Artificial Intelligence 1

Roman Barták
Department of Theoretical Computer Science and Mathematical Logic

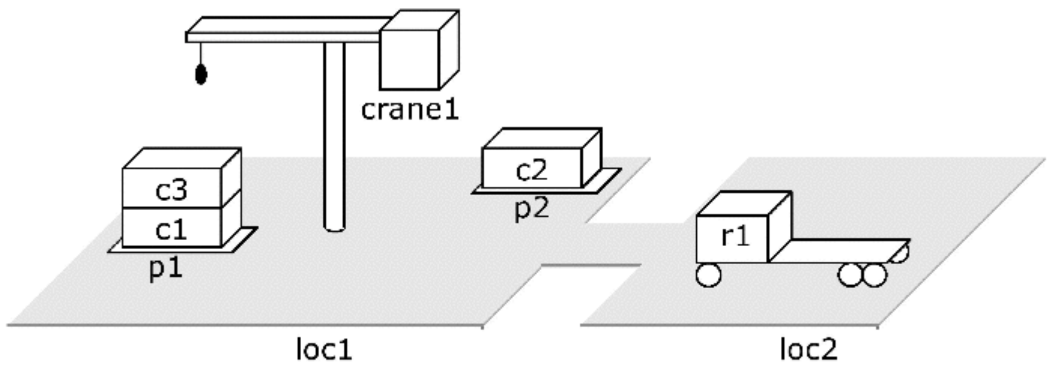
Automated Planning

Today we will explore techniques for **action planning** – how to find a sequence of actions to reach a given goal.

- **problem representation**
  - situation calculus (pure logical representation)
  - using sets of predicates (instead of formulas)
  - planning domain vs. planning problem
- **planning techniques**
  - state-space planning
    - forward and backward
  - plan-space planning
    - partially ordered plans

We can simplify the full FOL model into a so called **classical representation** of planning problems.

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!

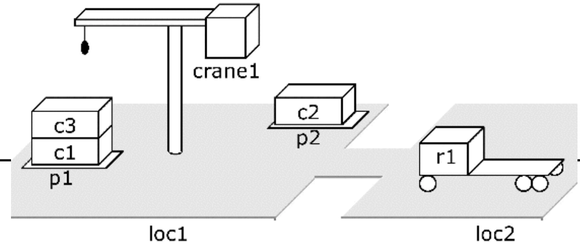
operator o is a triple (name(o), precond(o), effects(o))

- **name(o): name of the operator** in the form n(x_1,…,x_k)
  - n: a symbol of the operator (a unique name for each operator)
  - x_1,…,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)

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 (let S be a set of literals)**:**

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

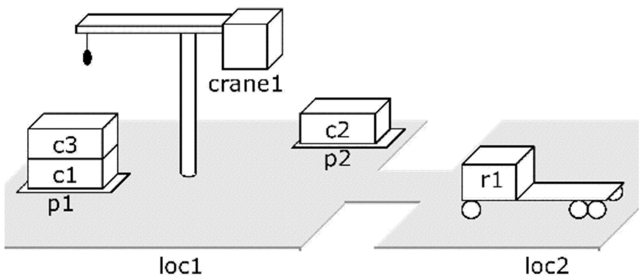
Action **a** is **applicable** to state **s** if and only if

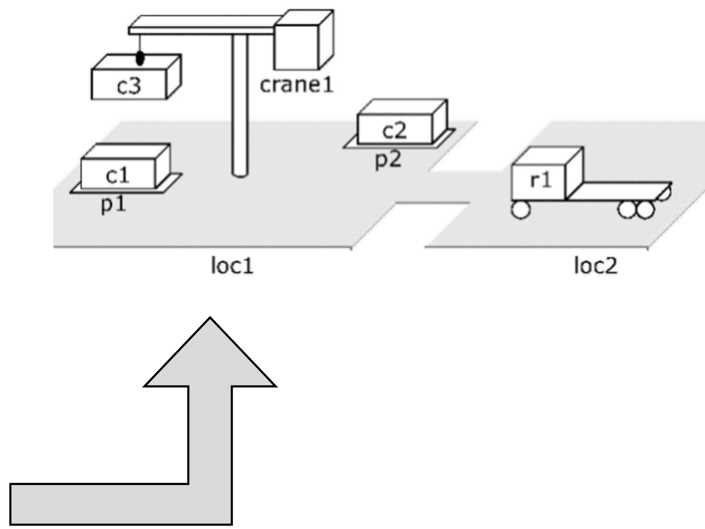
$$\text{precond}^+(\mathbf{a}) \subseteq \mathbf{s} \;\wedge\; \text{precond}^-(\mathbf{a}) \cap \mathbf{s} = \varnothing$$

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** $A = \{\text{all instantiated operators from } O \text{ over } L\}$
  - 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

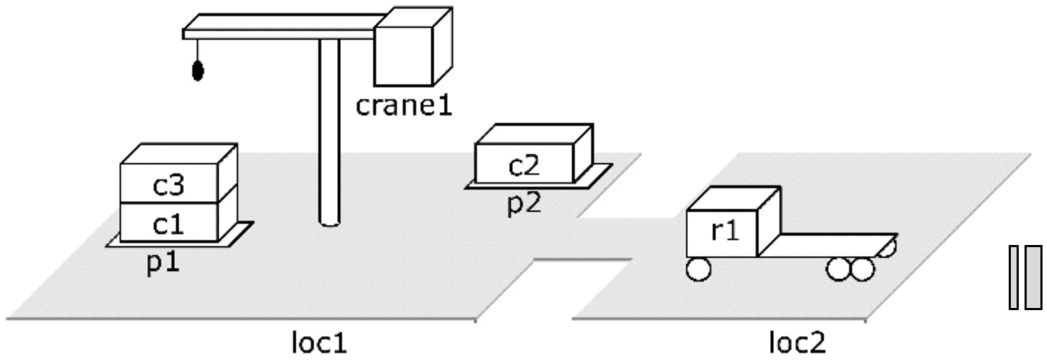
Plan is a sequence of actions $\langle a_1,a_2,\ldots,a_k\rangle$.

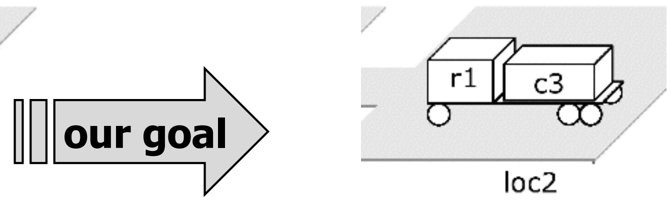
Plan $\pi = \langle a_1,a_2,\ldots,a_k\rangle$ is a **solution plan** for problem P iff $\gamma^*(s_0,\pi)$ satisfies the goal condition g.

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

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

Classical representation: example plan



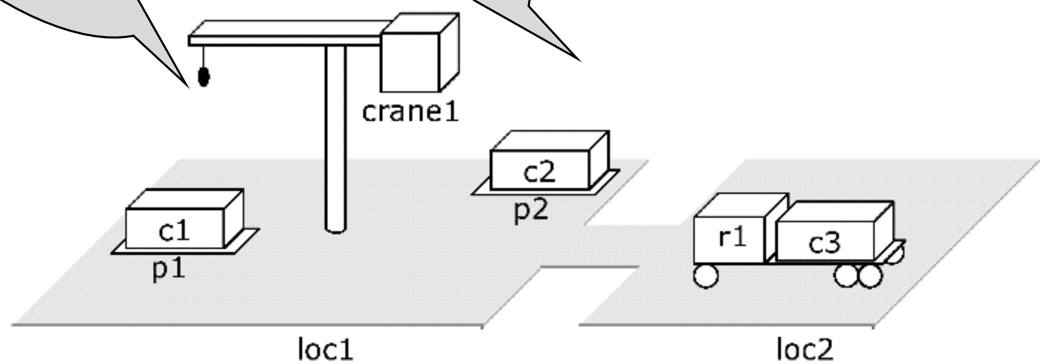


s_1 = {attached(p1,loc1), in(c1,p1), in(c3,p1), top(c3,p1), on(c3,c1), on(c1,pallet), attached(p2,loc1), in(c2,p2), top(c2,p2), on(c2,pallet), belong(crane1,loc1), empty(crane1), adjacent(loc1,loc2), adjacent(loc2,loc1), at(r1,loc2), occupied(loc2), unloaded(r1)}.

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

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

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



The search space corresponds to the state space of the planning problem.

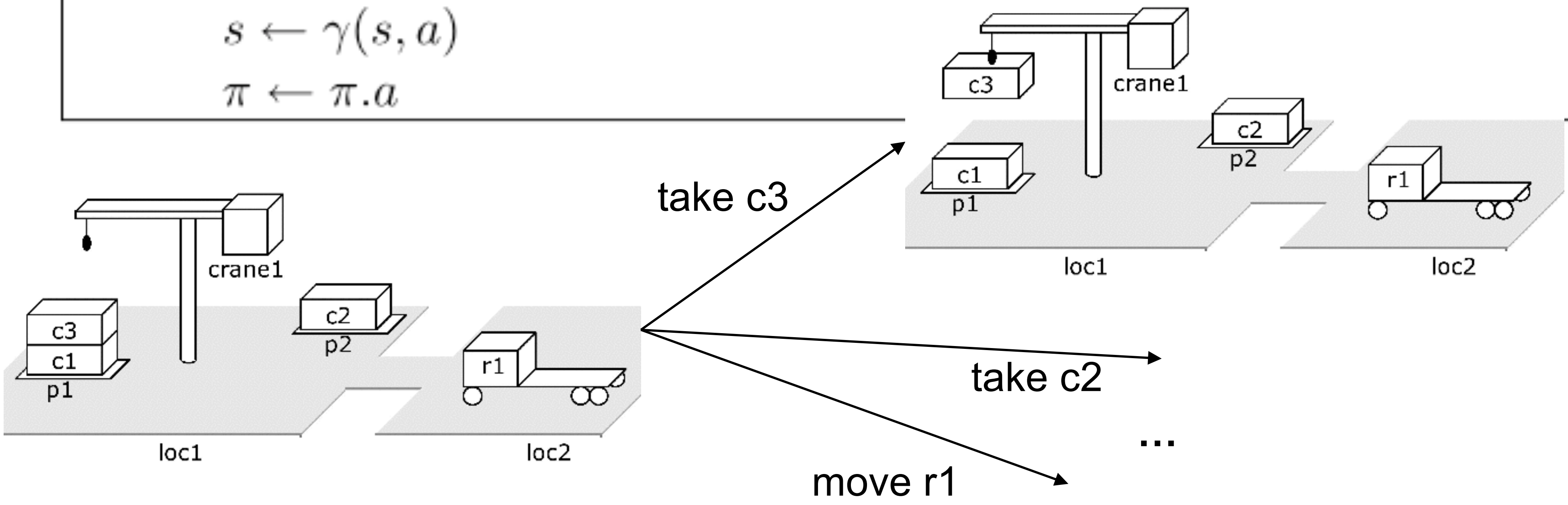
- search nodes correspond to world states
- arcs correspond to state transitions by means of actions
- the task is to find a path from the initial state to some goal state

Basic approaches

- forward search (progression)
  - start in the initial state and apply actions until reaching a goal state
- backward search (regression)
  - start with the goal and apply actions in the reverse order until a subgoal satisfying the initial state is reached
  - lifting (actions are only partially instantiated)

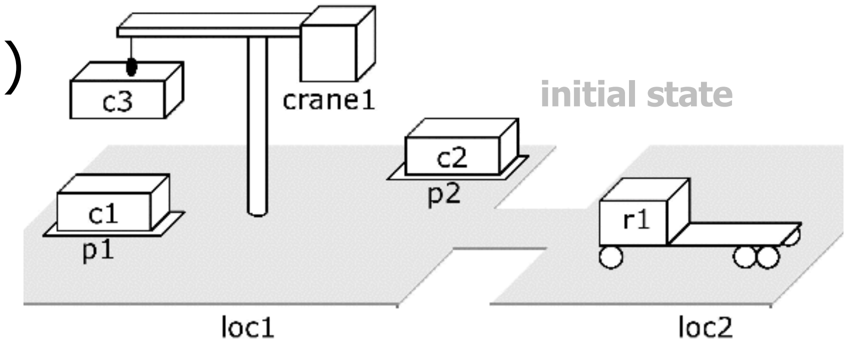
Forward planning: algorithm

Forward-search(O, s_0, g)
    $s \leftarrow s_0$
    $\pi \leftarrow$ the empty plan
    loop
        if s satisfies g then return π
        $E \leftarrow \{a | a$ is a ground instance of an operator in *O*,
                and $\mathrm{precond}(a)$ is true in $s\}$
        if $E = \emptyset$ then return failure
        nondeterministically choose an action $a \in E$
        $s \leftarrow \gamma(s, a)$
        $\pi \leftarrow \pi.a$



Forward planning: an example

{belong(crane1,loc1), adjacent(loc2,loc1)
holding(crane1,c3), unloaded(r1),
at(r1,loc2), ¬occupied(loc1),
occupied(loc2),…}



move(r1,loc2,loc1)

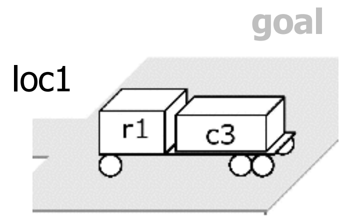
move(r, l, m)
;; robot r moves from location l to location m
precond: adjacent(l, m), at(r, l), ¬occupied(m)
effects: at(r, m), occupied(m), ¬occupied(l), ¬at(r, l)

{belong(crane1,loc1),
adjacent(loc2,loc1), holding(crane1,c3), unloaded(r1),
at(r1,loc1), occupied(loc1), …}

load(crane1,loc1,c3,r1)

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)

{belong(crane1,loc1), adjacent(loc2,loc1),
empty(crane1), loaded(r1,c3),
at(r1,loc1), occupied(loc1), …}



Goal = {at(r1,loc1),loaded(r1,c3)}

Start with a goal (not a goal state as there might be more goal states) and through sub-goals try to reach the initial state.

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

- action **a** contributes to goal **g**: **g** $\cap$ effects(**a**) $\neq \emptyset$
- effects of action **a** are not conflicting goal **g**:
  - $g^- \cap$ effects$^+$(a) = $\emptyset$
  - $g^+ \cap$ effects$^-$(a) = $\emptyset$

A **regression set** of the goal **g** for (relevant) action **a** is
γ^{-1}(g,a) = (g - effects(a)) $\cup$ precond(a)

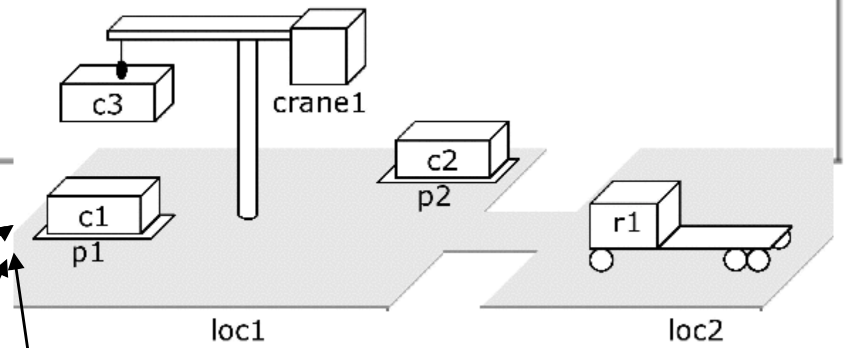
Example:

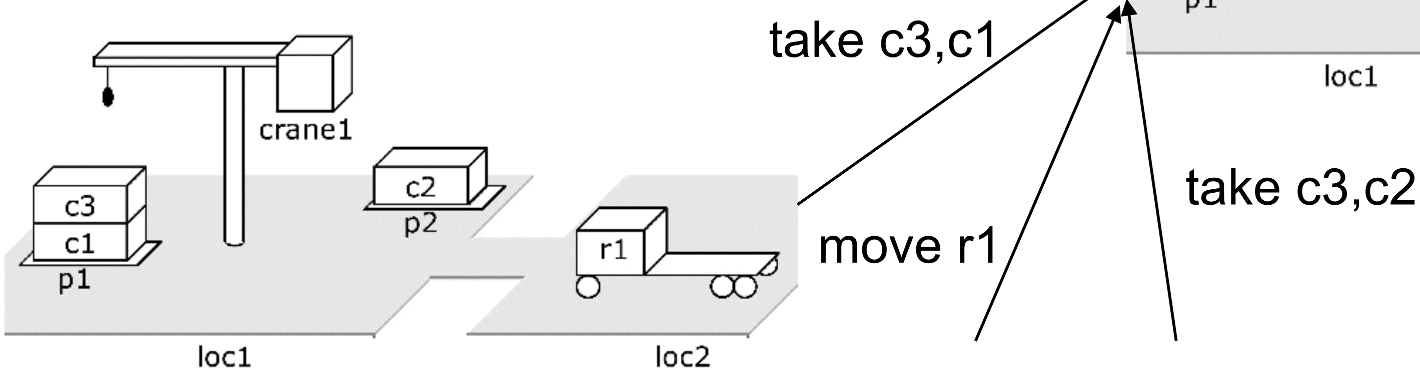
goal: **{on(a,b), on(b,c)}**
action **stack(a,b)** is relevant
by backward application of the action we get a new goal:
{holding(a), clear(b), on(b,c)}

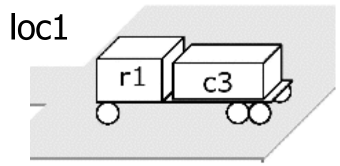
stack(*x*,*y*)
Precond: holding(*x*), clear(*y*)
Effects: $\neg$ holding(*x*), $\neg$ clear(*y*),
on(*x*,*y*), clear(*x*), handempty

Backward planning: algorithm

Backward-search(O, s_0, g)
  $\pi \leftarrow$ the empty plan
  loop
    if s_0 satisfies g then return π
    $A \leftarrow \{a | a$ is a ground instance of an operator in O
        and $\gamma^{-1}(g, a)$ is defined$\}$
    if $A = \emptyset$ then return failure
    nondeterministically choose an action $a \in A$
    $\pi \leftarrow a.\pi$
    $g \leftarrow \gamma^{-1}(g, a)$







Goal = {at(r1,loc1),loaded(r1,c3)}

load(crane1,loc1,c3,r1)

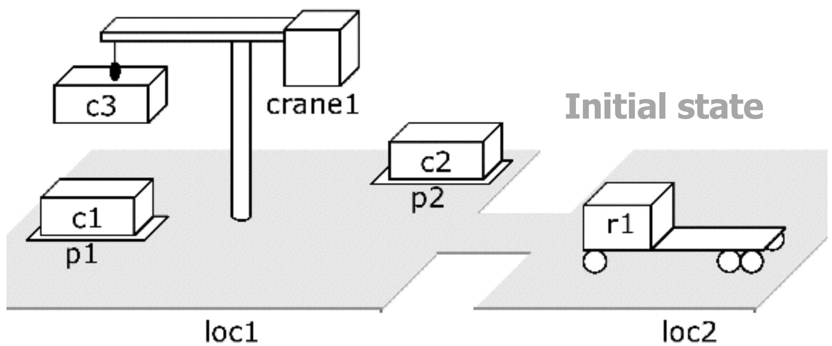
load(k, l, c, r)
;; crane k at location l loads container c onto robot r
precond: belong(k, l), holding(k, c), at(r, l), unloaded(r)
effects: empty(k), $\neg$ holding(k, c), loaded(r, c), $\neg$ unloaded(r)

{at(r1,loc1), belong(crane1,loc1),
holding(crane1,c3), unloaded(r1)}

move(r1,loc2,loc1)

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)

{belong(crane1,loc1), holding(crane1,c3),
unloaded(r1),
adjacent(loc2,loc1),
at(r1,loc2),
¬ occupied(loc1)}



Lifted-backward-search(O, s_0, g)
  $\pi \leftarrow$ the empty plan
  loop
    if s_0 satisfies g then return π
    $A \leftarrow \{(o, \theta) | o$ is a standardization of an operator in O,
        θ is an mgu for an atom of g and an atom of effects (o),
        and $\gamma^{-1}(\theta(g), \theta(o))$ is defined$\}$
    if $A = \emptyset$ then return failure
    nondeterministically choose a pair $(o, \theta) \in A$
    $\pi \leftarrow$ the concatenation of $\theta(o)$ and $\theta(\pi)$
    $g \leftarrow \gamma^{-1}(\theta(g), \theta(o))$

Notes:

- standardization = a copy with fresh variables
- mgu = most general unifier
- by using the variables we can decrease the branching factor but the trade off is more complicated loop check

The principle of plan space planning is similar to backward planning:

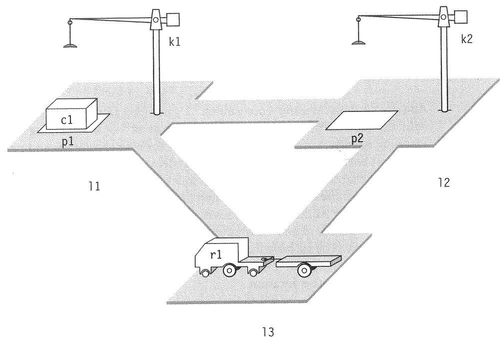
- start from an **"empty" plan** containing just the description of initial state and goal
- **add other actions** to satisfy not yet covered (open) goals
- if necessary **add other relations** between actions in the plan

Planning is realised as **repairing flaws in a partial plan**

- go from one partial plan to another partial plan until a complete plan is found

Assume a partial plan with the following two actions:

- take(k1,c1,p1,l1)
- load(k1,c1,r1,l1)



Possible modifications of the plan:

- **adding a new action**
  - to apply action **load**, robot r1 must be at location l1
  - action **move**(r1,l,l1) moves robot r1 to location l1 from some location l
- **binding the variables**
  - action **move** is used for the right robot and the right location
- **ordering some actions**
  - the robot must **move** to the location before the action **load** can be used
  - the order with respect to action **take** is not relevant
- **adding a causal relation**
  - new action is added to move the robot to a given location that is a precondition of another action
  - the causal relation between **move** and **load** ensures that no other action between them moves the robot to another location

The initial state and the goal are encoded using two **special actions** in the initial partial plan:

- **Action a_0 represents the initial state** in such a way that atoms from the initial state define effects of the action and there are no preconditions. This action will be before all other actions in the partial plan.
- **Action a_∞ represents the goal** in a similar way – atoms from the goal define the precondition of that action and there is no effect. This action will be after all other actions.

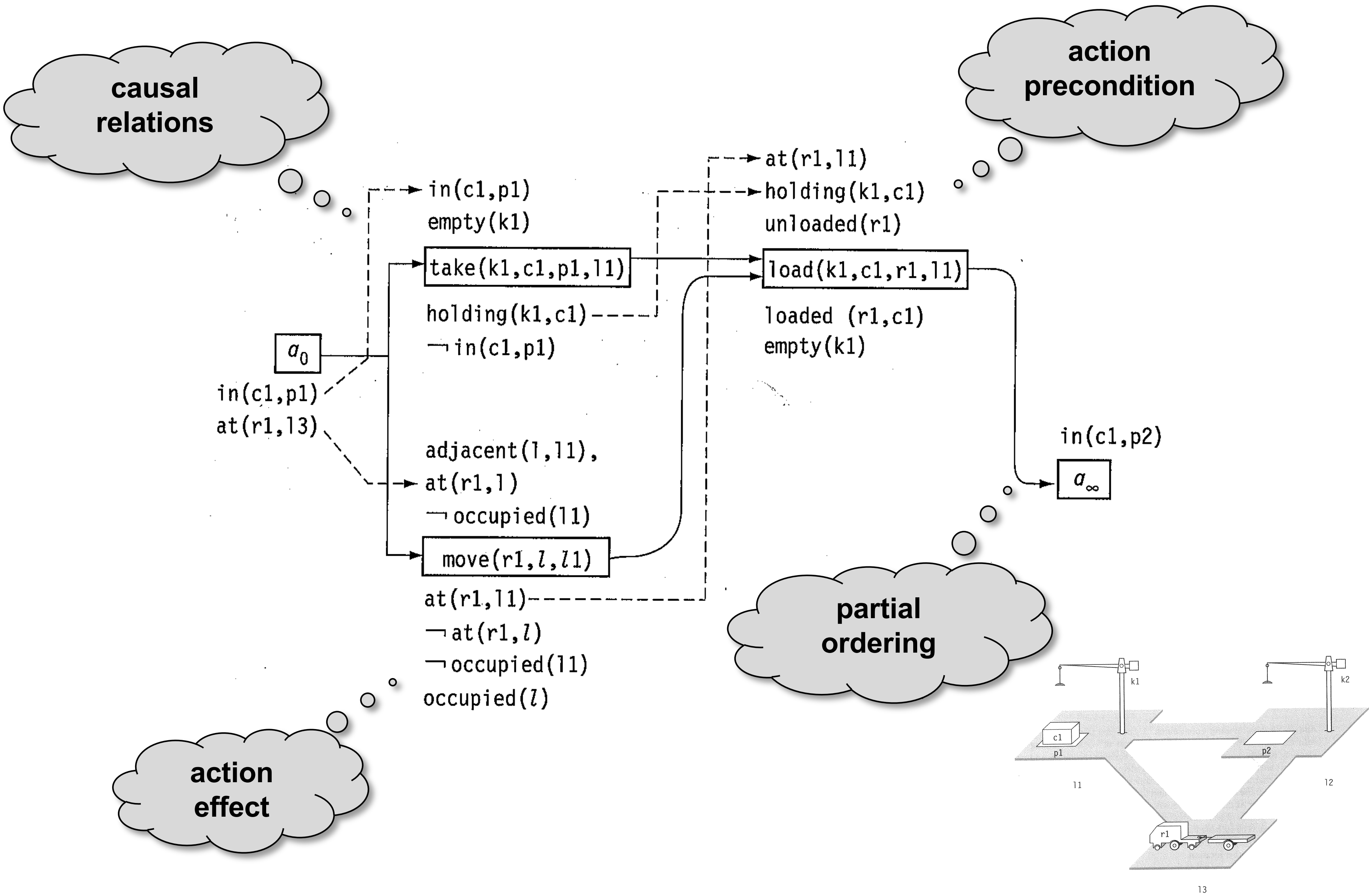
Planning is realised by **repairing flaws** in the partial plan.

The search nodes correspond to partial plans.

A partial plan Π is a tuple (A,<,B,L), where

- A is a set of partially instantiated planning operators $\{a_1,\ldots,a_k\}$
- $<$ is a partial order on A ($a_i<a_j$)
- B is set of constraints in the form $x=y$, $x\neq y$ or $x\in D_i$
- L is a set of causal relations ($a_i \rightarrow^p a_j$)
  - a_i,a_j are ordered actions $a_i<a_j$
  - p is a literal that is effect of a_i and precondition of a_j
  - B contains relations that bind the corresponding variables in p

Partial plan: an example



Open goal is an example of a **flaw**.

This is a precondition **p** of some operator **b** in the partial plan such that no action was decided to satisfy this precondition (there is no causal relation $a_i \rightarrow^p b$).

The open goal p of action b can be resolved by:

– finding an operator **a** (either present in the partial plan or a new one) that can give **p** (**p** is among the effects of **a** and **a** can be before **b**)

– binding the variables from **p**

– adding a causal relation $\mathbf{a} \rightarrow^{\mathbf{p}} \mathbf{b}$

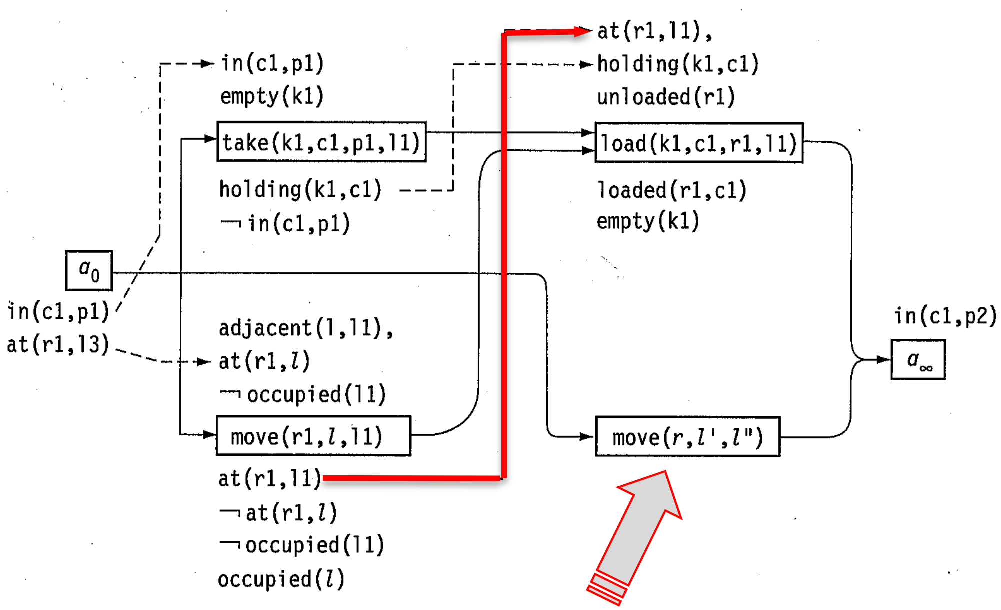
Threat is another example of **flaw**.

It is an action that can influence existing causal relation.

- Let $a_i \rightarrow^p a_j$ be a causal relation and action **b** has among its effects a literal unifiable with the negation of **p** and action **b** can be between actions a_i and a_j. Then **b** is threat for that causal relation.

We can **remove the threat** by one of the following ways:

- ordering **b** before $\mathbf{a_i}$
- ordering **b** after $\mathbf{a_j}$
- binding variables in **b** in such a way that **p** does not bind with the negation of **p**



Partial plan Π = (A,<,B,L) is a **solution plan** for the problem P = (Σ,s_0,g) if:

- partial ordering < and constraints B are globally consistent
  - there are no cycles in the partial ordering
  - we can assign variables in such a way that constraints from B hold
- Any linearly ordered sequence of fully instantiated actions from A satisfying < and B goes from s_0 to a state satisfying g.

Hmm, but this definition **does not say how** to verify that a partial plan is a solution plan!

Claim: Partial plan Π = (A,<,B,L) is a solution plan if:

- there are no flaws (no open goals and no threats)
- partial ordering < and constraints B are globally consistent

PSP = Plan-Space Planning

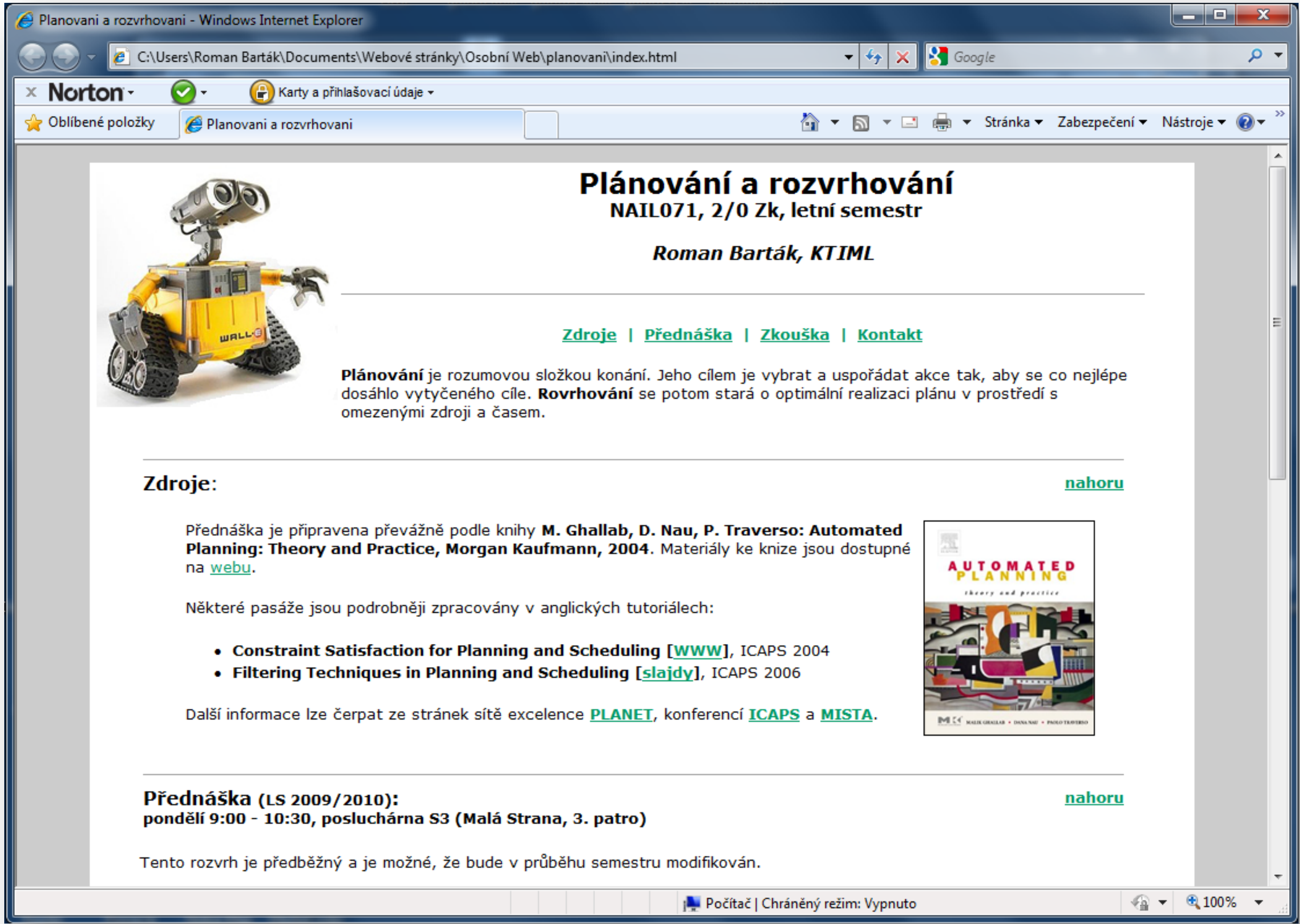
PSP(π)
    $flaws \leftarrow$ OpenGoals(π) $\cup$ Threats(π)
    if $flaws = \emptyset$ then return(π)
    select any flaw $\phi \in flaws$
    $resolvers \leftarrow$ Resolve(ϕ, π)
    if $resolvers = \emptyset$ then return(failure)
    nondeterministically choose a resolver $\rho \in resolvers$
    $\pi' \leftarrow$ Refine(ρ, π)
    return(PSP(π'))
end

Notes:

- The selection of flaw is deterministic (all flaws must be resolved).
- The resolvent is selected non-deterministically (search in case of failure).

Course **Planning and scheduling**

- http://ktiml.mff.cuni.cz/~bartak/planovani/



An **agent view** of Artificial Intelligence

- an agent is an entity perceiving environment and acting upon it
- a **rational agent** maximizes expected performance

Problem solving with simple state space

- **search** techniques
- exploiting extra information –> heuristic search **A***
- factored states –> **constraint satisfaction**
- more agents –> **adversarial search** (games)

Knowledge representation

- propositional and first-order **logic**
- **inference** procedures

Automated planning

- situation calculus
- state-space and plan-space planning

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