Planning & Scheduling

Roman Barták
Department of Theoretical Computer Science and Mathematical Logic

Planning Problem Formalization

Today program

- **Problem Formalisation for Classical Planning**
  - conceptual model
    - state transitions
    - goals
    - initial assumptions
  - set representation (propositional logic)
    - states and actions
  - classical representation (first-order logic)
    - operators
    - planning domain and planning problem
  - some extensions

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

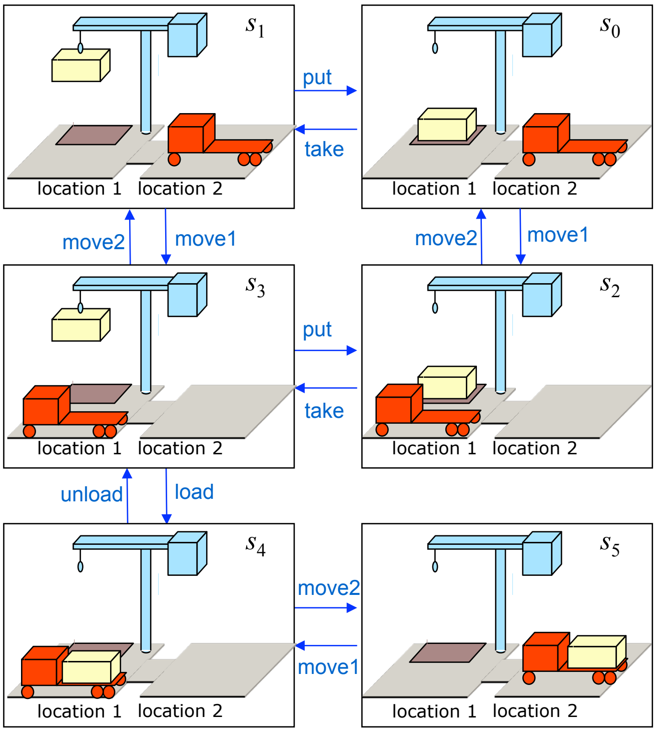
System Σ modelling states and transitions:

- **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: $S \times A \times E \rightarrow P(S)$
  - actions and events are sometimes applied separately
    γ: $S \times (A \cup E) \rightarrow P(S)$

- 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?

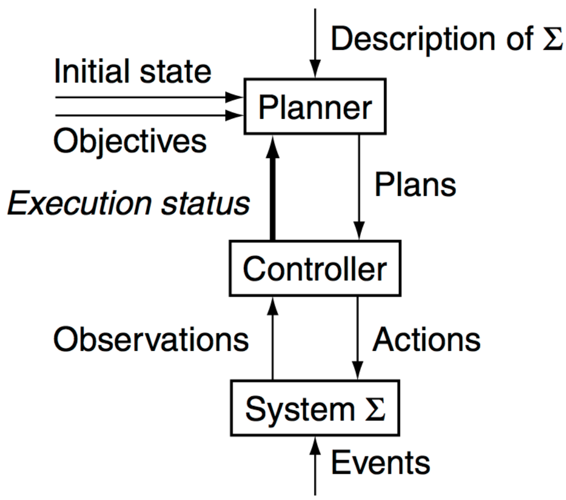
- **goal state** or a set of of goal states
- **satisfaction of some constraint** over a sequence of visited states
  - for example, some states must be excluded or some states must be visited
- **optimisation of some objective function** over a sequence of visited states (actions)
  - for example, maximal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



Σ = (S,A,E,γ)

- S = {s_0, …, s_5}
- E = {} resp. {ε}
- 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 state

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

Some 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.

Classical 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 g

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

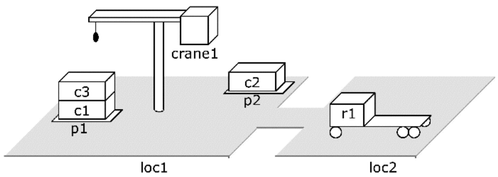
Planning in the restricted model reduces to "path finding" in the graph defined by states and state transitions.

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

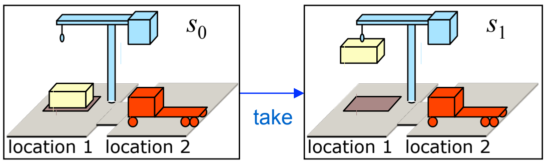
- **How to efficiently solve planning problems?**
  – How to find a path in a graph with 10^{277} nodes?

Set representation

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}$^+$



Set representation: a planning domain

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

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

- S ⊆ P(L), i.e. **state** s is a subset of L describing which propositions hold in that state
  - if **p** ∈ **s**, then **p** holds in **s**
  - if **p** ∉ **s**, then **p** does not hold in **s**
- **action** a ∈ A is a triple of subsets of L
  a = (precond(a), effects$^-$(a), effects$^+$(a))
  - effects$^-$(a) ∩ effects$^+$(a) = ∅
  - action **a** is applicable to state **s** iff precond(**a**) ⊆ **s**
- **transition function** γ:
  - γ(**s**,**a**) = (**s** – effects$^-$(**a**)) ∪ effects$^+$(**a**), if **a** is applicable to **s**

Set representation: a planning problem

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

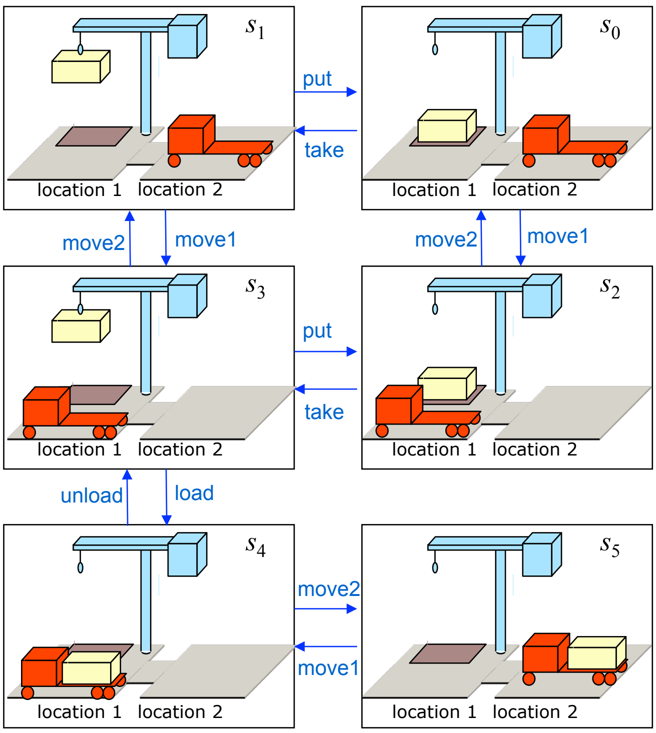
Plan π is a sequence of actions $\langle a_1, a_2, \ldots, a_k \rangle$

- **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

Set representation: reachability

Direct successors of state s:

$\Gamma(s) = \{\gamma(s,a) \mid a \in A$ 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:

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

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

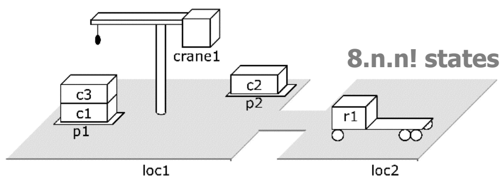
$\Gamma_\infty^{-1}(g) = \Gamma^{-1}(g) \cup \Gamma^{-2}(g) \cup \ldots$

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

Set representation: properties

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

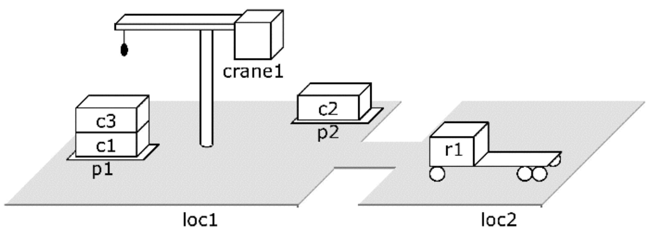
- **Classical representation** generalize the set representation by exploiting **first-order logic**.
  - **State** is a set of logical atoms that are true in a given state.
  - **Action** is an instance of planning operator that changes truth value of some atoms.

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)

Classical representation: states

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!

Classical representation: planning operators

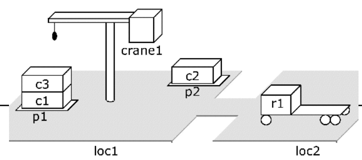
operator o is a triple (name(o), precond(o), effects(o))

- **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!)

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)

Classical representation: actions

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)

Classical representation: action usage

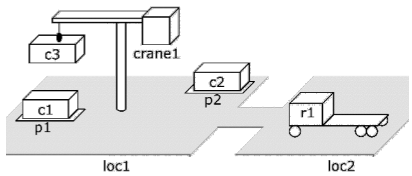
Notation:

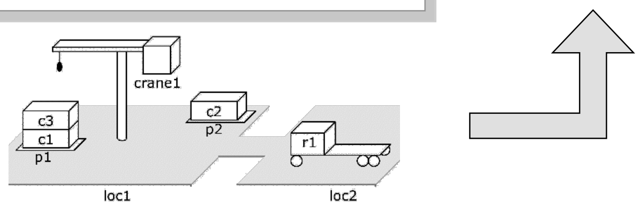
- S^+ = {positive atoms in S}
- S^- = {atoms, whose negation is in S}

Action **a** is **applicable** to state **s** if any only
$\text{precond}^+(\mathbf{a}) \subseteq \mathbf{s} \;\wedge\; \text{precond}^-(\mathbf{a}) \cap \mathbf{s} = \emptyset$

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)
```





Classical representation: a planning domain

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** $S \subseteq P$({all instantiated atoms from L})
- **actions** A = {all instantiated operators from O over L}
  - action **a** is **applicable** to state **s** if $\text{precond}^+(\mathbf{a}) \subseteq \mathbf{s} \;\wedge\; \text{precond}^-(\mathbf{a}) \cap \mathbf{s} = \emptyset$
- **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 $\mathbf{g}^+ \subseteq \mathbf{s} \wedge \mathbf{g}^- \cap \mathbf{s} = \emptyset$
    - $S_\mathbf{g} = \{\mathbf{s} \in S \mid \mathbf{s}$ 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)

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

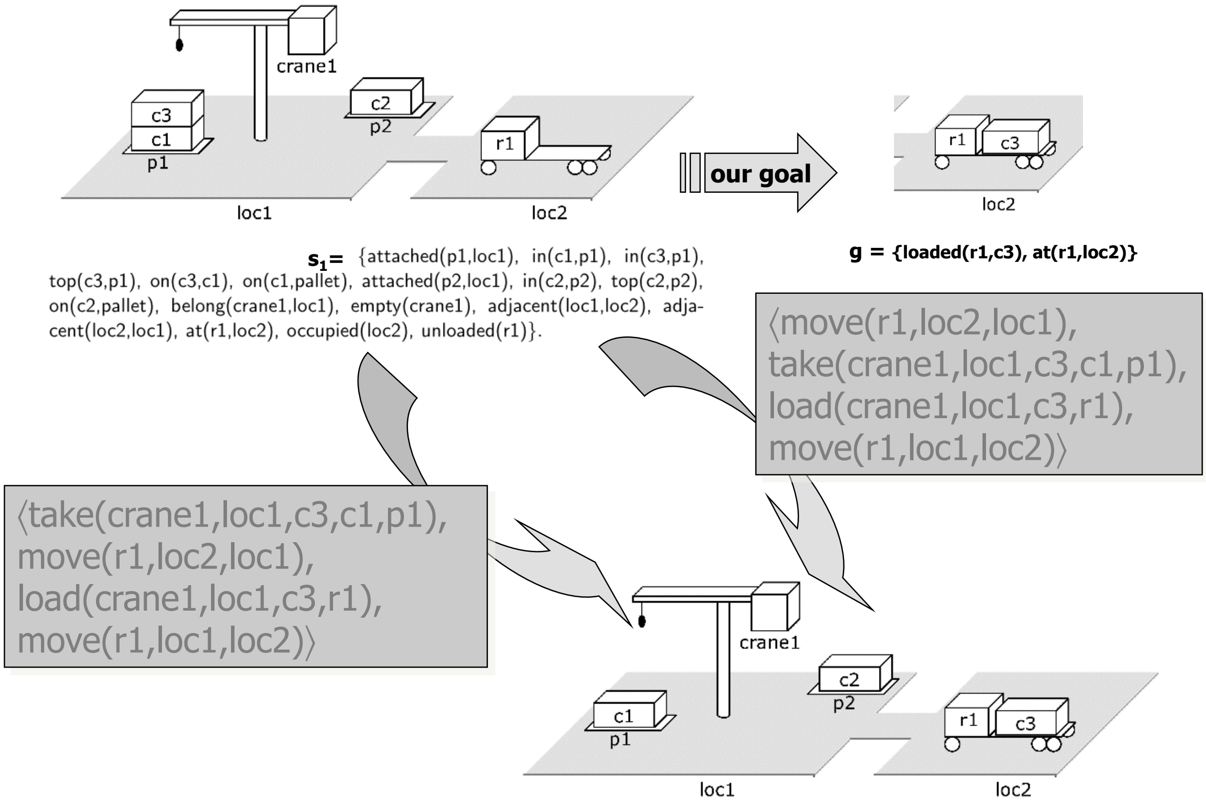
action effects are not in conflict with g:

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

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

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

Classical representation: an example plan



Classical representation: extensions

- **Syntactical extensions**
  - typed variables (each constant from a language has a type)
    - *Example: type robot: rob1, rob2, rob3*
  - existential quantification of goals (closed formula!)
    - *Example: ∃x,y (on(x,c1) ∧ on(y,c2))*
- **Conditional operators**
  - one operator encapsulates several “mini” operators, each with own preconditions and effects
  - all mini operators with satisfied preconditions are applied together
  - *Example: Switch on/off can be done using the same operator*
- **Disjunctive preconditions**
  - precondition can be described using any logical formula
  - *Example: a robot R goes from A to B either if A and B are connected via a road or R is a four-wheel-drive car*

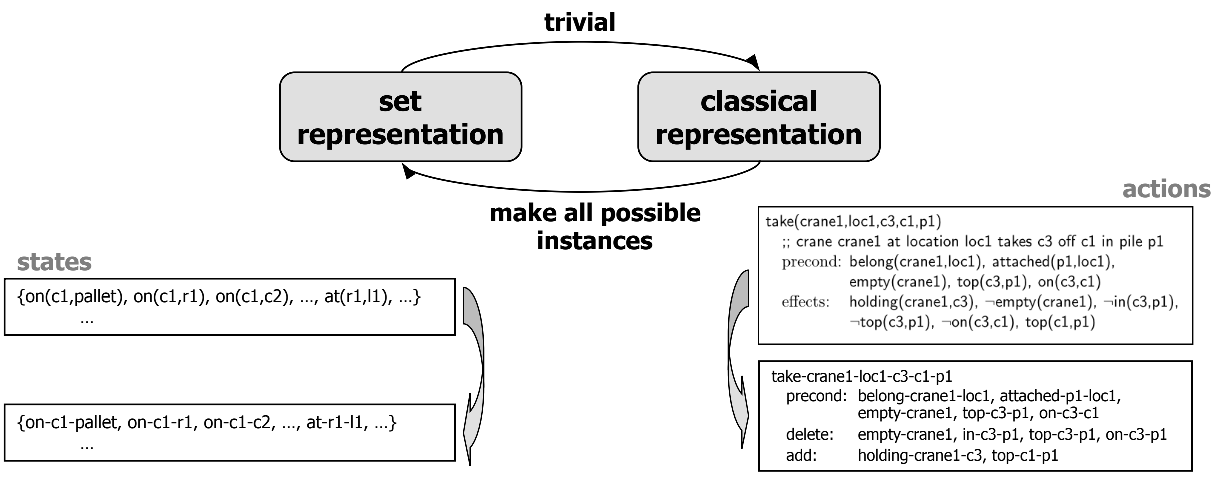
Classical representation: extensions

- **Attached procedures (to operators)**
  - they are used to verify more complex preconditions (for example numerical preconditions)
  - *Example: weight(c) ≤ maxweight(r)*
- **Axioms**
  - for automated inference of some facts
  - *Example: ∀ l,l‘ (adjacent(l,l‘) ⇔ adjacent(l‘,l))*
  - This must be done carefully for fluents:
    - *∀k (¬∃x holding(k,x) ⇒ empty(k))*
    - *∀k (∃x holding(k,x) ⇒ ¬empty(k))*

    Fluents can be split into two sets:
      - **Primal atoms**, that can be used both in preconditions and effects (for example, holding)
      - **Secondary atoms**, that can be used in preconditions only; cannot be used in effects (for example, empty)

Comparison of representations

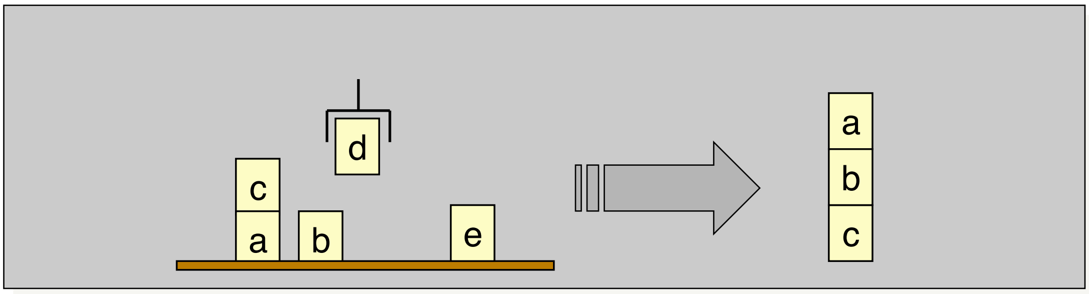
- **Expressive power** of both representations is **identical**.
- However, the translation from the classical representation to the set representation brings **exponential increase of size**.



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 computer hand that can hold at most one block

situation example



Department of Theoretical Computer Science and Mathematical Logic
bartak@ktiml.mff.cuni.cz