Klasické plánování

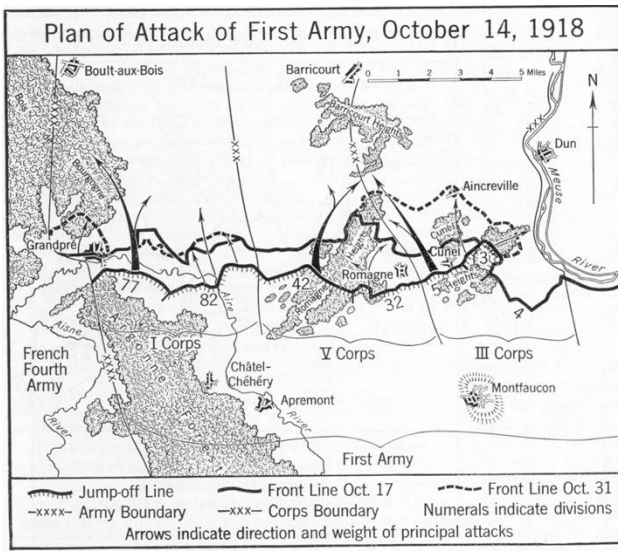
Radek Mařík

CVUT FEL, K13133

16. dubna 2014

Obsah

Koncept plánu [Nau09]



Plán

- mnoho definic a pohledů ....
- Schéma, program nebo metoda připravená dopředu k dosažení nějakého cíle.

Definice plánu [Nau09, Pec10]

Plánování

- Uvažování o hypotetické interakci mezi agentem a prostředím vzhledem k dané úloze.
- Motivací k procesu plánování je zdůvodnění, že daná dle plánu provedená sada akcí změní prostředí tak, aby bylo dosaženo vytčeného cíle či řešení zadání úlohy.

Plánování a rozvrhování [Nau09]

- **Rozvrhování** . . . přiřazuje řadě procesů zdroje s časovým omezením,
- **Plánování** . . . uvažuje možné interakce mezi komponentami plánu.

Plánování

- Dáno: počáteční stav, cílový stav, operátory.
- Hledáme sekvenci operátorů, které dosáhnou na cílový stav z počátečního stavu
  - výběr příslušných akcí, uspořádáním akcí, zahrnutí kauzalit

Rozvrhování

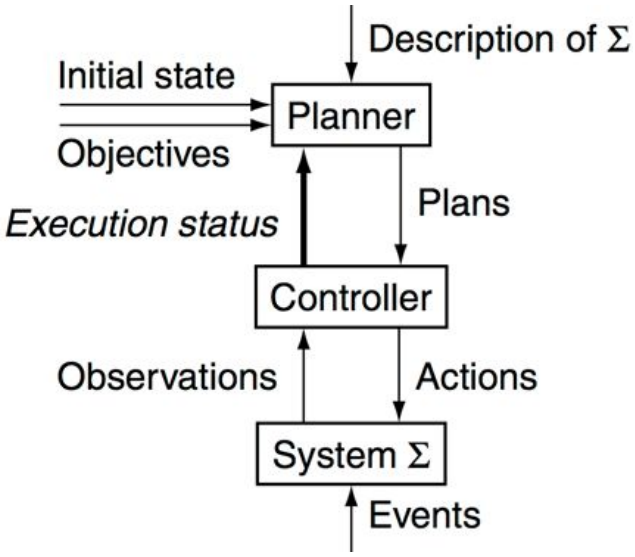
- Dáno: zdroje, akce, omezení.
- Hledáme rozvrh, který vyhovuje omezením
  - uspořádání akcí, přiřazení zdrojů, splnění omezení.

Reálné aplikace - průzkum vesmíru [Nau09]

Projekty

- Autonomní plánování, rozvrhování, řízení
  - NASA: JPL a Ames.
- Remote Agent Experiment (REX)
  - Deep Space 1.
- Mars Exploration Rover (MER)

Koncepční model plánování I [Nau09]



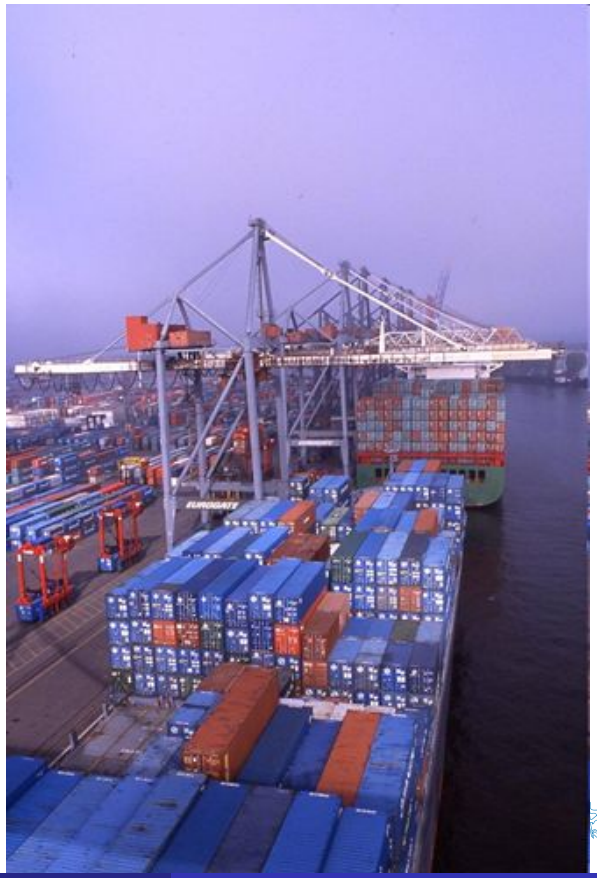
Prostředí

- Systém s přechody mezi stavy
- $\Sigma = (S, A, E, \gamma)$
- $S = \{\text{states}\}$
- $A = \{\text{actions}\}$
- $E = \{\text{exogenous events}\}$
- $\gamma = \{\text{state-transition function}\}$

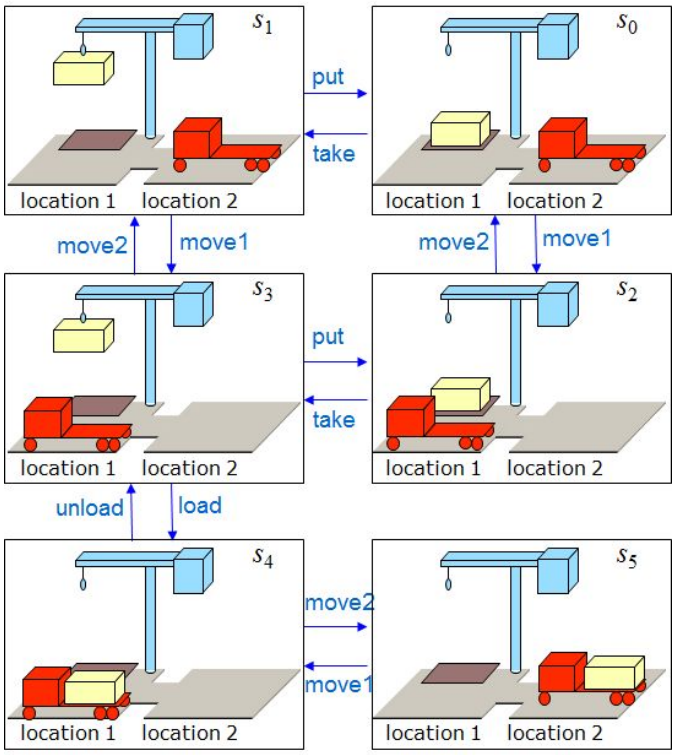
Roboty v docích [Wic11]

Ilustrace plánovacích procedur

- přístav s několika místy (doky)
- lodě,
- sklady s kontejnery
- parkovací místa pro
  - vlaky,
  - nákladní auta
- cíl:
  - vyložit/naložit lodě atd.
  - přemísťovat kontejnery pomocí robotů



Příklad přechodového systému [Nau09]



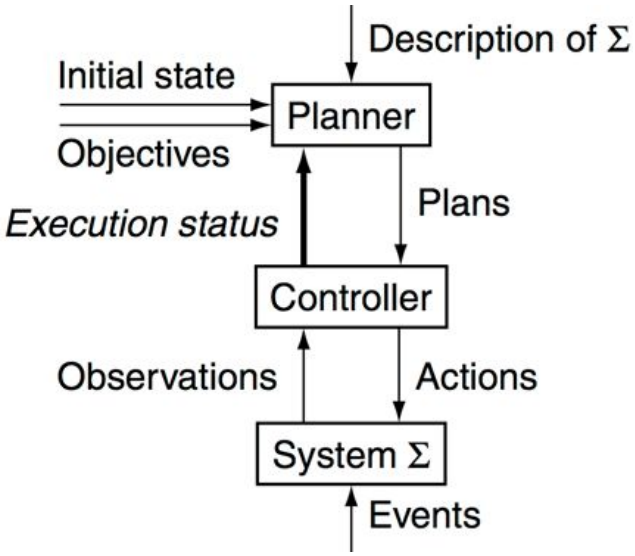
Systém s přechody mezi stavy

- $\Sigma = (S, A, E, \gamma)$
- $S = \{\text{states}\}$
- $A = \{\text{actions}\}$
- $E = \{\text{exogenous events}\}$
- $\gamma = S \times (A \cup E) \rightarrow 2^S$

Systém s přechody mezi stavy

- $S = \{s_0, s_1, \ldots, s_5\}$
- $A = \{move_1, move_2, put, take, load, unload\}$
- $E = \{\}$
- $\gamma = \{\text{viz šipky}\}$

Koncepční model plánování I [Nau09]



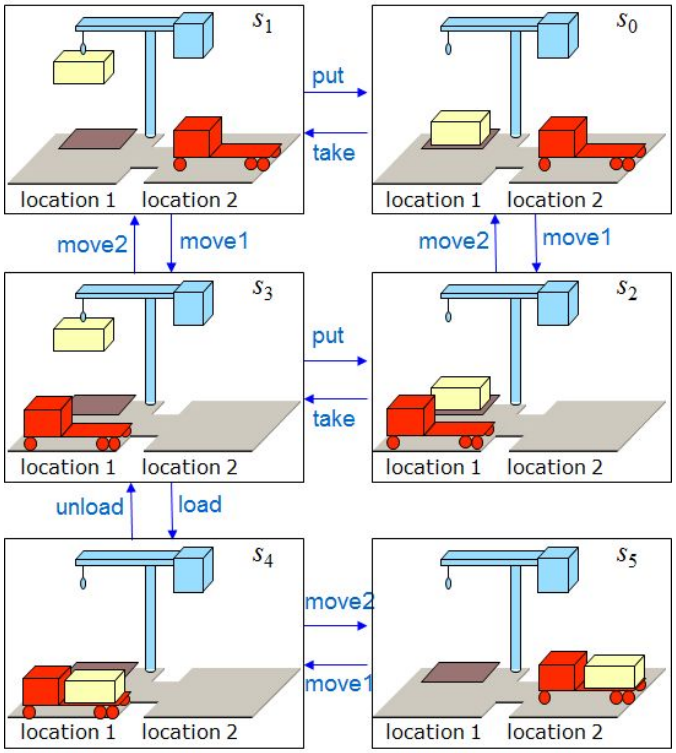
Řadič

- pozorovací funkce $h : S \rightarrow O$
- za pozorování $o \in O$ produkuje akci $a \in A$

Plánovač

- produkuje plány
- instrukce pro řadič

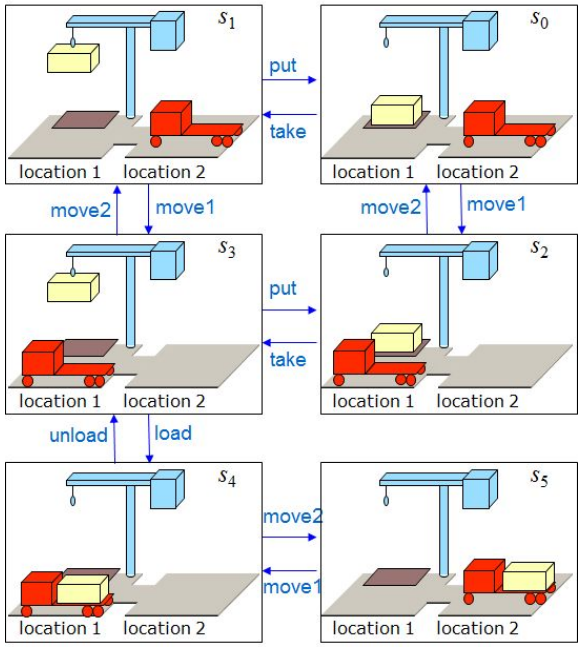
Plánovací úloha [Nau09]



angl. Planning problem

- popis systému Σ
- počáteční stav nebo množina stavů
  - počáteční stav $= s_0$
- Zadání úlohy
  - cílový stav
  - množina cílových stavů
  - množina úloh,
  - "trajektorie" stavů,
  - hodnotící funkce
  - cílový stav $= s_5$

Plán [Nau09]



Klasický plán

- sekvence akcí
- $< take, move_1, load, move_2 >$

Postup, taktika (angl. policy)

- částečná funkce z S do A
- $\{(s_0, take), (s_1, move_1), (s_3, load), (s_4, move_2)\}$

Typy plánovačů [Nau09]

Plánovače pro specifickou doménu

- navrženy a vyladěny pro danou doménu
- nebudou pracovat dobře (jestli vůbec) v jiné doméně
- většina úspěšných reálných plánovačů pracuje tímto způsobem

Plánovače nezávislé na doméně

- v principu pracuje v jakékoliv plánovací doméně
- nepoužívá žádnou doménově specifickou znalost s vyjímkou definic základních akcí
- v praxi je neproveditelné, aby pracovalo pro každou možnou doménu,
- zavedení řady zjednodušujících předpokladů
  - klasické plánování
  - zaměřena většina výzkumu automatizovaného plánování

Omezující předpoklady [Nau09]

- **A0: Konečný systém**
  - konečně mnoho stavů, akcí, událostí
- **A1: Plně pozorovatelný**
  - řadič vždy může pozorovat současný stav
- **A2: Deterministický**
  - každá akce má pouze jediný výstup
- **A3: Statický** (žádné vnější události)
  - změny jsou způsobeny pouze akcemi řadiče
- **A4: Dosažitelnost cílů**
  - existence množiny cílových stavů S_g
- **A5: Sekvenční plány**
  - plán je lineárně uspořádaná sekvence akcí $< a_0, a_1, \ldots, a_n >$
- **A6: Implicitní čas**
  - žádné časové prodlevy, lineární sekvence okamžitých stavů
- **A7: Off-line plánování**
  - plánovač nezná aktuální stav běhu

Klasické plánování [Nau09]

- Vychází ze všech 8 omezujících předpokladů
  - Offline generování sekvencí akcí pro deterministické, statické, konečné systémy s úplnou znalostí, dosažitelnými cíli a implicitním časem.
- Redukuje se na následující úlohu:
  - Dáno (Σ, s_0, S_g)
  - Nalezni sekvenci akcí $\pi =< a_0, a_1, \ldots, a_n >$, která produkuje sekvenci přechodů mezi stavu $< s_0, s_1, \ldots, s_n >$ takovou, že $s_n \in S_g$.
- Metodou je vyhledávání cesty v grafu
  - uzly = stavy
  - hrany = akce
- Je to triviální?

Klasické plánování - příklad [Nau09]

Převozy nákladů letadly

- 10 letišť
- 50 letadel
- 200 kusů nákladu
  - počet stavů $50^{10} \times 200^{50+10} \approx 10^{155}$

Realita

- Počet částic ve vesmíru je pouze okolo 10^{87}

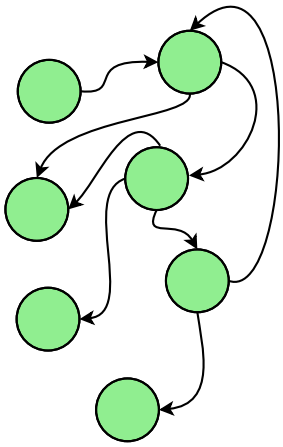
Výzkum automatizovaného plánování

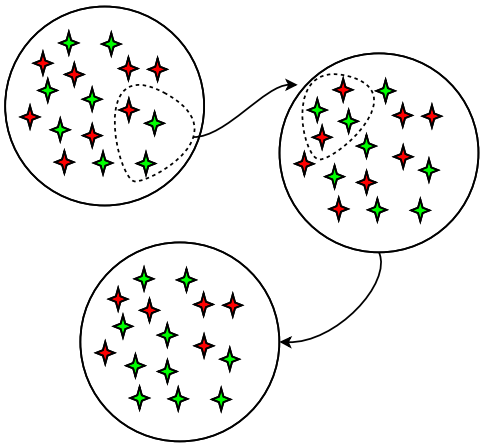
- převážně klasické plánování
- existují desítky (stovky) různých algoritmů

Klasické representace [Wic11]

- **reprezentace pomocí výroků**
  - stav světa je množina výroků
  - akce obsahuje předpoklady ve formě výroků a efekty ve formě výroků, které se přidávají či smažou
- **STRIPS reprezentace**
  - podobně jako výroková reprezentace
  - místo výroků se však pracuje s literály prvého řádu
- **reprezentace pomocí stavových proměnných**
  - stav je k-tice stavových proměnných $\{x_1, \ldots, x_n\}$
  - akce je parciální funkce nad stavy

Rozložená reprezentace stavu





Reprezentace stavu světa

- atomická . . . stav je dále nedělitelný
- rozložená . . . stav je kolekce proměnných

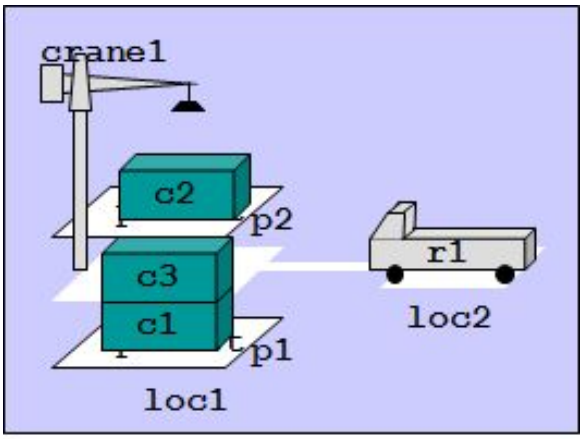
STRIPS - reprezentace stavů [Wic11]

- Nechť $\mathcal{L}$ je jazyk prvého řádu
  - s konečně mnoha predikátovými symboly,
  - s konečným počtem konstatních symbolů,
  - a žádnými funkčními symboly
- **stav STRIPS plánovací domény** je množina základních atomů z $\mathcal{L}$:
  - (základní) atom p platí ve stavu $s \Leftrightarrow p \in s$
  - s splňuje množinu (základních) literálů g ($s \models g$), jestliže
    - každý pozitivní literál v g je v s
    - každý negativní literál v g není v s

Příklad STRIPS stavu [Wic11]

Stav v DWR doméně

$$
\begin{aligned}
state = \{ & attached(p_1, loc_1), \\
& attached(p_2, loc_1), \\
& in(c_1, p_1), in(c_3, p_1), \\
& top(c_3, p_1), on(c_3, c_1), \\
& on(c_1, pallet), in(c_2, p_2), \\
& top(c_2, p_2), on(c_2, pallet), \\
& belong(crane_1, loc_1), \\
& empty(crane_1), \\
& adjacent(loc_1, loc_2), \\
& adjacent(loc_2, loc_1), \\
& at(r_1, loc_2), \\
& occupied(loc_2), \\
& unloaded(r_1)\}
\end{aligned}
$$



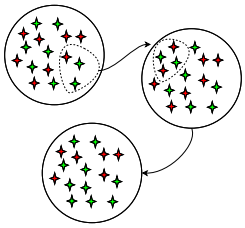
STRIPS - reprezentace operátorů a akcí [Wic11]

- **plánovací operátor** STRIPS plánovací domény je trojice
  - $o = (name(o), precond(o), effects(o))$,
    - kde jméno operátoru $name(o)$
    - je syntaktický výraz tvaru $n(x_1, \ldots, x_k)$,
    - kde n je (jednoznačný) symbol
    - a $x_1, \ldots, x_k$ jsou všechny proměnné,
    - které se vyskytují v o, a
  - vstupní podmínky $precond(o)$ a efekty $effects(o)$ operátoru jsou množiny literálů.
- **akce** STRIPS plánovací domény je uzavřená instance plánovacího operátorů.

Příklad STRIPS operátorů [Wic11]

- $move(r, l, m)$
  - robot r se přesune z místa l do sousedního místa m
  - **precond:** $adjacent(l, m), at(r, l), \neg occupied(m)$
  - **effects:** $at(r, m), occupied(m), \neg occupied(l), \neg at(r, l)$
- $load(k, l, c, r)$
  - jeřáb k na místě l naloží kontejner c na robot r
  - **precond:** $belong(k, l), holding(k, c), at(r, l), unloaded(r)$
  - **effects:** $empty(k), \neg holding(k, c), loaded(r, c), \neg unloaded(r)$
- $put(k, l, c, d, p)$
  - jeřáb k na místě l položí kontejner c na d v svazku p
  - **precond:** $belong(k, l), attached(p, l), holding(k, c), top(d, p)$
  - **effects:**
    $\neg holding(k, c), empty(k), in(c, p), top(c, p), on(c, d), \neg top(d, p)$

Aplikovatelnost a přechody mezi stavy [Wic11]



- Nechť L je množina literálů
  - L^+ je množina atomů, které jsou pozitivní literály v L,
  - L^- je množina všech atomů, jejichž negace jsou v L
- Nechť a je akce a s je stav.
- Potom a je **aplikovatelná** v $s \Leftrightarrow$:
  - $precond^+(a) \subseteq s$; a
  - $precond^-(a) \cap s == \{\}$
- Stavová přechodová funkce y pro akci aplikovatelnou ve stavu s je definována jako:
  - $y(s, a) = (s - effects^-(a)) \cup effects^+(a)$

STRIPS: plánovací domény [Wic11]

- Nechť $\mathcal{L}$ je jazyk prvého řádu bez funkcí.
- **STRIPS plánovací doména** na $\mathcal{L}$ omezený stavově-přechodový systém $\Sigma = (S, A, \gamma)$ takový, že:
  - Sje množina STRIPS stavů, tj. množina uzavřených atomů,
  - A je množina uzavřených instancí nějakých STRIPS plánovacích operátorů O
  - $y : S \times A \rightarrow S$ kde
    - $y(s, a) = (s - effects^-(a)) \cup effects^+(a)$, jestliže a je aplikovatelná v s
    - $y(s, a) =$ nedefinovaná jinak
  - S je uzavřená v rámci y

STRIPS: plánovací úloha [Wic11]

- **STRIPS plánovací úloha** je trojice $\mathcal{P} = (\Sigma, s_i, g)$, kde:
  - $\Sigma = (S, A, \gamma)$ je STRIPS plánovací doména na daném jazyce prvého řádu $\mathcal{L}$
  - $s_i \in S$ je počáteční stav
  - g je množina uzavřených literálů popisující cíl tak, že množina cílových stavů je $S_g = \{s \in S | s \models g\}$

Příklad STRIPS plánovací úlohy [Wic11]

Stav v DWR plánovací úloha

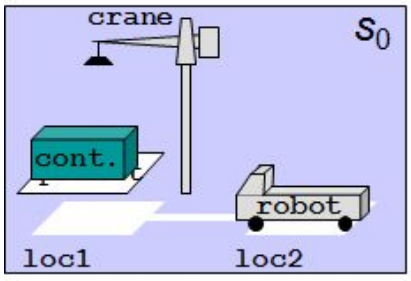
- Σ: STRIPS plánovací doména DWR
- s_i: jakýkoliv stav

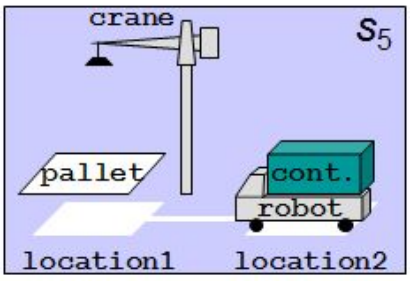
$$
\begin{aligned}
s_0 = \{&attached(pile, loc_1),\\
&in(cont, pile), top(cont, pile),\\
&on(cont, pallet), belong(crane, loc_1),\\
&empty(crane), adjacent(loc_1, loc_2),\\
&adjacent(loc_2, loc_1),\\
&at(robot, loc_2),\\
&occupied(loc_2),\\
&unloaded(robot)\}
\end{aligned}
$$

- $g \subset L$

$$
\begin{aligned}
g = \{&\neg unloaded(robot),\\
&at(robot, loc_2)\}
\end{aligned}
$$

tj. $S_g = \{s_5\}$





Základy PDDL [Wic11]

Planning Domain Definition Language (PDDL)

- http://cs-www.cs.yale.edu/homes/dvm/
- vlastnosti jazyka (verze 1.x):
  - základní forma STRIPS akcí
  - různá rozšíření jako explicitní požadavky
- používán k definicím:
  - plánovacích domén:
    - požadavků,
    - typů,
    - predikátů,
    - možných operátorů
  - plánovacích problémů:
    - objekty,
    - rigidní a průběžné relace,
    - počáteční situace,
    - popis cíle.
- dnes již verze 3.x

Plánovací doména "Opice"

```
(define (domain MONKEY)
  (:requirements :strips :typing)
  (:types monkey box location fruit)
  (:predicates
    (isClear ?b - box)                  (onBox ?m - monkey ?b - box)
    (onFloor ?m - monkey)               (atM ?m - monkey ?loc - location)
    (atB ?b - box    ?loc - location)   (atF ?f - fruit  ?loc - location)
    (hasFruit ?m - monkey ?fruit))
  (:action GOTO
    :parameters (?m - monkey ?loc1 ?loc2 - location)
    :precondition (and (onFloor ?m) (atM ?m ?loc1))
    :effect (and (atM ?m ?loc2) (not (atM ?m ?loc1))))
  (:action PUSH
    :parameters (?m - monkey ?b - box ?loc1 ?loc2 - location)
    :precondition (and (onFloor ?m) (atM ?m ?loc1) (atB ?b ?loc1) (isClear ?b))
    :effect (and (atM ?m ?loc2) (atB ?b ?loc2)
                 (not (atM ?m ?loc1))
                 (not (atB ?b ?loc1))))
  (:action CLIMB
    :parameters (?m - monkey ?b - box ?loc1 - location)
    :precondition (and (onFloor ?m) (atM ?m ?loc1) (atB ?b ?loc1) (isClear ?b))
    :effect (and (onBox ?m ?b) (not (isClear ?b)) (not (onFloor ?m)))  )
  (:action GRAB-FRUIT
    :parameters (?m - monkey ?b - box ?f - fruit ?loc1 - location)
    :precondition (and (onBox ?m ?b) (atB ?b ?loc1) (atF ?f ?loc1))
    :effect (and (hasFruit ?m ?f))))
```

Plánovací problém "Opice"

```
(define (problem MONKEY1)
  (:domain MONKEY)
  (:objects monkeyJudy - monkey
            bananas - fruit
            boxA - box
            locX locY locZ  - location)
  (:init (and
    (onFloor monkeyJudy)
    (atM monkeyJudy locX)
    (atB boxA locY)
    (atF bananas locZ)
    (isClear boxA)
    )
  )

  (:goal (and (hasFruit monkeyJudy bananas))
)
```


```
)
```


Řešení plánovacího problému "Opice"

```
Begin plan
1 (goto monkeyjudy locx locy)
2 (push monkeyjudy boxa locy locz)
3 (climb monkeyjudy boxa locz)
4 (grab-fruit monkeyjudy boxa bananas locz)
End plan
```

STRIPS reprezentace - příklad

- pocatecni stav: *start*
- *cil: writeValue(a, 3)*
- *akce:*

```
startPlan:      PAR [] 
                PRE [start]
                ADD [declared(server, a), declared(server, b)]
                DEL [start]
connectServer:  PAR [A]
                PRE [declared(server, A)]
                ADD [bound(server,A)]
                DEL [declared(server, A)]
writing:        PAR [a, 3]
                PRE [bound(server,a)]
                ADD [writeValue(a, 3)]
                DEL []
```

Výsledek příkladu

- cíl: *writeValue(a,3);*
- *výsledná sekvence:*
  1. *start*
  2. *connectServer(a)*
  3. *writing(a, 3)*

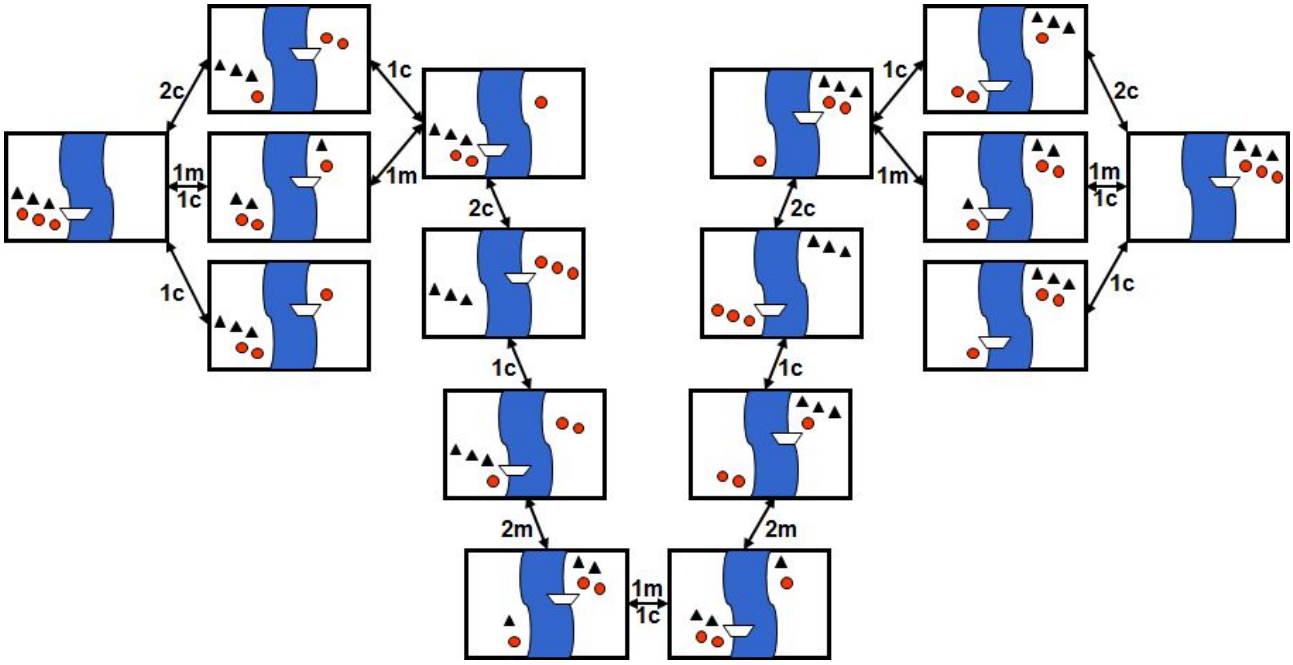
Plánování - hledání logických formulí do hloubky

```
% Depth first search
% ==================
depthFirstSearch(AnsPath) :-
  initialState(Init),
  depthFirst([Init], AnsPath).

depthFirst([S|_], [S]) :-
  finalState(S), !.
depthFirst([S|Path], [S|AnsPath]) :-
  extend([S|Path], S1),
  depthFirst([S1, S |Path], AnsPath).

extend([S|Path], S1) :-
  nextState(S, S1),
  not(memberState(S1, [S|Path])).

memberState(S, Path) :-
  member(S,Path).
```

Plánování - specifikace úlohy v Prologu

```
% Farmer, Wolf, Goat, Cabbage
% ===========================
initialState([n,n,n,n]).
finalState([s,s,s,s]).

nextState(S, S1) :-  move(S, S1), safe(S1).

move([F, W, G, C], [F1,  W,  G,  C]) :- cross(F, F1).
move([F, F, G, C], [F1, F1,  G,  C]) :- cross(F, F1).
move([F, W, F, C], [F1,  W, F1,  C]) :- cross(F, F1).
move([F, W, G, F], [F1,  W,  G, F1]) :- cross(F, F1).

safe([F, W, G, C]) :- F=G, !; F=W, F=C.

cross(n,s).
cross(s,n).
%-------
t1(AnsPath) :- depthFirstSearch(AnsPath).
```

Prohledávání stavového prostoru [Wic11]



hra Misionáři a kanibalové

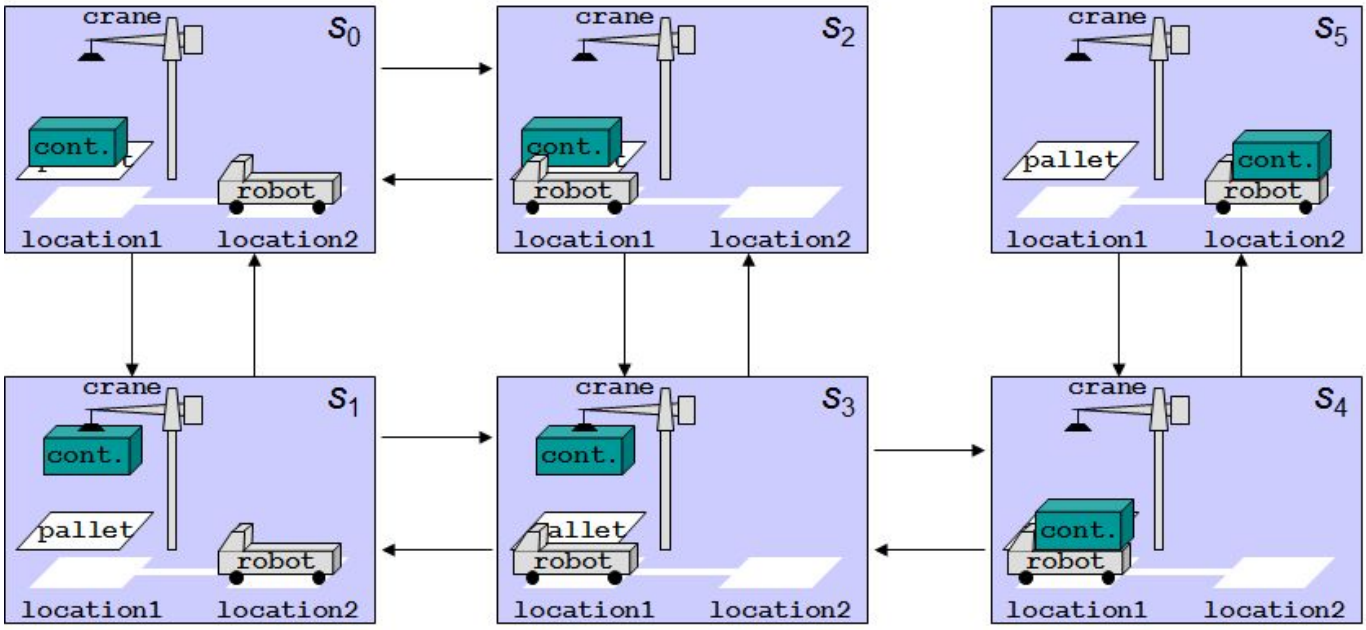
- přepravit 3 kanibaly a 3 misionáře přes řeku
- jakmile je někde více kanibalů než misionářů, jsou misionáři sněděni

Plánování ve stavovém prostoru [Wic11]

Myšlenka:

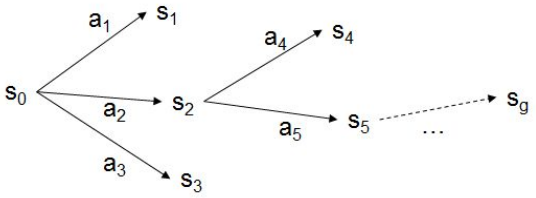
- aplikace standardních prohledávacích algoritmu
  - do šířky,
  - do hloubky,
  - A^*, atd.
- k řešení plánovací úlohy
  - prohledavaný prostor je podmnožina stavového prostoru
  - uzly odpovídají stavům světa
  - hrany korespondují přechodům mezi stavy
  - cesta v prohledávaném prostoru odpovídá plánu

Plánování ve stavovém prostoru - příklad [Wic11]



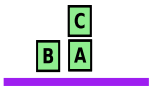
- uzly: zavřené atomy
- hrany: akce (uzavřené instance operátorů)

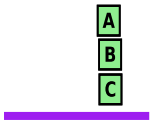
Progresivní hledání [Wic11]



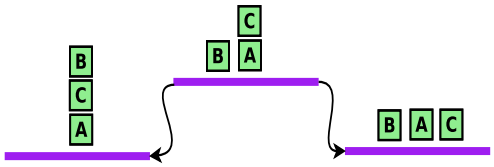
1. Forward-search(O, s_0, g)
   1. $s \leftarrow s_0$
   2. $\pi \leftarrow$ the empty plan
   3. loop
      1. jestliže $s \models g$, potom vrať π
      2. $E \leftarrow \{a | a$ je uzavřená instance operátoru $\in O$ a $precond(a)$ je pravdivá v $s\}$
      3. jestliže $E == \emptyset$, potom vrať *FAILURE*
      4. nedeterministicky vyber akci $a \in E$
      5. $s \leftarrow \gamma(s, a)$
      6. $\pi \leftarrow \pi.a$

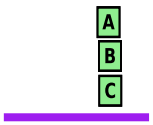
Příklad prohledávání prostoru stavů 1 [Wic11]



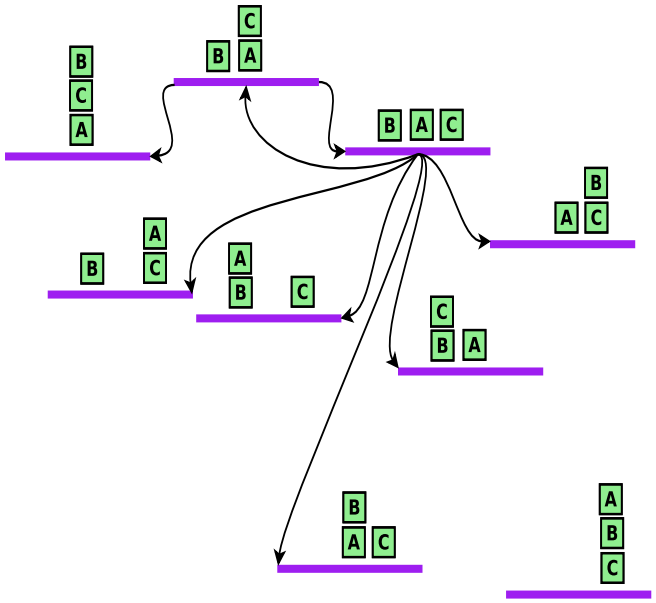


Příklad prohledávání prostoru stavů 2 [Wic11]

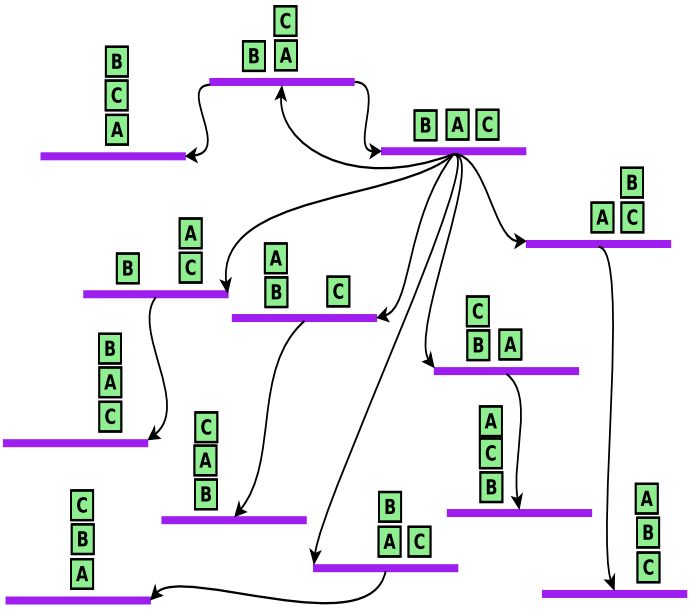




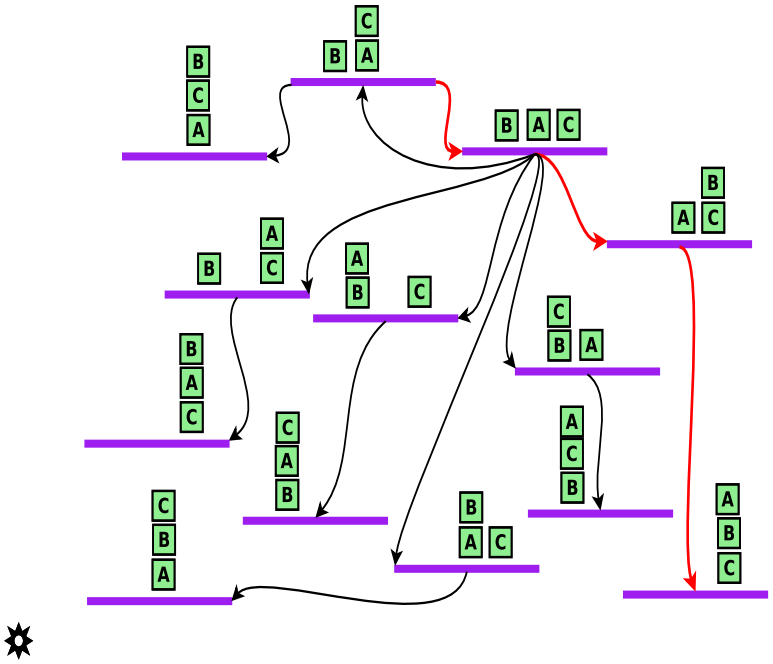
Příklad prohledávání prostoru stavů 3 [Wic11]



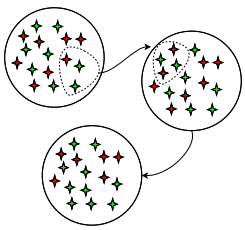
Příklad prohledávání prostoru stavů 4 [Wic11]



Příklad prohledávání prostoru stavů 5 [Wic11]

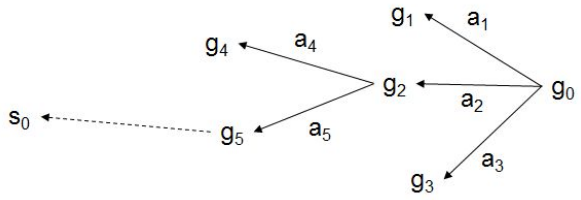


Relevantní akce [Nau09]



- Nechť $\mathcal{P} = (\Sigma, s_i, g)$ je STRIPS plánovací úloha.
- Akce a je **relevantní** pro cíl g, jestliže
  - a způsobí, že alespoň jeden z literálů g je pravdivý
    - $g \cap \mathit{effects}(a) \neq \emptyset$
  - a nezpůsobí, že ani jeden z literálů g je nepravdivý
    - $g^+ \cap \mathit{effects}^-(a) = \emptyset \bigwedge g^- \cap \mathit{effects}^+(a) = \emptyset$
- **Regresní množina** cíle g pro relevatní akci $a \in A$ je:
  - $\gamma^{-1}(g, a) = (g - \mathit{effects}(a)) \cup \mathit{precond}(a)$

Zpětné hledání [Wic11]

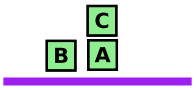


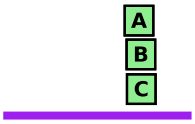
1. Backward-search(O, s_0, g)
   1. $\pi \leftarrow$ the empty plan
   2. loop
      1. jestliže $s_0 \models g$, potom vrať π
      2. $A \leftarrow \{a | a$ je uzavřená instance operátoru $\in O$ a $\gamma^{-1}(g, a)$ je definováno $\}$
      3. jestliže $A == \emptyset$, potom vrať *FAILURE*
      4. nedeterministicky vyber akci $a \in A$
      5. $\pi \leftarrow a.\pi$
      6. $g \leftarrow \gamma^{-1}(s, a)$

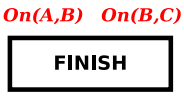
Sussmanova anomálie

- interakce podcílů, jejichž řešení se musí proložit, aby je bylo možné splnit současně,

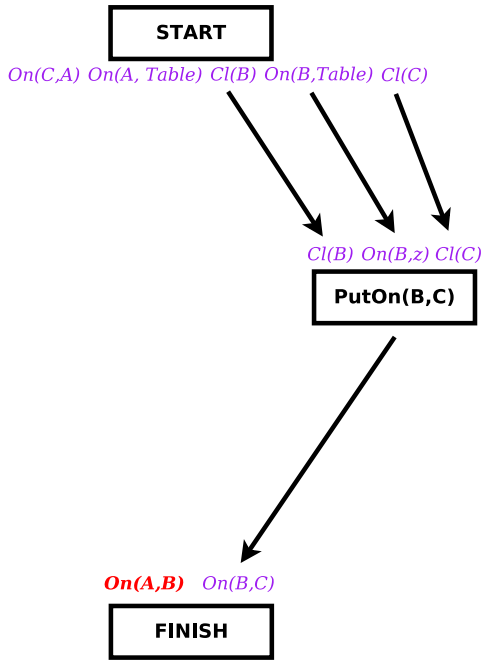
Sussmanova anomálie - příklad s kostkami I [Nau09]

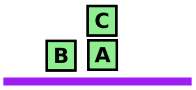


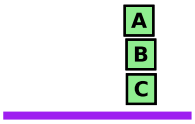




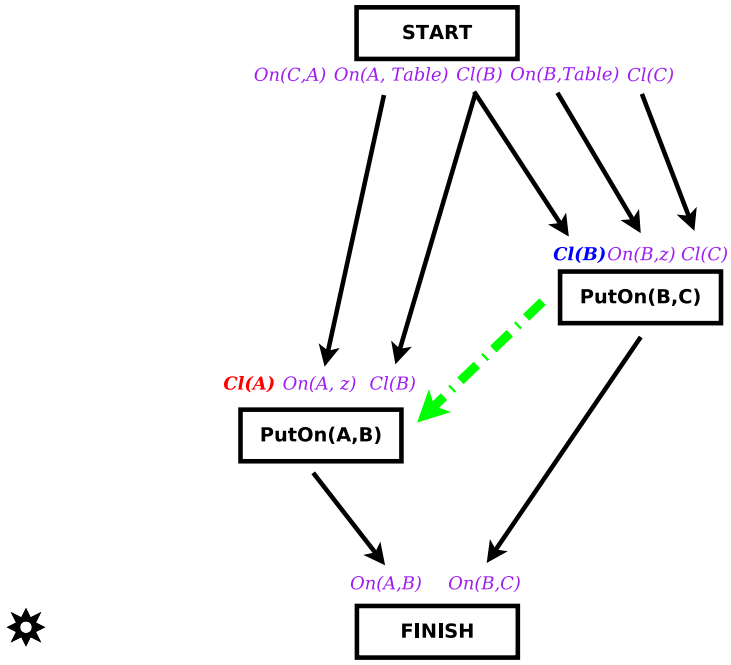
Sussmanova anomálie - příklad s kostkami II [Nau09]

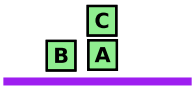


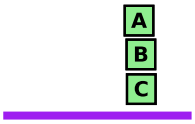




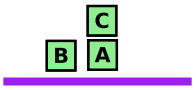
Sussmanova anomálie - příklad s kostkami III [Nau09]

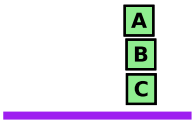






Sussmanova anomálie - příklad s kostkami IV [Nau09]





Prohledávání prostoru stavů vs. plánů [Wic11]

prohledávání stavového prostoru

- prohledávání grafu, jehož uzly reprezentují stavy světa

prohledávání prostoru plánů

- prohledávání grafu, jehož uzly reprezentují částečné plány
- uzly: částečně určené plány
- hrany: operace zjemnění plánů
- řešení: částečně uspořádané plány
- částečný plán:
  - podmožina akcí
  - podmnožina organizační struktury
    - časové uspořádání akcí
    - zdůvodnění: co akce znamená pro plán
  - podmnožina vazeb proměnných

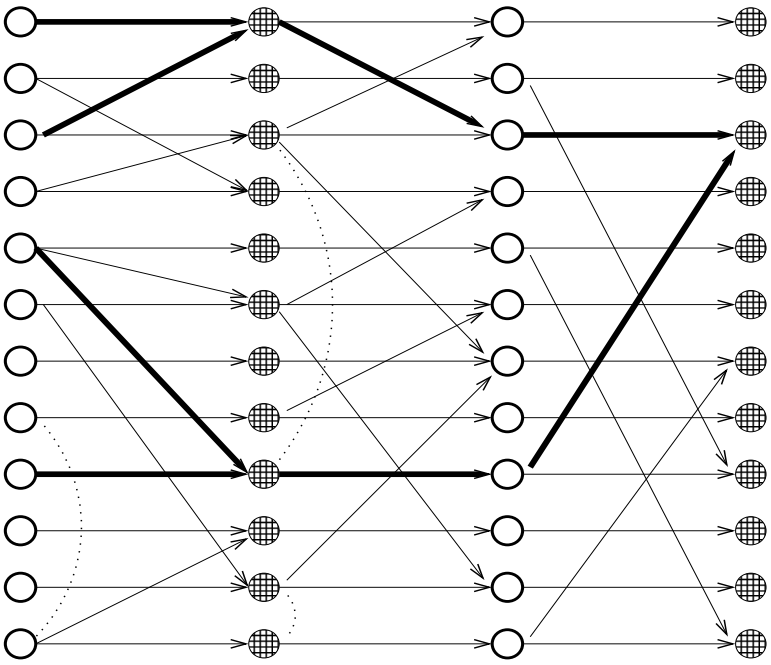
Plánování v prostoru plánů - omezující podmínky [Nau09]

- podmínka předcházení
  - a musí předcházet b
- vazební podmínka
  - podmínky nerovnosti, např. $v_1 \neq v_2$ nebo $v_1 \neq c$
  - podmínky rovnosti a substituce, např. $v_1 = v_2$ nebo $v_1 = c$
- kauzální vazby
  - použij akci a k vytvoření podmínky p potřebné pro akci b

GRAPHPLAN plánovač

- 1997
- plány jsou reprezentovány pomocí *plánovacího grafu*,
  - myšlenka je velmi podobná dynamickému programování či řešení toku sítí,
- Všechny plány jsou konstruovány souběžně.
  - rozšiřování grafu (dopředný běh)
  - vyhledání plánu (zpětný běh)
- Plánovač udržuje relaci binární vzájemné výlučnosti (*mutex*) mezi uzly representující aplikované akce a výroky popisující stav.
- Problém s cyklením odstraněn.
- Nelze používat parametrizované akční schémata (instance).
  - Vytváří obrovský prostor výroků.
- Existuje řada podpůrných strategií, které podstatně urychlují plánovaní.
- Implementace jsou schopny zvládnout plány s 50-100 voláními metod do minut.

GraphPlan - plánovací graf



Implementace plánovačů

Počáteční pokusy

- STRIPS [1971] . . . , první plánovač, zpětné plánovaní dle podmínek akcí

Prostor stavů/plánů

- WARPLAN [1973] . . . lineární plánovač, Sussmanova anomálie řešení posouvání akcí
- PWEAK, TWEAK [1987], UCPOP [1992] . . . plánovače s častečným uspořádáním

Plánovací grafy

- GRAPHPLAN[1997] . . . první plánovač s plánovacím grafem
- Blackbox [1998] . . . kombinuje GRAPHPLAN a SATPLAN
- FF [2000] . . . plánovací graf jako heuristika s velmi rychlým dopředným a lokálním prohledáváním

PDDL domény [Wic11]

<domain> ::=
(define (domain <name>)
[<extension-def>]
[<require-def>]
[<types-def>]:typing
[<constants-def>]
[<domain-vars-def>]:expression-evaluation
[<predicates-def>]
[<timeless-def>]
[<safety-def>]:safety-constraints
<structure-def>*)

<extension-def> ::=
(:extends <domain name>+)
<require-def> ::=
(:requirements <require-key>+)
<require-key> ::=
:strips | :typing | ...

<types-def> ::= (:types <typed list (name)>)
<constants-def> ::=
(:constants <typed list (name)>)
<domain-vars-def> ::=
(:domain-variables
<typed list(domain-var-declaration)>)
<predicates-def> ::=
(:predicates <atomic formula skeleton>+)
<atomic formula skeleton> ::=
(<predicate> <typed list (variable)>)
<predicate> ::= <name>
<variable> ::= ?<name>
<timeless-def> ::=
(:timeless <literal (name)>+)
<structure-def> ::= <action-def>
<structure-def> ::=:domain-axioms <axiom-def>
<structure-def> ::=:action-expansions <method-def>

PDDL typy [Wic11]

- PDDL types syntax

<typed list (x)> ::= x*

<typed list (x)> ::=:typing
x+ - <type> <typed list(x)>

<type> ::= <name>

<type> ::= (either <type>+)

<type> ::=:fluents (fluent <type>)

PDDL příklad: DWR typy [Wic11]

```
(define (domain dock-worker-robot)

    (:requirements :strips :typing )

    (:types
        location     ;there are several connected locations
        pile         ;is attached to a location,
                     ;it holds a pallet and a stack of containers
        robot        ;holds at most 1 container,
                     ;only 1 robot per location
        crane        ;belongs to a location to pickup containers
        container )

...)
```

PDDL predikáty: příklad [Wic11]

```
(:predicates
    (adjacent ?l1 ?l2 - location)          ;location ?l1 is adjacent to ?l2
    (attached ?p - pile ?l - location)     ;pile ?p attached to location ?l
    (belong ?k - crane ?l - location)      ;crane ?k belongs to location ?l

    (at ?r - robot ?l - location)          ;robot ?r is at location ?l
    (occupied ?l - location)               ;there is a robot at location ?l
    (loaded ?r - robot ?c - container )    ;robot ?r is loaded with container ?c
    (unloaded ?r - robot)                  ;robot ?r is empty

    (holding ?k - crane ?c - container)    ;crane ?k is holding a container ?c
    (empty ?k - crane)                     ;crane ?k is empty

    (in ?c - container ?p - pile)          ;container ?c is within pile ?p
    (top ?c - container ?p - pile)         ;container ?c is on top of pile ?p
    (on ?c1 - container ?c2 - container);container ?c1 is on container ?c2
)
```

PDDL akce [Wic11]

<action-def> ::=
(:action <action functor>
:parameters (<typed list (variable)>)
<action-def body>)
<action functor> ::= <name>
<action-def body> ::=
[:vars (<typed list(variable)>)]:existential-preconditions :conditional-effects
[:precondition <GD>]
[:expansion <action spec>]:action−expansions
[:expansion :methods]:action−expansions
[:maintain <GD>]:action−expansions
[:effect <effect>]
[:only-in-expansions <boolean>]:action−expansions

PDDL popis cíle [Wic11]

<GD> ::= <atomic formula(term)>
<GD> ::= (and <GD>+)
<GD> ::= <literal(term)>
<GD> ::=:disjunctive−preconditions (or <GD>+)
<GD> ::=:disjunctive−preconditions (not <GD>)
<GD> ::=:disjunctive−preconditions (imply <GD> <GD>)
<GD> ::=:existential−preconditions (exists (<typed list(variable)>) <GD>)
<GD> ::=:universal−preconditions (forall (<typed list(variable)>) <GD>)
<literal(t)> ::= <atomic formula(t)>
<literal(t)> ::= (not <atomic formula(t)>)
<atomic formula(t)> ::= (<predicate> t*)
<term> ::= <name>

PDDL efekty [Wic11]

<effect> ::= (and <effect>+)

<effect> ::= <atomic formula(term)>

<effect> ::= (not <atomic formula(term)>)

<effect> ::=:conditional-effects
(forall (<variable>*) <effect>)

<effect> ::=:conditional-effects
(when <GD> <effect>)

<effect> ::=:fluents (change <fluent> <expression>)

PDDL operátor: příklad [Wic11]

```
;; moves a robot between two adjacent locations
(:action move
   :parameters (?r - robot ?from ?to - location)
   :precondition (and
      (adjacent ?from ?to) (at ?r ?from)
      (not (occupied ?to)))
   :effect (and
      (at ?r ?to) (occupied ?to)
      (not (occupied ?from)) (not (at ?r ?from)) ))
```

PDDL problém [Wic11]

<problem> ::= (define (problem <name>)
(:domain <name>)
[<require-def>]
[<situation>]
[<object declaration>]
[<init>]
<goal>+
[<length-spec>])
<object declaration> ::= (:objects <typed list (name)>)
<situation> ::= (:situation <initsit name>)
<initsit name> ::= <name>
<init> ::= (:init <literal(name)>+)
<goal> ::= (:goal <GD>)
<goal> ::=:action–expansions (:expansion <action spec(action-term)>)
<length-spec> ::= (:length [(:serial <integer>)] [(:parallel <integer>)])

PDDL problém: DWR příklad [Wic11]

```
;; a simple DWR problem with 1 robot and 2
locations
(define (problem dwrpb1)
    (:domain dock-worker-robot)
    (:objects
        r1 - robot
        l1 l2 - location
        k1 k2 - crane
        p1 q1 p2 q2 - pile
        ca cb cc cd ce cf pallet - container)
    (:init
        (adjacent l1 l2)
        (adjacent l2 l1)
        (attached p1 l1)
        (attached q1 l1)
        (attached p2 l2)
        (attached q2 l2)
        (belong k1 l1)
        (belong k2 l2)

        (in ca p1) (in cb p1) (in cc p1)
        (on ca pallet) (on cb ca) (on cc cb)
        (top cc p1)

        (in cd q1) (in ce q1) (in cf q1)
        (on cd pallet) (on ce cd) (on cf ce)
        (top cf q1)

        (top pallet p2)
        (top pallet q2)

        (at r1 l1)
        (unloaded r1)
        (occupied l1)

        (empty k1)
        (empty k2))

    ;; task is to move all containers to locations l2
    ;; ca and cc in pile p2, the rest in q2
    (:goal (and
        (in ca p2) (in cc p2)
        (in cb q2) (in cd q2) (in ce q2) (in cf q2))))
```

Literatura I

Dana Nau.
CMSC 722, ai planning (fall 2009), lecture notes.
http://www.cs.umd.edu/class/fall2009/cmsc722/, 2009.

Michal Pechoucek.
A4m33pah, lecture notes.
http://cw.felk.cvut.cz/doku.php/courses/a4m33pah/prednasky, February 2010.

Gerhard Wickler.
A4m33pah, lecture notes.
http://cw.felk.cvut.cz/doku.php/courses/a4m33pah/prednasky, February 2011.