Introduction to Artificial Intelligence

Roman Barták
Department of Theoretical Computer Science and Mathematical Logic

So far, we expressed static information only. How can we represent **information changing with time**, for example location of agent?

Using time-annotated propositional variables (**fluents**):

$\mathbf{L^t_{x,y}}$ agent is at cell (x,y) at time t.

Observation model:

Connects observation, for example about breeze, with information in the world model.

$$\mathbf{L^t_{x,y} \Rightarrow (Breeze^t \Leftrightarrow B_{x,y})}$$

As we are going to ask queries whether a square is OK to move into, that is, the square contains no pit nor live Wumpus, we can introduce special propositional variables:

$$\mathbf{OK^t_{x,y} \Leftrightarrow \quad (\neg P_{x,y} \wedge \neg (W_{x,y} \wedge WumpusAlife^t)}$$

Transition model:

Describes evolution of world after applying actions, for example using **effect axioms**:

$$\mathbf{L^t_{x,y} \wedge FacingEast^t \wedge Forward^t \Rightarrow (L^{t+1}_{x+1,y} \wedge \neg L^{t+1}_{x,y})}$$

Effect axioms do not say what is not changed by the action (**frame problem**)

For example, one cannot deduce anything about validity of $HaveArrow^1$ after performing action $Forward^0$.

We can use **frame axioms** explicitly asserting all the propositions that remain the same:

$$\mathbf{Forward^t \Rightarrow (HaveArrow^t \Leftrightarrow HaveArrow^{t+1})}$$

This gives many frame axioms, which makes reasoning inefficient.

Instead, we can write axioms about fluents rather than about actions, for example, **successor-state axiom** defining the truth value of fluent at time (t+1) in terms of fluents and actions at time t.

$$\mathbf{F^{t+1} \Leftrightarrow ActionCausesF^t \vee (F^t \wedge \neg ActionCausesNotF^t)}$$

$$\mathbf{HaveArrow^{t+1} \Leftrightarrow (HaveArrow^t \wedge \neg Shoot^t)}$$

Hybrid Wumpus agent

```
function HYBRID-WUMPUS-AGENT(percept) returns an action
  inputs: percept, a list, [stench,breeze,glitter,bump,scream]
  persistent: KB, a knowledge base, initially the atemporal "wumpus physics"
              t, a counter, initially 0, indicating time
              plan, an action sequence, initially empty

  TELL(KB,MAKE-PERCEPT-SENTENCE(percept,t))
  TELL the KB the temporal "physics" sentences for time t
  safe ← {[x, y] : ASK(KB, OK^t_{x,y}) = true}
  if ASK(KB, Glitter^t) = true then
    plan ← [Grab] + PLAN-ROUTE(current,{[1,1]},safe) + [Climb]
  if plan is empty then
    unvisited ← {[x, y] : ASK(KB, L^{t'}_{x,y}) = false for all t' ≤ t}
    plan ← PLAN-ROUTE(current,unvisited ∩ safe,safe)
  if plan is empty and ASK(KB, HaveArrow^t) = true then
    possible_wumpus ← {[x, y] : ASK(KB, ¬ W_{x,y}) = false}
    plan ← PLAN-SHOT(current,possible_wumpus,safe)
  if plan is empty then   // no choice but to take a risk
    not_unsafe ← {[x, y] : ASK(KB, ¬ OK^t_{x,y}) = false}
    plan ← PLAN-ROUTE(current,unvisited ∩ not_unsafe,safe)
  if plan is empty then
    plan ← PLAN-ROUTE(current,{[1, 1]},safe) + [Climb]
  action ← POP(plan)
  TELL(KB,MAKE-ACTION-SENTENCE(action,t))
  t ← t + 1
  return action
```

```
function PLAN-ROUTE(current,goals,allowed) returns an action sequence
  inputs: current, the agent's current position
          goals, a set of squares; try to plan a route to one of them
          allowed, a set of squares that can form part of the route

  problem ← ROUTE-PROBLEM(current,goals,allowed)
  return A*-GRAPH-SEARCH(problem)
```

Hybrid agent combines various approaches to problem solving.

Logic is used to deduce state of the world based on percepts.

Condition-action rules are used to prioritize goals.

Plan-Shot uses Plan-Route to plan a sequence of actions that will line up with this shot.

A* search is used for route planning. The path from the current cell to some goal cell through allowed cells is looked for.

Hybrid planner uses propositional inference to determine properties of states (safe squares), but uses A* search to make plans.

Can we use logical inference to find a plan completely?

Idea:

1. Construct a **propositional sentence** that includes
   - **Init^0**: assertions describing the initial state
   - **Transition^1,...,Transition^t**: successor-state axioms for all possible actions
   - **Goal^t**: assertions that the goal is achieved at time **t**
     example: (**HaveGold^t** $\wedge$ **ClimbedOut^t**)
2. Present the sentence to a **SAT solver**
   - If a model is found (formula is satisfiable), then the goal is achievable.
   - If the sentence is unsatisfiable, then no plan (of length **t**) exists to achieve the goal.
3. Assuming a model is found, **extract** the **plan** from the model
   - Plan consists of actions for which the corresponding variables are assigned true.

How do we know the number of steps to achieve the goal?

We can try to find plans with incrementally increased length.

SATPlan always finds the shortest plan, if one exists

```
function SATPLAN(init, transition, goal, T_max) returns solution or failure
  inputs: init, transition, goal, constitute a description of the problem
          T_max, an upper limit for plan length

  for t = 0 to T_max do
    cnf ← TRANSLATE-TO-SAT(init, transition, goal, t)
    model ← SAT-SOLVER(cnf)
    if model is not null then
      return EXTRACT-SOLUTION(model)
  return failure
```

Note: we need to know completely the initial state (the values of all propositional variables for the initial state are set).

For example, $\mathbf{L^0_{1,1}}$ is set to **true** to describe the initial location of agent, but we also need to set $\mathbf{L^0_{1,2}}$ etc. to **false** to say where the agent is not present at time 0.

We have effect (and frame) axioms or successor-state axioms to describe how the **state changes**.

But we also need to describe **when an action is applicable**, for example shooting is not possible, if agent has no arrow.

Preconditions axioms describe when action is applicable

$$\textbf{Shoot}^t \Rightarrow \textbf{HaveArrow}^t$$

Can the plan obtained from the model contain **multiple simultaneous actions**?

Yes, for example action variables $\textbf{Forward}^0$ and $\textbf{Shoot}^0$ can be both true.

To eliminate parallel actions, we introduce **action exclusion axioms**:

$$\forall t\ \forall i \neq j: (\neg A^t_i \vee \neg A^t_j)$$

We may allow multiple simultaneous actions that do no interfere with each other (such pairs of actions are not part of action exclusion axioms)

Interference means that actions have no conflicting effects and do not destroy (or set) preconditions of each other.

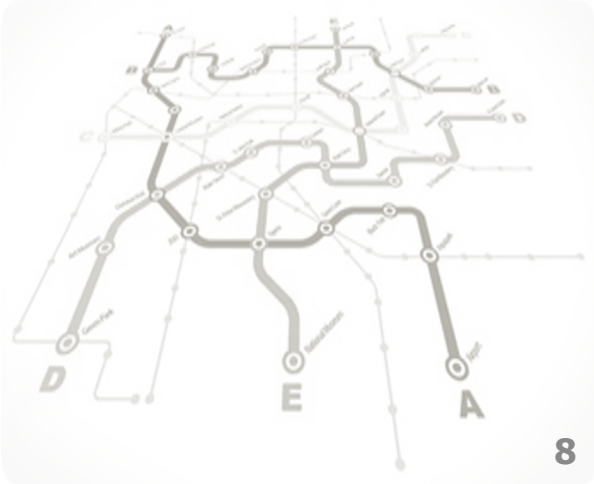
Planning is about finding actions to achieve agent's goal.

We can **find plans by search** (problem solving).

- But it deals with atomic representation of states and thus needs good domain-specific heuristics.

We can do **planning by propositional inference**.

- Using domain-independent heuristics based on the logical structure of the problem.
- However, it may be swamped when there are many actions and states.
- *For example,*
  in the Wumpus world, action Forward has to be repeated for all four agent orientations, T time steps and n^2 locations.

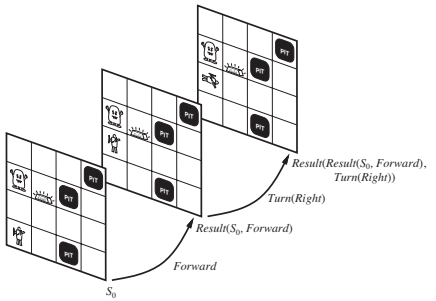


Can we get rid of writing the same axioms repeatedly for different times and for similar actions?

Can we do planning using logical inference only (no wrapper)?

Using **first-order logic** helps there.

Situation calculus:



- uses special terms to describe **situations** (give names to states)
  - $\mathbf{s_0}$: name of initial situation
  - **Results(s,a)**: name of situation after applying action **a** to situation **s**
- **states** are described using relations among objects,
  Example: **at(robot, location)**
  - if the validity of relation changes in states, then we add extra argument describing at which situation the relation holds (**fluents**): **at(robot,location,s)**
  - if the validity does not change, then no extra argument is needed (**rigid/eternal predicates**): **connected(loc1,loc2)**
- action's preconditions are described using a **possibility axiom**
  - **Phi(s)** $\Rightarrow$ **Poss(a,s),** where **Phi(s)** is a formula describing the preconditions
  - *Example*: **at(a,l,s)** $\wedge$ **connected(l,l')** $\Rightarrow$ **Poss(go(a,l,l'), s)**
- properties of next state are described using **successor-state axioms** for each fluent
  - **Poss(a,s)** $\Rightarrow$ **(F(Result(s,a))** $\Leftrightarrow$ **F is made true by a** $\vee$ **(F(s)** $\wedge$ **F is not made false by a)**
- **planning** is then realized by asking whether there exists a situation such that the goal condition is true there
  - $\exists$ **s: HaveGold(a,s)** $\wedge$ **ClimbedOut(a,s)**

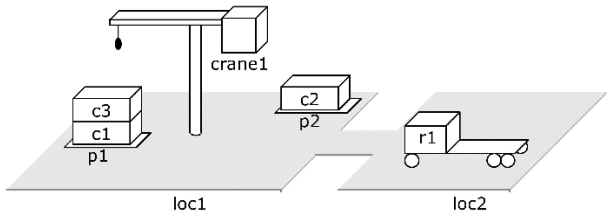
Uses **factored representation of states** (state is represented using a vector of variables) and **actions schemata** (operators) to describe capabilities of agent (how to change the world).

Planning is then realized via search (in the state space) with domain-independent heuristics.

Example (size of state space):

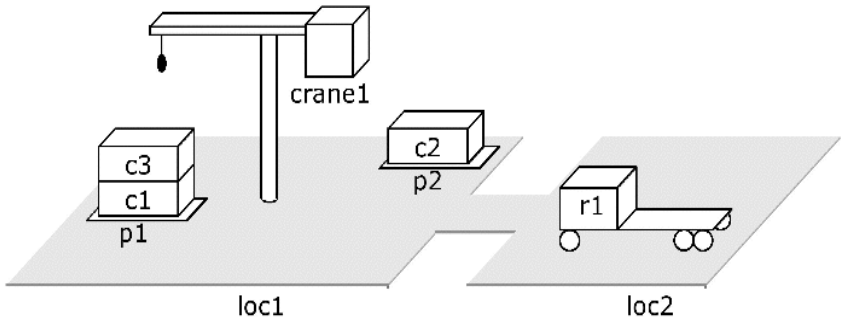
5 locations, 3 piles per location, 100 containers, 3 robots

↳ **10^{277} states**



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

State *s* 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).

Goal *g* is a set of instantiated literals:

state **s** satisfies the goal condition **g** if and only if

$\mathbf{g}^+ \subseteq \mathbf{s} \ \wedge \ \mathbf{g}^- \cap \mathbf{s} = \varnothing$

Action schema (operator) describes the action without specifying the particular objects to which the action applies.

Lifted representation (lifts from propositional logic to a restricted subset of first-order logic).

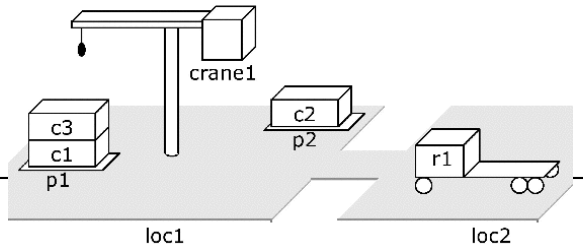
Operator o is a triple (name(o), precond(o), effects(o))

- **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)

An action is a fully instantiated operator

– substitute constants to variables



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), ¬ empty(k), ¬ in(c, p), ¬ top(c, p), ¬ on(c, d), top(d, p)

operator

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)

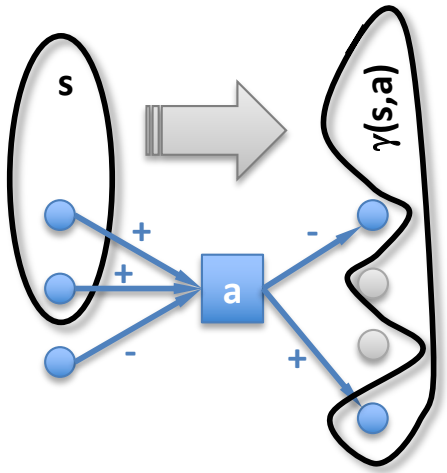
action

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

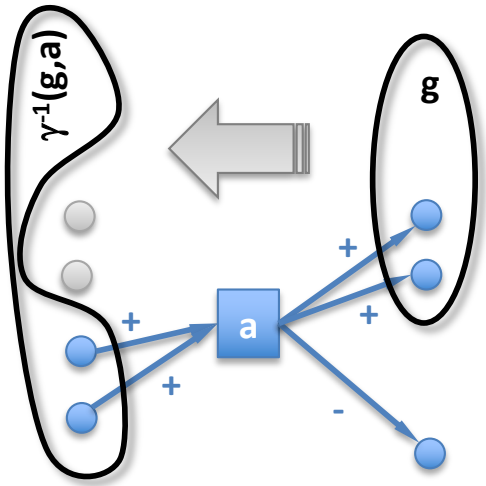
$\gamma(\mathbf{s},\mathbf{a}) = (\mathbf{s} - \text{effects}^-(\mathbf{a})) \cup \text{effects}^+(\mathbf{a})$

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

action contributes to g: $g \cap \text{effects}(a) \neq \varnothing$

action effects are not in conflict with g:

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



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

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

Description of operators **O** is usually called a **domain model** (it also implicitly includes description of predicate symbols in states)**.**

Planning problem P is a triple **(O,s_0,g),** where:

- **O** is a planning domain model (description of operators)
- **s_0** is an initial state (s_0 provides the particular constants – objects – used in actions)
- **g** is a goal condition (a set of instantiated literals)

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.

Example: Blockworld problem

Constants

- blocks: a,b,c

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

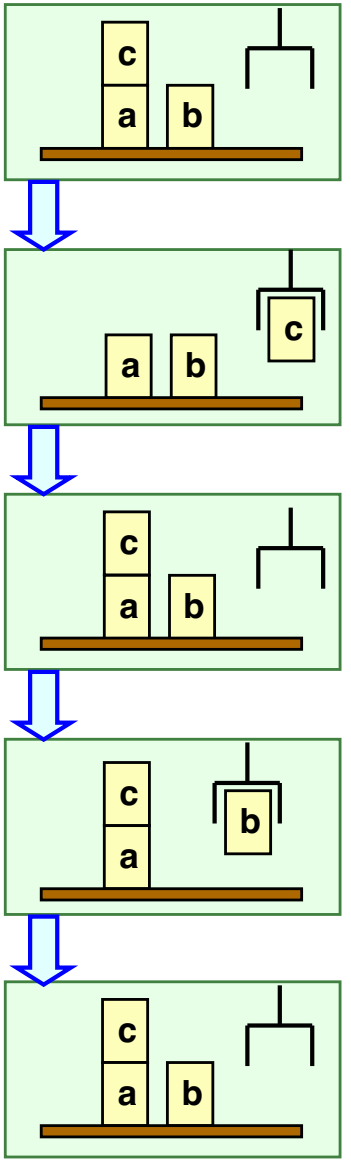
Operators

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



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 (progression) planning goes from the initial state to a goal state.

Non-determinism is implemented by a **search algorithm**.

- Uninformed search can only solve small problems.
- We need **informed search** (A*) with good heuristic to guide the search algorithm.

```
Backward-search(O, s_0, g)
    π ← the empty plan
    loop
        if s_0 satisfies g then return π
        A ← {a | a is a ground instance of an operator in O
                    and γ^{-1}(g, a) is defined}
        if A = ∅ then return failure
        nondeterministically choose an action a ∈ A
        π ← a.π
        g ← γ^{-1}(g, a)
```

Backward (regression) planning starts with the goal and applies the actions backward until it reaches the initial state.

Only **relevant actions** are explored

=> **smaller branching factor** in comparison to forward search.

Uses state sets rather than individual states

=> **harder** to come up with good heuristics.

We need heuristics to guide the search algorithm.

A **heuristic function** *h(s)* estimates the distance from a state *s* to a goal.

Admissible heuristic (one that does not overestimate) allows usage of A* algorithm to find optimal solutions.

Admissible heuristic can be derived from **solving a relaxed problem** (problem, where some constraints are removed, to make the problem easier to solve).

Ignore action preconditions

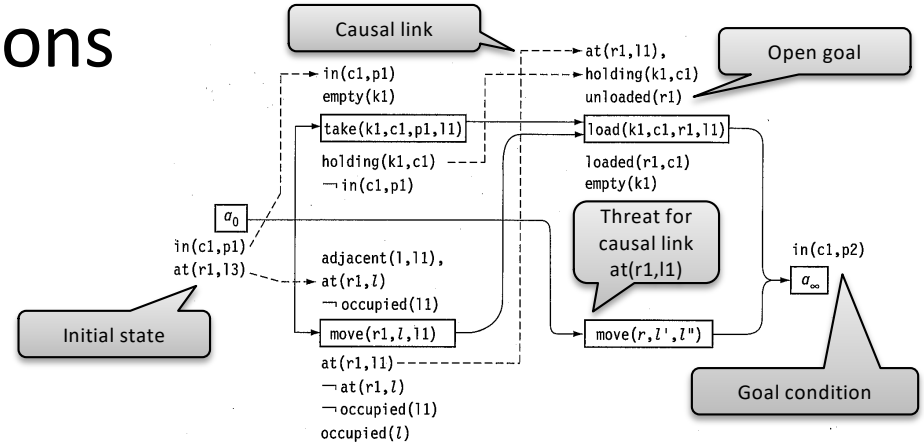
- all actions are always applicable
- find a set of actions of minimal size to cover the goal conditions (set-cover problem)
- *note*: we also ignore that some actions undo effects of other actions

Ignore negative effects

- actions do not destroy preconditions of other actions so the set of valid propositions in the state is monotonically enlarging
- we can progressively add all applicable actions until all goal conditions are covered (smallest set of actions leading to the goal is found then)

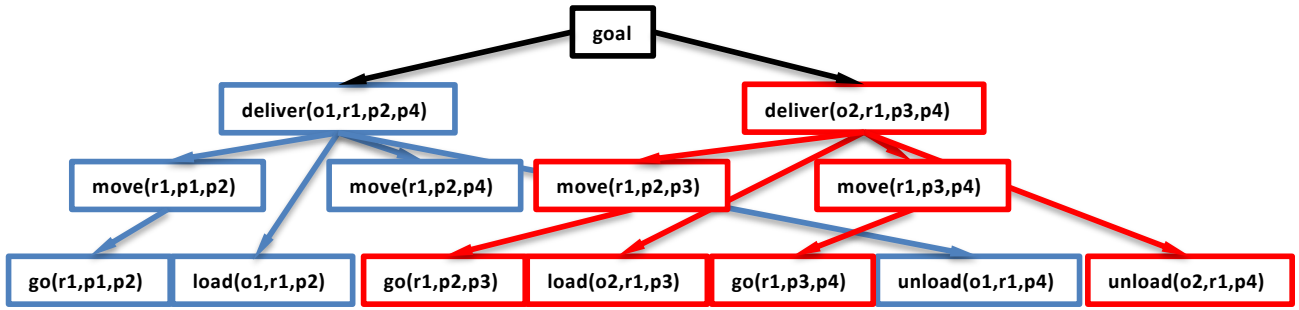
Plan-space planning (partial order planning)

Planning is realized by refining initial (empty) plan by adding actions and relations to fulfill **open goals** and to resolve **causal threats**.



Hierarchical planning

Planning is realized by **decomposing tasks** to sub-tasks until primitive tasks (actions) are obtained.

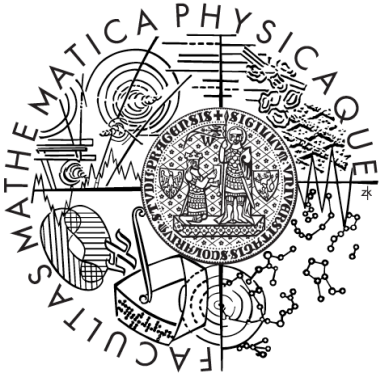


Automated planning – devising a plan of action to achieve agent's goals – is one of critical parts of AI.

- **Planning domain model** describes planning operators (capabilities of an agent) via preconditions and effects.
- **Planning task** is to find a sequence of actions from the current state to a state satisfying the goal condition.
- Planning is (frequently) realized by **state-space search** in forward direction (**progression**) or backward direction (**regression**).
- **Domain-independent heuristics** can be used thanks to factored representation of states.

For more information
course **Planning and Scheduling**

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



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