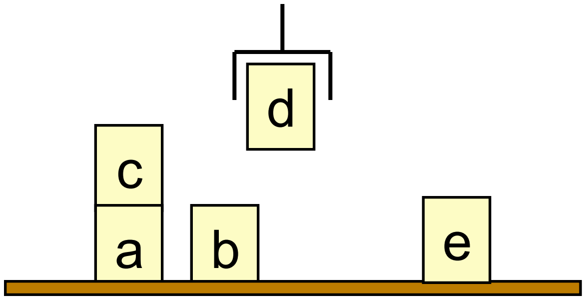
pickup(x)
Precond: ontable(x), clear(x), handempty
Effects: ¬ontable(x), ¬clear(x), ¬handempty, holding(x)

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

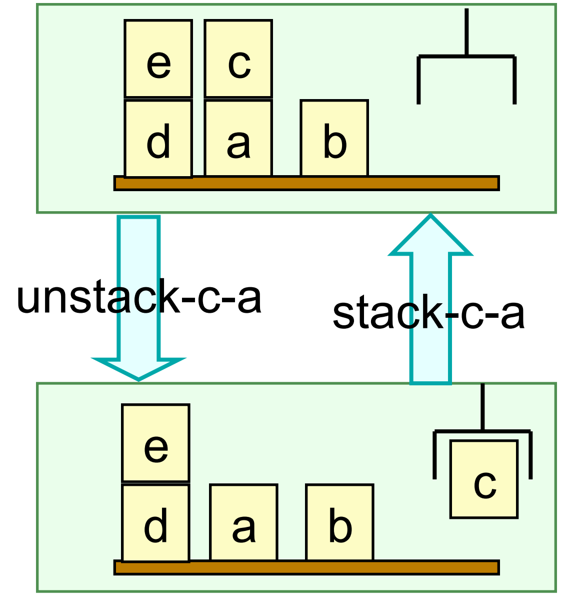
- 60 actions
- 50 if we exclude nonsensical ones, e.g., unstack-e-e
- Here are four of them:

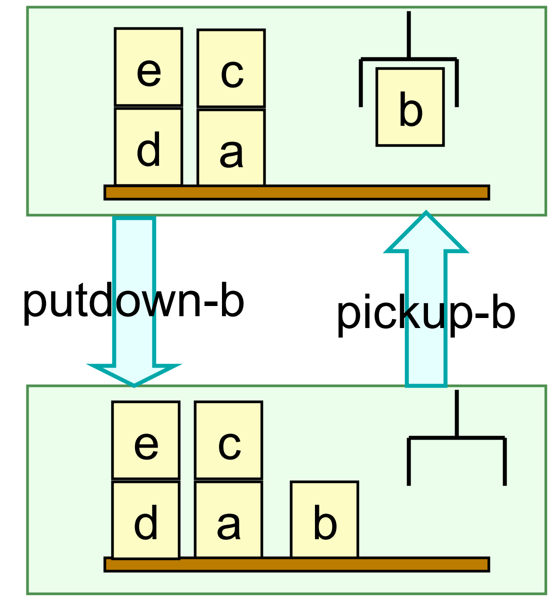
unstack-c-a
- Pre: on-c-a, clear-c, handempty
- Del: on-c-a, clear-c, handempty
- Add: holding-c, clear-a

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

pickup-b
- Pre: ontable-b, clear-b, handempty
- Del: ontable-b, clear-b, handempty
- Add: holding-b

putdown-b
- Pre: holding-b
- Del: holding-b
- Add: ontable-b, clear-b, handempty





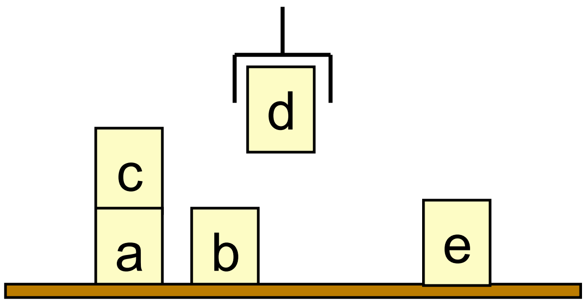
State-Variable Representation: Symbols

- Constant symbols:

  a, b, c, d, e   of type block

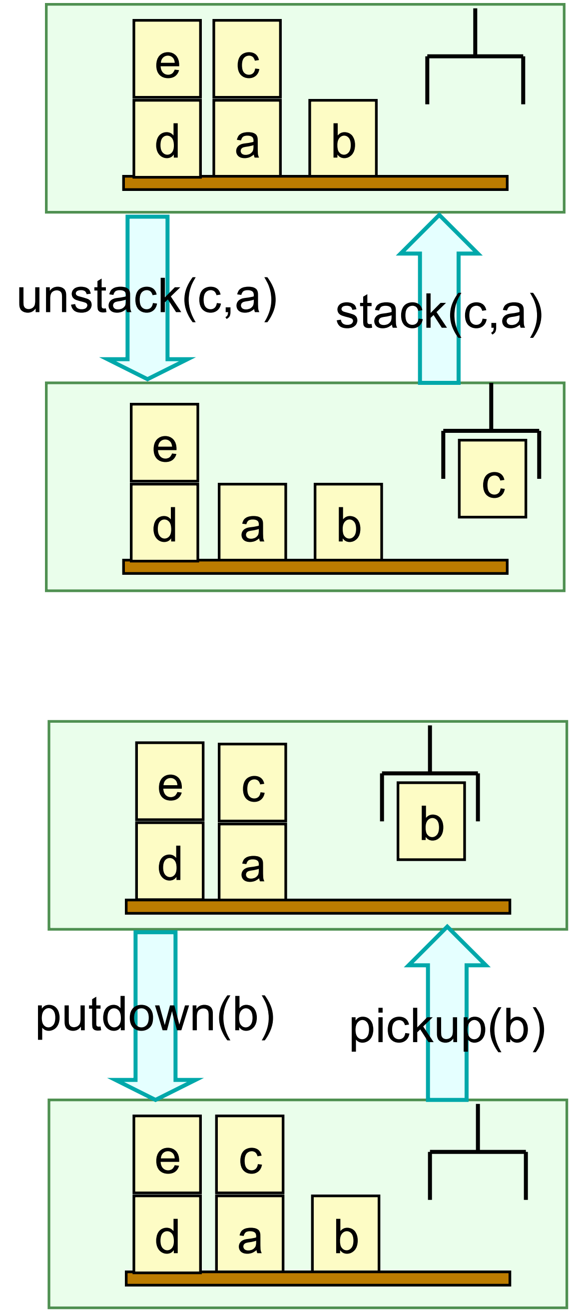
  0, 1, table, nil   of type other

- State variables:



pos(x) = y	if block x is on block y
pos(x) = table	if block x is on the table
pos(x) = nil	if block x is being held
clear(x) = 1	if block x has nothing on it
clear(x) = 0	if block x is being held or has another block on it
holding = x	if the robot hand is holding block x
holding = nil	if the robot hand is holding nothing

State-Variable Operators



With data types:

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

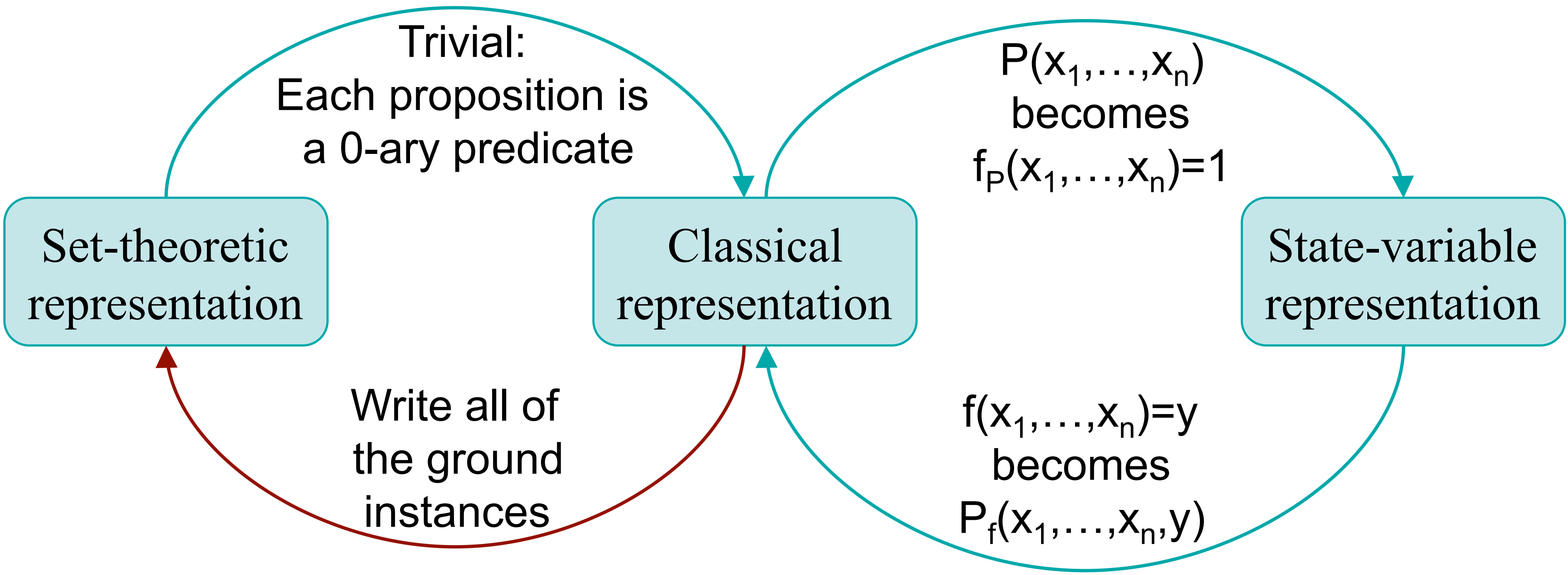
stack(x : block, y : block)
  Precond: holding=x, clear(x)=0, clear(y)=1
  Effects: holding←nil, clear(y)←0, pos(x)←y, clear(x)←1

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 and most computer scientists
  - Useful in non-classical planning problems as a way to handle numbers, functions, time