Artificial Intelligence 1

Roman Barták
Department of Theoretical Computer Science and Mathematical Logic

Automated Planning

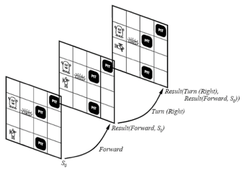
Introduction

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

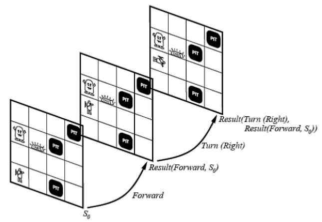
Actions and situations

- So far we modelled a static world only.
- **How to reason about actions and their effects in time?**
- **In propositional logic** we need a copy of each action for each time (situation):
  - $L^t_{x,y} \wedge FacingRight^t \wedge Forward^t \Rightarrow L^{t+1}_{x+1,y}$
  - We need an upper bound for the number of steps to reach a goal but this will lead to a huge number of formulas.
- Can we do it better in **first order logic**?
  - We do not need copies of axioms describing state changes; this can be implemented using a universal quantifier for time (situation)
  - ∀t P is the result of action A in time t+1



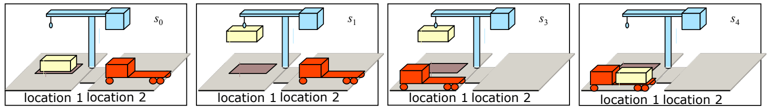
Situation calculus

- **actions** are represented by terms
  - Go(x,y)
  - Grab(g)
  - Release(g)
- **situation** is also a term
  - initial situation: S_0
  - situation after applying action *a* to state *s*: Result(a,s)
- **fluent** is a predicates changing with time
  - the situation is in the last argument of that term
  - Holding(G, S_0)
- **rigid** (eternal) **predicates**
  - Gold(G)
  - Adjacent(x,y)



Situation calculus: plans

- We need to reason about sequences of actions – about **plans**.
  - Result([],s) = s
  - Result([a|seq],s) = Result(seq, Result(a,s))
- What are the typical tasks related to plans?
  - **projection task** – what is the state/situation after applying a given sequence of actions?
    - At(Agent, [1,1] , S_0) ∧ At(G, [1,2], S_0) ∧ ¬Holding(o, S_0)
    - At(G, [1,1], Result([Go([1,1],[1,2]),Grab(G),Go([1,2],[1,1])], S_0))
  - **planning task** – which sequence of actions reaches a given state/situation?
    - ∃seq At(G, [1,1], Result(seq, S_0))



Situation calculus: actions

- Each **action** can be described using two axioms:
  - **possibility axiom:** Preconditions ⇒ Poss(a,s)
    - At(Agent,x,s) ∧ Adjacent(x,y) ⇒ Poss(Go(x,y),s)
    - Gold(g) ∧ At(Agent,x,s) ∧ At(g,x,s) ⇒ Poss(Grab(g),s)
    - Holding(g,s) ⇒ Poss(Release(g),s)
  - **effect axiom:** Poss(a,s) ⇒ Changes
    - Poss(Go(x,y),s) ⇒ At(Agent,y,Result(Go(x,y),s))
    - Poss(Grab(g),s) ⇒ Holding(g,Result(Grab(g),s))
    - Poss(Release(g),s) ⇒ ¬Holding(g,Result(Release(g),s))
- Beware! This is not enough to deduce that a plan reaches a given goal.
  - we can deduce At(Agent, [1,2], Result(Go([1,1],[1,2]), S_0))
  - but we **cannot deduce** At(G, [1,2], Result(Go([1,1],[1,2]), S_0))
  - Effect axioms describe what has been changed in the world but say nothing about the property that everything else is not changed!
  - This is a so called **frame problem.**

Frame problem

- We need to represent properties that are not changed by actions.
- A simple **frame axiom** says what is not changed:

  At(o,x,s) ∧ o≠Agent ∧ ¬Holding(o,s) ⇒
  At(o,x,Result(Go(y,z),s))

  - for F fluents and A actions we need O(FA) frame axioms
  - This is a lot especially taking in account that most predicates are not changed.

Frame problem: better axioms

Can we use less axioms to model the frame problem?

- **successor-state axiom**

  Poss(a,s) ⇒
  (fluent holds in Result(a,s) ⇔
  fluent is effect of a ∨ (fluent holds in s ∧ a does not change fluent))

  - We get F axioms (F is the number of fluents) with O(AE) literals in total (A is the number of actions, E is the number of effects).

  Examples:

  Poss(a,s) ⇒
  (At(Agent,y,Result(a,s)) ⇔ a=Go(x,y) ∨ (At(Agent,y,s) ∧ a≠Go(y,z)))

  Poss(a,s) ⇒
  (Holding(g,Result(a,s)) ⇔ a=Grab(g) ∨ (Holding(g,s) ∧ a≠Release(g)))

  - **Beware of implicit effects!**
    - If an agent holds some object and the agent moves then also the object moves.
    - This is called a **ramification problem**.

  Poss(a,s) ⇒
  (At(o,y,Result(a,s)) ⇔
  (a=Go(x,y) ∧ (o=Agent ∨ Holding(o,s))) ∨
  (At(o,y,s) ∧ ¬∃z (y≠z ∧ a=Go(y,z) ∧ (o=Agent ∨ Holding(o,s)))))

Frame problem: even better axioms

- Successor-state axiom is still too big with O(AE/F) literals in average.
  - To solve the projection task with t actions, the time complexity depends on the total number of actions – O(AEt) – rather than on the actions in plan.
  - If we know each action, cannot we do it better say O(Et)?
- **classical successor-state axiom:**

  $Poss(a,s) \Rightarrow$
  $(F_i(Result(a,s)) \Leftrightarrow \underbrace{(a=A_1 \vee a=A_2 \vee \ldots)}_{\text{actions having } F_i \text{ among effects}} \vee (F_i(s) \wedge \underbrace{a \neq A_3 \wedge a \neq A_4 \ldots}_{\text{actions having } \neg F_i \text{ among effects}}))$
- We can introduce **positive** and **negative effects** of actions:
  - **PosEffect(a, F_i)** action a causes F_i to become true
  - **NegEffect(a, F_i)** action a causes F_i to become false
- **modified successor state axiom:**

  $Poss(a,s) \Rightarrow (F_i(Result(a,s)) \Leftrightarrow PossEffect(a, F_i) \vee (F_i(s) \wedge \neg NegEffect(a,F_i)))$
  $PosEffect(A_1, F_i)$
  $PosEffect(A_2, F_i)$
  $NegEffect(A_3, F_i)$
  $NegEffect(A_4, F_i)$

Hidden assumptions

Example:
- Assume the following claim:
  - *„In summer we will teach courses CS101, CS102, CS106, and EE101"*
  - so in FOL we have the facts
    - Course(CS,101), Course(CS, 102), Course(CS,106), Course(EE,101)
- How many courses will we teach in summer?
  - Something between one and infinity!!

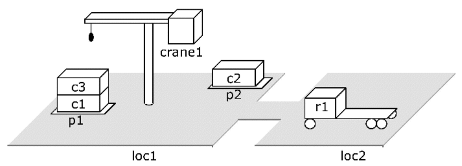
Why?
- We usually assume having a complete information about the world, i.e., what is not explicitly said does not hold – this is called a **closed world assumption** (CWA)
- There is no such assumption in FOL, so we need to complete the knowledge base:

  Course(d,n) $\Leftrightarrow$
  [d,n] = [CS,101] $\vee$ [d,n] = [CS,102] $\vee$ [d,n] = [CS,206] $\vee$ [d,n] = [EE,101]
- We also assumed that different names (constants) denote different objects – this is called a **unique name assumption** (UNA)
- Again, we need to explicitly describe that objects are different:
  - [CS,101] $\neq$ [CS,102], …

Classical representation: states

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!

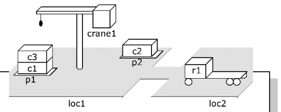
Classical representation: operators

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)

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)

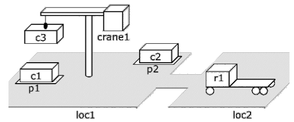
Notation:

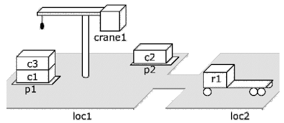
– S^+ = {positive atoms in S}
– S^- = {atoms, whose negation is in S}

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

The result of application of action a to **s** is
γ(**s**,**a**) = (**s** – effects$^-$(**a**)) $\cup$ effects$^+$(**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)





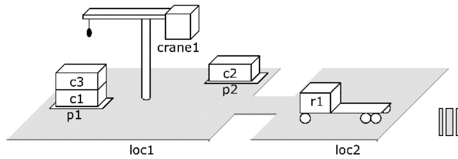
Let L be a language and O be a set of operators.

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

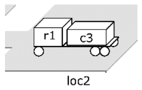
– **states** S $\subseteq$ P({all instantiated atoms from L})
– **actions** A = {all instantiated operators from O over L}
  - action **a** is **applicable** to state **s** if precond$^+$(**a**) $\subseteq$ **s** $\wedge$ precond$^-$(**a**) $\cap$ **s** = $\emptyset$
– **transition function** γ:
  -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

- **Planning problem** P is a triple (Σ,s_0,g):
  – Σ = (S,A,γ) 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 **g**$^+ \subseteq$ **s** $\wedge$ **g**$^- \cap$ **s** = $\emptyset$
    - S_g = {**s** $\in$ S | **s** satisfies **g**} – a set of goal states
- **Plan** is a sequence of actions $\langle a_1, a_2, \ldots, a_k \rangle$.
- Plan $\langle a_1, a_2, \ldots, a_k \rangle$ 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
  – s_0 provides the particular constants (objects)

Classical representation: example plan



our goal

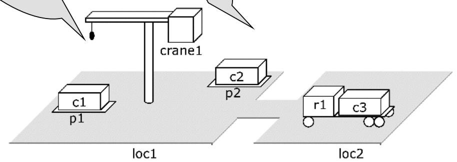


s_1 = {attached(p1,loc1), in(c1,p1), in(c3,p1), top(c3,p1), on(c3,c1), on(c1,pallet), attached(p2,loc1), in(c2,p2), top(c2,p2), on(c2,pallet), belong(crane1,loc1), empty(crane1), adjacent(loc1,loc2), adjacent(loc2,loc1), at(r1,loc2), occupied(loc2), unloaded(r1)}.

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

⟨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)⟩

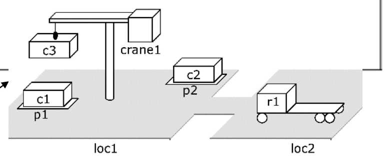


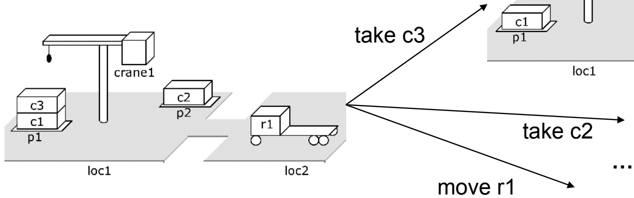
State-space planning

- **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

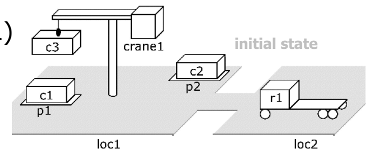
Forward-search(O, s_0, g)
  $s \leftarrow s_0$
  $\pi \leftarrow$ the empty plan
  loop
    if s satisfies g then return π
    $E \leftarrow \{a | a$ is a ground instance of an operator in O,
        and $\mathrm{precond}(a)$ is true in $s\}$
    if $E = \emptyset$ then return failure
    nondeterministically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

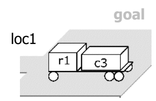
move(r, l, m)
;; robot r moves from location l to location m
precond: adjacent(l, m), at(r, l), ¬ occupied(m)
effects: at(r, m), occupied(m), ¬ occupied(l), ¬ at(r, l)

{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)}

Backward planning

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** ∩ effects(**a**) ≠ ∅
- effects of action **a** are not conflicting goal **g**:
  - $g^- \cap \text{effects}^+(a) = \emptyset$
  - $g^+ \cap \text{effects}^-(a) = \emptyset$

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

Example:

goal: **{on(a,b), on(b,c)}**

action **stack(a,b)** is relevant

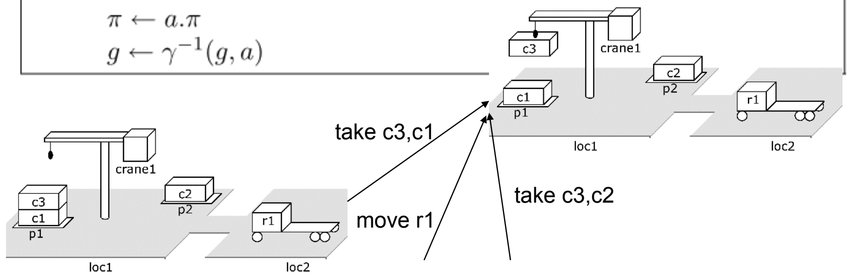
by backward application of the action we get a new goal:

{holding(a), clear(b), on(b,c)}

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

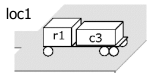
Backward planning: algorithm

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 $a \in A$
    $\pi \leftarrow a.\pi$
    $g \leftarrow \gamma^{-1}(g,a)$



Backward planning: an example

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



load(crane1,loc1,c3,r1)

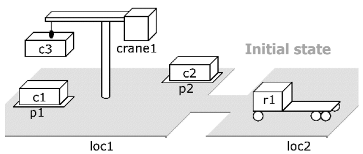
load(k,l,c,r)
;; crane k at location l loads container c onto robot r
precond: belong(k,l), holding(k,c), at(r,l), unloaded(r)
effects: empty(k), ¬ holding(k,c), loaded(r,c), ¬ unloaded(r)

{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), ¬ occupied(m)
effects: at(r,m), occupied(m), ¬ occupied(l), ¬ at(r,l)

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



Backward planning: lifting

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:

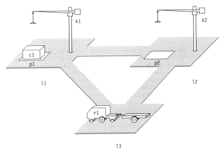
- standardization = a copy with fresh variables
- mgu = most general unifier
- by using the variables we can decrease the branching factor but the trade off is more complicated loop check

Plan-space planning: a core idea

- 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

Plan space planning: an example

- 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

Plan space planning: the initial plan

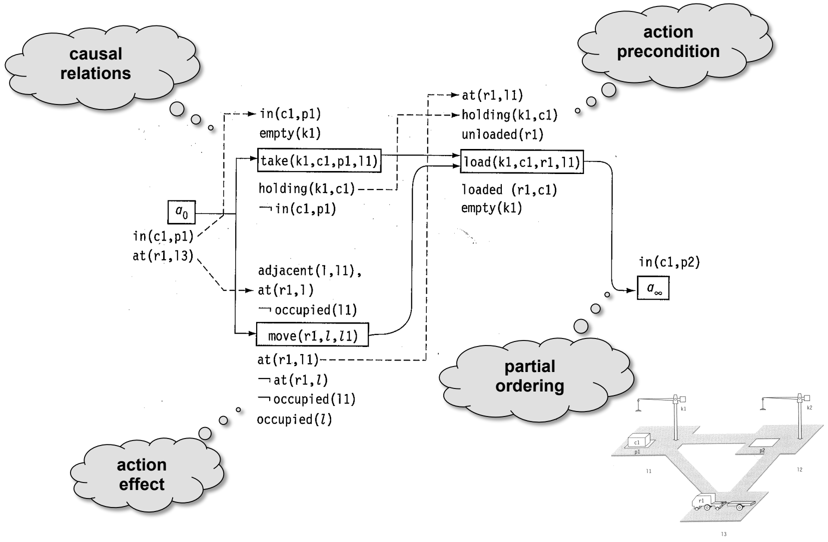
- **The initial state and the goal** are encoded using two **special actions** in the initial partial plan:
  - **Action a_0 represents the initial state** in such a way that atoms from the initial state define effects of the action and there are no preconditions. This action will be before all other 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.

Search nodes and partial plans

The search nodes correspond to partial plans.

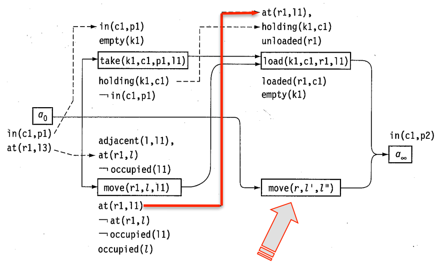
A partial plan Π is a tuple (A,<,B,L), where

- 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 ($a_i \xrightarrow{p} a_j$)
  - 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



- **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 $a_i \to^p b$).
- **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 $\mathbf{a} \to^{\mathbf{p}} \mathbf{b}$

- **Threat** is another example of **flaw**.
- This is action that can influence existing causal relation.
  - Let $a_i \to^p a_j$ 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 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 $\Pi = (A,<,B,L)$ is a **solution plan** for the problem $P = (\Sigma, s_0, g)$ 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 $\Pi = (A,<,B,L)$ is a solution plan if:
- there are no flaws (no open goals and no threats)
- partial ordering < and constraints B are globally consistent

Plan-space planning: algorithm

- **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 summary

- 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***
  - structured 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

More on automated planning

- Course **Planning and scheduling**
  - http://ktiml.mff.cuni.cz/~bartak/planovani/

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