Umělá inteligence I

Roman Barták, KTIML

roman.bartak@mff.cuni.cz
http://ktiml.mff.cuni.cz/~bartak

12

Plánování

Klasickou plánovací úlohu tedy můžeme řešit odvozovacími metodami predikátové logiky.

Nešlo by to lépe?

- **Reprezentace plánovacího problému**
  - množiny predikátů (místo formulí)
- **Plánovací algoritmy**
  - plánování v prostoru stavů
    - odvozovací mechanismy z logiky přizpůsobíme nové reprezentaci
    - dopředné a zpětné plánování
  - plánování v prostoru plánů
    - stavíme plán po kouskách

Klasická reprezentace

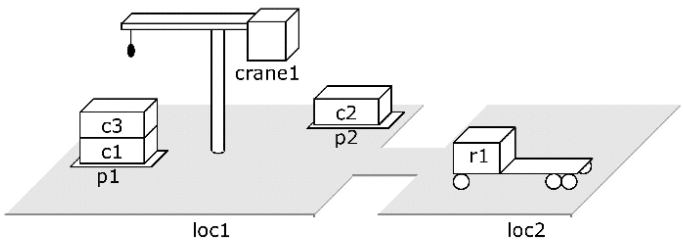
- **Klasická reprezentace** plánovacího problému používá **predikátovou logice**:
  - **Stavy** jsou množiny logických atomů, které jsou v dané interpretaci buď pravda nebo nepravda.
  - **Akce** jsou reprezentovány plánovacími operátory, které mění pravdivostní hodnotu těchto atomů.

Přesněji:

- L (**jazyk**) je konečná množina predikátových symbolů a konstant (nemáme funkce!).
- **Atom** je predikátový symbol s argumenty, např. on(c3,c1).
- Můžeme používat **proměnné**, např. on(x,y).

Reprezentace stavů

- **Stav je množina instanciovaných atomů** (bez proměnných). Je jich konečně mnoho!



{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)}.

- pravdivostní hodnota některých atomů se mění
  - **flexibilní atomy** (fluent)
  - např. at(r1,loc2)
- některé atomy nemění svojí pravdivostní hodnotu s různými stavy
  - **neměnné atomy** (rigid)
  - např. adjacent(loc1,loc2)

- **Předpoklad uzavřeného světa** (closed world assumption) Atom, který není ve stavu explicitně uveden, neplatí!

Plánovací operátory

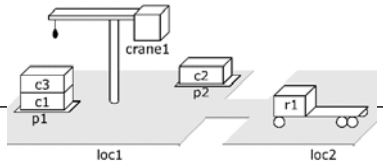
operátor o je trojice:
(name(o), precond(o), effects(o))

- **name(o): jméno operátoru** ve tvaru $n(x_1,\ldots,x_k)$
  - n: symbol operátoru (jednoznačný pro každý operátor)
  - $x_1,\ldots,x_k$: symboly proměnných (parametry operátoru)
    - musí obsahovat všechny symboly proměnných v operátoru!
- **precond(o): předpoklady**
  - literály, které musí být splnitelné, aby šlo operátor použít
- **effects(o): efekty**
  - literály, které se stanou pravdivými aplikací operátoru (nesmí to být neměnné atomy!)

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)

Akce

Akce jsou plně instanciované operátory
– za proměnné jsou dosazeny konstanty

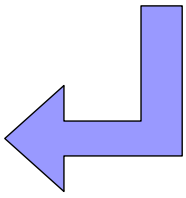


take(k, l, c, d, p)
;; crane k at location l takes c off of d in pile p
precond: belong(k, l), attached(p, l), empty(k), top(c, p), on(c, d)
effects: holding(k, c), $\neg$ empty(k), $\neg$ in(c, p), $\neg$ top(c, p), $\neg$ on(c, d), top(d, p)

operátor

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)

akce



Aplikace akce

Notace:

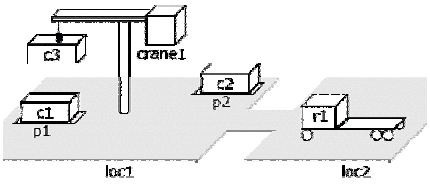
- S^+ = {pozitivní atomy v S}
- S^- = {atomy, jejichž negace je v S}

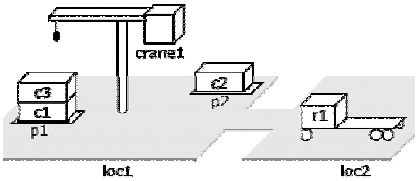
Akce a je **použitelná** na stav s právě když
$\text{precond}^+(a) \subseteq s \;\&\; \text{precond}^-(a) \cap s = \varnothing$

Výsledkem aplikace akce a na s je
$\gamma(s,a) = (s - \text{effects}^-(a)) \cup \text{effects}^+(a)$

Implicitně řeší problém rámce

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)





Plánovací doména

Nechť L je jazyk a O je množina operátorů.

Plánovací doména Σ nad jazykem L a s operátory O je trojice (S,A,γ):

- **stavy** $S \subseteq P(\{\text{všechny instanciované atomy nad L}\})$
- **akce** A = {všechny instanciované operátory z O nad L}
  - akce a je **použitelná** na stav s, pokud
    $\text{precond}^+(a) \subseteq s \;\&\; \text{precond}^-(a) \cap s = \varnothing$
- **přechodová funkce** γ:
  - $\gamma(s,a) = (s - \text{effects}^-(a)) \cup \text{effects}^+(a)$, je-li a použitelná na s
  - S je uzavřená vzhledem ke γ (je-li $s \in S$, potom pro každou akci a aplikovatelnou na s platí $\gamma(s,a) \in S$)

Plánovací problém

Plánovací problém P je trojice (Σ, s_0, g):

- $\Sigma=(S,A,\gamma)$ je plánovací doména
- s_0 je počáteční stav, $s_0 \in S$
- g je množina instanciovaných literálů
  - stav s splňuje g právě tehdy, když $g^+ \subseteq s$ & $g^- \cap s = \emptyset$
  - $S_g = \{s \in S \mid s \text{ splňuje } g\}$ - množina cílových stavů

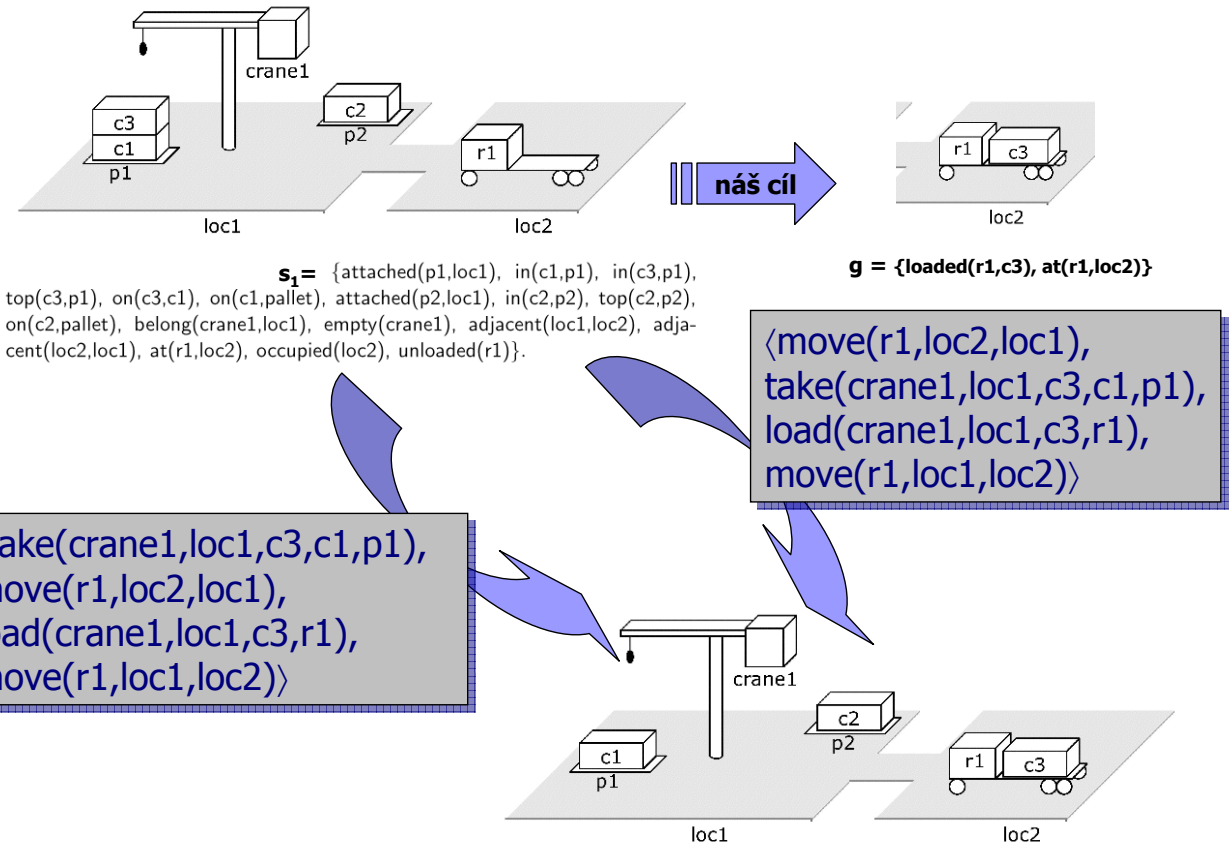
Zápis plánovacího problému je trojice (O, s_0, g).

Plány a řešení

Plán π je posloupnost akcí $\langle a_1, a_2, \ldots, a_k \rangle$.
Plán π je **řešením** P právě když $\gamma(s_0,\pi)$ splňuje g.

- **Dopředné ověření existence plánu**
  Přímí následníci stavu s: $\Gamma(s) = \{\gamma(s,a) \mid a \in A \text{ je aplikovatelná na } s\}$
  Dosažitelné stavy: $\Gamma_\infty(s) = \Gamma(s) \cup \Gamma^2(s) \cup \ldots$
  Plánovací problém má řešení právě když $S_g \cap \Gamma_\infty(s_0) \neq \emptyset$.
- **Zpětné ověření existence plánu**
  Akce a je relevantní pro cíl g právě když:
  akce přispívá do g: $g \cap \text{effects}(a) \neq \emptyset$
  efekty akce nejsou v konfliktu s g: $g^- \cap \text{effects}^+(a) = \emptyset \wedge g^+ \cap \text{effects}^-(a) = \emptyset$
  Regresní (zpětná) množina cíle g pro (relevantní) akci a:
  $\gamma^{-1}(g,a) = (g - \text{effects}(a)) \cup \text{precond}(a)$
  $\Gamma^{-1}(g) = \{\gamma^{-1}(g,a) \mid a \in A \text{ je relevantní pro } g\}$
  $\Gamma_\infty^{-1}(g) = \Gamma^{-1}(g) \cup \Gamma^{-2}(g) \cup \ldots$
  Plánovací problém má řešení právě když s_0 je nadmnožinou nějakého prvku z $\Gamma_\infty^{-1}(g)$.

Ukázka plánu

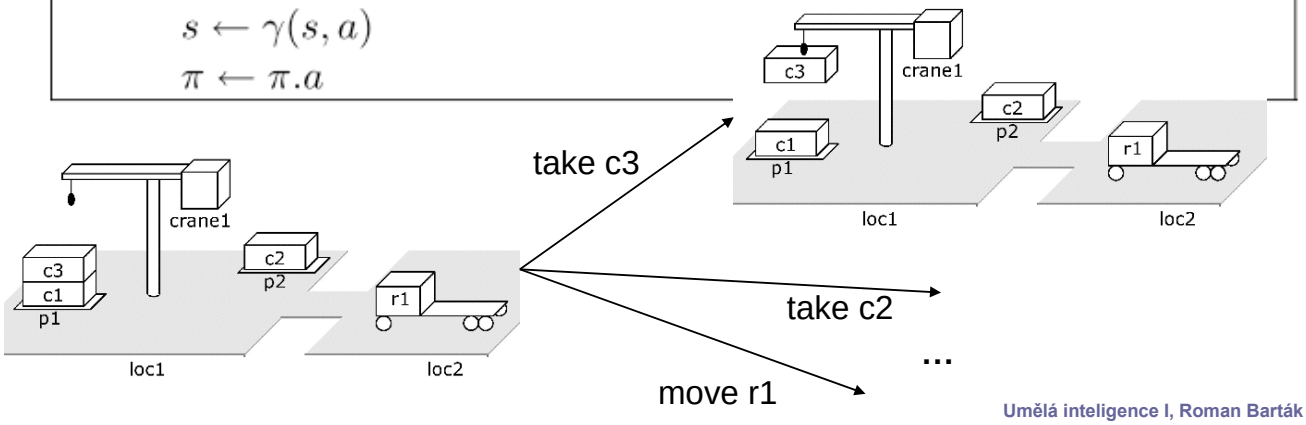


Plánování se stavy

- **Prohledávaný prostor odpovídá stavovému prostoru plánovacího problému.**
  - uzly odpovídají stavům
  - hrany odpovídají stavovým přechodům pomocí akcí
  - cílem je najít cestu mezi počátečním stavem a některým koncovým stavem
- **Typy prohledávání**
  - dopředné (forward search, progression)
    - začínáme v počátečním stavu a jdeme k některému stavu cílovému
  - zpětné (backward search, regression)
    - začínáme s cílem (pozor to není stav, ale reprezentace množiny stavů!) a jdeme přes podcíle k počátečním stavu
    - liftovaná verze (částečné instanciování akcí, tj. použijeme proměnné)

Dopředné plánování

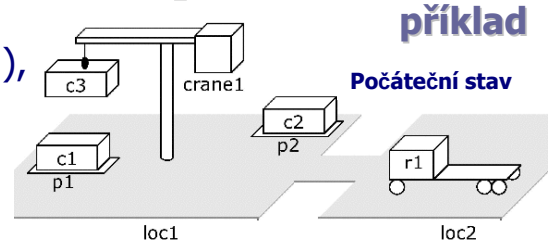
Forward-search(O, s_0, g)
- $s \leftarrow s_0$
- $\pi \leftarrow$ the empty plan
- loop
  - if s satisfies g then return π
  - $E \leftarrow \{a | a$ is a ground instance an operator in O, and $\text{precond}(a)$ is true in $s\}$
  - if $E = \emptyset$ then return failure
  - nondeterministically choose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



Dopředné plánování

příklad

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



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), ¬occupied(loc2), …}

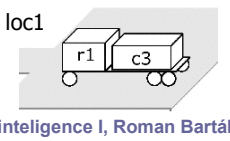
load(crane1,loc1,c3,r1)

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), ¬occupied(loc2), …}

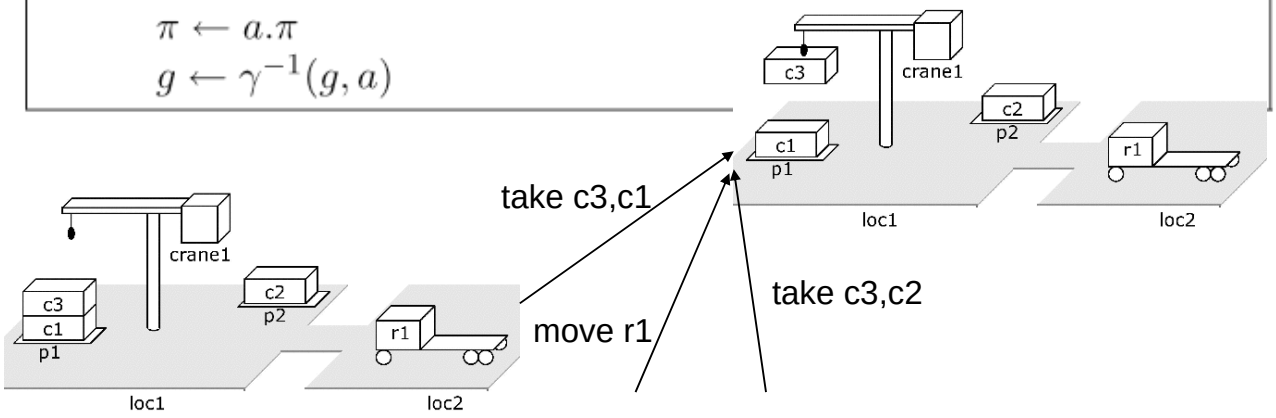
Cíl = {at(r1,loc1),loaded(r1,c3)}

Cíl



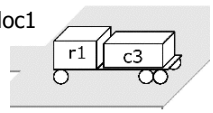
Zpětné plánování

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



Zpětné plánování

příklad



Cíl = {at(r1,loc1),loaded(r1,c3)}

load(crane1,loc1,c3,r1)

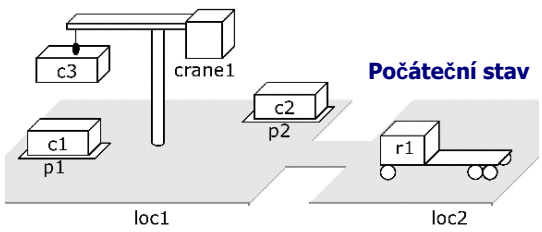
load(k, l, c, r)
  ;; crane k at location l loads container c onto robot r
  precond: belong(k, l), holding(k, c), at(r, l), unloaded(r)
  effects: empty(k), $\neg$ holding(k, c), loaded(r, c), $\neg$ unloaded(r)

{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)}



Počáteční stav

Zpětné plánování-liftování

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 $\theta(\pi)$
    $g \leftarrow \gamma^{-1}(\theta(g), \theta(o))$

- standardizace = kopie s novými proměnnými
- mgu = most general unifier (nejobecnější unifikace)
- použití volných proměnných zmenšuje větvení, ale komplikuje detekci cyklu

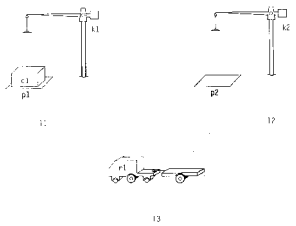
Plánování v prostoru plánů

- Princip podobný **zpětnému plánování** ve stavovém prostoru:
  - začínáme z **„prázdného" plánu** obsahujícího popis počátečního stavu a cíle
  - **přidáváme další akce**, které plní dosud nesplněné (otevřené) cíle
  - případně **přidáváme vazby** mezi již přítomnými akcemi
- Na plánování se můžeme dívat jako na **opravovaní kazů v částečném plánu**
  - přecházíme od jednoho částečného plánu k dalšímu dokud nenajdeme úplný plán

Plánování v prostoru plánů

příklad

- Předpokládejme, že v částečném plánu zatím máme akce:
  - take(k1,c1,p1,l1)
  - load(k1,c1,r1,l1)



- **Možné úpravy plánu:**
  - **Přidání akce**
    - aby šlo použít **load**, musí být robot r1 na místě l1
    - přesuň robota r1 na místo l1 **move**(r1,l,l1)
  - **Svázání proměnných**
    - akce **move** se týká správného robota a správného místa
  - **Přidání podmínky uspořádání**
    - přesunutí robota se musí uskutečnit před **load**
    - na pořadí vzhledem k **take** ale nezáleží
  - **Přidání kauzální (příčinné) vazby**
    - nová akce byla přidána, aby se robot dostal tam, kam má
    - kauzální vazba mezi **move** a **load** nám zajistí, že mezi těmito akcemi robota někdo zase neodvolá

Principy PSP

- **Počáteční stav i cíl** zakódujeme jako **speciální akce**, které jsou v prvotním částečném plánu:
  - **Akce a_0 reprezentuje počáteční stav** tak, že nemá žádné předpoklady a počáteční stav je zakódován jako efekt. Tato akce je před všemi ostatními akcemi.
  - **Akce a_∞ reprezentuje cíl**, který je zakódován jako předpoklad, efekt akce je prázdný. Tato akce je za všemi ostatními akcemi.
- **Plánování** bude založeno na **odstraňování kazů** (flaws) částečného plánu.
  - Budeme přecházet od jednoho částečného plánu k dalšímu, dokud nenajdeme řešící plán.

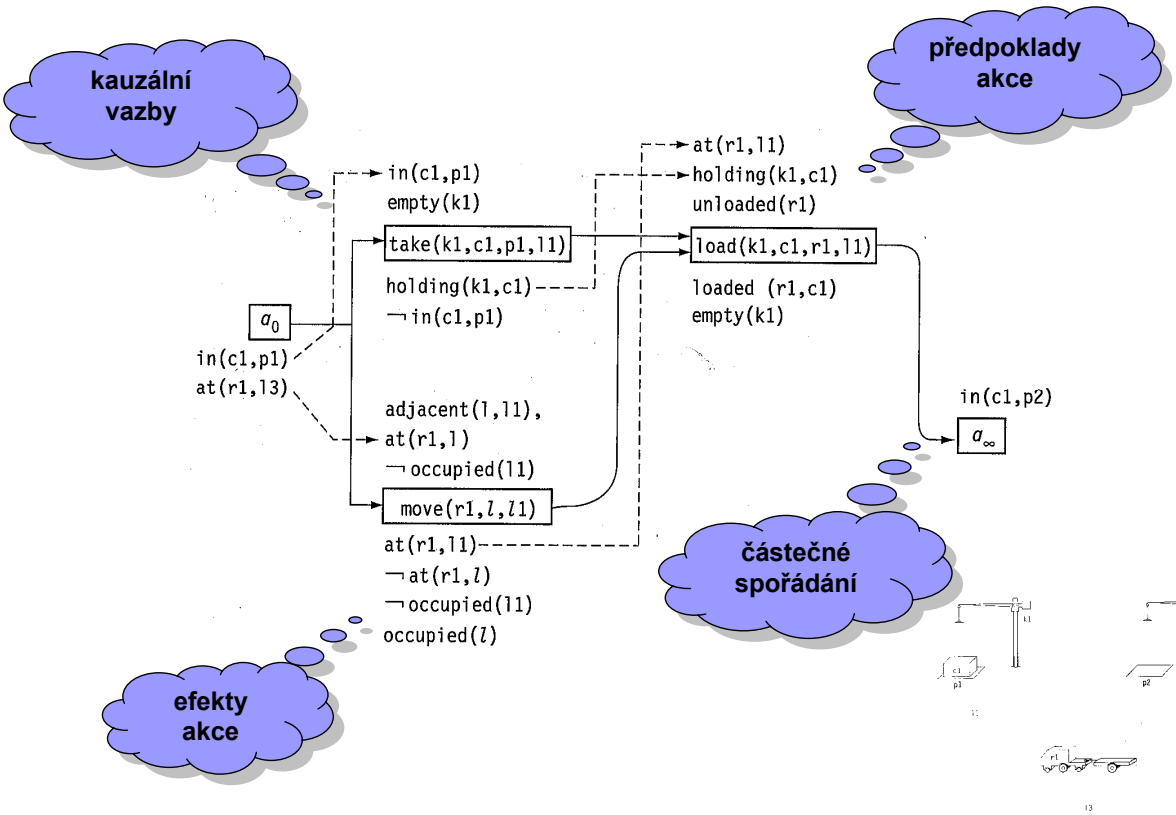
Uzly jsou plány

Uzly prohledávaného prostoru jsou tvořeny částečnými plány.

Částečný plán Π je čtveřice (A,<,B,L), kde

- A je množina částečně instanciovaných plánovacích operátorů $\{a_1,\ldots,a_k\}$
- $<$ je částečné uspořádání na A ($a_i<a_j$)
- B je množina vazeb tvaru x=y, x≠y nebo $x\in D_i$
- L je množina kauzálních vztahů tvaru $(a_i \rightarrow^p a_j)$
  - a_i,a_j jsou akce uspořádané $a_i<a_j$
  - p je výraz, který je efektem a_i a předpokladem a_j
  - v B jsou vazby svazující příslušné proměnné v p

Částečný plán: příklad

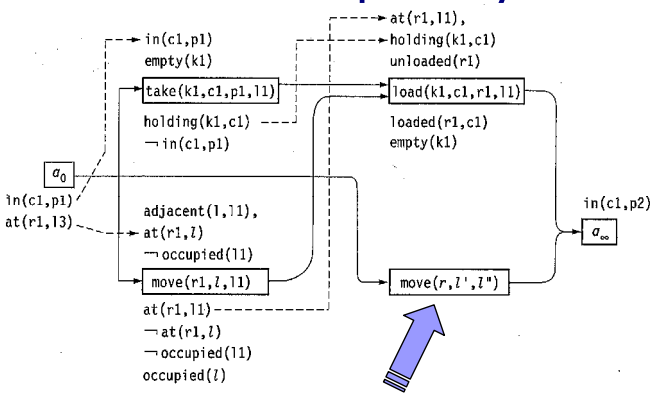


Otevřený cíl

- **Otevřený cíl** (open goal) je **kazem plánu.**
- Jedná se o předpoklad p nějakého operátoru b, o kterém zatím nebylo rozhodnuto, jak ho splnit (neexistuje kauzální vazba $a_i \rightarrow^p b$).
- **Odstranění otevřeného cíle p akce b:**
  - najdi operátor a (buď již přítomný v plánu nebo nový),
  - který lze použít na splnění p (má p mezi efekty a může být před b)
  - svaž proměnné
  - vytvoř kauzální vazbu

Hrozba

- **Hrozba** (threat) je dalším **kazem plánu.**
- Jedná se o akci, která může porušit kauzální vazbu.
  - Přesněji, je-li $a_i \rightarrow^p a_j$ kauzální vazba a akce b má efekt unifikovatelný s negací p a může se nacházet mezi a_i a a_j , potom je b hrozbou (může porušit platnost kauzální vazby).
- **Odstranění hrozby** lze udělat třemi způsoby:
  - uspořádáním b před a_i
  - uspořádáním b za a_j
  - navázáním proměnných v b tak, že neruší platnost p



Řešící plán

- **Částečný plán** Π = (A,<,B,L) je řešícím plánem pro problém P = (Σ,s_0,g) pokud:
  - částečné uspořádání < a vazby B jsou globálně konzistentní
    - v částečném uspořádání nejsou cykly
    - mohu proměnné přiřadit hodnotu z příslušné domény tak, že najdu hodnoty ostatních proměnných splňující B
  - libovolná lineárně uspořádaná posloupnost plně instanciovaných akcí A splňující < a vazby B vede z s_0 do stavu splňujícího g
- Definice nám bohužel přímo **nedává výpočtovou proceduru**, jak ověřit, zda je plán řešící!

  Tvrzení: Částečný plán Π = (A,<,B,L) je řešící pokud:
  - nemá žádné kazy, tj. otevřené cíle ani hrozby
  - částečné uspořádání < a vazby B jsou globálně konzistentní

Procedura PSP

- **PSP = Plan-Space Planning** (plánování v prostoru plánů)

```
PSP(π)
    flaws ← OpenGoals(π) ∪ Threats(π)
    if flaws = ∅ then return(π)
    select any flaw φ ∈ flaws
    resolvers ← Resolve(φ, π)
    if resolvers = ∅ then return(failure)
    nondeterministically choose a resolver ρ ∈ resolvers
    π' ← Refine(ρ, π)
    return(PSP(π'))
end
```

- Volba kazu je deterministická (musí se odstranit všechny kazy).
- Volba zjemnění je nedeterministická (v případě neúspěchu se zkouší další alternativa).

Detaily PSP

- Otevřené cíle lze efektivně zjistit udržováním **agendy předpokladů akcí**. Přidání kauzální vazby pro p vyřadí p z agendy.
- **Všechny hrozby** lze najít prozkoumáním všech trojic akcí ($O(n^3)$) nebo inkrementálně: po přidání akce se zjistí, komu je hrozbou ($O(n^2)$), a po přidání kauzální vazby se ověří její hrozby ($O(n)$).
- Pro odstraněné otevřených cílů a hrozeb se používají pouze **konzistentní** zjemnění plánu.
  - konzistence uspořádání buď detekcí cyklů nebo lépe udržováním tranzitivního uzávěru
  - konzistence vztahů B
    - pokud není negace, lze rychle (například pomocí AC)
    - je-li přítomna negace jedná se o NP-úplný problém

Více o plánování

- Přednáška **Plánování a rozvrhování**
  - http://kti.mff.cuni.cz/~bartak/planovani/