Planning and Scheduling: Representations for Classical Planning

Hochschule Bonn-Rhein-Sieg

b-it Bonn-Aachen International Center for Information Technology

Prof. Dr.-Ing. Gerhard K. Kraetzschmar

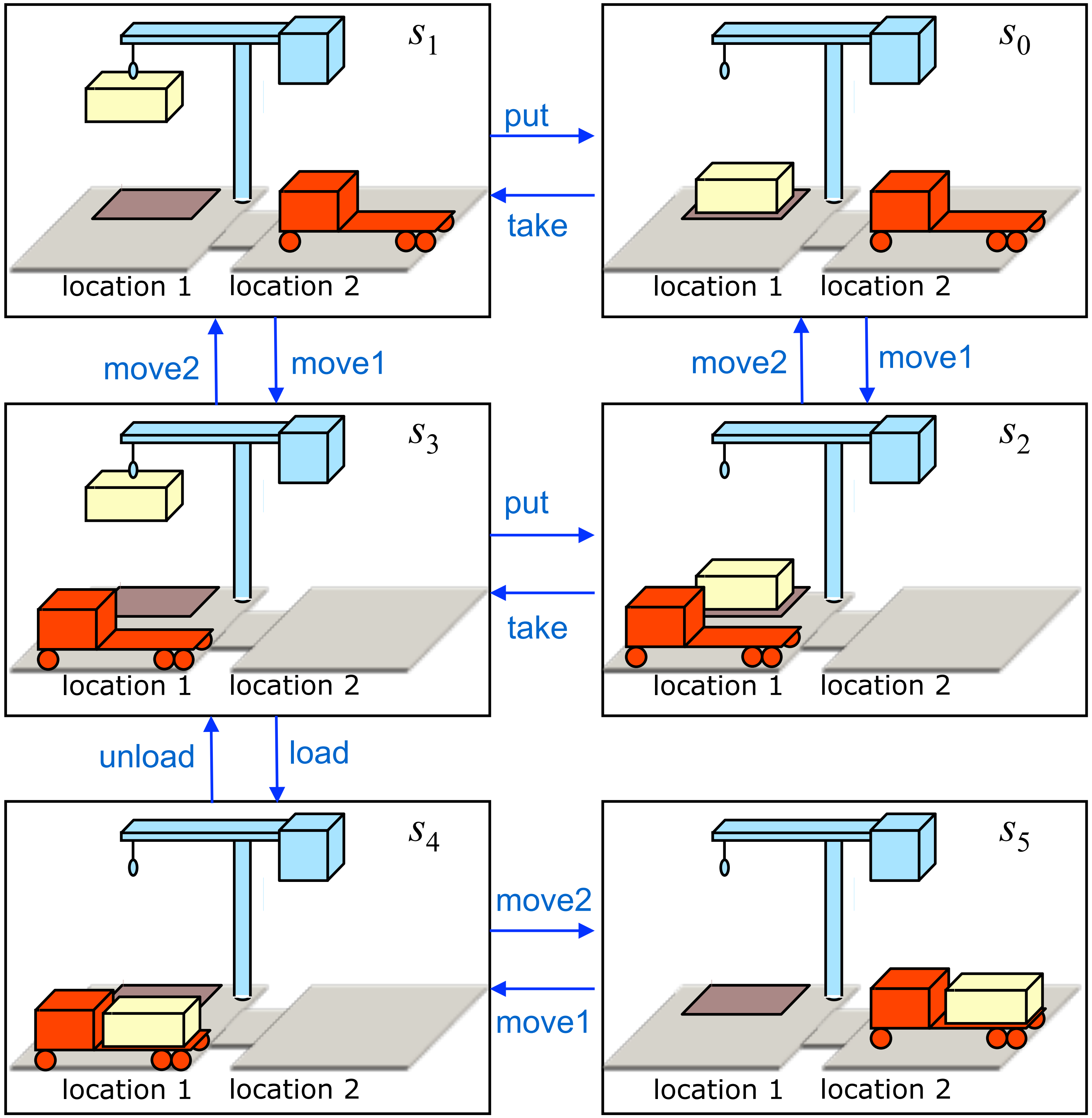
Tuesday, October 08, 13

Acknowledgements

- These slides refer to Chapter 2 of the textbook:
  Malik Ghallab, Dana Nau, and Paolo Traverso:
  Automated Planning: Theory and Practice
  Morgan Kaufmann, 2004
- These slides are an adaptation of slides by Dana Nau
- The contributions of these authors are gratefully acknowledged

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

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 $on(c_1, c_3), on(c_1, x)$
  - Ground expression: contains no variable symbols $on(c_1, c_3)$
  - Nonground expression: at least one variable symbol $on(c_1, x)$
  - 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 an expression e: result of applying a substitution θ to e
    - Replace variables of e simultaneously, not sequentially
- 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

Example of a State

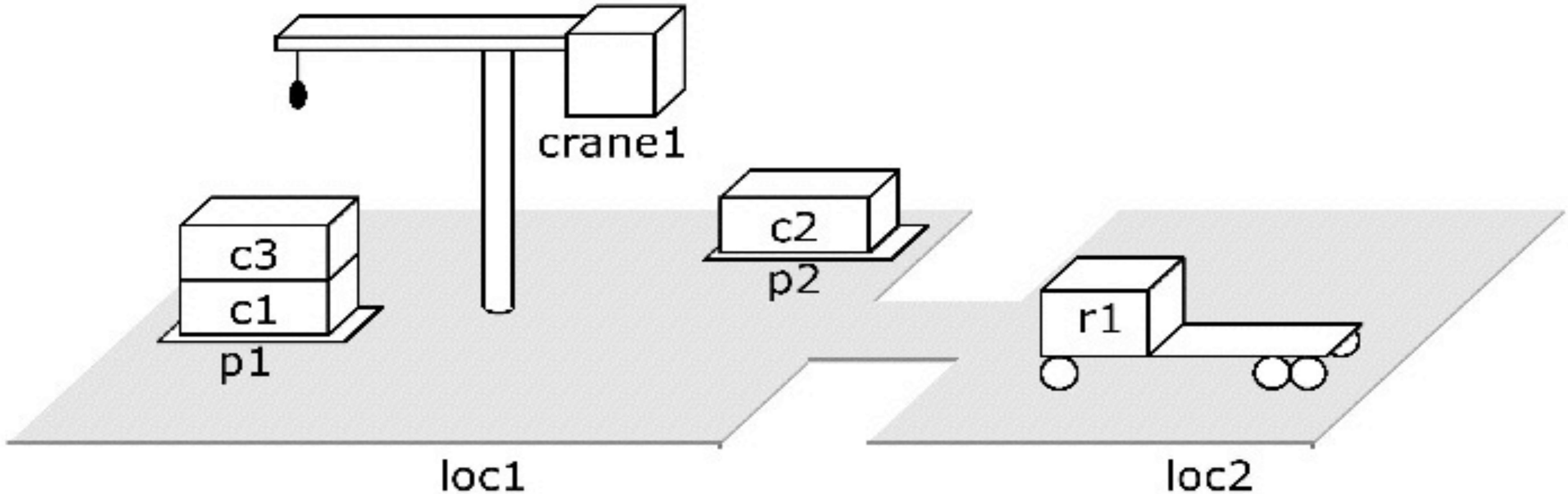


Figure 2.2: The DWR state s_1={attached(p1,loc1), in(c1,p1), in(c3,p1), top(c3,p1), on(c3,c1), on(c1,pallet), attached(p2,loc1), in(c2,p2), top(c2,p2), on(c2,pallet), belong(crane1,loc1), empty(crane1), adjacent(loc1,loc2), adjacent(loc2,loc1), at(r1,loc2), occupied(loc2), unloaded(r1)}.

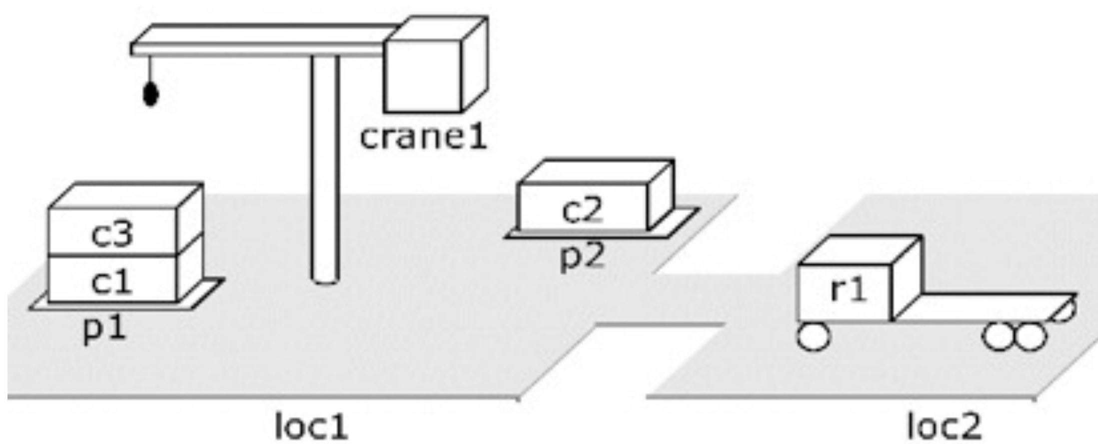
Operators

- Operator: a triple o=(name(o), precond(o), effects(o))
  - 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

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



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)

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

Notation

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

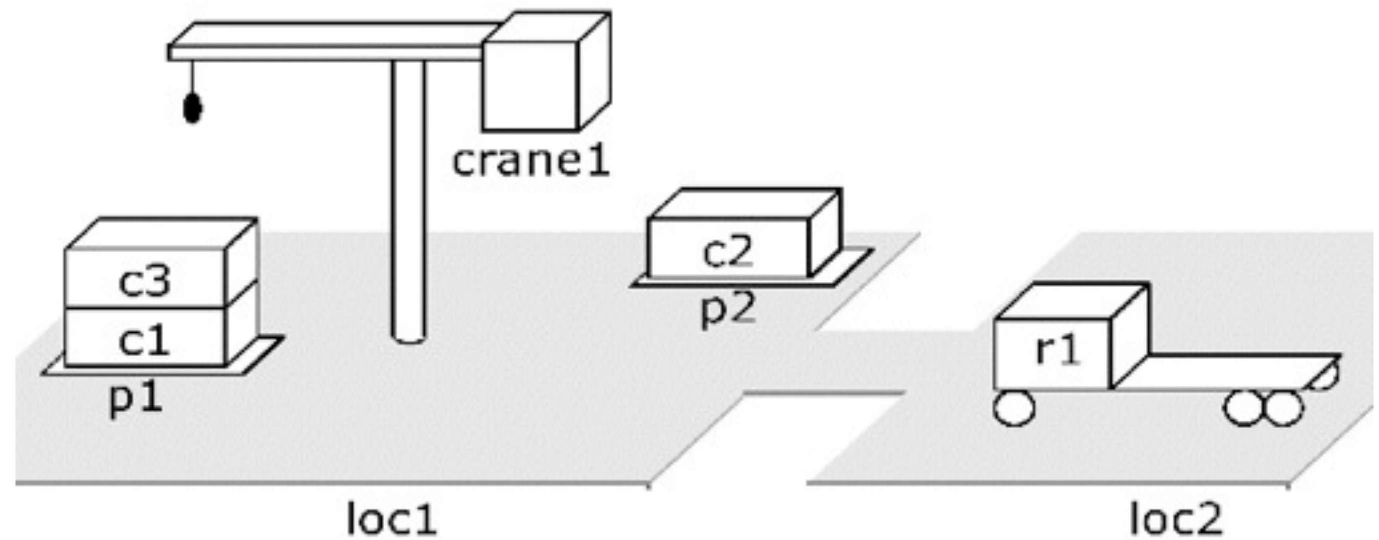
take(k, l, c, d, p)
;; crane k at location l takes c off of d in pile p
precond: belong(k, l), attached(p, l), empty(k), top(c, p), on(c, d)
effects: holding(k, c), $\neg$ empty(k), $\neg$ in(c, p), $\neg$ top(c, p), $\neg$ on(c, d), top(d, p)

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

Applicability

- An action a is applicable to a state s if s satisfies $precond(a)$
  - i.e., if $precond^+(a) \subseteq s \wedge precond^-(a) \cap s = \varnothing$

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

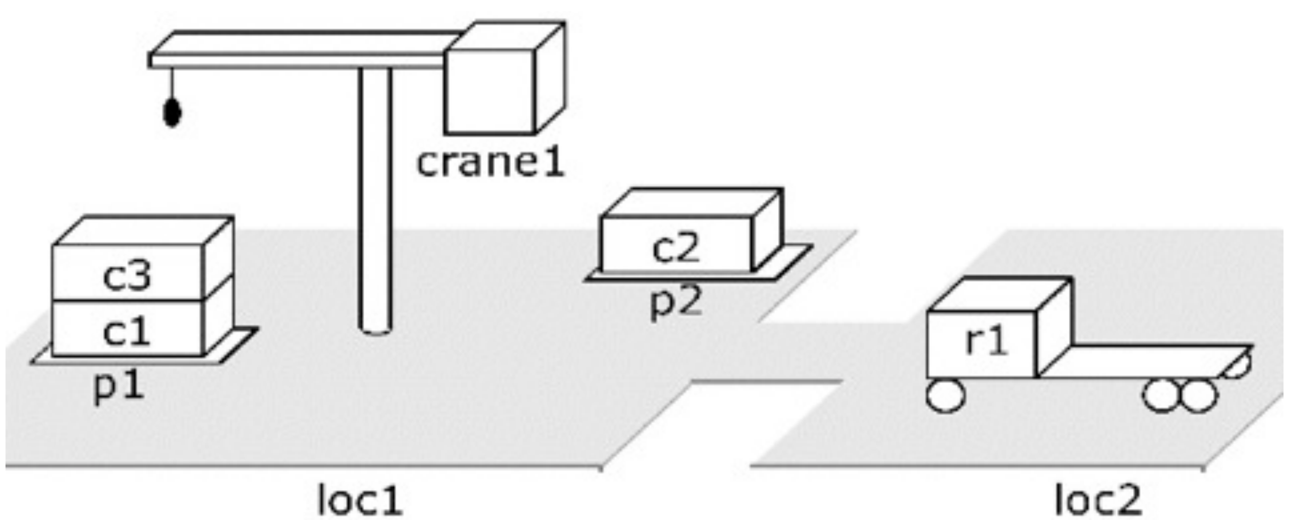


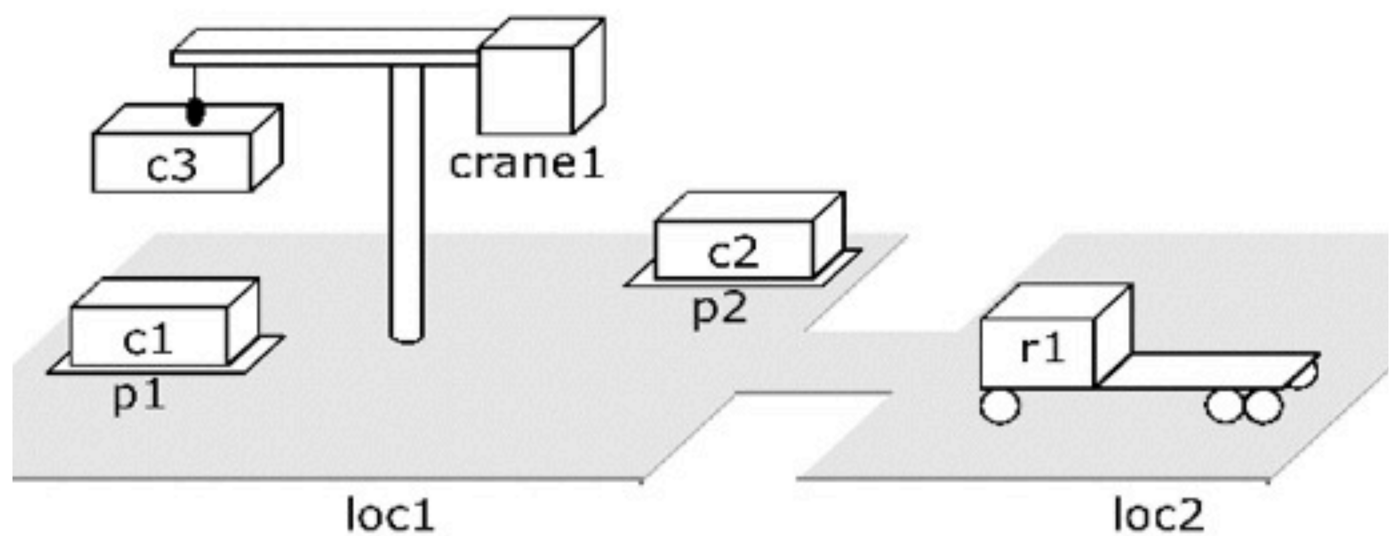
take(crane1,loc1,c3,c1,p1)
;; crane crane1 at location loc1 takes c3 off c1 in pile p1
precond: belong(crane1,loc1), attached(p1,loc1),
empty(crane1), top(c3,p1), on(c3,c1)
effects: holding(crane1,c3), ¬empty(crane1), ¬in(c3,p1),
¬top(c3,p1), ¬on(c3,c1), top(c1,p1)

Result of Performing an Action

- If a is applicable to s, the result of performing it is
  - $\gamma(s,a) = (s \setminus effects^{-}(a)) \cup effects^{+}(a)$
  - 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), ¬empty(crane1), ¬in(c3,p1),
¬top(c3,p1), ¬on(c3,c1), top(c1,p1)

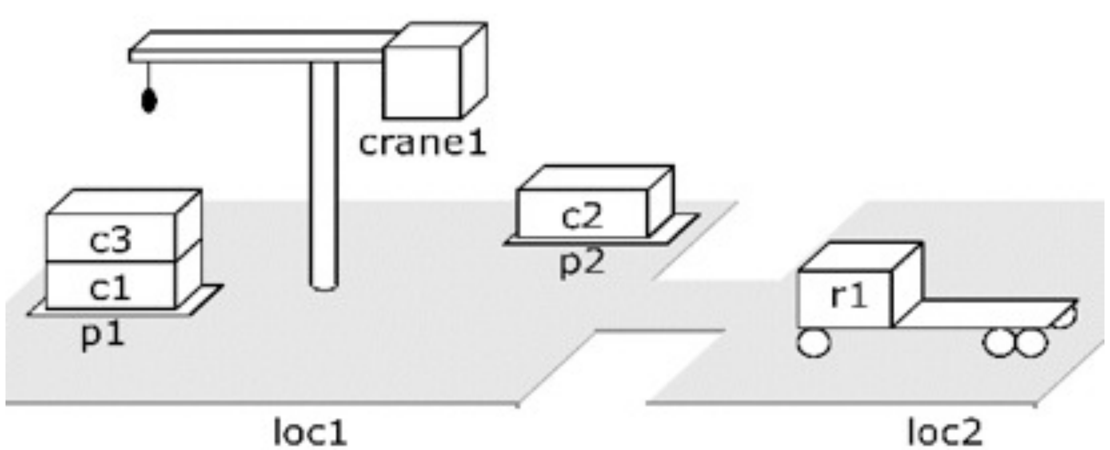




Planning Domains: Language plus Operators

- 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(r, c)

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)
  - The actual planning problem: $P = (\Sigma, s_0, S_g)$
    - s_0, S_g are as above
    - $\Sigma = (S, A, \gamma)$ is a state-transition system
    - 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
- We often say "planning problem" when we 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

$$
\begin{aligned}
\gamma(s_0, a_1) &= s_1 \\
\gamma(s_1, a_2) &= s_2 \\
&\vdots \\
\gamma(s_{n-1}, a_n) &= s_n \\
s_n &\vdash g
\end{aligned}
$$

Example: DWR – Dock Worker Robot

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

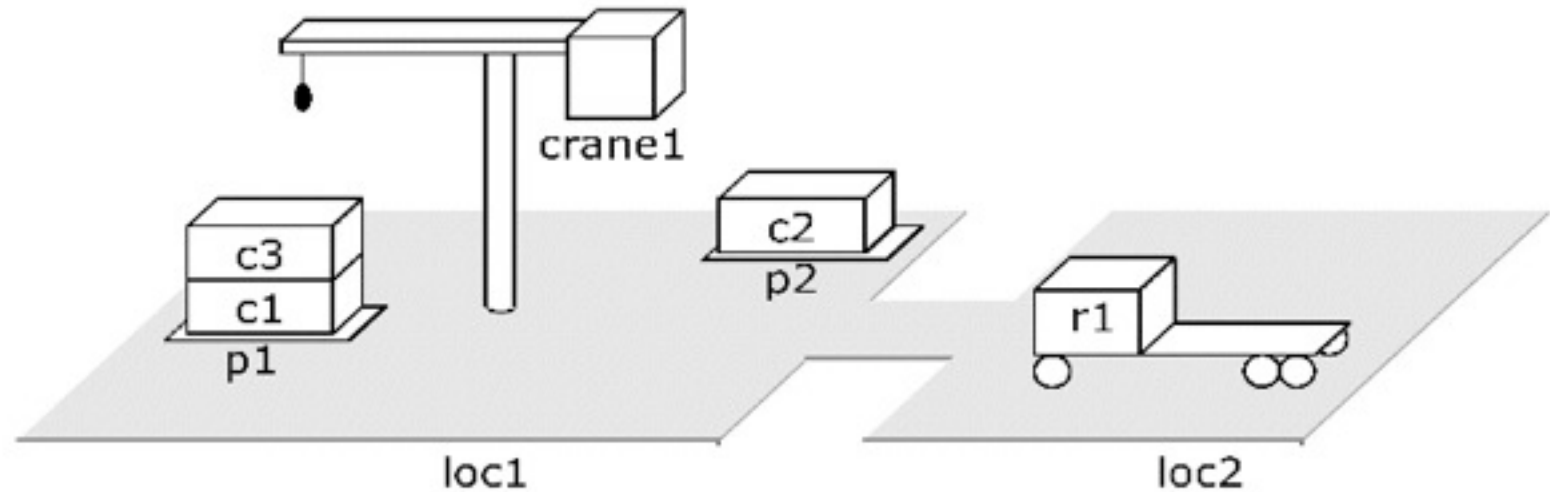
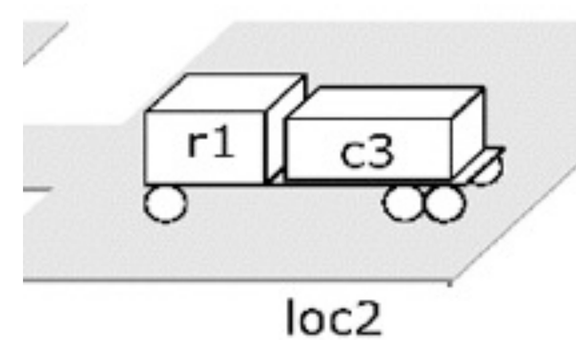
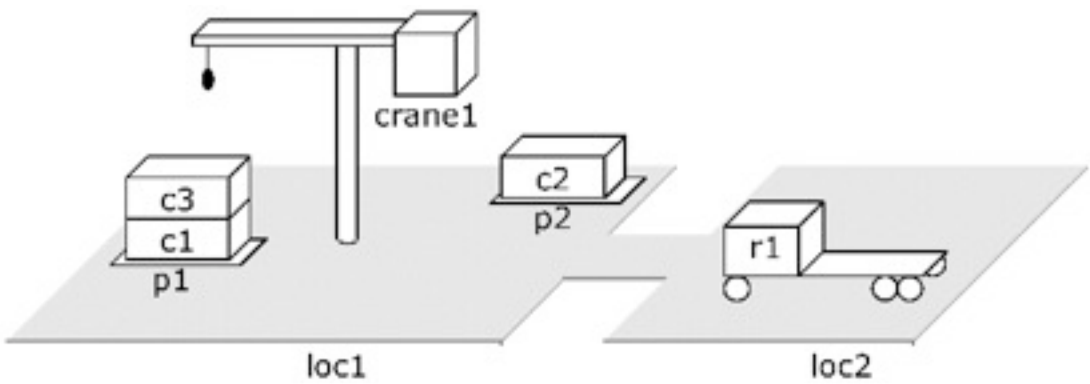


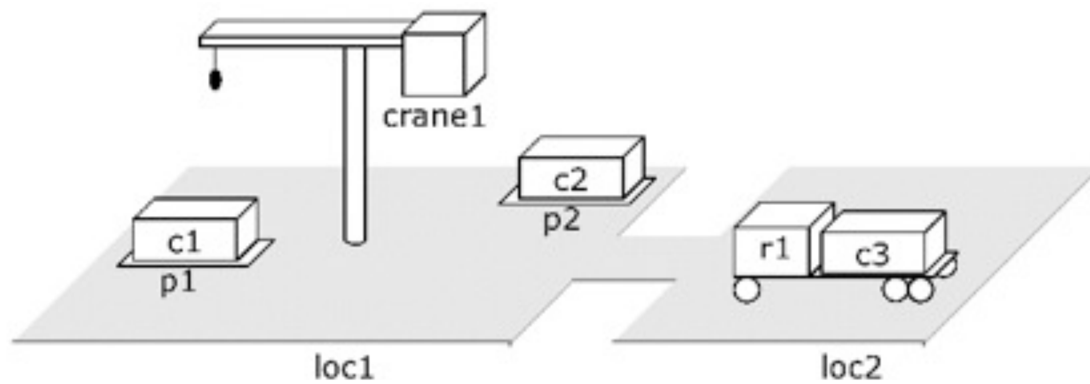
Figure 2.2: The DWR state s_1={attached(p1,loc1), in(c1,p1), in(c3,p1), top(c3,p1), on(c3,c1), on(c1,pallet), attached(p2,loc1), in(c2,p2), top(c2,p2), on(c2,pallet), belong(crane1,loc1), empty(crane1), adjacent(loc1,loc2), adjacent(loc2,loc1), at(r1,loc2), occupied(loc2), unloaded(r1)}.

  - $g_1 = \{loaded(r_1, c_3), at(r_1, loc_2)\}$



Example





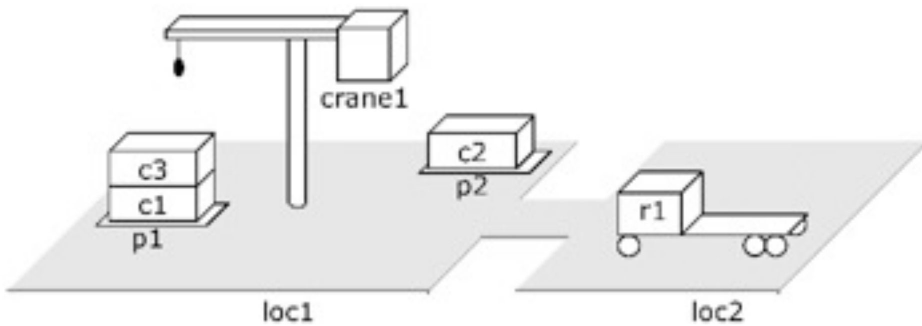
- Here are three solutions for P:

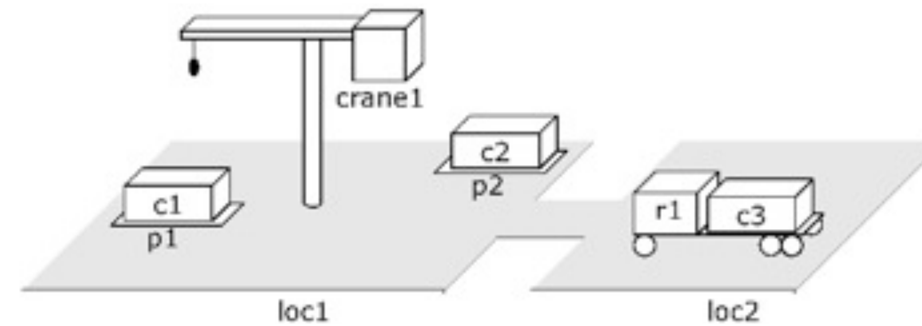
$$\langle\, take(crane_1, loc_1, c_1, p_1)\\ move(r_1, loc_2, loc_1),\\ load(crane_1, loc_1, c_3, r_1),\\ move(r_1, loc_1, loc_2) \quad \rangle$$

$$\langle\, move(r_1, loc_2, loc_1),\\ take(crane_1, loc_1, c_1, p_1)\\ load(crane_1, loc_1, c_3, r_1),\\ move(r_1, loc_1, loc_2) \quad \rangle$$

$$\langle\, take(crane_1, loc_1, c_1, p_1)\\ move(r_1, loc_2, loc_1),\\ move(r_1, loc_1, loc_2),\\ move(r_1, loc_2, loc_1),\\ load(crane_1, loc_1, c_3, r_1),\\ move(r_1, loc_1, loc_2) \quad \rangle$$

Example





- This one is redundant: can remove actions and still have a solution

$$\langle take(crane_1, loc_1, c_1, p_1)\\ move(r_1, loc_2, loc_1),\\ move(r_1, loc_1, loc_2),\\ move(r_1, loc_2, loc_1),\\ load(crane_1, loc_1, c_3, r_1),\\ move(r_1, loc_1, loc_2) \quad \rangle$$

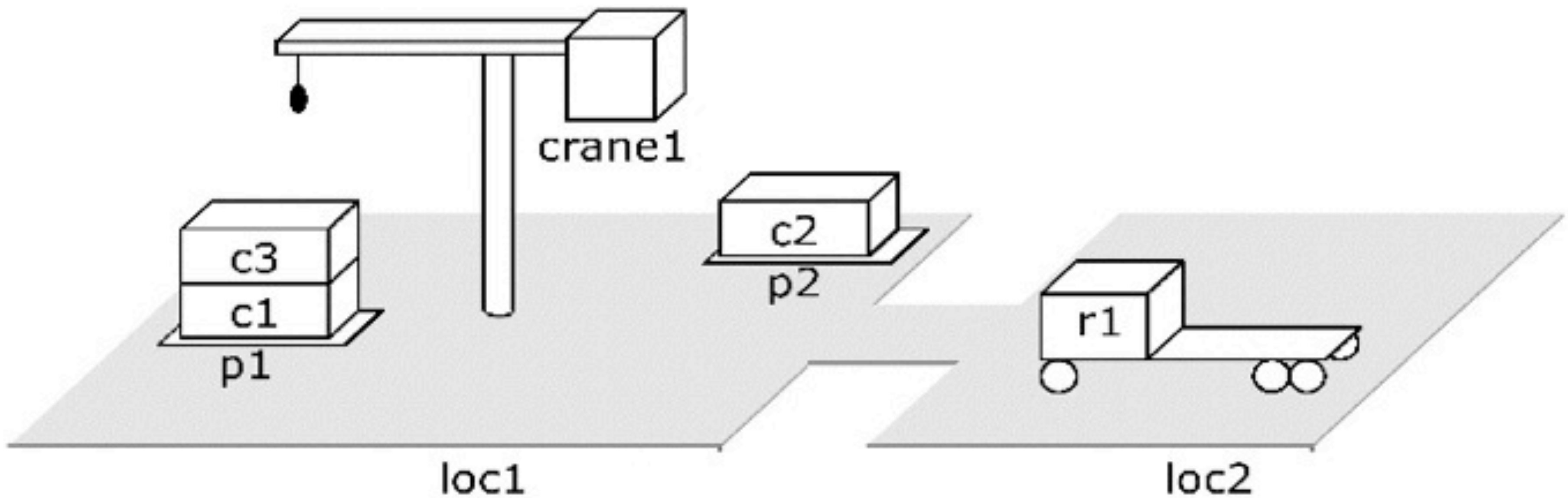
- These two are irredundant and shortest

$$\langle take(crane_1, loc_1, c_1, p_1)\\ move(r_1, loc_2, loc_1),\\ load(crane_1, loc_1, c_3, r_1),\\ move(r_1, loc_1, loc_2) \quad \rangle$$

$$\langle move(r_1, loc_2, loc_1),\\ take(crane_1, loc_1, c_1, p_1)\\ load(crane_1, loc_1, c_3, r_1),\\ move(r_1, loc_1, loc_2) \quad \rangle$$

Set-Theoretic Representation

- Like classical representation, but restricted to propositional logic



- States:
  - Instead of a collection of ground atoms …
  - $\{on(c_1, pallet), on(c_1, r1), on(c1, c2), \ldots, at(r_1, l_1), at(r_1, l_2), \ldots\}$
  - … use a collection of propositions (Boolean variables):
  - $\{\text{on-c1-pallet}, \text{on-c1-r1}, \text{on-c1-c2}, \ldots, \text{at-r1-l1}, \text{at-r1-l2}, \ldots\}$

Set-Theoretic Representation

- Instead of an operator like this one,

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)

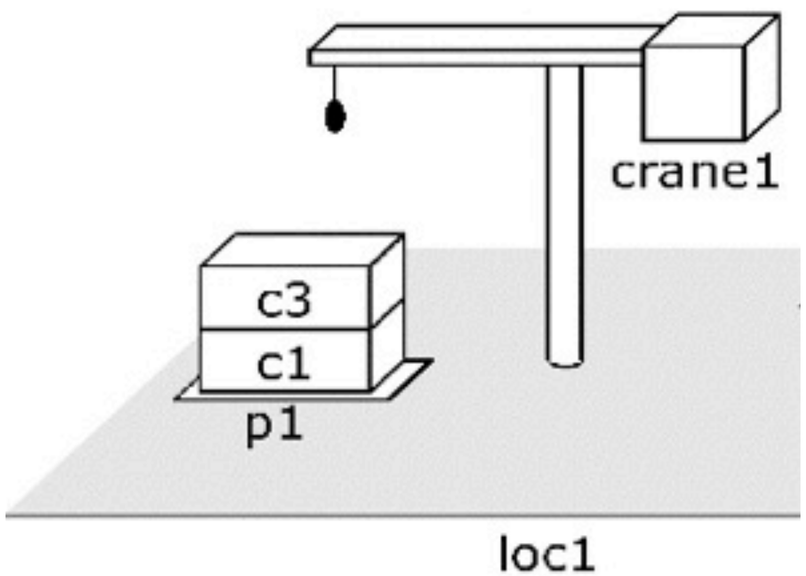
- … there are lots of actions like this one

```
take-crane1-loc1-c3-c1-p1
        precond: belong-crane-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 blow-up
  - If a classical operator contains n atoms and each atom has arity k,
  - then it corresponds to c^{nk} actions where $c = \|\{constantsymbols\}\|$

State-Variable Representation

- A state variable is like a field in a record structure



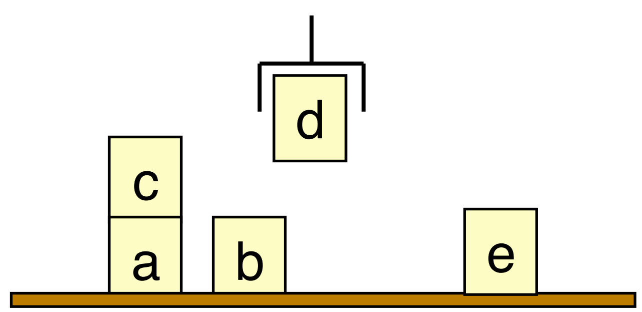
$\mathsf{load}(c, r, l)$
;; robot r loads container c at location l
precond: $\mathsf{rloc}(r) = l, \mathsf{cpos}(c) = l, \mathsf{rload}(r) = \mathsf{nil}$
effects: $\mathsf{rload}(r) \leftarrow c, \mathsf{cpos}(c) \leftarrow r$

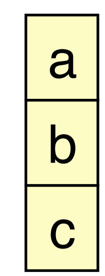
$\mathsf{unload}(c, r, l)$
;; robot r unloads container c at location l
precond: $\mathsf{rloc}(r) = l, \mathsf{rload}(r) = c$
effects: $\mathsf{rload}(r) \leftarrow \mathsf{nil}, \mathsf{cpos}(c) \leftarrow l$

- $\{top(p_1) = c_3, cpos(c_3) = c_1, cpos(c_1) = pallet, \ldots\}$

- Classical and state-variable representations take similar amounts of space
  - Each can be translated into the other in low-order polynomial time

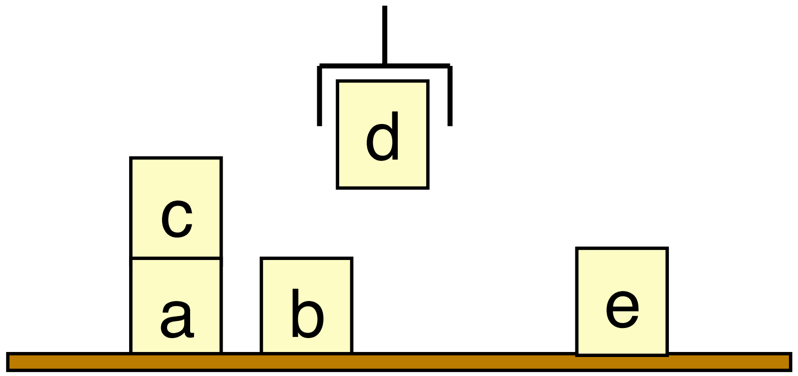
Example: The Blocks World

- 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 

    goal 

- 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

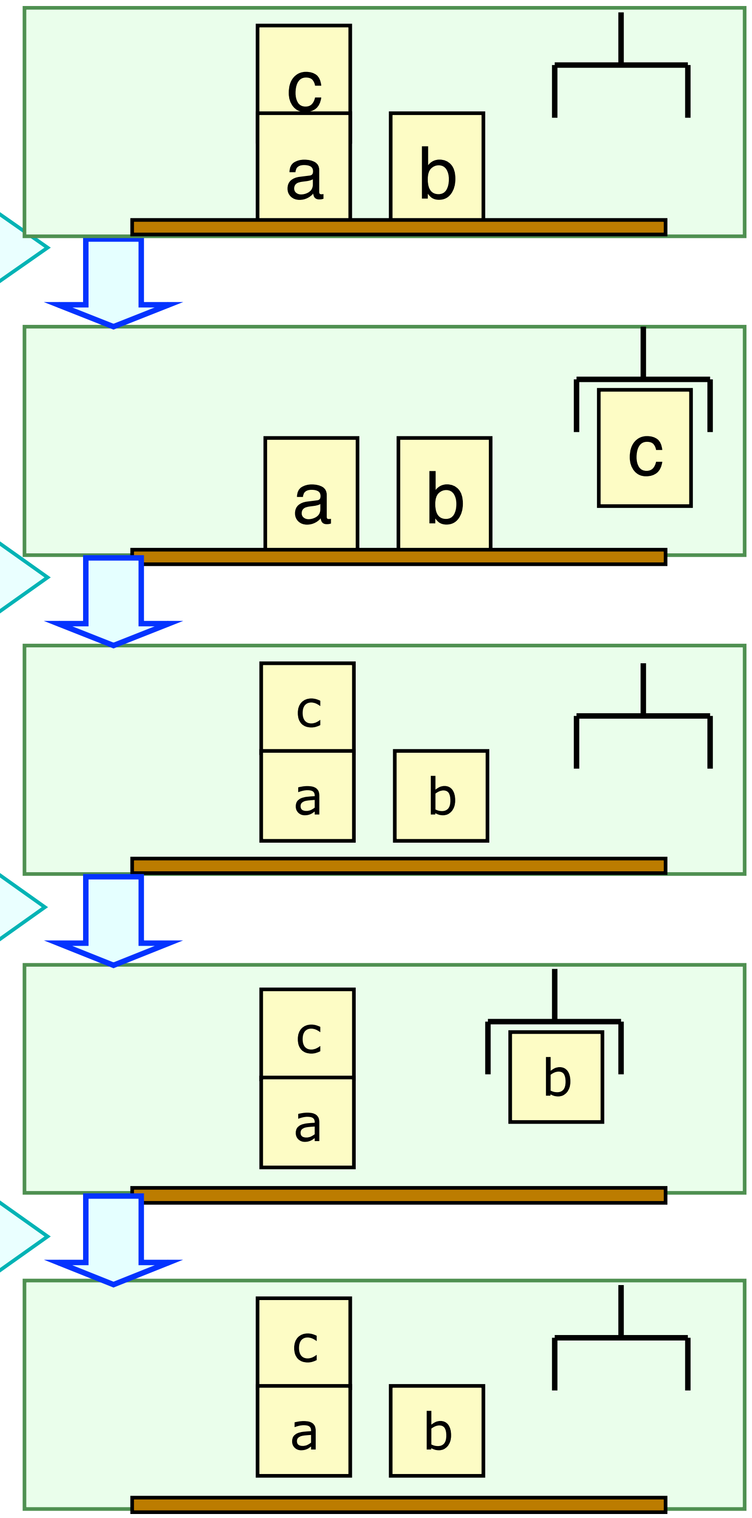
Classical Operators

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

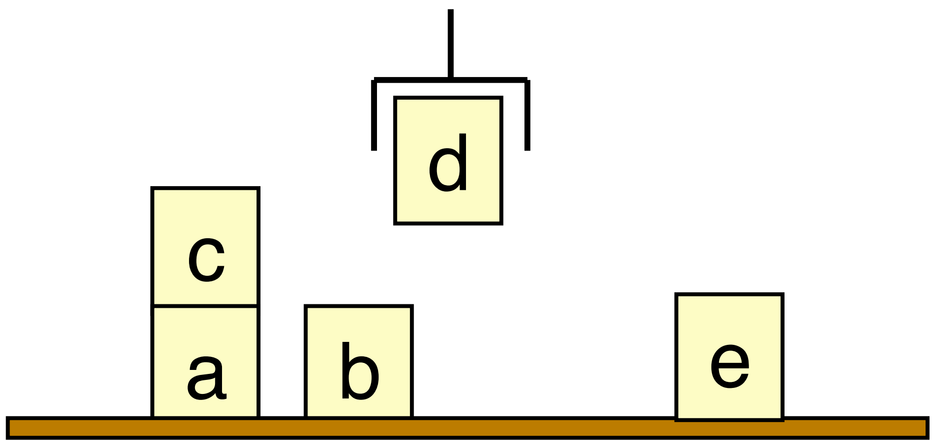
stack(*x*,*y*)
Precond: holding(*x*), clear(*y*)
Effects: ~holding(*x*), ~clear(*y*), on(*x*,*y*), clear(*x*), handempty

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

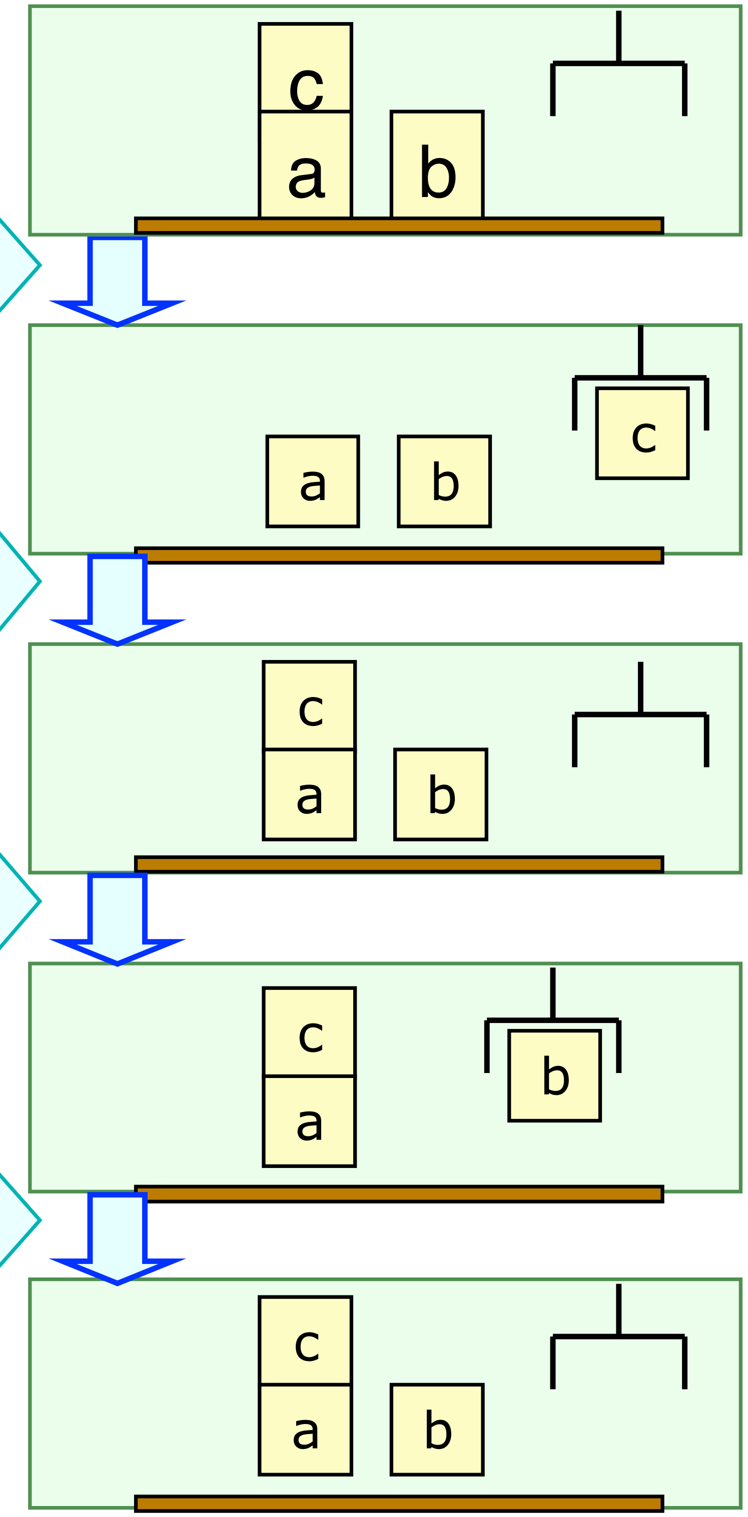
- Fifty different actions
- Four of them:

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

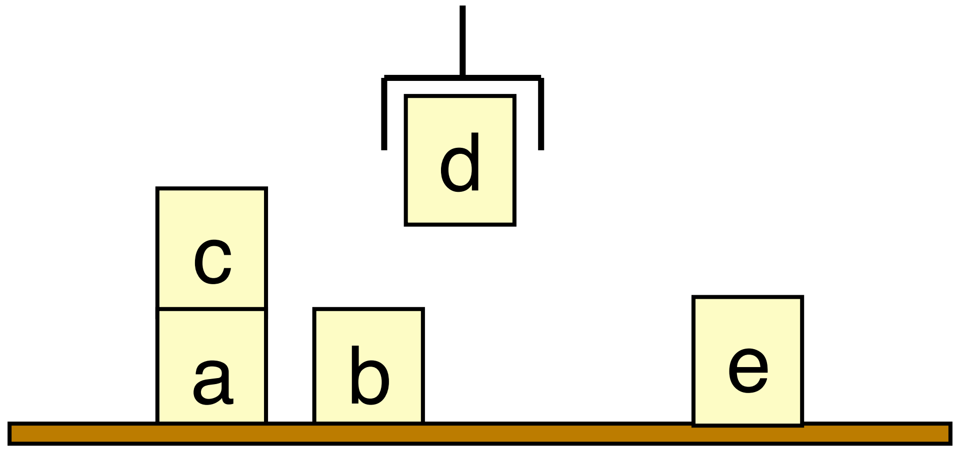
pickup-c
Pre: ontable-c, clear-c, handempty
Del: ontable-c, clear-c, handempty
Add: holding-c

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:
  - 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



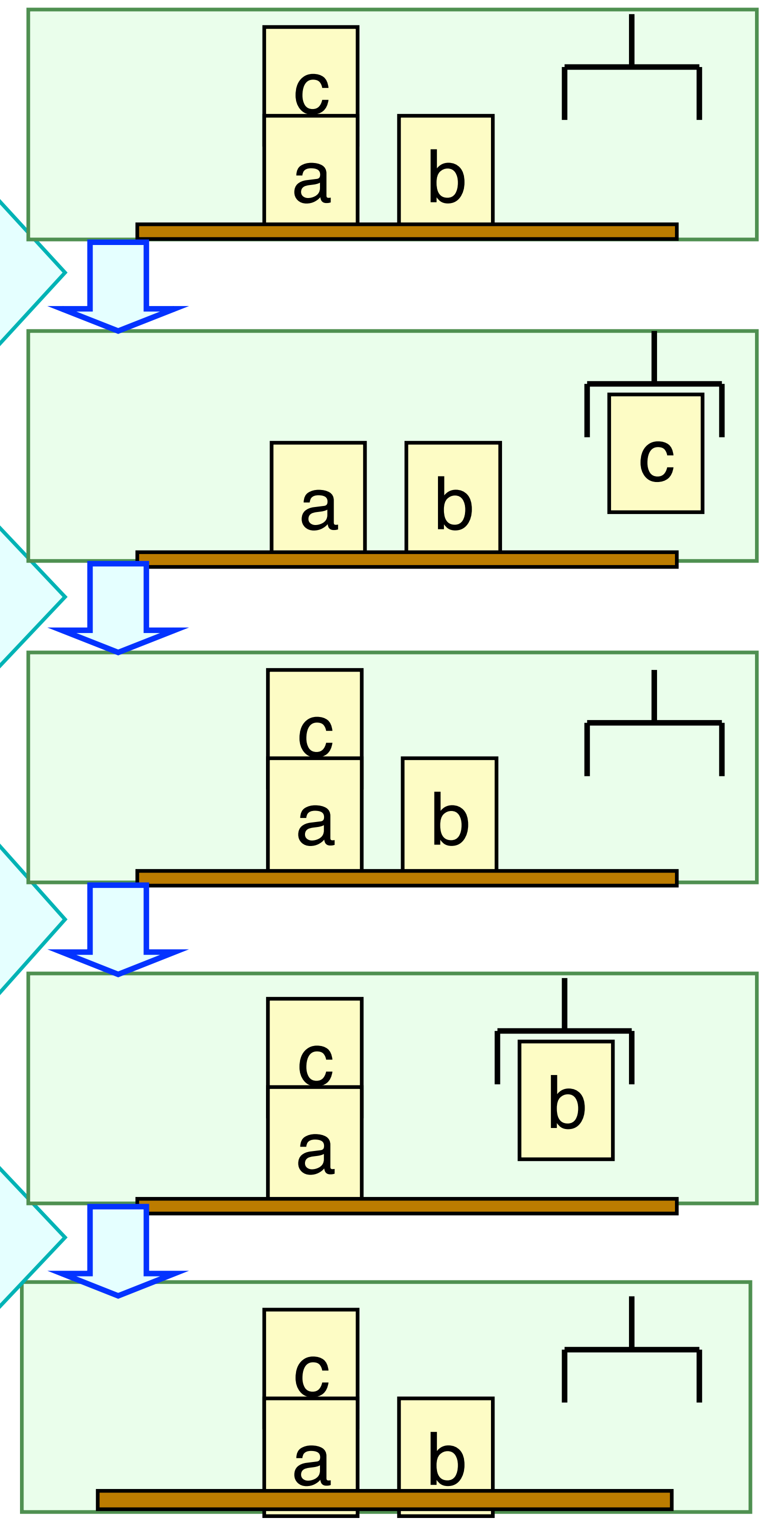
State-Variable Operators

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

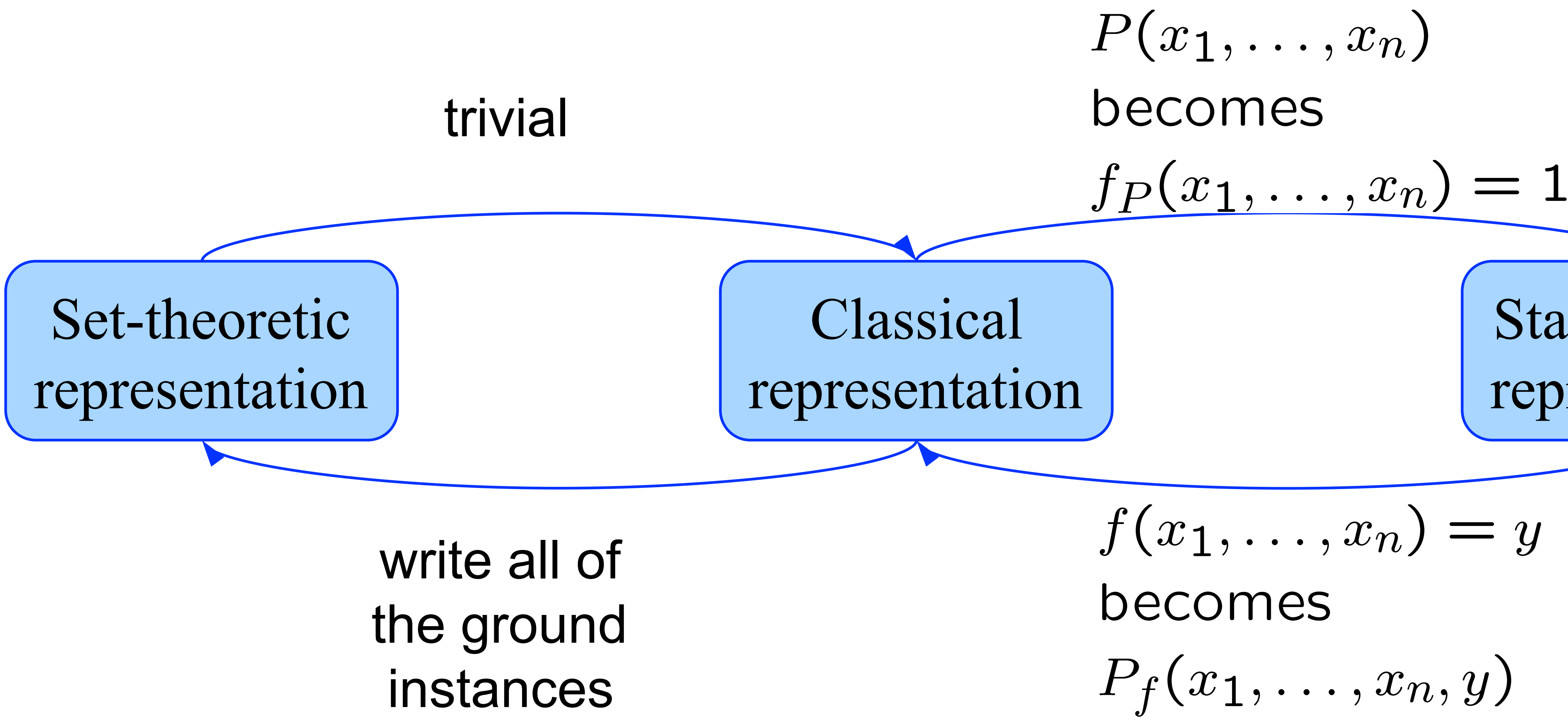
pickup(x : block)
Precond: pos(x)=table, clear(x)=1, holding=nil
Effects: pos(x)=nil, clear(x)=0, holding=x

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, except for the following:
  - Converting to set-theoretic from either of the others can incur 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
  - Less natural for logicians, more natural for engineers
  - Useful in non-classical planning problems as a way to handle numbers, functions, time