# Plánování a rozvrhování

9

**Roman Barták, KTIML**

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

## Dále

- Zatím jsme se zabývali řešením obecných plánovacích problémů. Otázka dne je, můžeme nějak zlepšit jejich praktickou efektivitu?
  - **heuristiky**
    - problémově nezávislý průvodce
  - **řídící pravidla**
    - problémově závislé prořezání stavů
  - **hierarchické plánování**
    - problémově závislé návody na řešení úloh

# Heuristiky
## pro plánování

## Heuristiky

- **Heuristiky** slouží pro výběr dalšího uzlu v prohledávání (používali jsme zde nedeterminismus).
  - Pozn.: Pokud víme, jaký uzel vybrat, hovoříme o **orákulu** a problém řešíme deterministicky.
- Snahou je, aby heuristika byla co **nejblíže orákulu** a zároveň se **efektivně počítala**.
- Typický přístup k heuristikám je založen na řešení **relaxovaného problému** (některé předpoklady a podmínky nejsou uvažovány).
  - vyřešíme relaxované problémy pro následné uzly
  - vybereme uzel, který dává nejlepší (relaxované) řešení
- Pokud hledáme optimální řešení v nějaké cenové míře, potom heuristika h(u) odhaduje cenu h*(u) nejlepšího řešení dosažitelného z uzlu u.
  - heuristika je **přípustná**, pokud $h(u) \leq h^*(u)$ (pro minimalizaci ceny)
  - algoritmy používající přípustné heuristiky garantují nalezení optima

# Základní heuristika
## pro plánování se stavy

- Heuristika bude **odhadovat počet akcí** z daného stavu do cílového stavu resp. k dosažení daného atomu či množiny atomů.
- Založena bude na „vyřešení" **relaxovaného problému**.
  - uvažujeme pouze pozitivní efekty
  - předpokládáme, že různé atomy v cíli se dosahují nezávisle na sobě

- **Nultý pokus:**
  - $\Delta_0(s,p) = 0$ pokud $p \in s$
  - $\Delta_0(s,g) = 0$ pokud $g \subseteq s$
  - $\Delta_0(s,p) = \infty$ pokud $p \notin s$ a $\forall a \in A, p \notin effects^+(a)$
  - $\Delta_0(s,p) = \min_a\{1+\Delta_0(s,precond(a)) \mid p \in effects^+(a)\}$
  - $\Delta_0(s,g) = \Sigma_{p \in g}\Delta_0(s,p)$

Tato heuristika ale **není přípustná** (pro hledání optimálního plánu), protože se nejedná o dolní odhad vzdálenosti!

```
Delta(s)
    for each p do: if p ∈ s then Δ0(s,p) ← 0, else Δ0(s,p) ← ∞
    U ← s
    iterate
        for each a such that precond(a) ⊆ U do
            U ← U ∪ effects+(a)
            for each p ∈ effects+(a) do
                Δ0(s,p) ← min{Δ0(s,p), 1 + Σ_{q∈precond(a)} Δ0(s,q)}
    until no change occurs in the above updates
end
```



# Přípustné heuristiky
## pro plánování se stavy

- **První pokus o přípustnou heuristiku**
  - ...
  - $\Delta_1(s,g) = \max\{\Delta_0(s,p) \mid p \in g\}$
  - Umožňuje bezpečné oříznutí větve, kde je hodnota heuristiky větší než dosud nejlepší nalezené řešení.
  - Experimenty ale ukázaly, že $\Delta_1$ není tak informovaná jako $\Delta_0$ (menší zaměření = větší prohledávání)
- **Druhý pokus o přípustnou heuristiku**
  Zkusme zjišťovat dosažitelnost dvojice atomů najednou.
  - ...
  - $\Delta_2(s,p) = \min_a\{1+\Delta_2(s,precond(a)) \mid p \in effects^+(a)\}$
  - $\Delta_2(s,\{p,q\}) = \min\{$
    $\min_a\{1+\Delta_2(s,precond(a)) \mid \{p,q\} \in effects^+(a)\},$
    $\min_a\{1+\Delta_2(s,\{q\}\cup precond(a)) \mid p \in effects^+(a)\},$
    $\min_a\{1+\Delta_2(s,\{p\}\cup precond(a)) \mid q \in effects^+(a)\}\}$
  - $\Delta_2(s,g) = \max_{p,q}\{\Delta_2(s,\{p,q\}) \mid \{p,q\} \subseteq g\}$
- Uvedený princip můžeme zobecnit na libovolně velké množiny, pro k>2 už je ale výpočtová složitost příliš velká.
- **Co na to Graphplan?**
  - Popsaný princip částečně odpovídá expanzní fázi Graphplanu, Graphplan má ale navíc mutexy.
  - Heuristiku $\Delta_2$ můžeme přizpůsobit do podoby Graphplanu tak, že dosažení dvojice atomů dvěma nezávislými akcemi budeme považovat za jeden krok.

# Použití heuristik
## pro plánování se stavy

### Dopředné plánování

- Preferujeme akci vedoucí do stavu, jehož heuristická vzdálenost od cíle je nejkratší.
- Heuristika se počítá v každém kroku algoritmu.

```
Heuristic-forward-search(π, s, g, A)
    if s satisfies g then return π
    options ← {a ∈ A | a applicable to s}
    for each a ∈ options do Delta(γ(s, a))
    while options ≠ ∅ do
        a ← argmin{Δ0(γ(s, a), g) | a ∈ options}
        options ← options − {a}
        π' ← Heuristic-forward-search(π.a, γ(s, a), g, A)
        if π' ≠ failure then return(π')
    return(failure)
end
```

### Zpětné plánování

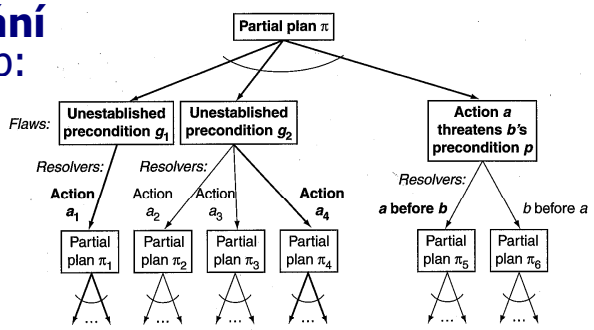
- Na úvod vypočteme heuristické vzdálenosti z počátku $s_0$ do všech atomů: $\Delta(s_0,p)$
  - lze dělat inkrementálně
- Preferujeme akce, jejichž předpoklady jsou heuristicky blíže počátku.

```
Backward-search(π, s0, g, A)
    if s0 satisfies g then return(π)
    options ← {a ∈ A | a relevant for g}
    while options ≠ ∅ do
        a ← argmin{Δ0(s0, γ^-1(g, a)) | a ∈ options}
        options ← options − {a}
        π' ← Backward-search(a.π, s0, γ^-1(g, a), A)
        if π' ≠ failure then return(π')
    return failure
end
```

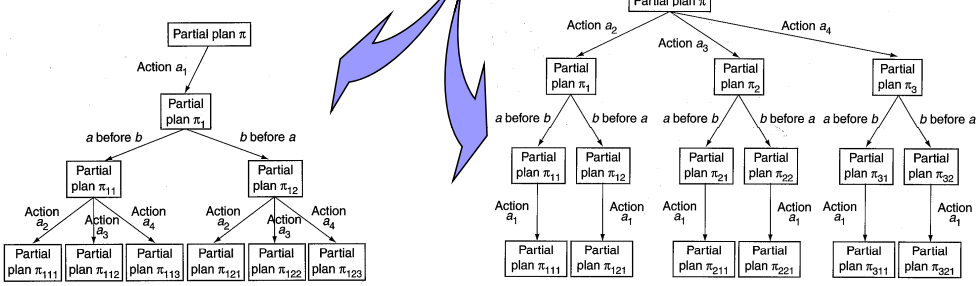


# Heuristiky
## pro plánování s plány

- Plánování v prostoru plánů je **prohledávání AND-OR grafu**. Máme zde dva typy voleb:
  - výběr kazu k opravě (AND uzel)
  - výběr způsobu opravy (OR uzel)




- **Volba kazu k opravě**
  - Vlastně se jedná o **serializaci AND-OR grafu**, tj. AND uzel se rozpadne do několika samostatných uzlů.
  - Jaká serializace je lepší?




  - Lepší je serializace, která vede k menšímu počtu uzlů v grafu.
  - **FAF (fewest alternatives first) heuristika**
    - nejdříve se opravují kazy s menším počtem alternativních způsobů opravy

# Volba opravy
## pro plánování s plány

**Jaký způsob opravy kazu zkusit nejdříve?**

- Nechť $\{\pi_1, \ldots, \pi_m\}$ jsou částečné plány získané různými opravami kazu a $g_\pi$ je množina otevřených cílů v $\pi$.
- **Nultý pokus**
  Preferujeme částečný plán s menším počtem otevřených cílů
  ⇨ $\eta_0(\pi) = |g_\pi|$
  - To nám ale **o složitosti,** jak cíle dosáhnout, **moc neříká**.
- **Další pokus**
  Vygenerujme AND-OR graf pro $\pi$ do dané hloubky k a spočteme počet nových akcí v grafu a počet otevřených cílů, které nejsou v $s_0$
  ⇨ $\eta_k(\pi)$
  - To je ale **výpočtově dost náročné**.
- **Ještě vylepšení**
  Uděláme plánovací graf pro původní cíl (vytvoří se jen jednou). Najdeme otevřený cíl p v $\pi$, který se do grafu přidal poslední, a na cestě z $s_0$ do p spočteme počet akcí, které nejsou v $\pi$
  ⇨ $\eta(\pi)$



# Řídící pravidla

# Prořezávání

- Heuristiky se snaží navést plánovač směrem k cíli tím, že uspořádají alternativy pro prozkoumání. Neřeší ale **problém s počtem alternativ**.
- Můžeme **nevhodné alternativy detekovat a odstranit**?
- **Příklad** (svět kostek)
  - Pokud je kostka na „správném" místě konzistentním s cílem, potom akce, která s ní pohne, pouze plán prodlouží.
  - Pokud je kostka na nesprávném místě a existuje akce, která ji přesouvá na správné místo, potom každá akce, která kostku přesouvá jinam prodlužuje plán.
- Doménově závislé informace umožňují prořezat prohledávaný prostor, jak je ale vyjádřit pro obecný plánovací algoritmus?
  - **řídící pravidla (control rules)**



simple temporal logic

# Temporální logiky

- Potřebujeme formalismus pro vyjadřování vztahů mezi vlastnostmi aktuálního stavu světa a stavů následujících.
- **jednoduché temporální logiky**
  - obohacují predikátovou logiku o **modální operátory**
    - $\phi_1 \cup \phi_2$ (až do) $\phi_2$ platí teď nebo v některém příštím stavu a $\phi_1$ platí až do té doby
    - $\Box\ \phi$ (vždy) $\phi$ platí teď a v každém z příštích stavů
    - $\Diamond\ \phi$ (časem) $\phi$ platí teď nebo v některém z příštích stavů
    - $\bigcirc\ \phi$ (příště) $\phi$ platí v následujícím stavu
    - GOAL($\phi$) $\phi$ (nemá modální operátory) platí v cílovém stavu
  - $\phi$ je libovolná formule vyjadřující nějaký vztah mezi objekty světa (může obsahovat i modální operátory)

# Modální operátory

## interpretace

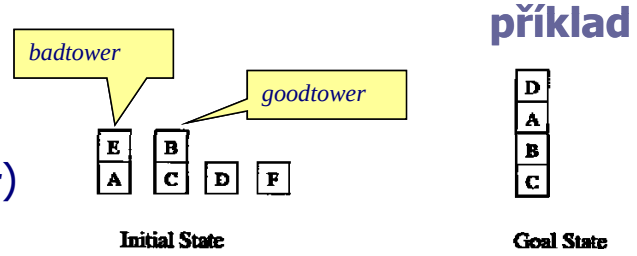
- Pro **interpretaci** modální formule nestačí jediný stav, ale budeme pracovat s trojicí **(S, $s_i$, g)**
  - S = $\langle s_0, s_1, \ldots \rangle$ je nekonečná posloupnost stavů
  - $s_i \in S$ je aktuální stav
  - g je cílová formule
- Plán $\pi = \langle a_1, a_2, \ldots, a_n \rangle$ dává konečnou posloupnost stavů $S_\pi = \langle s_0, s_1, \ldots, s_n \rangle$, kde $s_{i+1} = \gamma(s_i, a_{i+1})$, ze které může udělat nekonečnou posloupnost $\langle s_0, s_1, \ldots, s_{n-1}, s_n, s_n, s_n, \ldots \rangle$.
- $(S, s_i, g) \models \phi$ definujeme takto:
  - $(S, s_i, g) \models \phi$ iff $s_i \models \phi$ pro atom $\phi$
  - $(S, s_i, g) \models \phi_1 \wedge \phi_2$ iff $(S, s_i, g) \models \phi_1$ a $(S, s_i, g) \models \phi_2$
  - ...
  - $(S, s_i, g) \models \phi_1 \cup \phi_2$ iff existuje $j \geq i$ tž. $(S, s_j, g) \models \phi_2$ a pro každé k: $i \leq k < j$ $(S, s_k, g) \models \phi_1$
  - $(S, s_i, g) \models \Box \phi$ iff $(S, s_j, g) \models \phi$ pro každé $j \geq i$
  - $(S, s_i, g) \models \Diamond \phi$ iff $(S, s_j, g) \models \phi$ pro nějaké $j \geq i$
  - $(S, s_i, g) \models \bigcirc \phi$ iff $(S, s_{i+1}, g) \models \phi$
  - $(S, s_i, g) \models \text{GOAL}(\phi)$ iff $\phi \in g$



# Řídící pravidlo

## příklad

- Dobrá věž (*goodtower*) je taková, ve které není potřeba přesouvat žádnou kostku. Špatná věž (*badtower*) je věž, která není dobrá.




$$goodtower(x) \triangleq clear(x) \wedge \neg \text{GOAL}(holding(x)) \wedge goodtowerbelow(x)$$

$$goodtowerbelow(x) \triangleq (ontable(x) \wedge \neg \exists [y{:}\text{GOAL}(on(x,y))]\,)$$
$$\vee\, \exists [y{:}on(x,y)]\, \neg \text{GOAL}(ontable(x)) \wedge \neg \text{GOAL}(holding(y)) \wedge \neg \text{GOAL}(clear(y))$$
$$\wedge\, \forall [z{:}\text{GOAL}(on(x,z))]\, z = y \wedge \forall [z{:}\text{GOAL}(on(z,y))]\, z = x$$
$$\wedge\, goodtowerbelow(y)$$

$$badtower(x) \triangleq clear(x) \wedge \neg goodtower(x)$$

**Řídící pravidlo:**

*goodtower zůstává goodtower*

$$\Box\big(\forall [x{:}clear(x)]\, goodtower(x) \Rightarrow \bigcirc(clear(x) \vee \exists [y{:}on(y,x)]\, goodtower(y))$$
$$\wedge\, badtower(x) \Rightarrow \bigcirc(\neg \exists [y{:}on(y,x)]\,)$$
$$\wedge\, (ontable(x) \wedge \exists [y{:}\text{GOAL}(on(x,y))]\, \neg goodtower(y))$$
$$\Rightarrow \bigcirc(\neg holding(x))\big)$$

*nedávej nic na badtower*

*neber kostku ze stolu, pokud ji nemůžeš dát na goodtower*

# Progrese

- Abychom mohli použít řídící pravidla při plánování, potřebujeme vyjádřit **pokrok řídící formule ze stavu $s_i$ do stavu $s_{i+1}$.**
  - hledáme formuli progr($\phi$, $s_i$), která je pravdivá ve stavu $s_{i+1}$, pokud je $\phi$ pravdivá ve stavu $s_i$
- $\phi$ neobsahuje modální operátory
  - progr($\phi$, $s_i$) = true pokud $s_i \models \phi$
    = false pokud neplatí $s_i \models \phi$
- $\phi$ s logickými spojkami
  - progr($\phi_1 \wedge \phi_2$, $s_i$) = progr($\phi_1$, $s_i$) $\wedge$ progr($\phi_2$, $s_i$)
  - progr($\neg\phi$, $s_i$) = $\neg$progr($\phi$, $s_i$)
- $\phi$ s kvantifikátory (nemáme funkční symboly, jen k konstant $c_j$)
  - progr($\forall x\ \phi$, $s_i$) = progr($\phi\{x/c_1\}$, $s_i$) $\wedge$ ... $\wedge$ progr($\phi\{x/c_k\}$, $s_i$)
  - progr($\exists\ x\ \phi$, $s_i$) = progr($\phi\{x/c_1\}$, $s_i$) $\vee$ ... $\vee$ progr($\phi\{x/c_k\}$, $s_i$)
- $\phi$ s modálními operátory
  - progr($\phi_1 \cup \phi_2$, $s_i$) = (($\phi_1 \cup \phi_2$) $\wedge$ progr ($\phi_1$, $s_i$)) $\vee$ progr ($\phi_2$, $s_i$)
  - progr($\Box\ \phi$, $s_i$) = ($\Box\ \phi$) $\wedge$ progr($\phi$, $s_i$)
  - progr($\Diamond\ \phi$, $s_i$) = ($\Diamond\ \phi$) $\vee$ progr($\phi$, $s_i$)
  - progr($\bigcirc\ \phi$, $s_i$) = $\phi$

**Technické poznámky:**

- progress($\phi$, $s_i$) získáme z progr($\phi$, $s_i$) vyčištěním (true $\wedge$ d $\rightarrow$ d, $\neg$true $\rightarrow$ false, ...)
- můžeme rozšířit na posloupnost stavů $\langle s_0, \ldots, s_n \rangle$
  progress($\phi$, $\langle s_0, \ldots, s_n \rangle$) = $\phi$ pokud n = 0
  = progress(progress($\phi$, $\langle s_0, \ldots, s_{n-1} \rangle$), $s_n$) jinak



# Progrese

## vlastnosti

- $(S,s_i,g) \models \phi$ právě tehdy když $(S,s_{i+1},g) \models$ progress($\phi$, $s_i$).
  - tj. progress se chová tak, jak jsme chtěli
- $(S,s_0,g) \models \phi$ potom pro každý prefix S' = $\langle s_0, \ldots, s_i \rangle$ od S platí progress($\phi$,S') $\neq$ false.
  - pokud je řídící pravidlo splněno, tak progres nehlásí chybu
- Je-li plán $\pi$ aplikovatelný na $s_0$ a progress($\phi$, $S_\pi$) = false, potom neexistuje prodloužení S' od $S_\pi$ tž. (S',$s_0$,g) $\models \phi$.
  - pokud progress hlásí chybu, nelze splnit řídící pravidlo

V plánovacím algoritmu budeme postupně přes progress upravovat řídící formuli pro další stavy a pokud bude „hlásit neúspěch" víme, že daným směrem neexistuje plán, který by splňoval řídící pravidlo.

# Řízené plánování

- Dopředné prohledávání ve stavovém prostoru můžeme upravit na využití řídících pravidel.
  - Pokud částečný plán $S_\pi$ nesplňuje řídící podmínku progress($\phi$, $S_\pi$), nemusíme tento plán dále rozšiřovat.