Plánování a rozvrhování

5

Roman Barták, KTIML

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

Na úvod

Klasické plánování

- **uzly** prohledávaného prostoru odpovídají **částečným plánům**
- **řešení**, které je z daného uzlu dosažitelné, **obsahuje všechny akce** částečného plánu

- plánování ve stavovém prostoru
- plánování v prostoru plánů

Neoklasické plánování

- **uzly** prohledávaného prostoru odpovídají **několika částečným plánům**
- **ne každá akce** z částečných plánů v uzlu se objeví **v řešení**, které je z daného uzlu dosažitelné

- plánování s plánovacím grafem

Problém dosavadních technik

- **volba špatné akce**, která se projeví pozdě, vede k neefektivnímu prohledávání
- **velký větvící faktor** je prapůvodcem tohoto problému, protože dává k dispozici na výběr hodně akcí

Jak objevit „dobré" akce?

- **budeme řešit „relaxovaný" problém**, jehož řešení bude obsahovat všechna řešení původního problému
  - **relaxovaný problém** = z původního problému odstraníme některá omezení (např. negativní efekty)
- při výběru akcí budeme zkoušet pouze akce z řešení relaxovaného problému

Co pro to potřebujeme?

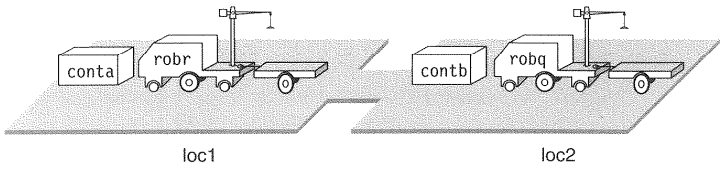
- reprezentace několika plánů najednou tak, že ne všechny akce z reprezentace se musí objevit v konečném řešení

Plánování
s plánovacím grafem

Základní myšlenky

- Místo strategie nejmenších závazků (least-commitment) jdeme opačným směrem, **strategie silných závazků**:
  - akce jsou plně instaciovány
  - je zvoleno jejich místo v plánu
- Plánování s plánovacím grafem je založeno na dvou principech:
  - **analýza dosažitelnosti**
    - zjišťujeme, zda je daný stav dosažitelný z počátku v daném počtu kroků
  - **disjunktivní zjemňování**
    - kazy opravujeme použitím disjunkce rozkladů a zachycením interferencí rozkladů formou omezení

Příprava



move(r, l, l') ;; robot r at location l moves to a connected location l'
precond: at(r, l), adjacent(l, l')
effects: at(r, l'), ¬ at(r, l)

load(c, r, l) ;; robot r loads container c at location l
precond: at(r, l), in(c, l), unloaded(r)
effects: loaded(r, c), ¬ in(c, l), ¬ unloaded(r)

unload(c, r, l) ;; robot r unloads container c at location l
precond: at(r, l), loaded(r, c)
effects: unloaded(r), in(c, l), ¬ loaded(r, c)

Pracovat budeme s instaciovanými atomy a akcemi, jejichž zápis zkrátíme:

Atomy:

- r_1, r_2, q_1, q_2 – pozice robota
- a_1, a_2, a_r, a_q, b_1, b_2, b_r, b_q – pozice kontejneru
- u_r,u_q – robot je prázdný

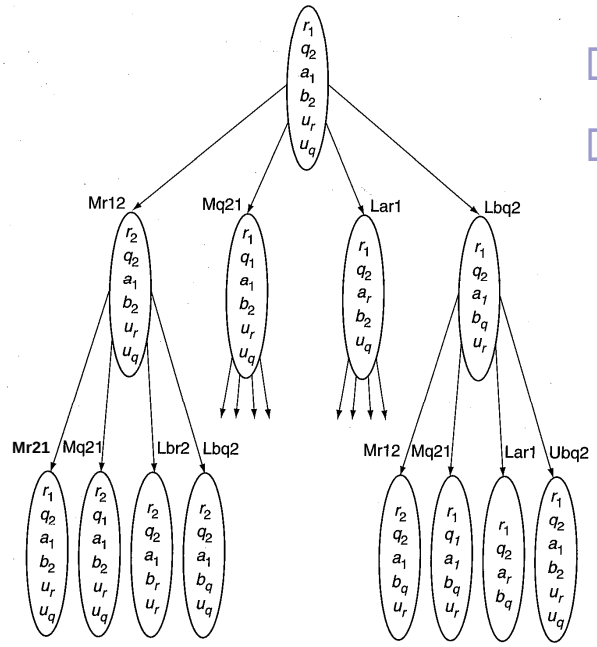
Počáteční stav je tedy $\{r_1,q_2,a_1,b_2,u_r,u_q\}$.

Akce:

- Mr12, Mr21, Mq12, Mq21 – pohyb robota
- Lar1, Lar2, Laq1, Laq2, Lbr1, Lbr2, Lbq1, Lbq2 – naložení kontejneru na robota
- Uar1, Uar2, Uaq1, Uaq2, Ubr1, Ubr2, Ubq1, Ubq2 – vyložení kontejneru z robota

Strom dosažitelnosti

Uzly odpovídají stavům a hrany přechodům mezi stavy



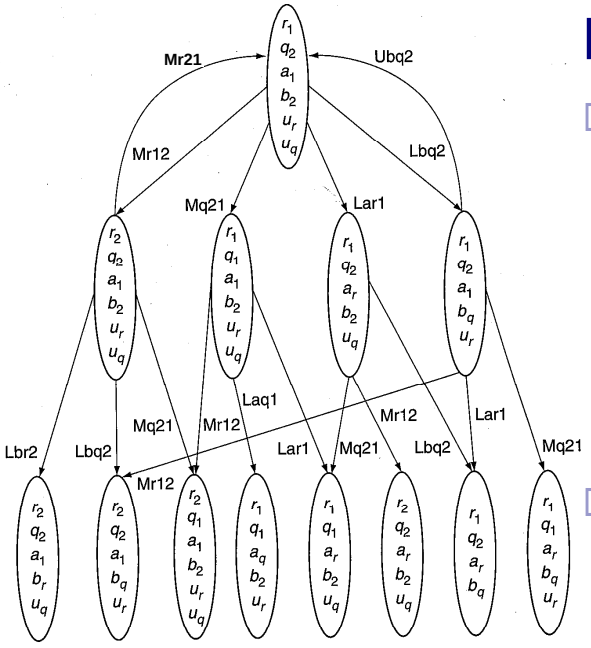
- **Kořen** – stav s_0
- Strom dosažitelnosti hloubky d s kořenem s_0 **řeší všechny plánovací problémy**, kde je cílový stav dosažitelný z s_0 za d nebo méně akcí.
  - **Plán existuje, pokud je cílový stav někde ve stromu dosažitelnosti!**

Problém:

Strom dosažitelnosti obsahuje $O(k^d)$ uzlů, pro k=počet akcí použitelných na stav.

Graf dosažitelnosti

- Některé stavy ve stromu dosažitelnosti se opakují (stejný stav lze dosáhnout různými cestami). Tyto stavy není třeba rozlišovat různými uzly!



Dostáváme **graf dosažitelnosti**

- **Problém**
  - Graf dosažitelnosti je pořád **neprakticky velký**.
    - až tak velký, že počet uzlů odpovídá počtu stavů v doméně
- **Co s tím?**
  - Zkusme analýzu dosažitelnosti udělat trochu volněji!

Volnější dosažitelnost

Graf dosažitelnosti poskytuje nutnou a postačující podmínku dosažitelnosti daného stavu.

- stav je dosažitelný právě tehdy, když se vyskytuje v grafu dosažitelnosti

Plánovací graf poskytuje pouze nutnou podmínku dosažitelnosti.

- pokud je stav dosažitelný, potom se vyskytuje v plánovacím grafu
- ale ne všechny stavy, které jsou v plánovacím grafu jsou skutečně dosažitelné

K čemu nám bude plánovací graf?

- pokud budeme schopni sestrojit takový plánovací graf rychle s malou spotřebou paměti, poskytne nám náhled zda a za jak dlouho můžeme dosáhnout cílový stav
- po konstrukci plánovacího grafu bude potřeba extrahovat příslušný plán

Aproximace stavů

Jak aproximovat graf/strom dosažitelnosti?

- uzly popisují aproximace stavů na dané úrovni

Jak aproximovat stav?

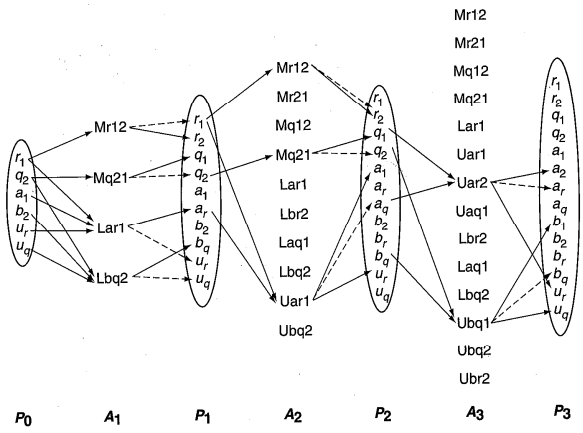
Připomeňme, že stav je množina instanciovaných atomů.

- aproximace stavu pomocí sjednocení atomů ze stavů dosažitelných na dané úrovni
- **Jak to uděláme?**
  - Na daný stav aplikujeme paralelně všechny použitelné akce, u kterých budeme ignorovat negativní efekty.
  - Důsledek: počet atomů v uzlech se nikdy nezmenší.
  - Bude dobré si vyznačit, která akce je zodpovědná za přidání atomu a jaký atom by chtěla smazat.

Plánovací graf

Plánovací graf je orientovaný vrstvený graf, kde

- každá vrstva obsahuje buď (exkluzivně)
  - instanciované atomy (stavová vrstva) nebo
  - akce (akční vrstva)
- **stavové a akční vrstvy** se střídají
  - nultá vrstva popisuje počáteční stav
  - následuje vrstva popisující akce použitelné na počáteční stav
  - poslední vrstva je stavová
- akční vrstva a následující stavová vrstva tvoří **úroveň**
- **hrany** vedou
  - z atomů do akce, pro kterou jsou to předpoklady
  - z akce do atomů popisujících efekty
    - hrany pro negativní efekty jsou označeny zvlášť
    - negativní efekty se ze stavu nemažou!



Vrstvený plán

Jak to bude s plány v plánovacím grafu?

- Místo jedné posloupnosti akcí budeme pracovat s posloupností množin akcí – **vrstvený plán**.
  - množina akcí na pozici j bude podmnožinou akcí z úrovně j
- Z vrstveného plánu se získá sekvenční plán tak, že se vezme libovolná permutace akcí z první množiny následovaná libovolnou permutací akcí z druhé množiny atd.

Příklad:

Vrstvený plán ⟨{a1,a2},{a3,a4},{a5,a6,a7}⟩ popisuje 2x2x6=24 sekvenčních plánů.

Nezávislé akce

- **Jak zaručíme, že akce vybrané do množiny ve vrstveném plánu můžeme použít v libovolném pořadí a vždy dospějeme do stejného stavu?**
  - Tyto akce musí být na sobě nezávislé!
- **Kdy jsou akce (a,b) na sobě závislé?**
  - Když je nelze uspořádat zcela libovolně, konkrétně:
    - a maže předpoklad b (tj. a nemůže být před b)
    - a maže pozitivní efekt b (výsledek pak záleží na pořadí akcí)
    - symetricky pro b
- **Akce (a,b) jsou nezávislé**, právě když:
  - $effects^-(a) \cap (precond(b) \cup effects^+(b)) = \varnothing$
  - $effects^-(b) \cap (precond(a) \cup effects^+(a)) = \varnothing$

Poznámka:
Nezávislost akcí je dána doménou, ne konkrétním plánovacím problémem!

Aplikace množiny akcí

Máme-li **množinu π navzájem nezávislých akcí**, potom

- **π je použitelná na stav s** právě když
  - předpoklady všech akcí jsou splněny
  - $precond(\pi) = \cup \{precond(a) \mid \forall a \in \pi\} \subseteq s$
- **výsledkem aplikace π na stav s** je stav
  - $\gamma(s,\pi) = (s - effects^-(\pi)) \cup effects^+(\pi)$
    - $effects^{-/+}(\pi)$ jsou sjednocení příslušných efektů akcí z π

Tvrzení:

- Je-li π množina navzájem nezávislých akcí použitelná na stav s, potom pro libovolnou permutaci akcí $\langle a_1,a_2,\ldots,a_k\rangle$ z π platí $\gamma(s,\pi)= \gamma(\ldots\gamma(\gamma(s,a_1),a_2)\ldots a_k)$.

Toto tvrzení ospravedlňuje použití vrstvených plánů!

Řešící vrstvený plán

- Vrstvený plán $\Pi=\langle\pi_1,\pi_2,\ldots,\pi_k\rangle$ je **řešením plánovacího problému** (O,s_0,g) právě když:
  - každá množina akcí π_i je nezávislá,
  - množina π_1 je použitelná na stav s_0, π_2 je použitelná na stav $\gamma(s_0,\pi_1)$ atd.,
  - $g \subseteq \gamma(\ldots\gamma(\gamma(s_0,\pi_1),\pi_2)\ldots\pi_k)$.

Tvrzení:

Je-li Π řešícím plánem problému (O,s_0,g) potom posloupnost akcí odpovídající libovolné permutaci prvků z π_1, následovaná libovolnou permutací z π_2 atd. převádí stav s_0 na stav splňující g.

Plánujeme

Jak budeme plánovat s plánovacím grafem?

- **Sestrojíme plánovací graf** tak, aby poslední stavová vrstva splňovala cílovou podmínku.
  - Přesněji, budeme chtít, aby v poslední vrstvě bylo možné mít všechny cílové atomy **najednou**.
- Z akčních vrstev **vybereme množiny nezávislých akcí** tak, abychom dostali všechny cílové atomy.
  - To můžeme dělat **zpětným chodem** od poslední úrovně, kde vybereme akce, které nám dají cíl, v předposlední úrovni vybereme akce, které dávají jejich předpoklady, atd.
  - Některý cílový atom ale může být splněn už v předchozí úrovni.

    Zajistíme, aby každý atom ve stavové vrstvě byl efektem nějaké akce z předchozí vrstvy.
    - Použijeme no-op akce, které mají stejný předpoklad (jeden atom), jako jediný svůj pozitivní efekt: **α_p je no-op akce pro p**, tj. $\text{precond}(\alpha_p)=\text{effect}^+(\alpha_p)=\{p\}$, $\text{effect}^-(\alpha_p)=\varnothing$

Mutex atomů

Jak zjistíme, zda dva atomy mohou/nemohou být společně v jedné vrstvě?

- **V nulté vrstvě** jsou dohromady **všechny atomy** (je to počáteční stav).
- **Dvě závislé akce** se **nemohou vyskytnout najednou** v první akční vrstvě a tudíž jejich pozitivní efekty nemohou být najednou v první stavové vrstvě (tedy pokud zrovna nejsou také pozitivními efekty jiných nezávislých akcí).
- **Dva atomy** také **nemohou být společně** v jedné vrstvě, pokud se jedná o pozitivní a negativní efekt téže akce (opět, pokud tyto atomy nepřidají jiné nezávislé akce).
  - *Máme-li no-op akce, jedná se o speciální případ předchozí podmínky (pokud b maže p, potom jsou α_p a b závislé).*
- Hovoříme o nekompatibilitě dvou atomů/akcí, resp. o jejich vzájemné výlučnosti (**mutual exclusion - mutex**).

Mutex akcí

Level	*Mutex elements*
A_1	{Mr12} × {Lar1}
	{Mq21} × {Lbq2}
P_1	$\{r_2\} \times \{r_1, a_r\}$
	$\{q_1\} \times \{q_2, b_q\}$
	$\{a_r\} \times \{a_1, u_r\}$
	$\{b_q\} \times \{b_2, u_q\}$
A_2	{Mr12} × {Mr21,Lar1,Uar1}
	{Mr21} × {Lbr2,Lar1*,Uar1*}
	{Mq12} × {Mq21,Laq1,Lbq2*,Ubq2*}
	{Mq21} × {Lbq2,Ubq2}
	{Lar1} × {Uar1,Laq1,Lbr2}
	{Lbr2} × {Ubq2,Lbq2,Uar1,Mr12*}
	{Laq1} × {Uar1,Ubq2,Lbq2,Mq21*}
	{Lbq2} × {Ubq2}
P_2	$\{b_r\} \times \{r_1, b_2, u_r, b_q, a_r\}$
	$\{a_q\} \times \{q_2, a_1, u_q, b_q, a_r\}$
	$\{r_1\} \times \{r_2\}$
	$\{q_1\} \times \{q_2\}$
	$\{a_r\} \times \{a_1, u_r\}$
	$\{b_q\} \times \{b_2, u_q\}$
A_3	{Mr12} × {Mr21,Lar1,Uar1,Lbr2*,Uar2*}
	{Mr21} × {Lbr2,Uar2,Ubr2}
	{Mq12} × {Mq21,Laq1,Uaq1,Ubq1,Ubq2*}
	{Mq21} × {Lbq2,Ubq2,Laq1*,Ubq1*}
	{Lar1} × {Uar1,Uaq1,Laq1,Uar2,Ubr2,Lbr2,Mr21*}
	{Lbr2} × {Ubr2,Ubq2,Lbq2,Uar1,Uar2,Ubq1*}
	{Laq1} × {Uar1,Uaq1,Ubq1,Ubq2,Lbq2,Uar2*}
	{Lbq2} × {Ubr2,Ubq2,Uaq1,Ubq1,Mq12*}
	{Uaq1} × {Uar1,Uar2,Ubq1,Ubq2,Mq21}*
	{Ubr2} × {Uar1,Uar2,Ubq1,Ubq2,Mr12}*
	{Uar1} × {Uar2,Mr21*}
	{Ubq1} × {Ubq2}
P_3	$\{a_2\} \times \{a_r, a_1, r_1, a_q, b_r\}$
	$\{b_1\} \times \{b_q, b_2, q_2, a_q, b_r\}$
	$\{a_r\} \times \{u_r, a_1, a_q, b_r\}$
	$\{b_q\} \times \{u_q, b_2, a_q, b_r\}$
	$\{a_q\} \times \{a_1, u_q\}$
	$\{b_r\} \times \{b_2, u_r\}$
	$\{r_1\} \times \{r_2\}$
	$\{q_1\} \times \{q_2\}$

Stejně jako závislost akcí vede k nekompatibilitě atomů, nekompatibilní atomy vedou k dalším nekompatibilitám akcí.

- **Dvě akce jsou nekompatibilní**, pokud jsou jejich předpoklady nekompatibilní.
- Akci, jejíž **předpoklady jsou mutex**, můžeme z grafu rovnou vyřadit.

Mutex formálně

Dvě akce *a* a *b* **jsou mutex** v úrovni A_i, pokud:

- *a* a *b* jsou závislé, nebo
- předpoklad *a* je mutex předpokladu *b* v úrovni P_{i-1}.

Množinu akčních mutexů pro A_i budeme značit μA_i.

Dva atomy (výroky) p a q **jsou mutex** v úrovni P_i, pokud:

- každá akce v A_i, která má p jako pozitivní efekt, je mutex každé akce, která má q jako pozitivní efekt, a
- v A_i není žádná akce, která by měla p i q jako pozitivní efekt.

Množinu výrokových mutexů pro P_i budeme značit μP_i.

Vlastnosti mutexu

Mutex je **symetrická relace.**

Množiny mutexů v plánovacím grafu **monotónně klesají**:

- pokud jsou *p* a *q* v P_{i-1} a $(p,q) \notin \mu P_{i-1}$, potom $(p,q) \notin \mu P_i$,
- pokud jsou *a* a *b* v A_{i-1} a $(a,b) \notin \mu A_{i-1}$, potom $(a,b) \notin \mu A_i$.

Důkaz:

- Když dva výroky nejsou mutex, tak jejich no-op akce také nejsou mutex a zajistí proto „přesun" do další vrstvy.
- Pokud dvě akce nejsou mutex, potom jsou nezávislé a jejich předpoklady nejsou mutex. Nezávislost akcí se mezi vrstvami nemění a předpoklady, které nejsou mutex, se do další vrstvy přenesou dle předchozího tvrzení.

Poznámka:

množiny akcí a výroků v plánovacím grafu **monotónně rostou** ($P_{i-1} \subseteq P_i$ a $A_{i-1} \subseteq A_i$).

Graphplan

Plánovací algoritmus založený na plánovacím grafu.

- Střídá fáze expanze grafu a extrakce plánu.
- **Expanze:**
  - Nejprve vytvoří plánovací graf až do vrstvy, kde jsou všechny cílové atomy a žádná dvojice z nich není mutex (to je nutná podmínka existence plánu).
  - Pokud fáze extrakce neuspěje, přidá další vrstvu (může se zastavit při splnění ukončovací podmínky a tím dokázat neexistenci řešení).
- **Extrakce:**
  - Z plánovacího grafu se pokusí vybrat vrstvený plán vedoucí k cílovým atomům.

Technické omezení:
akce nemají negativní předpoklady (to lze ale snadno odstranit)

Expanze grafu

- Plánovací graf uchováváme v podobě seznamu vrstev a mutexů

$$G = \langle P_0, A_1, \mu A_1, P_1, \mu P_1, \ldots, A_i, \mu A_i, P_i, \mu P_i \rangle$$

- Plánovací graf závisí pouze na operátorech O a počátečním stavu s_0 (kódován v P_0), ale nezávisí na cíli g!
- Rozšíření o další úroveň dělá procedura Expand(G):

Expand($\langle P_0, A_1, \mu A_1, P_1, \mu P_1, \ldots, A_{i-1}, \mu A_{i-1}, P_{i-1}, \mu P_{i-1} \rangle$)
  $A_i \leftarrow \{a \in A \mid \text{precond}(a) \subseteq P_{i-1} \text{ and } \text{precond}^2(a) \cap \mu P_{i-1} = \emptyset\}$
  $P_i \leftarrow \{p \mid \exists a \in A_i : p \in \text{effects}^+(a)\}$
  $\mu A_i \leftarrow \{(a,b) \in A_i^2, a \neq b \mid \text{effects}^-(a) \cap [\text{precond}(b) \cup \text{effects}^+(b)] \neq \emptyset$
    or $\text{effects}^-(b) \cap [\text{precond}(a) \cup \text{effects}^+(a)] \neq \emptyset$
    or $\exists (p,q) \in \mu P_{i-1} : p \in \text{precond}(a), q \in \text{precond}(b)\}$
  $\mu P_i \leftarrow \{(p,q) \in P_i^2, p \neq q \mid \forall a,b \in A_i, a \neq b :$
    $p \in \text{effects}^+(a), q \in \text{effects}^+(b) \Rightarrow (a,b) \in \mu A_i\}$
  for each $a \in A_i$ do: link a with precondition arcs to $\text{precond}(a)$ in P_{i-1}
    positive arcs to $\text{effects}^+(a)$ and negative arcs to $\text{effects}^-(a)$ in P_i
  return($\langle P_0, A_1, \mu A_1, \ldots, P_{i-1}, \mu P_{i-1}, A_i, \mu A_i, P_i, \mu P_i \rangle$)
end

Pevný bod

- Počet různých úrovní v plánovacím grafu je omezený. Od určité úrovně už se graf nemění – úroveň pevného bodu.
- **Úroveň pevného bodu** v plánovacím grafu G je taková úroveň κ, že $\forall i$, $i>\kappa$ jsou úrovně i s ní identické, tj. $P_i=P_\kappa$, $\mu P_i=\mu P_\kappa$, $A_i=A_\kappa$, $\mu A_i=\mu A_\kappa$.
  - *Každý plánovací graf G má úroveň pevného bodu κ, kterým je nejmenší takové k, že* $|P_{k-1}|=|P_k|$ *a* $|\mu P_{k-1}|=|\mu P_k|$.

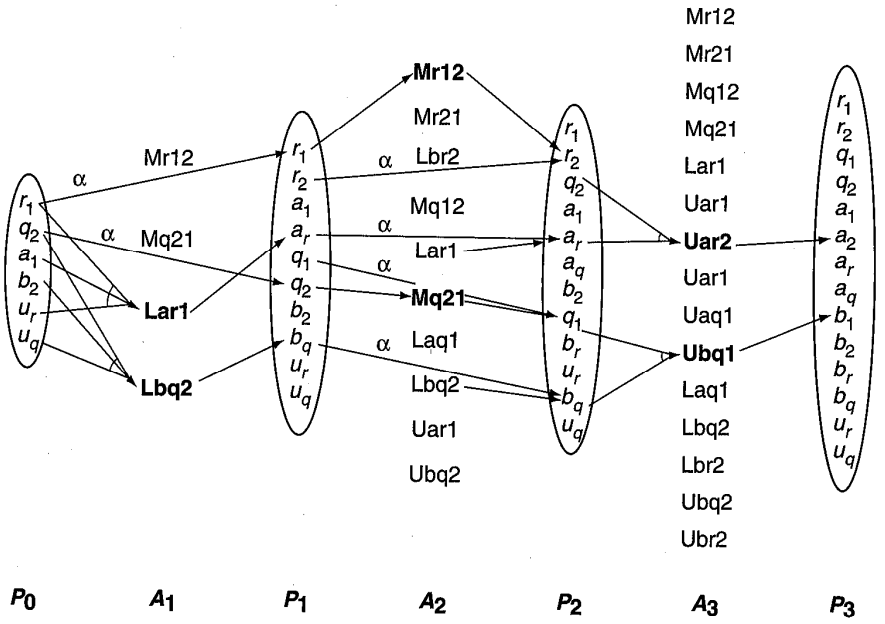
    Důkaz vychází z monotonie a konečnosti vrstev a mutexů.
  - Toto tvrzení nám dává efektivní metodu, jak najít pevný bod!

Extrakce plánu

- **Extrakce plánu** se provádí **zpětně** z úrovně P_i obsahující všechny cílové atomy, které nejsou po dvojicích mutex ($g^2 \cap \mu P_i=\varnothing$).
  - Nejdříve se najde množina akcí $\pi_i \subseteq A_i$, které nejsou po dvojicích mutex a které vedou k cílovým atomům.
  - Předpoklady akcí z π_i tvoří nový cíl pro úroveň P_i.
  - Pokud se cíl na úrovni P_j nepodaří splnit, vrací se k úrovni j+1 a hledá se alternativní množina π_{j+1}.
  - Pokud se dosáhne úrovně 0, je posloupnost $\langle \pi_1,\pi_2,\ldots,\pi_k \rangle$ řešením.
- Jedná se o **prohledávání AND/OR grafu**:
  - **OR větve** reprezentují alternativní akce poskytující daný atom
  - **AND větve** spojují předpoklady s vybranou akcí

Příklad extrakce

Level	Mutex elements
A_1	{Mr12} × {Lar1}
	{Mq21} × {Lbq2}
P_1	$\{r_2\} \times \{r_1, a_r\}$
	$\{q_1\} \times \{q_2, b_q\}$
	$\{a_r\} \times \{a_1, u_r\}$
	$\{b_q\} \times \{b_2, u_q\}$
A_2	{Mr12} × {Mr21,Lar1,Uar1}
	{Mr21} × {Lbr2,Lar1*,Uar1*}
	{Mq12} × {Mq21,Laq1,Lbq2*,Ubq2*}
	{Mq21} × {Lbq2,Ubq2}
	{Lar1} × {Uar1,Laq1,Lbr2}
	{Lbr2} × {Ubq2,Lbq2,Uar1,Mr12*}
	{Laq1} × {Uar1,Ubq2,Lbq2,Mq21*}
	{Lbq2} × {Ubq2}
P_2	$\{b_r\} \times \{r_1, b_2, u_r, b_q, a_r\}$
	$\{a_q\} \times \{q_2, a_1, u_q, b_q, a_r\}$
	$\{r_1\} \times \{r_2\}$
	$\{q_1\} \times \{q_2\}$
	$\{a_r\} \times \{a_1, u_r\}$
	$\{b_q\} \times \{b_2, u_q\}$
A_3	{Mr12} × {Mr21,Lar1,Uar1,Lbr2*,Uar2*}
	{Mr21} × {Lbr2,Uar2,Ubr2}
	{Mq12} × {Mq21,Laq1,Uaq1,Ubq1,Ubq2*}
	{Mq21} × {Lbq2,Ubq2,Laq1*,Ubq1*}
	{Lar1} × {Uar1,Uaq1,Laq1,Uar2,Ubr2,Lbr2,Mr21*}
	{Lbr2} × {Ubr2,Ubq2,Lbq2,Uar1,Uar2,Ubq1*}
	{Laq1} × {Uar1,Uaq1,Ubq1,Ubq2,Lbq2,Uar2*}
	{Lbq2} × {Ubr2,Ubq2,Uaq1,Ubq1,Mq12*}
	{Uaq1} × {Uar1,Uar2,Ubq1,Ubq2,Mq21}*
	{Ubr2} × {Uar1,Uar2,Ubq1,Ubq2,Mr12}*
	{Uar1} × {Uar2,Mr21*}
	{Ubq1} × {Ubq2}
P_3	$\{a_2\} \times \{a_r, a_1, r_1, a_q, b_r\}$
	$\{b_1\} \times \{b_q, b_2, q_2, a_q, b_r\}$
	$\{a_r\} \times \{u_r, a_1, a_q, b_r\}$
	$\{b_q\} \times \{u_q, b_2, a_q, b_r\}$
	$\{a_q\} \times \{a_1, u_q\}$
	$\{b_r\} \times \{b_2, u_r\}$
	$\{r_1\} \times \{r_2\}$
	$\{q_1\} \times \{q_2\}$



- nechť cílové atomy jsou $\{a_2,b_1\}$
- **tučně** vyznačené **akce** byly vybrány do **plánu**
- dostáváme vrstvený plán
  ⟨{Lar1,Lbq2}, {Mr12,Mq21}, {Uar2,Ubq1}⟩

Nogood

- **Mutex** je schopen zachytit **nekompatibilní dvojice**.
- Při splňování cíle na dané úrovni (tj. hledání akcí pro danou množinu výroků) můžeme zjistit, že daný **cíl nelze splnit**, tj. **množina výroků** jako celek je **nekompatibilní**.
  - Pokud se do stejné úrovně dostaneme později se stejným cílem, už víme, že se ho nepodaří splnit (jeho splnění záleží pouze na vrstvách s menším pořadovým číslem, tj. přidání další vrstvy na konec grafu jeho nesplnitelnost neovlivní).
- Nesplněné cíle je možné si pro každou vrstvu pamatovat formou tabulky ▽, tzv. **nogood** („není dobré“).
  - Pokud chceme splnit nějaký cíl na dané úrovni a tento cíl je nogood, můžeme se rovnou vrátit, protože takový cíl nelze splnit.

Algoritmy extrakce

```
Extract(G, g, i)
    if i = 0 then return (⟨⟩)
    if g ∈ ∇(i) then return(failure)
    π_i ← GP-Search(G, g, ∅, i)
    if π_i ≠ failure then return(π_i)
    ∇(i) ← ∇(i) ∪ {g}
    return(failure)
end
```

Extract se snaží splnit cíl g na úrovni i

- používá a upravuje tabulku ▽ obsahující nogoods

GPSearch hledá množinu akcí π_i splňující daný cíl.

- akce se do π_i přidávají jedna po druhé tak, aby π_i neobsahovala mutexy
- po splnění všech cílů se jde o úroveň níž

```
GP-Search(G, g, π_i, i)
    if g = ∅ then do
        Π ← Extract(G, ⋃{precond(a) | ∀a ∈ π_i}, i − 1)
        if Π = failure then return(failure)
        return(Π. ⟨π_i⟩)
    else do
        select any p ∈ g
        resolvers ← {a ∈ A_i | p ∈ effects⁺(a) and ∀b ∈ π_i : (a, b) ∉ μA_i}
        if resolvers = ∅ then return(failure)
        nondeterministically choose a ∈ resolvers
        return(GP-Search(G, g − effects⁺(a), π_i ∪ {a}, i))
end
```


Algoritmus Graphplan

```
Graphplan(A, s_0, g)
    i ← 0,   ∇ ← ∅ ,   P_0 ← s_0
    G ← ⟨P_0⟩
    until [g ⊆ P_i and g² ∩ μP_i = ∅] or Fixedpoint(G) do
        i ← i + 1
        G ← Expand(G)
    if g ⊈ P_i or g² ∩ μP_i ≠ ∅ then return(failure)
    Π ← Extract(G, g, i)
    if Fixedpoint(G) then η ← |∇(κ)|
    else η ← 0
    while Π = failure do
        i ← i + 1
        G ← Expand(G)
        Π ← Extract(G, g, i)
        if Π = failure and Fixedpoint(G) then
            if η = |∇(κ)| then return(failure)
            η ← |∇(κ)|
    return(Π)
end
```

- nejprve hledáme plánovací graf splňující podmínku g
- pokud neexistuje, končíme (není řešení)
- jinak extrahujeme vrstvený plán
- pokud se to nepovedlo
- přidáme další vrstvu
- a opět extrahujeme vrstvený plán
- algoritmus končí nalezením plánu nebo důkazem jeho neexistence.

Vlastnosti Graphplanu

Algoritmus Graphplan je korektní, úplný a vždy skončí.

Důkaz:

- pokud vrátí plán, jedná se o řešení
- pokud skončí neúspěchem, řešení neexistuje
  - Fixedpoint(G) a ($g \not\subset P_i$ nebo $g^2 \cap \mu P_i \neq \emptyset$)
    - pokud se do úrovně pevného bodu nenajde úroveň splňující cíl, pak už žádná další úroveň cíl nesplní
  - $|\nabla_{i-1}(\kappa)| = |\nabla_i(\kappa)|$
    - nogood tabulky pouze rostou – dostáváme $\nabla_{i-1}(\kappa) = \nabla_i(\kappa)$
    - pokud se nogood tabulka v pevném bodě nezmění, cíle, které nelze splnit na úrovni pevného bodu, už nepůjdou splnit nikdy - nogood tabulka se v dalších krocích „propaguje do dalších vrstev" $\nabla_{i-1}(\kappa) = \nabla_i(\kappa+1)$
- vždy skončí díky monotonii a konečnému počtu atomů a akcí (vrstvy se někdy začnou opakovat a nogoody se nasytí)

Závěrečné poznámky

- **Správa paměti**
  - plánovací graf pracuje s plně instanciovanými atomy a akcemi, což má vyšší paměťovou náročnost
  - díky monotonii vrstev a mutexů není potřeba uchovávat vrstvy explicitně – pro každý atom/akci si stačí pamatovat vrstvu, kdy se poprvé objevil(a), podobně pro mutexy
- **Prohledávání**
  - nejprve se vybírají atomy s menším počtem „podpůrných" akcí nebo atomy, které se v grafu objevily nejpozději
  - při výběru akce se preferují no-op akce nebo akce, které se v grafu objevily nejdříve
  - po výběru akce se pro zbývající atomy kontroluje, zda pořád mají některé podporující akce (kontrola dopředu)

Technika plánovacího grafu znamenala v plánování malou revoluci, protože umožnila **dramaticky zlepšit efektivitu** klasického plánování a **zvětšit velikost řešitelných problémů**.