Foundations of Automated Planning

Roman Barták

Charles University, Czech Republic

What is the content?

- automated planning
- but what is planning and what is the difference from scheduling?

Why could it be interesting to me?

- is it used somewhere?
- any applications?

What is the course about?

- problem formalisation
- solving approaches

Expected knowledge

– search algorithms, logic, constraint satisfaction and SAT

Topics

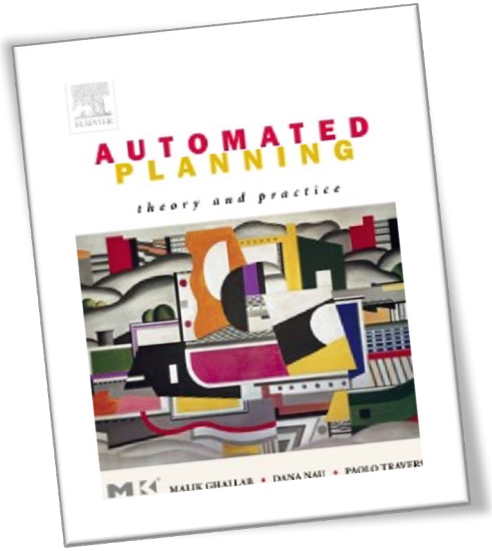
1. problem formalization
2. classical planning (STRIPS)
   - state-space and plan-space planning
3. neo-classical planning (Graphplan)
   - compilation to SAT and CSP
4. planning with time and resources
   - scheduling task inside planning
5. control knowledge and hierarchical planning
   - speeding-up planners

Automated Planning: Theory and Practice

M. Ghallab, D. Nau, P. Traverso

Morgan Kaufmann

https://projects.laas.fr/planning/aptp/

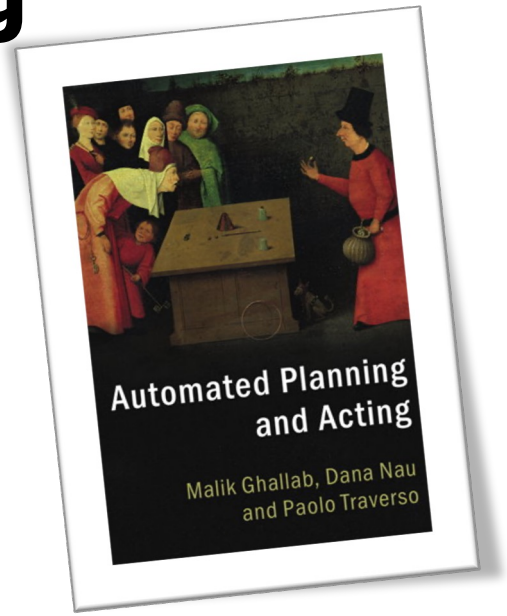


Automated Planning and Acting

M. Ghallab, D. Nau, P. Traverso

Cambridge University Press

https://projects.laas.fr/planning/



Project Shakey (1966-1972)

natural language processing

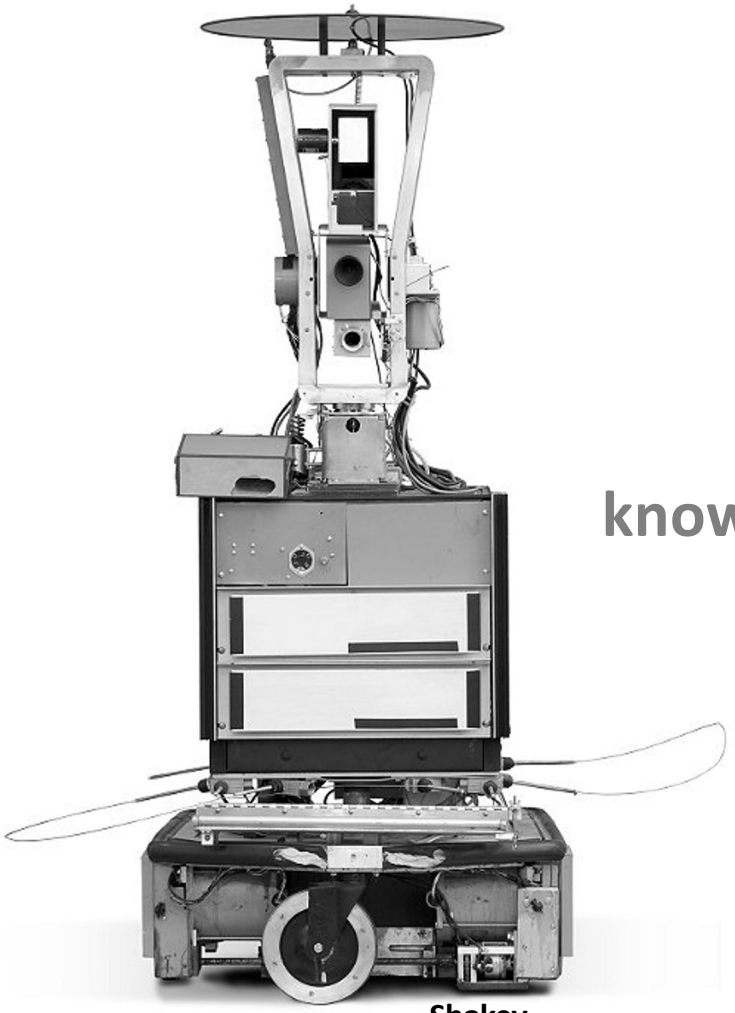
planning

machine learning

STRIPS

knowledge representation

computer vision



Shakey

algorithm A*

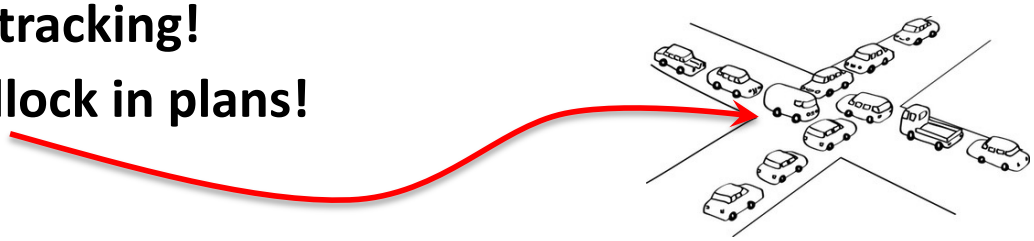
robotics

Launch: October 24, 1998

Target: Comet Borrelly

testing a payload of 12 advanced, high risk technologies

- **autonomous remote agent**
  - planning, execution, and monitoring spacecraft activities based on general commands from operators
  - three testing scenarios
    - 12 hours of low autonomy (execution and monitoring)
    - 6 days of high autonomy (operating camera, simulation of faults)
    - 2 days of high autonomy (keep direction)
      - » **beware of backtracking!**
      - » **beware of deadlock in plans!**

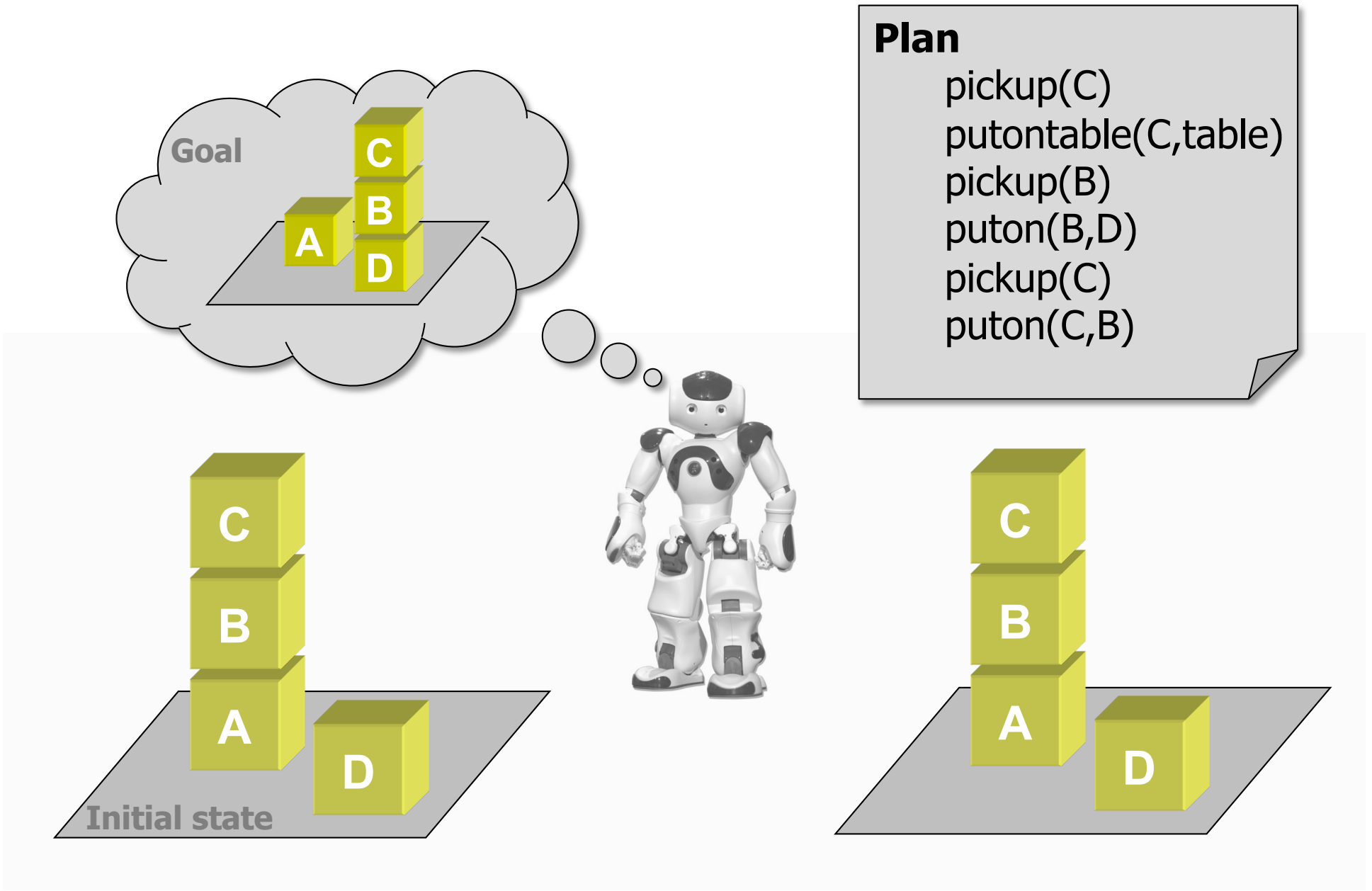


What?

What is planning and scheduling?

What is a difference between them?

What is planning?



Input:

- initial (current) state of the world
- description of actions that can change the world
- desired state of the world

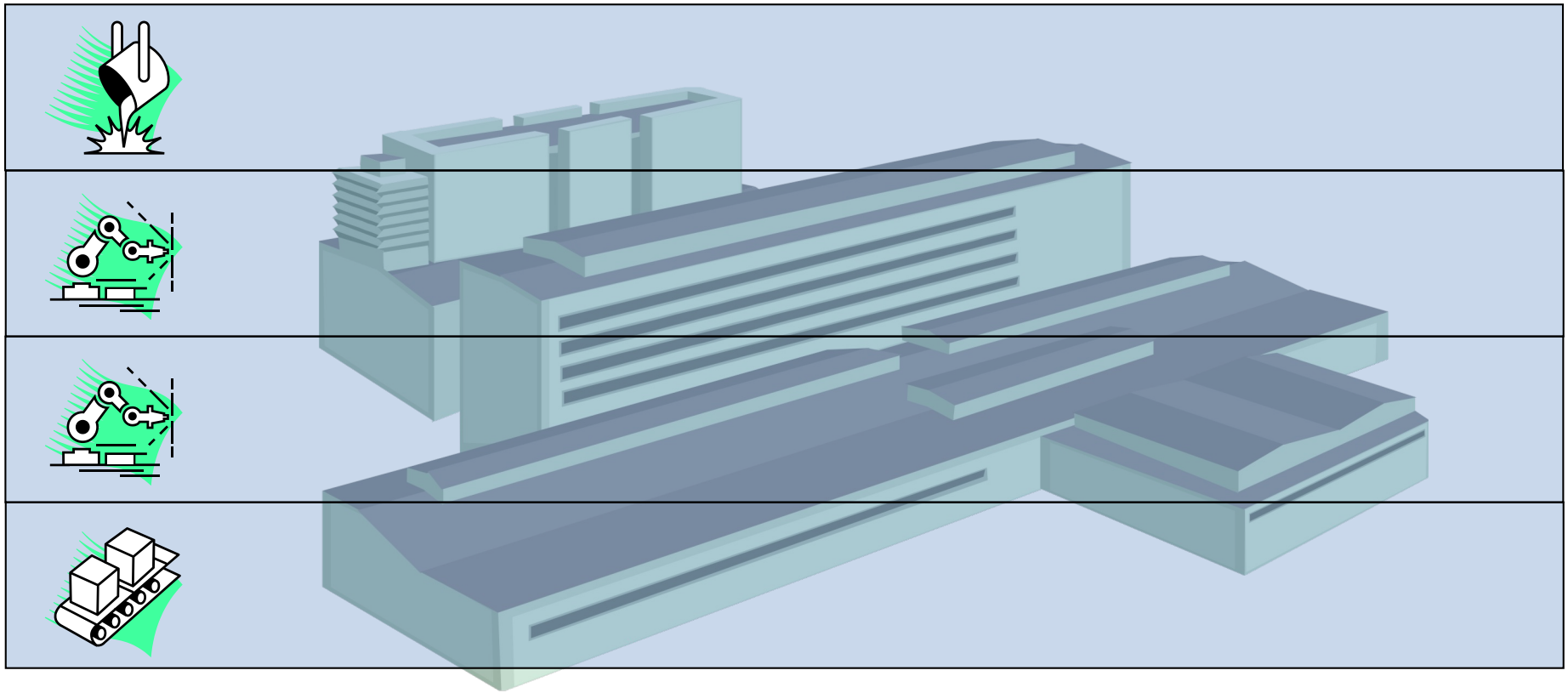
Output:

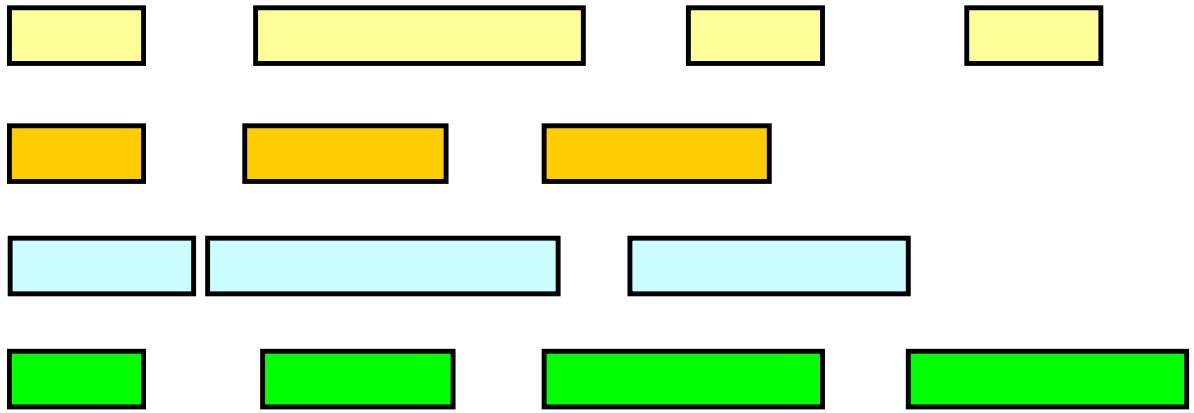
- a sequence of actions (a plan)

Properties:

- actions in the plan are unknown
- time and resources are not assumed

What is scheduling?





Input:

- a set of partially ordered activities
- available resources (machines, people, …)

Output:

- allocation of activities to time and resources (schedule)

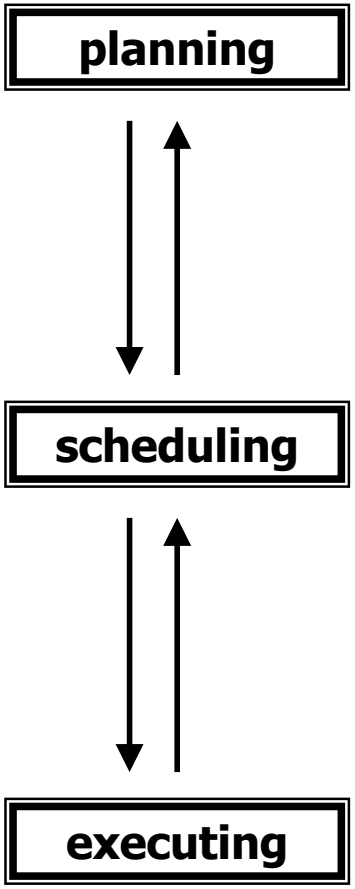
- **Properties:**
  - activities are known in advance
  - limited time and resources

Planning

- deciding which actions are necessary to achieve the goals
- topic of artificial intelligence
- complexity is usually worse than NP-c (in general, undecidable)

Scheduling

- deciding how to process the actions using given restricted resources and time
- topic of operations research
- complexity is typically NP-c



Planning deals with **causal relations** between actions and solves the problem which actions are necessary to reach a goal.

Scheduling focuses on **allocation** of actions to **time** and **space** (resources).

Sometimes both tasks are better to be solved together.

– For example, when there are many plans but only a few of them can be scheduled.

Let’s start

Planning problem formalisation

Planning deals with **selection and organization of actions** that are changing world states.

System Σ modelling states and transitions:

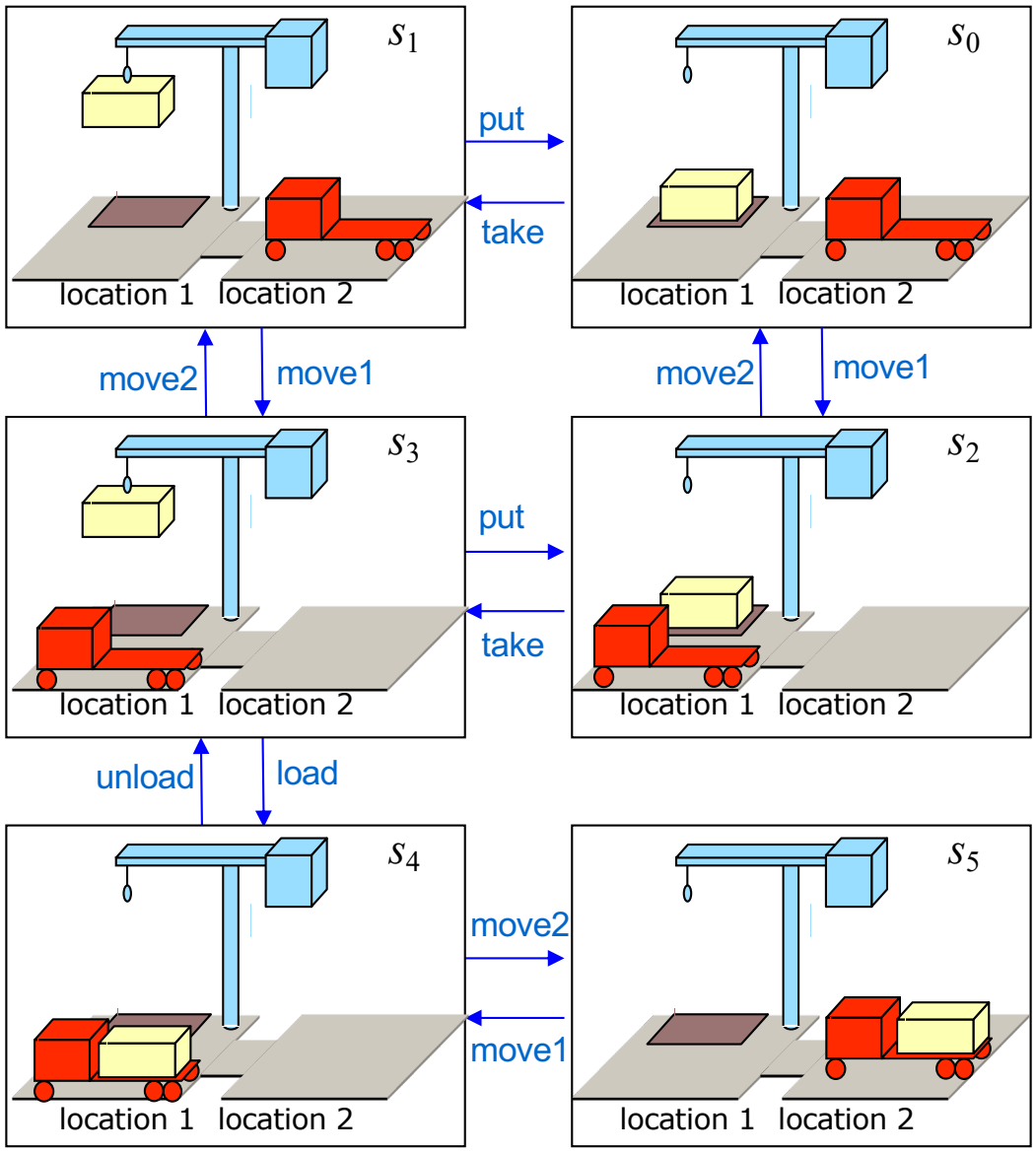
- **set of states S** (recursively enumerable)
- **set of actions A** (recursively enumerable)
  - actions are controlled by the planner!
  - no-op
- **set of events E** (recursively enumerable)
  - events are out of control of the planner!
  - neutral event ε
- **transition function** γ: $S \times A \times E \rightarrow 2^S$
  - actions and events are sometimes applied separately
    γ: $S \times (A \cup E) \rightarrow 2^S$

A **planning task** is to find which actions are applied to world states to reach some goal from a given initial state.

What is a goal?

- **goal state** or a set of goal states
- **satisfaction of some constraint** over a sequence of visited states
  - for example, some states must be excluded or some states must be visited
- **optimisation of some objective function** over a sequence of visited states (actions)
  - for example, maximal cost or a sum of costs

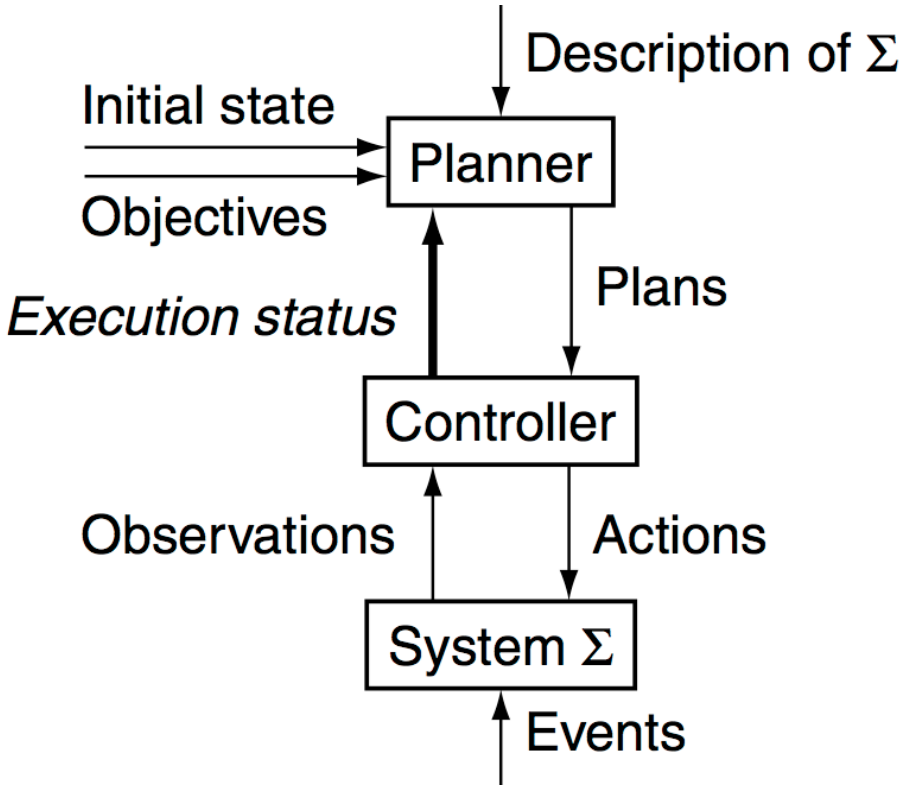
Example



$\Sigma = (S,A,E,\gamma)$

- $S = \{s_0, \ldots, s_5\}$
- $E = \{\}$ resp. $\{\varepsilon\}$
- A = {move1, move2, put, take, load, unload}
- γ: see figure

- init: s_0
- goal: s_5



A **planner** generates plans.

A **controller** takes care about plan execution.

- for each state it selects an action to execute
- observations (sensor input) are translated to world states

Dynamic planning involves re-planning when the state is not as expected.

Simplifying assumptions

- the system is **finite**
- the system is **fully observable**
  - We know completely the current state.
- the system is **deterministic**
  - $\forall s \in S\ \forall u \in (A \cup E)$: $|\gamma(s,u)| \leq 1$
- the system is **static**
  - There are no external events.
- the **goals** are **restricted**
  - The aim is to reach one of the goal states.
- the **plans** are **sequential**
  - A plan consists of a (linearly ordered) sequence of actions.
- **time** is **implicit**
  - Actions are instantaneous (no duration is assumed).
- **planning** is done **offline**
  - State of the world does not change during planning.

We will work with a deterministic, static, finite, and fully observable state-transition system with restricted goals and implicit time: $\Sigma = (S,A,\gamma)$.

Planning problem $P = (\Sigma, s_0, g)$:

- s_0 is the **initial state**
- g describes the **goal states**

A solution to the planning problem P is a sequence of actions $\langle a_1,a_2,\ldots,a_k\rangle$ with a corresponding sequence of states $\langle s_0,s_1,\ldots,s_k\rangle$ such that $s_i = \gamma(s_{i-1},a_i)$ and s_k satisfies goal condition g.

☞ **Classical planning (STRIPS planning)** ☜

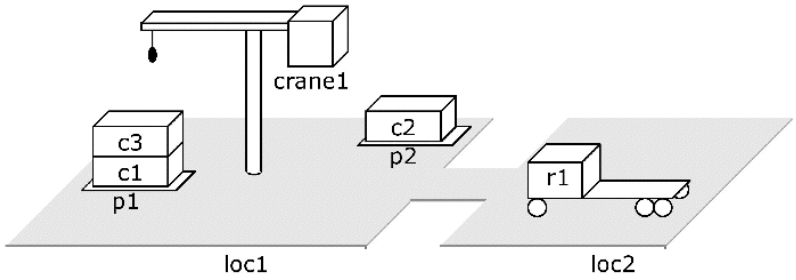
Planning in the restricted model reduces to “path finding” in the graph defined by states and state transitions.

Is it really so simple?

5 locations, 3 piles per location, 100 containers, 3 robots

↳ **10^{277} states**

This is 10^{190} times more than the largest estimate of the number of particles in the whole universe!



How to represent states and actions without enumerating the sets S and A?

- recall 10^{277} states with respect to the number of particles in the universe

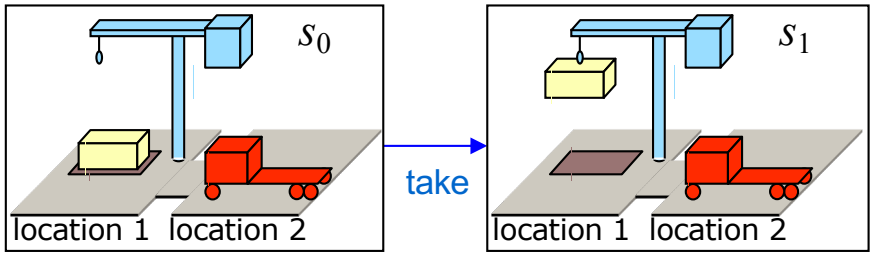
How to efficiently solve planning problems?

- How to find a path in a graph with 10^{277} nodes?

Each **state** is described using a **set of propositions** that hold at that state.
example: {onground, at2}

Each **action** is a syntactic expression describing:

- which propositions must hold in a state so the action is applicable to that state
  example: take: {onground}
- which propositions are added and deleted from the state to make a new state
  example:
  take: {onground}$^-$, {holding}$^+$



Let L= $\{p_1, \ldots, p_n\}$ be a finite set of propositional symbols (language).

A planning domain Σ over L is a triple (S,A,γ):

- $S \subseteq P(L)$, i.e. **state** s is a subset of L describing which propositions hold in that state
  - if **p** $\in$ **s**, then **p** holds in **s**
  - if **p** $\notin$ **s**, then **p** does not hold in **s**
- **action** a $\in$ A is a triple of subsets of L
  a = (precond(a), effects$^-$(a), effects$^+$(a))
  - effects$^-$(a) $\cap$ effects$^+$(a) = $\varnothing$
  - action **a** is applicable to state **s** iff precond(**a**) $\subseteq$ **s**
- **transition function γ**:
  - γ(**s**,**a**) = (**s** – effects$^-$(**a**)) $\cup$ effects$^+$(**a**), if **a** is applicable to **s**

Planning problem P is a triple (Σ, s_0, g):

- $\Sigma = (S, A, \gamma)$ is a planning domain over L
- s_0 is an initial state, $s_0 \in S$
- $g \subseteq L$ is a set of goal propositions

  $S_g = \{s \in S \mid g \subseteq s\}$ is a set of goal states

Plan π is a sequence of actions $\langle a_1, a_2, \ldots, a_k \rangle$

- **the length of plan** π is $k = |\pi|$
- **a state obtained by the plan** π (a transitive closure of γ)

  $\gamma(s,\pi) = s$, if k=0 (plan π is empty)

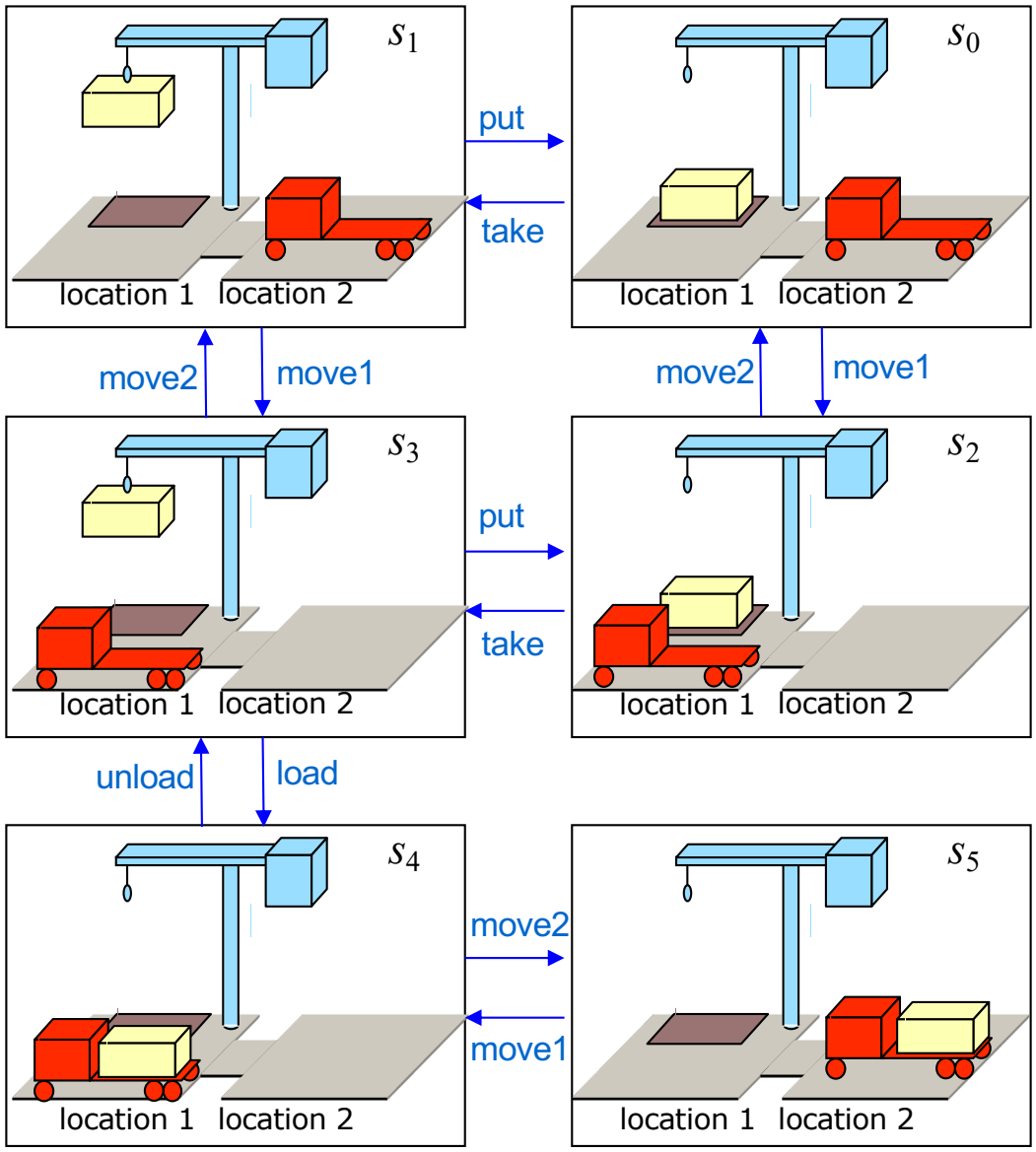
  $\gamma(s,\pi) = \gamma(\gamma(s,a_1), \langle a_2, \ldots, a_k \rangle)$, if k>0 and a_1 is applicable to s

  $\gamma(s,\pi)$ = undefined, otherwise

Plan π is a **solution plan** for P iff $g \subseteq \gamma(s_0,\pi)$.

- **redundant plan** contains a subsequence of actions that also solves P
- **minimal plan**: there is no shorter solution plan for P

Set representation: example



L = {onground, onrobot,
holding, at1, at2}

s_0 = {onground, at2}

g = {onrobot}

load = (
{holding,at1},
{holding},
{onrobot})

⟨take,move1,load,move2⟩
is a plan,
but not a minimal plan

Direct successors of state s:

$\Gamma(s) = \{\gamma(s,a) \mid a \in A \text{ is applicable to } s\}$

Reachable states:

$\Gamma_\infty(s) = \Gamma(s) \cup \Gamma^2(s) \cup \ldots$

Planning problem has a solution iff $S_g \cap \Gamma_\infty(s_0) \neq \emptyset$.

Action a is relevant for goal g if and only if:

$g \cap \text{effects}^+(a) \neq \emptyset$

$g \cap \text{effects}^-(a) = \emptyset$

Regression set for a goal g for (relevant) action a:

$\gamma^{-1}(g,a) = (g - \text{effects}^+(a)) \cup \text{precond}(a)$

$\Gamma^{-1}(g) = \{\gamma^{-1}(g,a) \mid a \in A \text{ is relevant for } g\}$

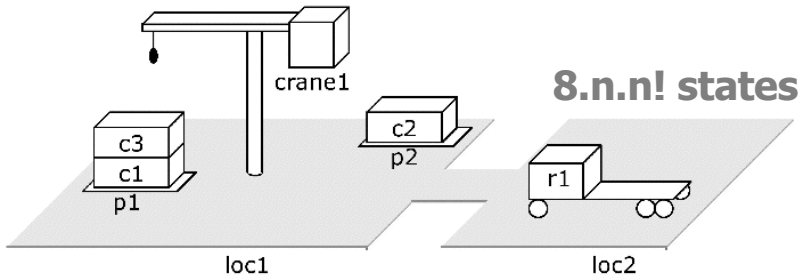
$\Gamma_\infty^{-1}(g) = \Gamma^{-1}(g) \cup \Gamma^{-2}(g) \cup \ldots$

Planning problem has a solution iff s_0 is a superset of some element in $\Gamma_\infty^{-1}(g)$.

Simplicity

- easy to read

How many states for n containers?



{nothing-on-c3, c3-on-c1,c1-on-pile1, nothing-on-c2, c2-on-pile2, crane-empty, robot-at-loc2}

Computations

- the transition function is easy to model/compute using set operations
- if precond(a) $\subseteq$ s, then
  $\gamma(s,a) = (s - effects^{-}(a)) \cup effects^{+}(a)$,

Expressivity

- some sets of propositions do not describe real states
  - {holding, onrobot, at2}
- for many domains, the set representation is still too large and not practical

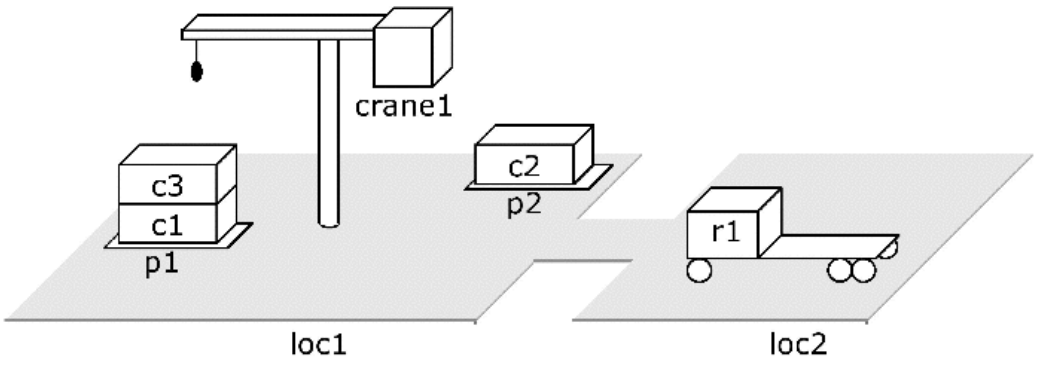
Classical representation generalizes the set representation by exploiting **first-order logic**.

- **State** is a set of logical atoms that are true in a given state.
- **Action** is an instance of a planning operator that changes truth values of some atoms.

More precisely:

- L (**language**) is a finite set of predicate symbols and constants (there are no function symbols!).
- **Atom** is a predicate symbol with arguments.
  example: on(c3,c1)
- We can use **variables** in the operators.
  example: on(x,y)

State is a set of instantiated atoms (no variables). There is a finite number of states!



{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)}.

The truth value of some atoms is changing in states:

- **fluents**
- *example: at(r1,loc2)*

The truth value of some state is the same in all states

- **rigid atoms**
- *example: adjacent(loc1,loc2)*

We will use a classical **closed world assumption**:

An atom that is not included in the state does not hold at that state!

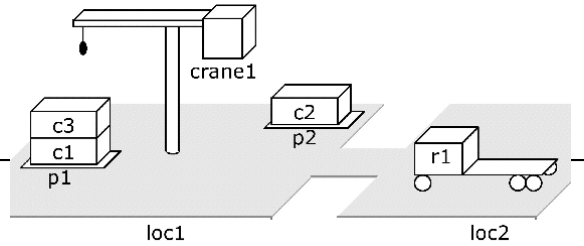
operator o is a triple (name(o), precond(o), effects(o))

- **name(o): name of the operator** in the form $n(x_1,\ldots,x_k)$
  - n: a symbol of the operator (a unique name for each operator)
  - $x_1,\ldots,x_k$: symbols for variables (operator parameters)
    - Must contain all variables appearing in the operator definition!
- **precond(o):**
  - literals that must hold in the state so the operator is applicable on it
- **effects(o):**
  - literals that will become true after operator application (only fluents can be there!)

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

An action is a fully instantiated operator

– substitute constants to variables



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)

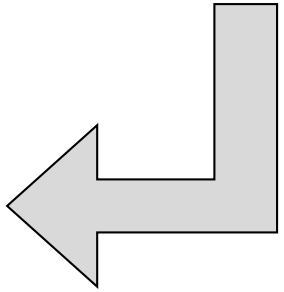
action

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:

- S^+ = {positive atoms in S}
- S^- = {atoms, whose negation is in S}

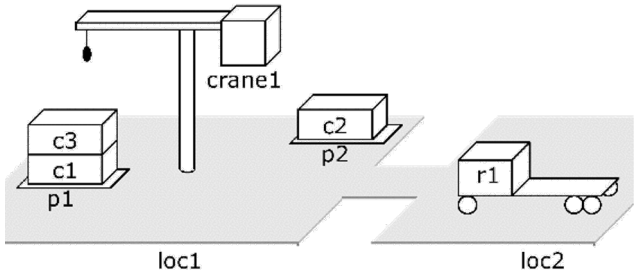
Action **a** is **applicable** to state **s** if and only if

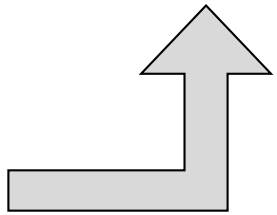
$\text{precond}^+(\mathbf{a}) \subseteq \mathbf{s} \ \wedge \ \text{precond}^-(\mathbf{a}) \cap \mathbf{s} = \varnothing$

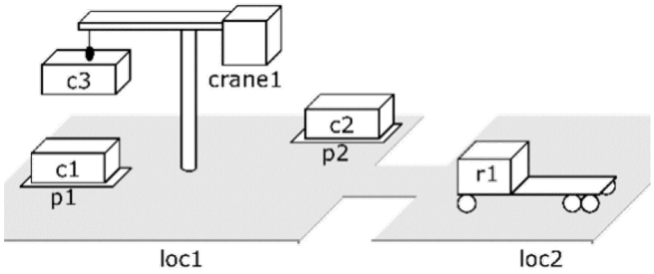
The result of application of action a to **s** is

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

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)







Let L be a language and O be a set of operators.

Planning domain Σ over language L with operators O is a triple (S,A,γ):

- **states** $S \subseteq P(\{\text{all instantiated atoms from L}\})$
- **actions** A = {all instantiated operators from O over L}
  - action **a** is **applicable** to state **s** if $\text{precond}^+(\mathbf{a}) \subseteq \mathbf{s} \ \wedge \ \text{precond}^-(\mathbf{a}) \cap \mathbf{s} = \varnothing$
- **transition function** γ:
  - $\gamma(\mathbf{s},\mathbf{a}) = (\mathbf{s} - \text{effects}^-(\mathbf{a})) \cup \text{effects}^+(\mathbf{a})$, if **a** is applicable on **s**
  - S is closed with respect to γ (if $\mathbf{s} \in S$, then for every action **a** applicable to **s** it holds $\gamma(\mathbf{s},\mathbf{a}) \in S$)

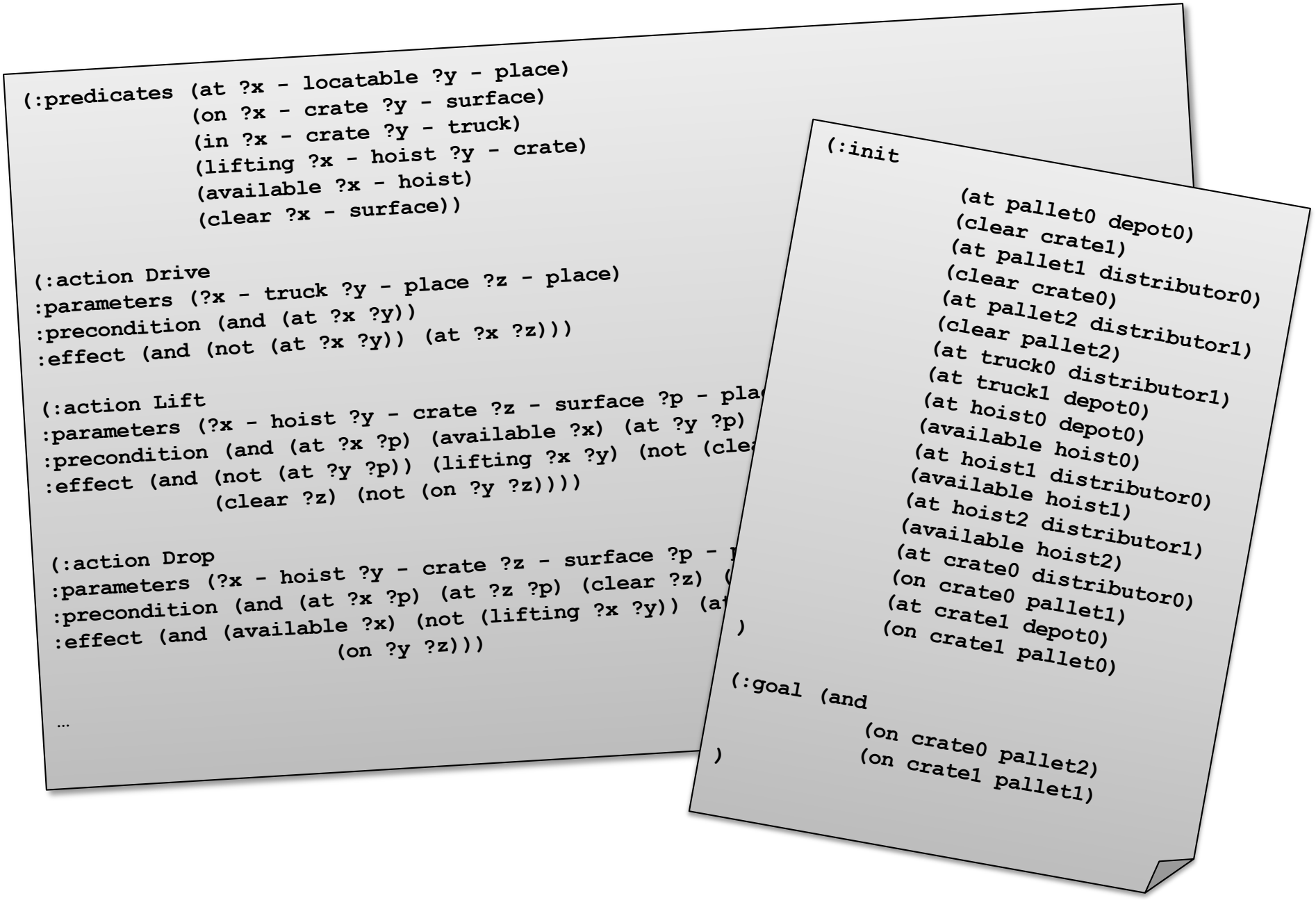
Planning problem P is a triple (Σ, s_0, g):

- $\Sigma = (S, A, \gamma)$ is a planning domain
- s_0 is an initial state, $s_0 \in S$
- g is a set of instantiated literals
  - state **s** satisfies the goal condition **g** if and only if $\mathbf{g}^+ \subseteq \mathbf{s} \ \wedge \ \mathbf{g}^- \cap \mathbf{s} = \varnothing$
  - $S_\mathbf{g} = \{\mathbf{s} \in S \mid \mathbf{s}$ satisfies $\mathbf{g}\}$ – a set of goal states

Usually the planning problem is given by a triple (O, s_0, g).

- O defines the the operators and predicates used
- s_0 provides the particular constants (objects)

Planning Domain Description Language (PDDL)



```
(:predicates (at ?x - locatable ?y - place)
             (on ?x - crate ?y - surface)
             (in ?x - crate ?y - truck)
             (lifting ?x - hoist ?y - crate)
             (available ?x - hoist)
             (clear ?x - surface))

(:action Drive
:parameters (?x - truck ?y - place ?z - place)
:precondition (and (at ?x ?y))
:effect (and (not (at ?x ?y)) (at ?x ?z)))

(:action Lift
:parameters (?x - hoist ?y - crate ?z - surface ?p - pla
:precondition (and (at ?x ?p) (available ?x) (at ?y ?p)
:effect (and (not (at ?y ?p)) (lifting ?x ?y) (not (cle
             (clear ?z) (not (on ?y ?z))))

(:action Drop
:parameters (?x - hoist ?y - crate ?z - surface ?p - p
:precondition (and (at ?x ?p) (at ?z ?p) (clear ?z) (
:effect (and (available ?x) (not (lifting ?x ?y)) (at
                    (on ?y ?z)))

…
```

```
(:init
            (at pallet0 depot0)
            (clear crate1)
            (at pallet1 distributor0)
            (clear crate0)
            (at pallet2 distributor1)
            (clear pallet2)
            (at truck0 distributor1)
            (at truck1 depot0)
            (at hoist0 depot0)
            (available hoist0)
            (at hoist1 distributor0)
            (available hoist1)
            (at hoist2 distributor1)
            (available hoist2)
            (at crate0 distributor0)
            (on crate0 pallet1)
            (at crate1 depot0)
            (on crate1 pallet0)
)

(:goal (and
            (on crate0 pallet2)
            (on crate1 pallet1)
)
```

Plan π is a sequence of actions $\langle a_1,a_2,\ldots,a_k\rangle$.

Plan π is a **solution of** P if and only if $\gamma(s_0,\pi)$ satisfies g.

- **Planning problem has a solutions iff $S_g \cap \Gamma_\infty(s_0) \neq \emptyset$.**
- **Planning problem has a solution iff s_0 is a superset of some element from $\Gamma_\infty^{-1}(g)$ (but γ^{-1} is defined a bit differently).**

Action a is relevant for a goal g if and only if :

action contributes to g: $g \cap effects(a) \neq \emptyset$

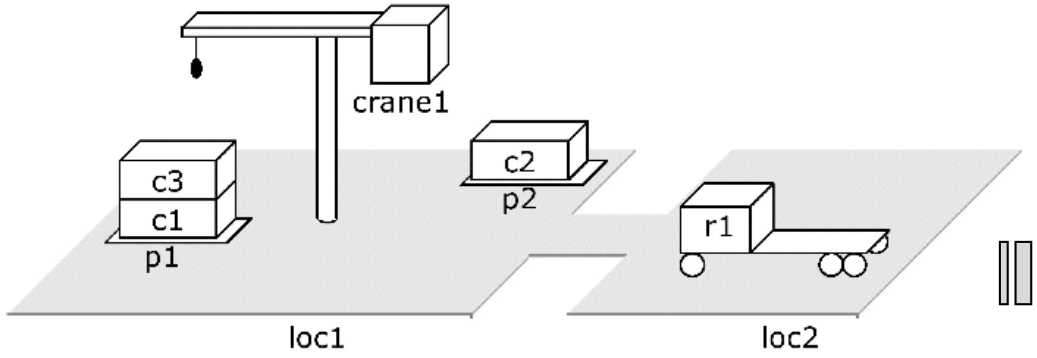
action effects are not in conflict with g:

- $g^- \cap effects^+(a) = \emptyset$
- $g^+ \cap effects^-(a) = \emptyset$

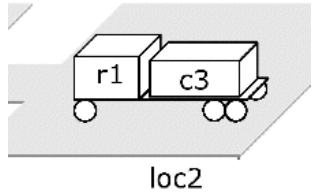
Regression set for a goal g for a (relevant) action a:

$\gamma^{-1}(g,a) = (g - effects(a)) \cup precond(a)$

Classical representation: an example plan



our goal

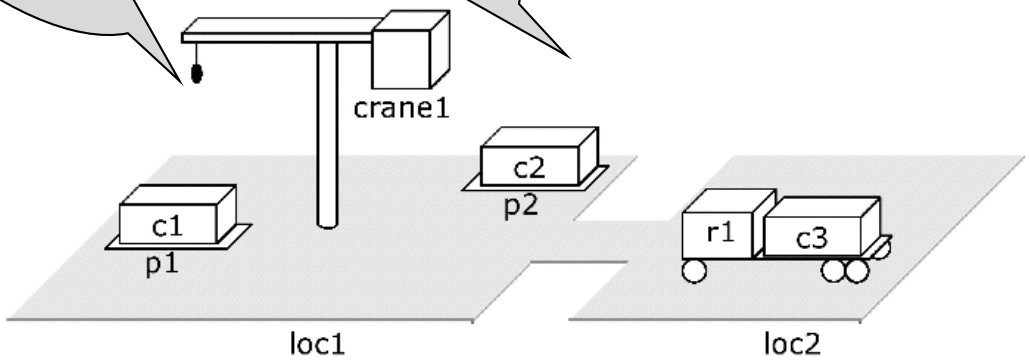


s_1 = {attached(p1,loc1), in(c1,p1), in(c3,p1), top(c3,p1), on(c3,c1), on(c1,pallet), attached(p2,loc1), in(c2,p2), top(c2,p2), on(c2,pallet), belong(crane1,loc1), empty(crane1), adjacent(loc1,loc2), adjacent(loc2,loc1), at(r1,loc2), occupied(loc2), unloaded(r1)}.

g = {loaded(r1,c3), at(r1,loc2)}

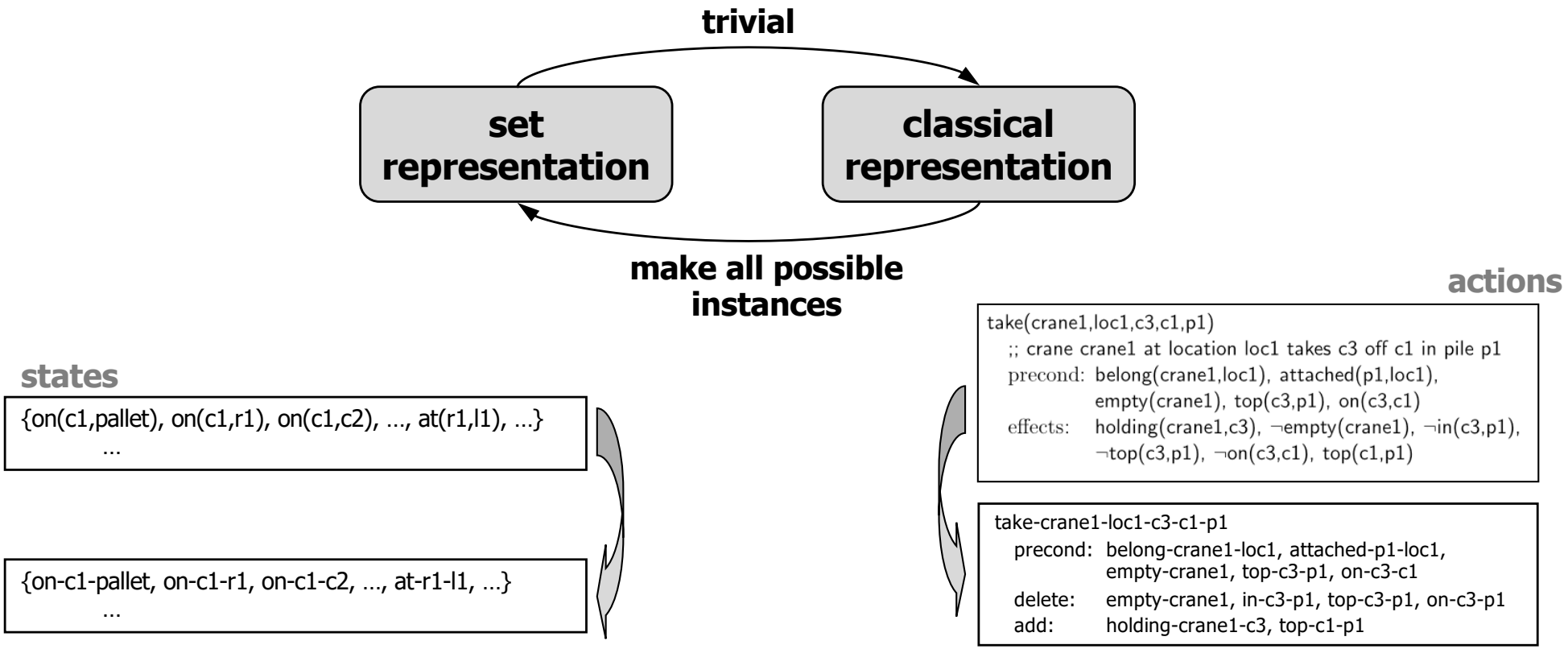
⟨take(crane1,loc1,c3,c1,p1),
move(r1,loc2,loc1),
load(crane1,loc1,c3,r1),
move(r1,loc1,loc2)⟩

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



Expressive power of both representations is **identical**.

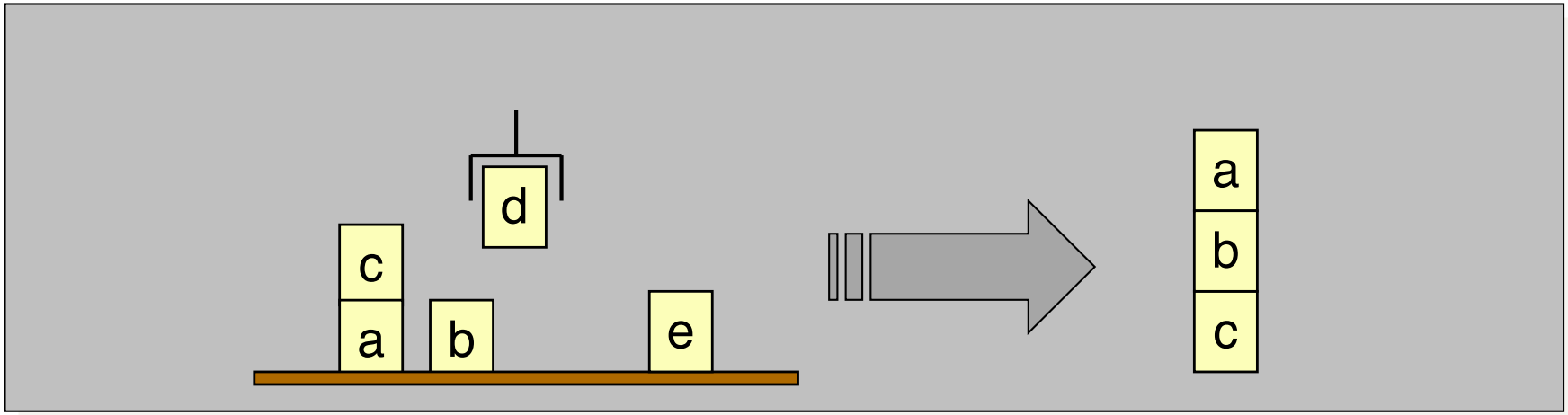
However, the translation from the classical representation to the set representation brings **exponential increase of size**.



The blocks world

- infinitely large table with a finite set of blocks
- the exact location of block on the table is irrelevant
- a block can be on the table or on another (single) block
- the planning domain deals with moving blocks by a robotic hand that can hold at most one block

situation example



Blockworld: classical representation

Constants

- blocks: a,b,c,d,e

Predicates:

- **ontable(*x*)**
  block *x* is on a table
- **on(*x*,*y*)**
  block *x* is on *y*
- **clear(*x*)**
  block *x* is free to move
- **holding(*x*)**
  the hand holds block *x*
- **handempty**
  the hand is empty

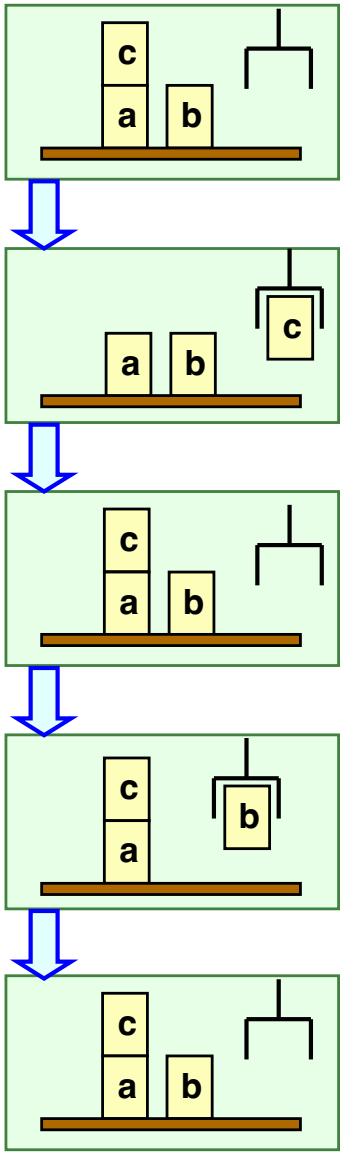
Operators

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

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



Blockworld: set representation

Propositions:

36 propositions for 5 blocks

- **ontable-a**
  block **a** is on table (5x)
- **on-c-a**
  block **c** is on block **a** (20x)
- **clear-c**
  block **c** is free to move (5x)
- **holding-d**
  the hand holds block **d** (5x)
- **handempty**
  the hand is empty (1x)

Actions

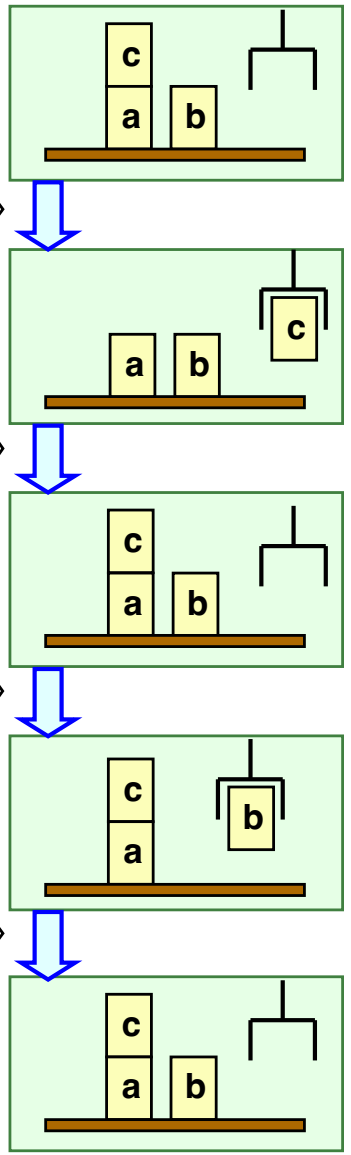
50 actions for 5 blocks

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



Charles University, Prague, Czech Republic
bartak@ktiml.mff.cuni.cz