Základy umělé inteligence

6. Plánování

Jiří Kubalík
Katedra kybernetiky, ČVUT-FEL

IDA
Intelligent Data Analysis
RESEARCH GROUP

http://cw.felk.cvut.cz/doku.php/courses/y33zui/start

Obsah přednášky

- Definice plánovacího problému, příklady plánovacích problémů
- Reprezentace plánovacího problému, jazyk STRIPS, příklady
- Plánování jako prohledávání stavového prostoru
- Partial-order planning, příklad

Obecná formulace plánování

:: **(Klasické) Plánování** – nalezení takové posloupnosti akcí, vedoucí z počátečního stavu k cíli a to v prostředí, které je

- plně pozorovatelné,
- deterministické,
- konečné,
- statické,
- diskrétní.

:: Nešlo by to řešit např. jednoduchým prohledáváním stavového prostoru?

- Ano, ovšem pouze ty nejjednodušší úlohy.
- Ale nás zajímají postupy, které zvládnou i složité úlohy.

Příklad: Koupě jedné knihy

:: Příklad: Chceme koupit knihu "AI: A Modern Approach" v e-shopu.

- Na skladě mají 10^9 knih, každá s jednoznačným 10-znakovým ISBN kódem.
- Pro každou knihu je definována akce $koupit(ISBN)$.
  Cílový stav je popsán jako $vlastnit(ISBN0137903952)$.

1. Použijeme k tomu **nějaký ze známých algoritmů prohledávání** (do hloubky, do šířky, s iter. prohlubováním, ...).
   - Museli bychom prověřit výsledek 10^9 akcí $koupit(ISBN)$, abychom našli tu, která vede na cílový stav $vlastnit(ISBN0137903952)$.

Příklad: Koupě jedné knihy

:: Příklad: Chceme koupit knihu "AI: A Modern Approach" v e-shopu.

- Na skladě mají 10^9 knih, každá s jednoznačným 10-znakovým ISBN kódem.
- Pro každou knihu je definována akce $koupit(ISBN)$.
  Cílový stav je popsán jako $vlastnit(ISBN0137903952)$.

1. Použijeme k tomu **nějaký ze známých algoritmů prohledávání** (do hloubky, do šířky, s iter. prohlubováním, ...).
   - Museli bychom prověřit výsledek 10^9 akcí $koupit(ISBN)$, abychom našli tu, která vede na cílový stav $vlastnit(ISBN0137903952)$.
2. **Rozumnější plánovací algoritmus**
   - ví, že $koupit(x)$ má za efekt $vlastnit(x)$,
   - proto rovnou vygeneruje akci $koupit(ISBN0137903952)$.

Příklad: Větší nákup

:: Příklad: Chceme koupit mléko, banány a akuvrtačku.

Prohledávací algoritmus

- **počáteční stav** – jsme doma a nemáme nic.
- **akce**:

  jdi_do_kina, jdi_do_skoly, $jdi\_do\_supermarketu$, cti_si, $zabehej\_si$, ...

  kup_listek, $najdi\_sal$, $najdi\_sedadlo$, ...

  $kup(chleb)$, $kup(mleko)$, $kup(banany)$, ...
- **cílový stav**: cíl $= vlastnit(mleko), vlastnit(banany), vlastnit(akuvrtacku)$.
- **heuristika** – počet nesplněných podcílů, počet chybějících položek

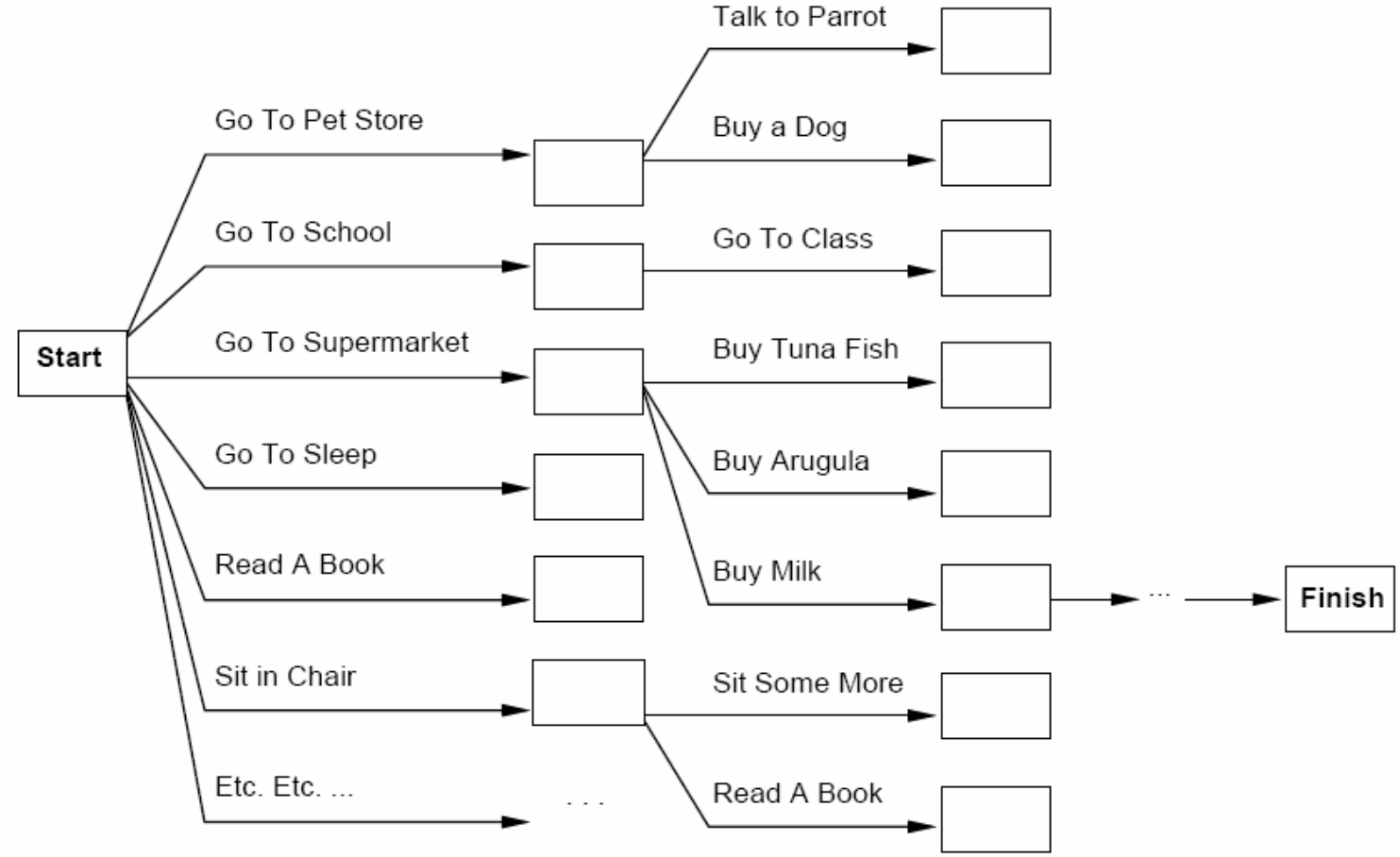
  Př.: cíl $= vlastnit(mleko), vlastnit(banany), vlastnit(akuvrtacku)$

  s $= vlastnit(mleko), vlastnit(banany)$

  $h(s) = 1$

Příklad: Větší nákup (2)

Nalézt správnou cestu (optimální sekvenci akcí) v takovémto grafu ...



©Russel, Norvig: Artificial Intelligence: A Modern Approach.

Plánování jako prohledávání stavového prostoru: Shrnutí

:: **Problémy** při použití prostého prohledávacího přístupu

- **velký větvící faktor** a délka cesty řešení
- **reprezentace plánu jako lineární posloupnosti akci**, začínající v počátečním stavu – neumožňuje vybírat flexibilně cíle, které budou aktuálně řešeny (dekompozice úlohy)

  např.: První akce při nákupu je přechod do obchodu. Prohledávač ale bude zbytečně zkoušet spoustu jiných míst než zjistí, kde se nakupuje.
- **cílový test a heuristika** jsou pouze černé skříňky – nelze je využít pro výběr relevantních akcí,

  ikdyž už jsme v obchodě, stejně musíme vyzkoušet akce
  - kup_chleb, kup_mleko, kup_banany, ...

    abychom zjistili, že
  - kup_chleb a spousta dalších kupovacích akcí nám nepomůže,
  - zatímco kup_mleko, kup_banany ano.

Hlavní rysy plánovacích algoritmů

:: **Otevřená reprezentace stavů, cílů a akcí**

- formální popis, většinou v predikátové logice 1. řádu,
- stavy a cíle jsou reprezentovány logickými větami,
- akce jsou reprezentovány logickým popisem *podmínek* (preconditions) a *efektů* (effects).

  př.: Jestliže cíl obsahuje $Have(Milk)$ a efektem akce $Buy(x)$ je $Have(x)$, potom může být užitečné přidat do plánu akci $Buy(Milk)$.

:: **Přidávání akcí do plánu, kdykoliv je to potřeba**

- pořadí v jakém jsou akce plánovány není svázáno s pořadím, v jakém jsou akce vykonávány. Zřejmé akce se naplánují první, čímž se redukuje větvící faktor a p-st backtrackingu v dalších fázích plánování.

  př.: akce $Buy(Milk)$ může být přidána do plánu dříve, než je jasné, kde se koupí (Toto může být specifikováno v podmínce akce).

:: **Využití nezávislosti podcílů pro dekompozici celého problému** – *rozděl-a-panuj*.

- řešit několik menších problémů je téměř vždy jednodušší, než řešit jeden velký problém (viz řešení problémů s omezeními).

Hlavní rysy plánovacích algoritmů (2)

:: **Využití nezávislosti podcílů pro dekompozici celého problému.**

- Zajišťování mléka, banánů a vrtačky se může rozdělit na
  - **nákup v supermarketu** – tento cíl dále rozpadne na
    - nákup banánů a
    - nákup mléka.
  - **návštěvu půjčovny nářadí** – s cílem získat vrtačku.

:: **Dekompozici nelze využít u problémů se silnými interakcemi mezi podcíli.**

- zkombinování částečných řešení je příliš náročné

  př.: Hra "Lišák"
  - najít individuální plány pro dopravení jednotlivých kostek na jejich správné pozice je jednoduché,
  - ovšem sloučit je do jednoho řešení je už problém.

STRIPS: Jazyk pro reprezentaci plánovacích problémů

:: STRIPS (STanford Research Institute Problem Solver)

:: Reprezentace stavů, akcí a cílů by měla umožnit využití logické struktury problému.

:: **Reprezentace stavů** – konjunkce pozitivních literálů

- literály výrokové a predikátové – $Unknown$, $Small$, $Have(Milk) \wedge At(Supermarket)$,
- literály obsahují pouze konstanty – proměnné a funkce nejsou povoleny,
  $At(x)$, $Buy(Cheapest(Cheese))$ jsou nepřípustné
  Tím je zaručena konečná množina stavů a akcí.
- **uzavřenost světa** – všechny podmínky, které nejsou vyjádřeny v popisu stavu, jsou považovány za neplatné.

:: **Reprezentace cílů** – může obsahovat proměnné

- stav s vyhovuje cíli g tehdy, když s obsahuje všechny literály g (může i další),
  př.: Cíl "*být v obchodě, kde prodávají mléko*" – $At(x) \wedge Sells(x, Milk)$

STRIPS: Jazyk pro reprezentaci plánovacích problémů 2

:: **Reprezentace akcí** – schéma akce

- **jméno a seznam parametrů** – pokud obsahuje proměnné, potom *schéma akce* definuje množinu akcí,
- **počáteční podmínky** – konjunkce pozitivních literálů, které specifikují, co musí být splněno, aby akce mohla být provedena,
- **efekt** – konjunkce literálů, které specifikují, jak se změní stav po provedení akce
  - pozitivní literály (*add list*) určují, co má platit ve výsledném stavu,
  - negativní literály (*delete list*) určují, co platit přestane.

Příklad: Přelet letadla odněkud nekam.

$Action(Fly(p, from, to))$

PODMÍNKY: $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$

EFEKT: $\neg At(p, from) \wedge At(p, to)$

:: **Schéma akce a je aplikovatelné na stav s**, pokud existuje způsob, jak instancializovat proměnné a tak, že každá z počátečních podmínek a je splněna v s, tzn. $Podminky(a) \subset s$.

Action Description Language (ADL)

:: ADL je rozšířením STRIPS.

STRIPS **Language**	ADL **Language**
Only positive literals in states: $Poor \wedge Unknown$	Positive and negative literals in states: $\neg Rich \wedge \neg Famous$
Closed World Assumption: Unmentioned literals are false.	Open World Assumption: Unmentioned literals are unknown.
Effect $P \wedge \neg Q$ means add P and delete Q.	Effect $P \wedge \neg Q$ means add P and $\neg Q$ and delete $\neg P$ and Q.
Only ground literals in goals: $Rich \wedge Famous$	Quantified variables in goals: $\exists x At(P_1, x) \wedge At(P_2, x)$ is the goal of having P_1 and P_2 in the same place.
Goals are conjunctions: $Rich \wedge Famous$	Goals allow conjunction and disjunction: $\neg Poor \wedge (Famous \vee Smart)$
Effects are conjunctions.	Conditional effects allowed: **when** P: E means E is an effect only if P is satisfied.
No support for equality.	Equality predicate $(x = y)$ is built in.
No support for types.	Variables can have types, as in $(p : Plane)$.

Akce ve STRIPS: Příklad

:: Příklad: Přelet letadla z JFK do SFO.

Počáteční stav je

$$At(P_1, JFK) \wedge At(P_2, SFO) \wedge Plane(P_1) \wedge Plane(P_2) \wedge Airport(JFK) \wedge Airport(SFO).$$

Tento stav vyhovuje počátečním podmínkám akce $Fly(p, from, to)$

$$At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$$

se substitucí

$$\{p/P_1, from/JFK, to/SFO\}.$$

Po aplikaci akce $Fly(P_1, JFK, SFO)$ se aktuální stav změní na

$$At(P_1, SFO) \wedge At(P_2, SFO) \wedge Plane(P_1) \wedge Plane(P_2) \wedge Airport(JFK) \wedge Airport(SFO)$$

:: *A co všechny ty objekty, které jsou v letadle? Jak vyjádříme, že také změnily letiště?*

Definice problémů ve STRIPS: Letecká nákladní přeprava

:: Letecká nákladní přeprava

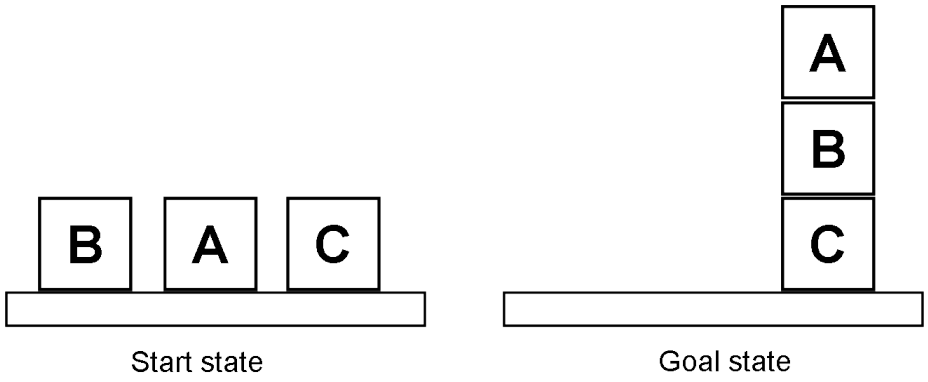
- **akce:** nakládání, vykládání, přelet z místa na místo.
- **predikáty** pro popis stavů:
  - $In(c, p)$ – náklad c je v letadle p,
  - $At(x, a)$ – objekt x (buď letadlo anebo náklad) je na letišti a,
  - $Cargo(c)$, $Plane(p)$, $Airport(a)$.

$$
\begin{aligned}
&Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK) \\
&\quad \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \\
&\quad \land Airport(JFK) \land Airport(SFO)) \\
&Goal(At(C_1, JFK) \land At(C_2, SFO)) \\
&Action(Load(c, p, a), \\
&\quad \text{PRECOND}: At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT}: \neg At(c, a) \land In(c, p)) \\
&Action(Unload(c, p, a), \\
&\quad \text{PRECOND}: In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT}: At(c, a) \land \neg In(c, p)) \\
&Action(Fly(p, from, to), \\
&\quad \text{PRECOND}: At(p, from) \land Plane(p) \land Airport(from) \land Airport(to) \\
&\quad \text{EFFECT}: \neg At(p, from) \land At(p, to))
\end{aligned}
$$

Definice problémů ve STRIPS: Svět kostek

:: **Svět kostek** – je dána množina kostek, které mohou ležet buď na desce stolu anebo mohou být navršeny na sobě. Vždy pouze jedna kostka může ležet na druhé kostce. Pouze jedna kostka může být přesouvána v dané chvíli. Cílem je sestavit kostky do požadované cílové konfigurace.

- **akce:**
  - $Move(b, x, y)$ – přesune blok b z x na y, pokud na x ani na y nic neleží.
- **predikáty** pro popis stavů:
  - $On(b, x)$ – blok b leží na x, kde x může být deska stolu nebo jiná kostka,
  - $Clear(b)$ – je splněn, když na b nic neleží.
- **poznámka** – *Jak je to s akcí* $Move(b, x, y)$*, když je buď* $x = Table$ anebo $y = Table$?



Definice problémů ve STRIPS: Svět kostek (2)

Musíme udělat dvě věci:

- **akce:**
  - $Move(b, x, y)$ – přesune blok b z x na y, pokud na x ani na y nic neleží.
  - $MoveToTable(b, x)$ – přesune blok b z x na desku stolu.

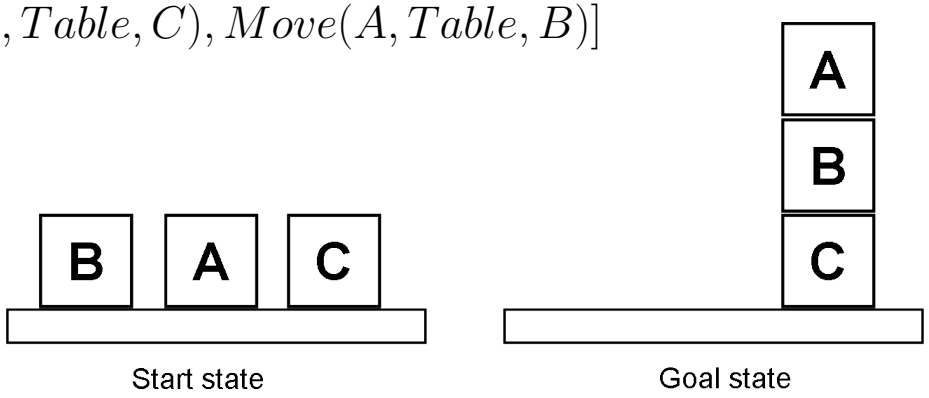
    Co tím vyřešíme?
- **predikáty** pro popis stavů:
  - $Clear(b)$ – je splněn, když je na b volné místo pro uložení kostky.

    Co tím vyřešíme?
- **poznámky**
  - *Co s akcí $Move(B, C, C)$?*
  - *Jak zajistíme, že plánovač správně rozliší, kdy použít $Move(b, x, y)$ a kdy $MoveToTable(b, x)$?*

Definice problémů ve STRIPS: Svět kostek (3)

$$
\begin{aligned}
&Init(On(A, Table) \wedge On(B, Table) \wedge On(C, Table) \\
&\quad \wedge Block(A) \wedge Block(B) \wedge Block(C) \\
&\quad \wedge Clear(A) \wedge Clear(B) \wedge Clear(C)) \\
&Goal(On(A, B) \wedge On(B, C)) \\
&Action(Move(b, x, y), \\
&\quad \text{PRECOND: } On(b, x) \wedge Clear(b) \wedge Clear(y) \wedge Block(b) \wedge \\
&\qquad (b \neq x) \wedge (b \neq y) \wedge (x \neq y), \\
&\quad \text{EFFECT: } On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y)) \\
&Action(MoveToTable(b, x), \\
&\quad \text{PRECOND: } On(b, x) \wedge Clear(b) \wedge Block(b) \wedge (b \neq x), \\
&\quad \text{EFFECT: } On(b, Table) \wedge Clear(x) \wedge \neg On(b, x))
\end{aligned}
$$

Řešení: $[Move(B, Table, C), Move(A, Table, B)]$



Plánování: Dopředná strategie prohledávání

:: Reprezentace (bez funkčních symbolů) zaručuje, že stavový prostor je konečný $\Longrightarrow$ jakýkoliv úplný algoritmus prohledávání je také úplným plánovacím algoritmem.

:: **Dopředná strategie prohledávání** (neefektivní)

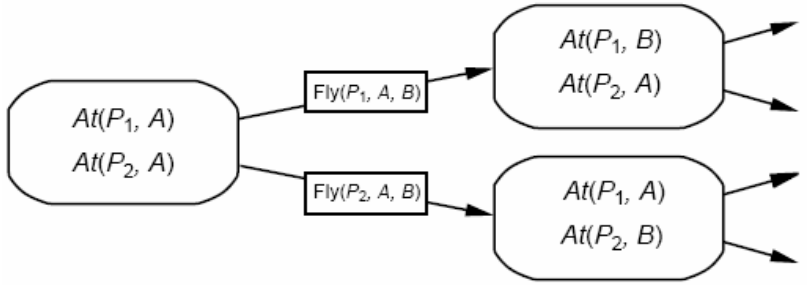
- **velký faktor větvení** – všechny použitelné akce jsou aplikovány na aktuální uzel.
- **bez dobré heuristiky nepoužitelné**.

  Př. Nákladní let. doprava – 10 letišť, každé letiště má 5 letadel a 20 balíků nákladu.

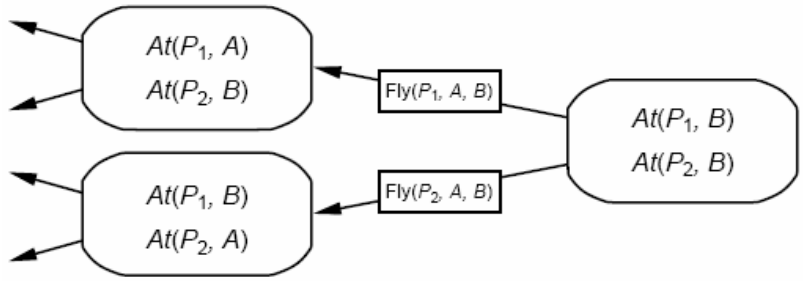
  Cílem je přepravit všechny balíky z letiště A do letiště B.

  Řekněme, že průměrný faktor větvení je 1000. Kolik uzlů má prohledávací strom hloubky 10?

  Přitom triviální řešení problému je: Nalož všechny balíky na jedno z letadel, která jsou na letišti A a preprav je na letiště B. *V jaké hloubce je toto řešení?*



Dopředná strategie



Zpětná strategie

Plánování: Zpětná strategie prohledávání

- **příznivý faktor větvení** – pouze relevantní akce jsou aplikovány při expandování aktuálního uzlu.

  Jaký je větvicí faktor u příkladu letecké přepravy?

  Akce je **relevantní**, jestliže generuje nějaký pozitivní literál (podmínku) cíle.

  Akce je **konzistentní**, jestliže neneguje (neruší) žádný z platných literálů cíle.

  Př.: Cíl je $At(C_1, B) \wedge At(C_2, B) \wedge \cdots \wedge At(C_{20}, B)$

  Nyní uvažujme literál $At(C_1, B)$. Pro ten existuje pouze jedna akce, která jej generuje

  $Unload(C_1, p, B)$ ... letadlo p není určeno.

  Kolik akcí je možné aplikovat na cílový stav?

- **Postup pro konstrukci předchůdců cíle** G s použitím relevantní a konzistentní akce A.
  - všechny pozitivní efekty A, které se vyskytují v G jsou odebrány,
  - všechny literály podmínky A jsou přidány, pokud tam již nejsou.

  Př.: Předchůdce cíle z příkladu nahoře je

  $$In(C_1, P_{12}) \wedge At(P_{12}, B) \wedge At(C_2, B) \wedge \cdots \wedge At(C_{20}, B)$$

  při substituci p/P_{12}.

Jaká je ukončovací podmínka u zpětné strategie prohledávání?

Forward a backward strategie prohledávání: Poznámky

:: Obě metody jsou příkladem zcela uspořádaného prohledávání – řešením je lineární sekvence akcí, vycházející z počátečního stavu a ukončená v cíli.

- neumí využít dekompozice problému; neumí zpracovávat odděleně nezávislé podproblémy.
- v každém kroku vybírají následující akci ze všech podproblémů.

:: **Least commitment** strategie

- flexibilní v pořadí plánování akcí,
- zřejmé nebo důležité akce jsou naplánovány jako první.

  Př.: Obchodní agent plánuje cestu z Prahy do Vancouveru.

  Nejdříve si najde lety Praha$\longrightarrow$ Londýn a Londýn$\longrightarrow$ Vancouver

  a potom až zajišťuje cestu na letiště v Praze a z letiště ve Vancouveru.

Partial-order planning: Motivační příklad

:: Příklad: Naplánuj akce pro nazutí bot.

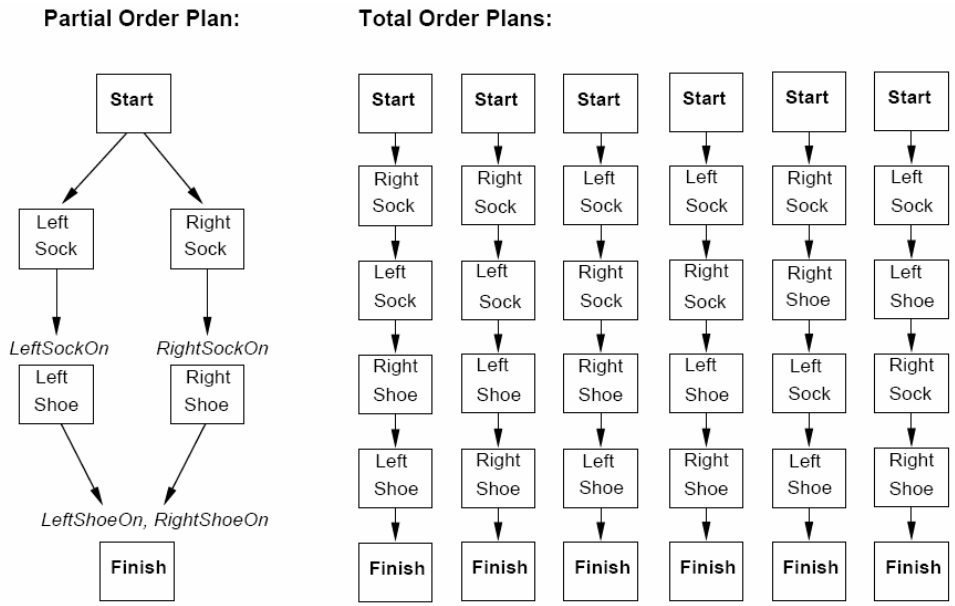
$$
\begin{aligned}
&Goal(RightShoeOn \wedge LeftShoeOn)\\
&Init()\\
&Action(RightShoe, \textsc{Precond}{:}RightSockOn, \textsc{Effect}{:}RightShoeOn)\\
&Action(RightSock, \textsc{Effect}{:}RightSockOn)\\
&Action(LeftShoe, \textsc{Precond}{:}LeftSockOn, \textsc{Effect}{:}LeftShoeOn)\\
&Action(LeftSock, \textsc{Effect}{:}LeftSockOn)\ .
\end{aligned}
$$

- plánovač by mohl "zpracovávat" každou nohu zvlášť, výsledkem by byly dvě sekvence $plan_1 = [RightSock, RightShoe]$ a $plan_2 = [LeftSock, LeftShoe]$, aniž by řešil jestli je nějaká akce plánu p_1 před anebo za jinou akcí plánu p_2.
- sekvence p_1 a p_2 by následně zkombinoval do výsledného plánu.

Partial-order planning

:: **Partial-order planning** (POP)

- plánovací alg., který může vložit dvě akce do plánu aniž by specifikoval, která bude první,
- řešení je reprezentováno grafem akcí, ne sekvencí akcí,
- **linearizace plánu** – výsledný plán; plně uspořádaná sekvence akcí.



©Russel, Norvig: Artificial Intelligence: A Modern Approach.

Částečně uspořádanému plánu odpovídá 6 možných úplně uspořádaných plánů (linearizací).

Partial-order plan: Formulace

:: **POP** je implementováno jako **prohledávání prostoru částečně uspořádaných plánů**.

- stavy prohledávacího prostoru jsou (většinou nedokončené) plány,
- začíná se s prázdným plánem,
- plán se postupně upravuje, dokud nemáme plán, který zcela řeší danou úlohu,
- akce při prohledávání jsou akce prováděné nad plány – přidej akci (např. $LeftSock$) do plánu.

:: **4 komponenty POP plánu**

- **Akce** – akce z mnoziny akcí definovaných v dané plánovací úloze. Počáteční prázdný plán obsahuje pouze akce
  - $Start.$ – nemá žádné počáteční podmínky, efektem akce jsou všechny literály počátečního stavu plánovacího problému.
  - $Finish$ – nemá žádný efekt, počáteční podmínky jsou cílové literály plánovacího problému.
- **Omezení na uspořádání** – omezení $A \prec B$ znamená, že akce A musí být vykonána *někdy* před akcí B.

  Cykly typu $A \prec B$ a $B \prec A$ jsou nepřípustné.

Partial-order plan: Formulace 2

- **Kauzální vazby** – kauzální vazba mezi akcemi A a B se zapisuje

  $$A \rightarrow^{p} B$$

  a znamená, že A zajistí podmínku p pro B.

  Kauzální vazba zabraňuje znegování podmínky p od okamžiku akce A až do akce B.

  Akce C **koliduje** s kauzální vazbou $A \rightarrow^{p} B$, jestliže C má efekt $\neg p$ a může přijít někdy po akci A a zároveň před akcí B.

  Př.: $RightSock \rightarrow^{RightSockOn} RightShoe$ říká, že $RightSockOn$ je efekt akce $RightSock$ a výchozí podmínkou akce $RightShoe$.

- **Otevřené výchozí podmínky** – výchozí podmínka je *otevřená*, jestliže není dosažena žádnou akcí v plánu. Cílem je redukovat množinu otevřených výchozích podmínek na prázdnou množinu.

:: **Konzistentní plán** neobsahuje cykly v omezeních na uspořádání ani konflikty v kauzálních vazbách.

- **řešení** je konzistentní plán s prázdnou množinou otevřených výchozích podmínek.

Partial-order plan: Příklad

Příklad: Nazouvání bot

$$\text{Actions:}\{RightSock, RightShoe, LeftSock, LeftShoe, Start, Finish\}$$
$$\text{Orderings:}\{RightSock \prec RightShoe, LeftSock \prec LeftShoe\}$$
$$\text{Links:}\{RightSock \xrightarrow{RightSockOn} RightShoe, LeftSock \xrightarrow{LeftSockOn} LeftShoe,$$
$$RightShoe \xrightarrow{RightShoeOn} Finish, LeftShoe \xrightarrow{LeftShoeOn} Finish\}$$
$$\text{Open Preconditions:}\{\}.$$

Partial-order planning: Formulace

:: **Formulace POP prohledávacího problému:**

- **počáteční stav** – počáteční plán s akcemi $Start$ a $Finish$, omezením na uspořádání $Start \prec Finish$, prázdnou množinou kauzálních vazeb a množinou otevřených výchozích podmínek, která obsahuje všechny výchozí podmínky akce $Finish$.
- Funkce **následník** vybere náhodně jednu z otevřených výchozích podmínek p akce B a **vygeneruje následující plán**, pro každý možný konzistentní způsob výběru akce A, která dosáhne p. Konzistence je zajištěna následovně:
  1. Kauzální vazba $A \rightarrow^p B$ a omezení na uspořádání $A \prec B$ jsou přidány do plánu. Pokud A je nová akce, přidej do plánu rovněž $Start \prec A$ a $A \prec Finish$.
  2. Vyřeš konflikty mezi novou kauzální vazbou a všemi existujícími akcemi a mezi akcí A a všemi existujícími kauzálními vazbami.
- **Cílový test** prověří, jestli je plán řešením původního plánovacího problému – množina otevřených výchozích podmínek je prázdná.

Partial-order planning: Příklad

:: Výměna pneumatiky

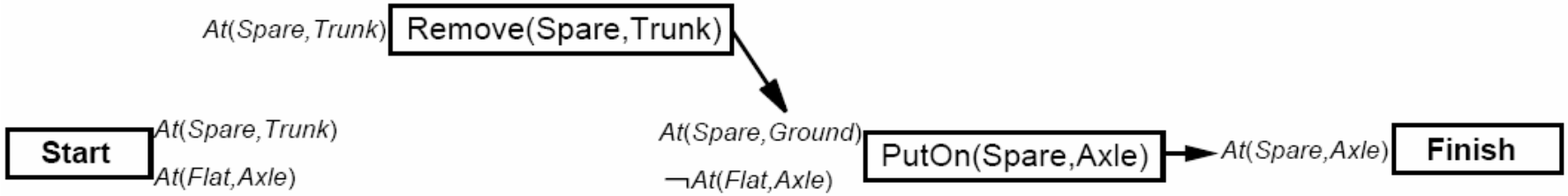
- **počáteční stav** – prasklá pneumatika je na ose, rezerva je v kufru.
- **cílový stav** – rezerva je na ose.
- **akce** – vyndej rezervu z kufru, odmontuj prasklou pneumatiku z osy, nasaď rezervu na osu, nech auto přes noc na ulici (efekt je, že všechny pneumatiky do rána zmizí).

$$
\begin{aligned}
&Init(At(Flat, Axle) \wedge At(Spare, Trunk))\\
&Goal(At(Spare, Axle))\\
&Action(Remove(Spare, Trunk),\\
&\quad \textsc{Precond}: At(Spare, Trunk)\\
&\quad \textsc{Effect}: \neg At(Spare, Trunk) \wedge At(Spare, Ground))\\
&Action(Remove(Flat, Axle),\\
&\quad \textsc{Precond}: At(Flat, Axle)\\
&\quad \textsc{Effect}: \neg At(Flat, Axle) \wedge At(Flat, Ground))\\
&Action(PutOn(Spare, Axle),\\
&\quad \textsc{Precond}: At(Spare, Ground) \wedge \neg At(Flat, Axle)\\
&\quad \textsc{Effect}: \neg At(Spare, Ground) \wedge At(Spare, Axle))\\
&Action(LeaveOvernight,\\
&\quad \textsc{Precond}:\\
&\quad \textsc{Effect}: \neg At(Spare, Ground) \wedge \neg At(Spare, Axle) \wedge \neg At(Spare, Trunk)\\
&\qquad\qquad \wedge \neg At(Flat, Ground) \wedge \neg At(Flat, Axle))
\end{aligned}
$$

Partial-order planning: Příklad (2)

:: **Postup hledání úplného částečně uspořádaného plánu:**

- Vybereme otevřenou výchozí podmínku $At(Spare, Axle)$ akce $Finish$.
  Vybereme akci, která dosáhne tuto podmínku – jediná možná je $PutOn(Spare, Axle)$.
- Vybereme výchozí podmínku $At(Spare, Ground)$ akce $PutOn(Spare, Axle)$. Zvolíme aplikovatelnou akci, která tuto podmínku dosáhne – jediná možná je $Remove(Spare, Trunk)$.



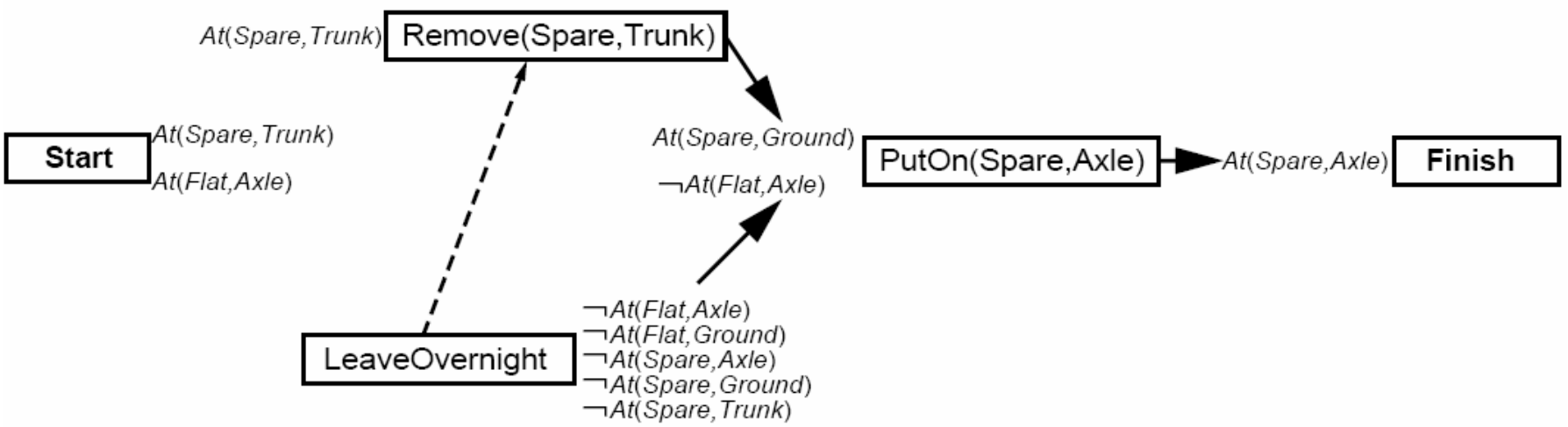
Partial-order planning: Příklad (3)

- Zvolíme otevřenou výchozí podmínku $\neg At(Flat, Axle)$ akce $PutOn(Spare, Axle)$.
  Vybereme aplikovatelnou akci $LeaveOvernight$, kterou jí můžeme dosáhnout.
  Akce $LeaveOvernight$ má efekt $\neg At(Spare, Ground)$, který koliduje s kauzální vazbou

$$Remove(Spare, Trunk) \rightarrow^{At(Spare,Ground)} PutOn(Spare, Axle).$$

  Tento **konflikt vyřešíme** přidáním omezení na uspořádání

$$LeaveOvernight \prec Remove(Spare, Trunk).$$



Partial-order planning: Příklad (4)

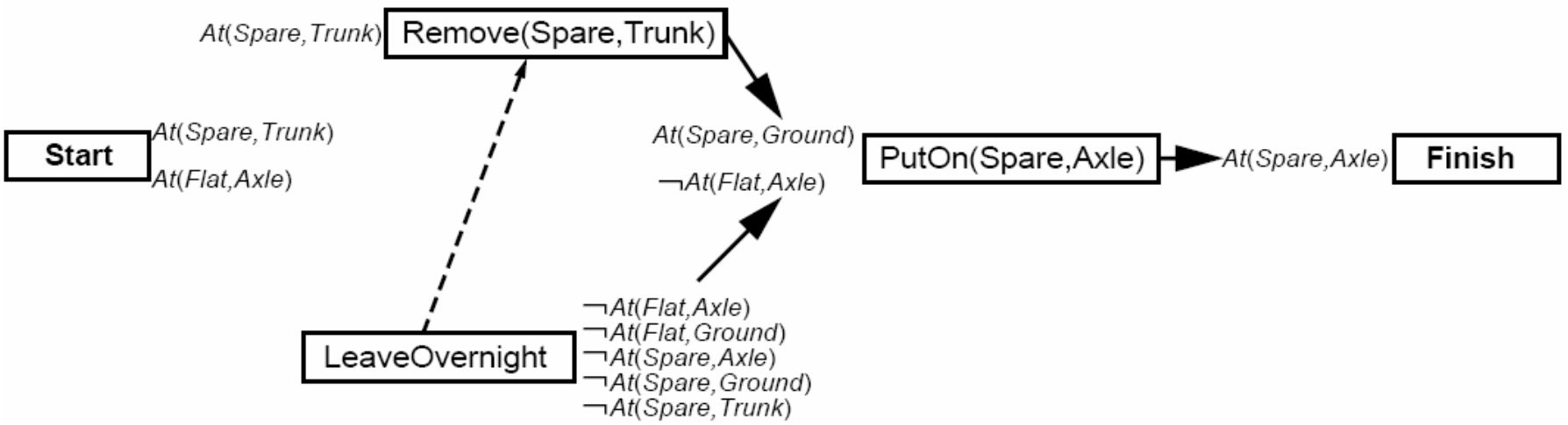
- Poslední otevřenou výchozí podmínkou je $At(Spare, Trunk)$ akce $Remove(Spare, Trunk)$. Jedinou akcí, kterou jí můžeme dosáhnout je akce $Start$, ovšem kauzální vazba

$$Start \rightarrow^{At(Spare,Trunk)} Remove(Spare, Trunk)$$

koliduje s efektem $\neg At(Spare, Trunk)$ akce $LeaveOvernight$.

Tento konflikt nelze odstranit, musíme se vrátit o krok zpět

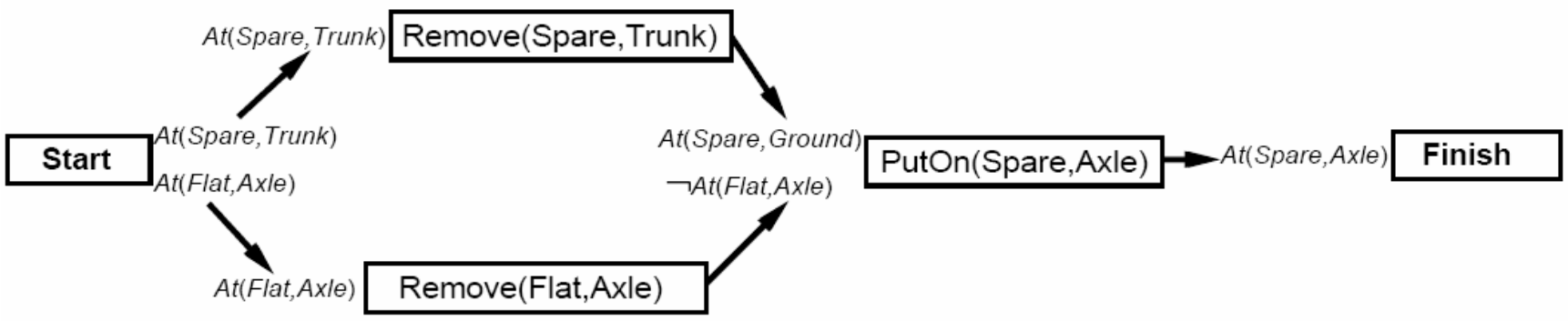
– odstranit akci $LeaveOvernight$ a poslední dvě kauzální vazby.



Partial-order planning: Příklad (5)

- Znovu uvažujeme otevřenou výchozí podmínku $\neg At(Flat, Axle)$ akce $PutOn(Spare, Axle)$. Pro její dosažení nyní zvolíme akci $Remove(Flat, Axle)$.
- Znovu vezmeme podmínku $At(Spare, Trunk)$ akce $Remove(Spare, Trunk)$ a zvolíme akci $Start$ pro její dosažení.
- Vezmeme podmínku $At(Flat, Axle)$ akce $Remove(Flat, Axle)$ a zvolíme akci $Start$ pro její dosažení.

  Tím získáme **kompletní konzistentní plán – řešení**.



- Kauzální vazby umožňují prořezávat prohledávaný prostor.
- Částečně uspořádaný plán poskytuje flexibilitu při konstrukci úplně uspořádaného plánu.

Partial-order planning s volnými proměnnými

Příklad: Předpokládejme úlohu ze světa kostek, kde máme otevřenou výchozí podmínku $On(A, B)$ a akci $Move(b, x, y)$ definovanou následovně

$$\begin{array}{l} Action(Move(b, x, y), \\ \quad \textsc{Precond}: On(b, x) \wedge Clear(b) \wedge Clear(y), \\ \quad \textsc{Effect}: On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y)) \end{array}$$

Tato akce zajistí cíl $On(A, B)$, protože její efekt $On(b, y)$ se unifikuje s $On(A, B)$ s použitím substituce $\{b/A, y/B\}$.

Aplikací této substituce na akci dostáváme akci

$$\begin{array}{l} Action(Move(A, x, B), \\ \quad \textsc{Precond}: On(A, x) \wedge Clear(A) \wedge Clear(B), \\ \quad \textsc{Effect}: On(A, B) \wedge Clear(x) \wedge \neg On(A, x) \wedge \neg Clear(B)) \end{array}$$

ve které zůstává proměnná x volná. Toto může být vyřešeno až v některém dalším kroku. Například akce $Start$ s efektem $On(A, D)$ může být použita k zajištění cíle $On(A, x)$ svázáním x a D.

Partial-order planning s volnými proměnnými

:: Takovéto pozdržené ohodnocení dané proměnné může být často mnohem efektivnější než opakované zkoušení různých hodnot proměnné pomocí backtrackingu.

:: Na druhou stranu, **použití predikátové logiky s proměnnými** pro reprezentaci akcí trošku komplikuje detekci a řešení konfliktů.

Příklad: Přidání akce $Move(A, x, B)$ do plánu znamená také přidání kauzální vazby

$$Move(A, x, B) \xrightarrow{On(A,B)} Finish$$

Pokud je v plánu i jiná akce M_2 s efektem $\neg On(A, z)$, bude M_2 v konfliktu když z je B. Musíme tedy rozšířit reprezentaci plánů o omezení nerovnosti typu $z \neq X$, kde z je proměnná a X je proměnná nebo konstanta.

V našem případě tedy omezení $x \neq B$ znamená, že plán může být rozšířen tak, že za z může být dosazena jakékoliv hodnota kromě B.

Literatura

Stuart Russell and Peter Norvig: Artificial Intelligence: A Modern Approach

- **Part IV Planning**

  11 Planning (http://aima.cs.berkeley.edu/2nd-ed/)