Plánování a rozvrhování

4

Roman Barták, KTIML

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

Pro zopakování

Plánovací algoritmy jsou založeny na prohledávání.

- **Plánování ve stavovém prostoru**
  - uzly odpovídají stavům
  - jdeme buď od počátečního stavu k cíli nebo obráceně

  Problémy:
  - **velký větvící faktor**
    - technika liftování zmenšuje větvící faktor (instanciace proměnných je odložena)
  - **alternativní pořadí akcí vedoucí k neúspěchům**
    - nemusíme vyžadovat konkrétní pořadí akcí, dokud není skutečně potřeba
  - **strategie nejmenších závazků** (least-commitment)
    - co není potřeba rozhodnout hned se odloží na později

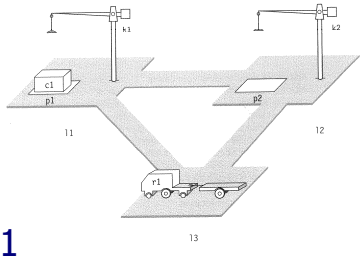
Plánování
v prostoru plánů

Základní myšlenka

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

Příklad

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



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

Plánování

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

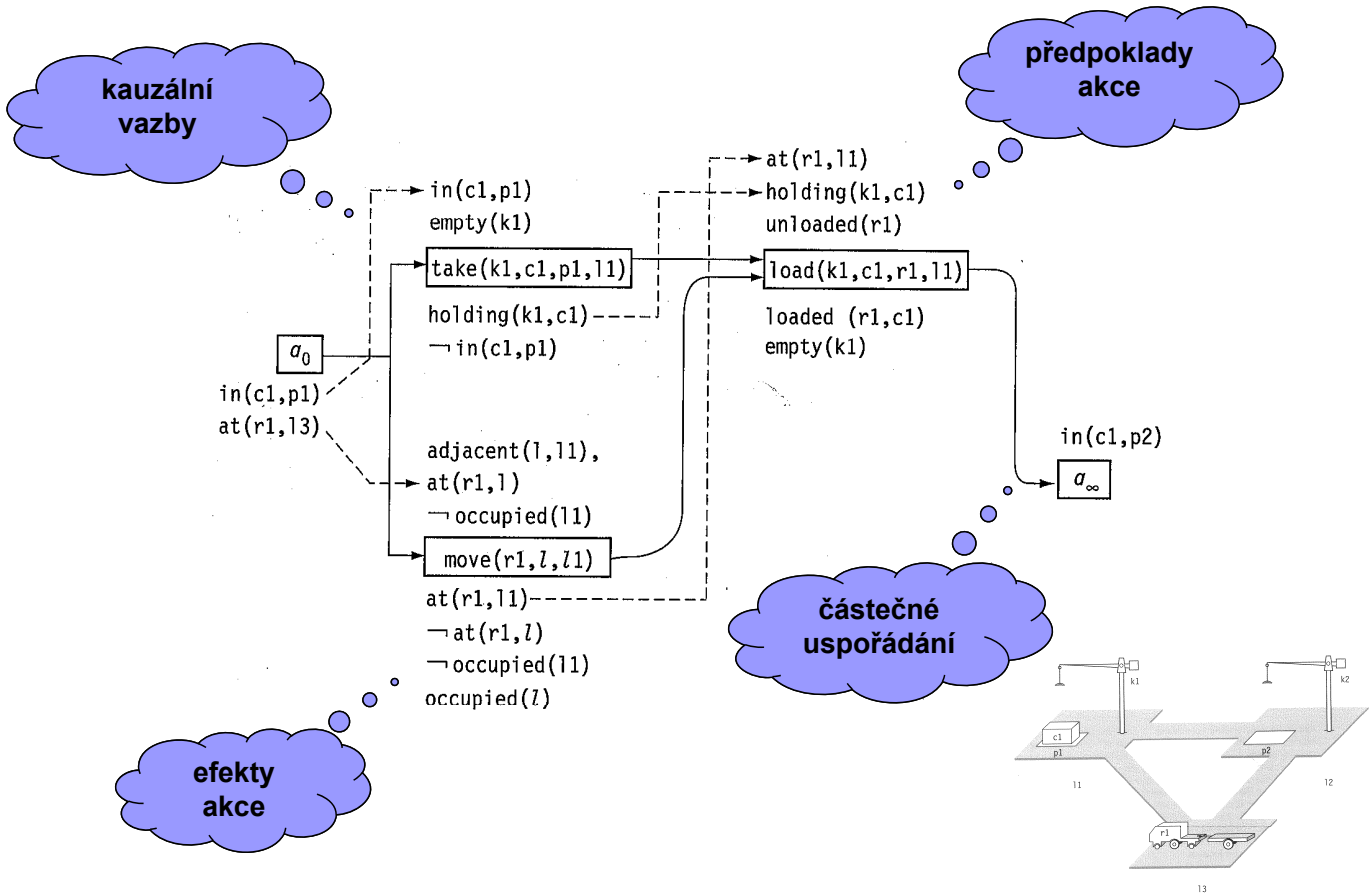
Uzly jsou plány

Uzly prohledávaného prostoru jsou tvořeny částečnými plány.

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

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

Částečný plán: příklad

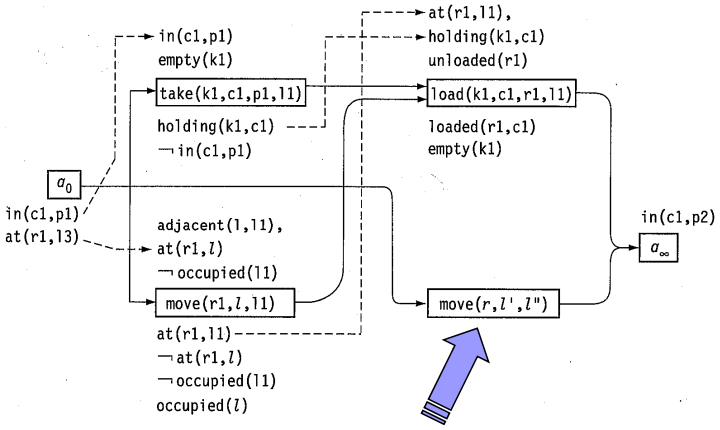


Otevřený cíl

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

Hrozba

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



Řešící plán

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

Řešící plán jinak

Jak efektivně ověřit, zda je daný plán řešící?

Tvrzení:

Částečný plán $\Pi = (A,<,B,L)$ je řešící pokud:

- nemá žádné kazy, tj. otevřené cíle ani hrozby
- částečné uspořádání < a vazby B jsou globálně konzistentní

Důkaz indukcí podle délky plánu

- $\{a_0,a_1,a_\infty\}$ je řešící plán
- pro větší množiny akcí vyber libovolnou z možných prvních akcí a spoj ji s akcí a_0

Procedura PSP

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

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

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

Detaily PSP

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

Vlastnosti PSP

Procedura PSP je **korektní a úplná**.

- **korektnost**
  - pokud program skončí, pak v něm není žádný kaz a plán je konzistentní.
- **úplnost**
  - pokud existuje plán, má procedura PSP vždy možnost vybrat ty správné akce

- Pozor na **deterministickou implementaci**!
  - **Prostor plánů není konečný!**
  - Úplná deterministická procedura musí garantovat prohledání všech plánů do dané délky, např. iterativní prohlubování (iterative deepening).

Procedura PoP

PoP je konkrétní (a populární) instance algoritmu PSP.

```
PoP(π, agenda)          ;; where π = (A, ≺, B, L)
    if agenda = ∅ then return(π)
    select any pair (a_j, p) in and remove it from agenda
    relevant ← Providers(p, π)
    if relevant = ∅ then return(failure)
    nondeterministically choose an action a_i ∈ relevant
    L ← L ∪ {⟨a_i --p--> a_j⟩}
    update B with the binding constraints of this causal link
    if a_i is a new action in A then do:
        update A with a_i
        update ≺ with (a_i ≺ a_j), (a_0 ≺ a_i ≺ a_∞)
        update agenda with all preconditions of a_i
    for each threat on ⟨a_i --p--> a_j⟩ or due to a_i do:
        resolvers ← set of resolvers for this threat
        if resolvers = ∅ then return(failure)
        nondeterministically choose a resolver in resolvers
        add that resolver to ≺ or to B
    return(PoP(π, agenda))
end
```

- **Agenda** je množina dvojic (a,p), kde p je otevřený předpoklad akce a.
- **Nejprve hledá akci** a_i pro pokrytí nějakého p z agendy.
- **Ve druhé fázi řeší všechny hrozby**, které vznikly přidáním akce a_i resp. kauzální vazby s a_i.

Příklad na závěr

- **Počáteční stav:**
  - At(Home), Sells(OBI,Drill), Sells(Tesco,Milk), Sells(Tesco,Banana)
  - tj. akce **Start:**
    Precond: none
    Effects: At(Home), Sells(OBI,Drill), Sells(Tesco,Milk), Sells(Tesco,Banana)
- **Cíl:**
  - Have(Drill), Have(Milk), Have(Banana), At(Home)
  - tj. akce **Finish:**
    Precond: Have(Drill), Have(Milk), Have(Banana), At(Home)
    Effects: none
- Máme k dispozici následující **operátory**:
  - **Go(*l*,*m*)** ;; jdi z místa *l* do místa *m*
    Precond: At(*l*)
    Effects: At(*m*), ¬At(*l*)
  - **Buy(*p*,*s*)** ;; na místě *s* kup *p*
    Precond: At(*s*), Sells(*s*,*p*)
    Effects: Have(*p*)

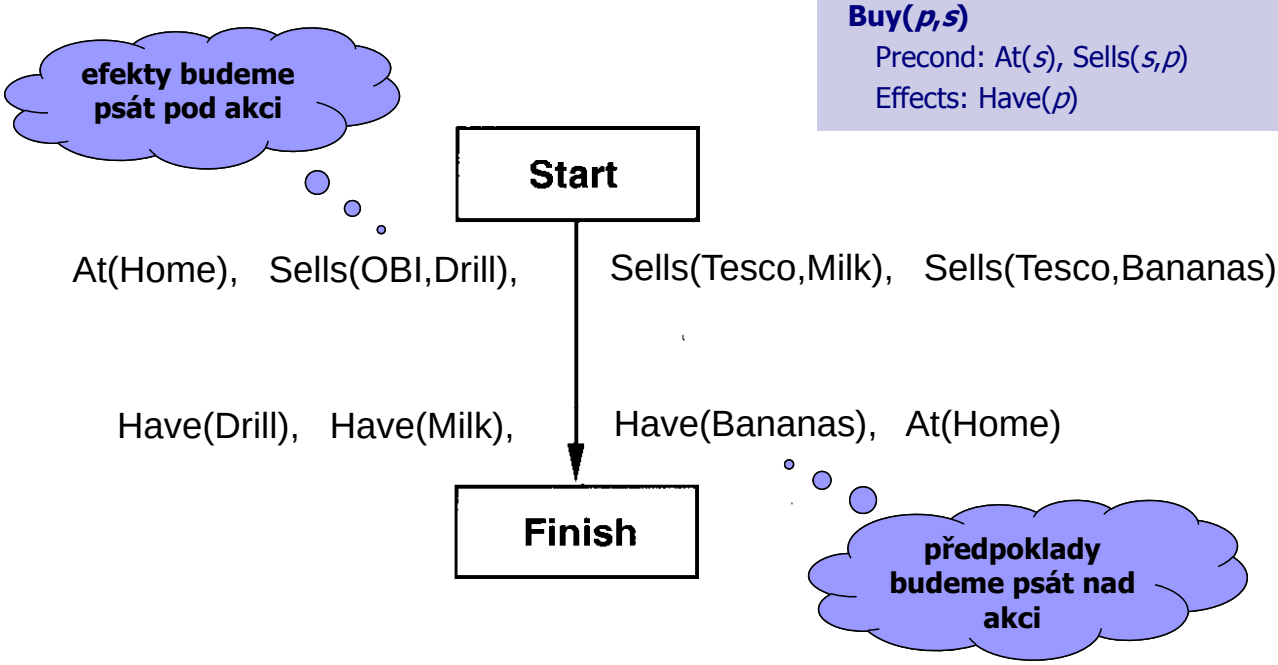
Příklad na závěr

- **Počáteční plán**

Operátory

Go(*l*,*m*)
Precond: At(*l*)
Effects: At(*m*), ¬At(*l*)

Buy(*p*,*s*)
Precond: At(*s*), Sells(*s*,*p*)
Effects: Have(*p*)



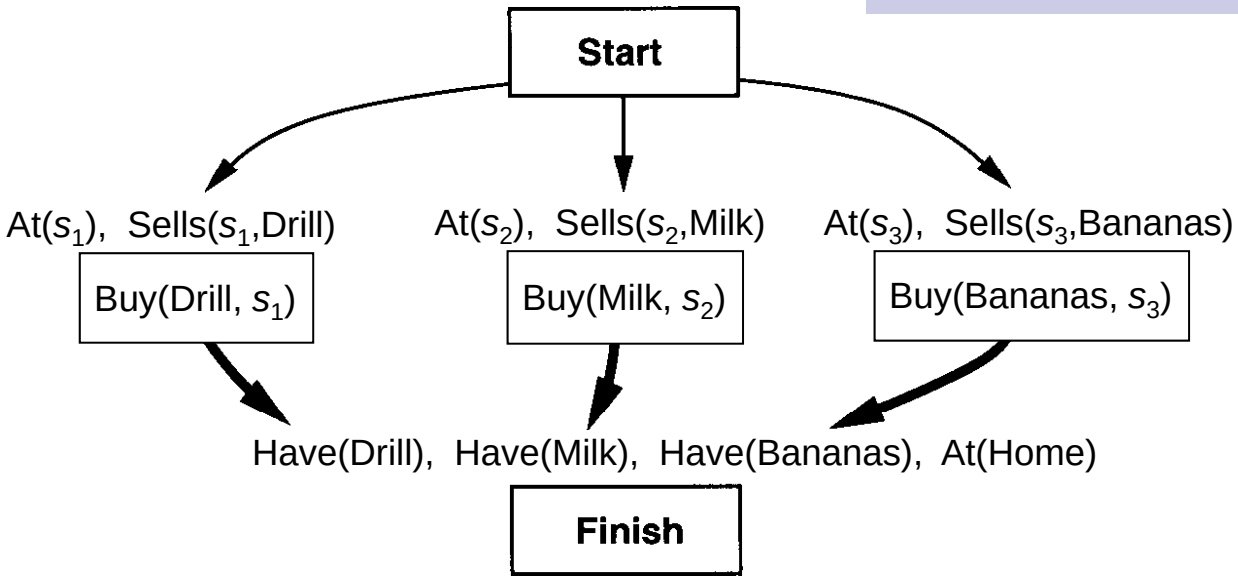
Příklad na závěr

- jediný způsob, jak **splnit otevřené cíle Have**, je **akcemi Buy** (které netvoří žádné hrozby)

Operátory

Go(*l*,*m*)
Precond: At(*l*)
Effects: At(*m*), ¬At(*l*)

Buy(*p*,*s*)
Precond: At(*s*), Sells(*s*,*p*)
Effects: Have(*p*)



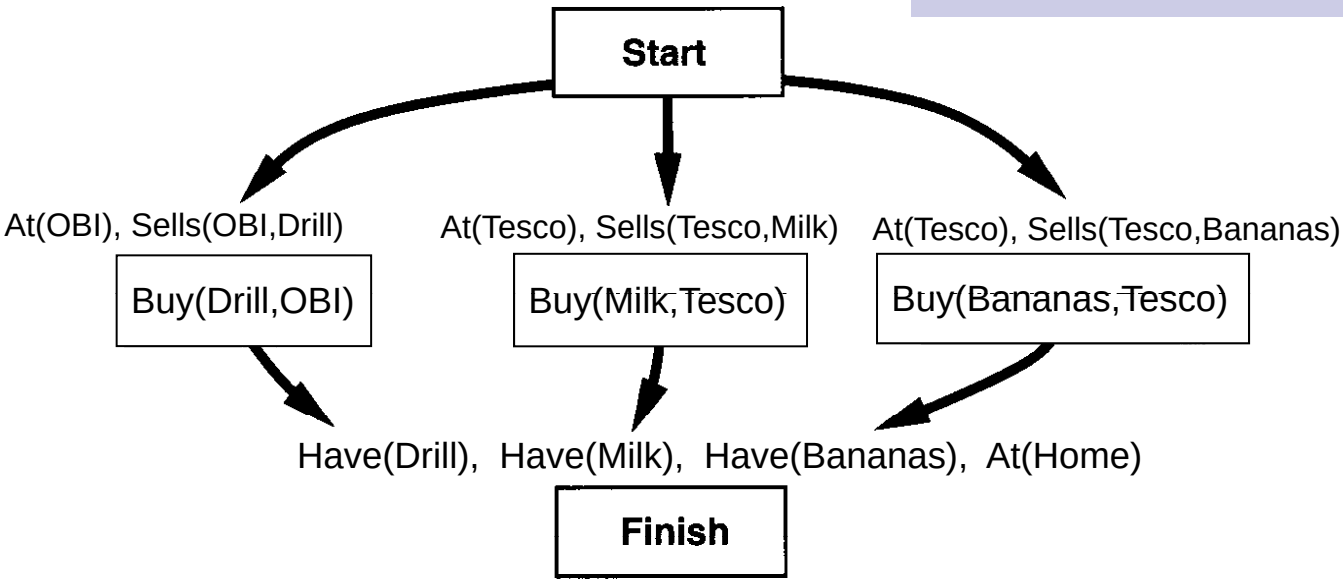
Příklad na závěr

- jediný způsob, jak **splnit předpoklady Sells** je dosazení příslušných **konstant**

Operátory

Go(*l*,*m*)
Precond: At(*l*)
Effects: At(*m*), ¬At(*l*)

Buy(*p*,*s*)
Precond: At(*s*), Sells(*s*,*p*)
Effects: Have(*p*)



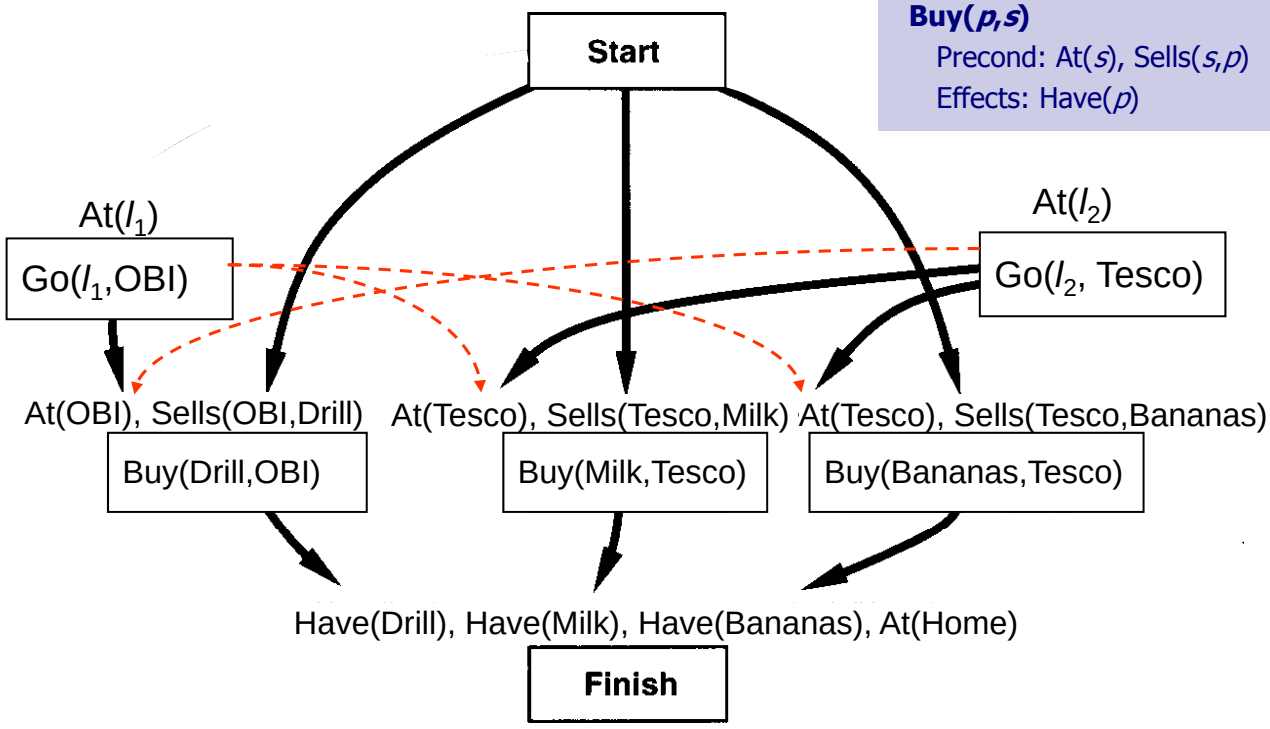
Příklad na závěr

- jediný způsob, jak **splnit otevřené cíle At**, je přidání **akcí Go**
  - přibyly nové hrozby!

Operátory

Go(*l*,*m*)
Precond: At(*l*)
Effects: At(*m*), ¬At(*l*)

Buy(*p*,*s*)
Precond: At(*s*), Sells(*s*,*p*)
Effects: Have(*p*)



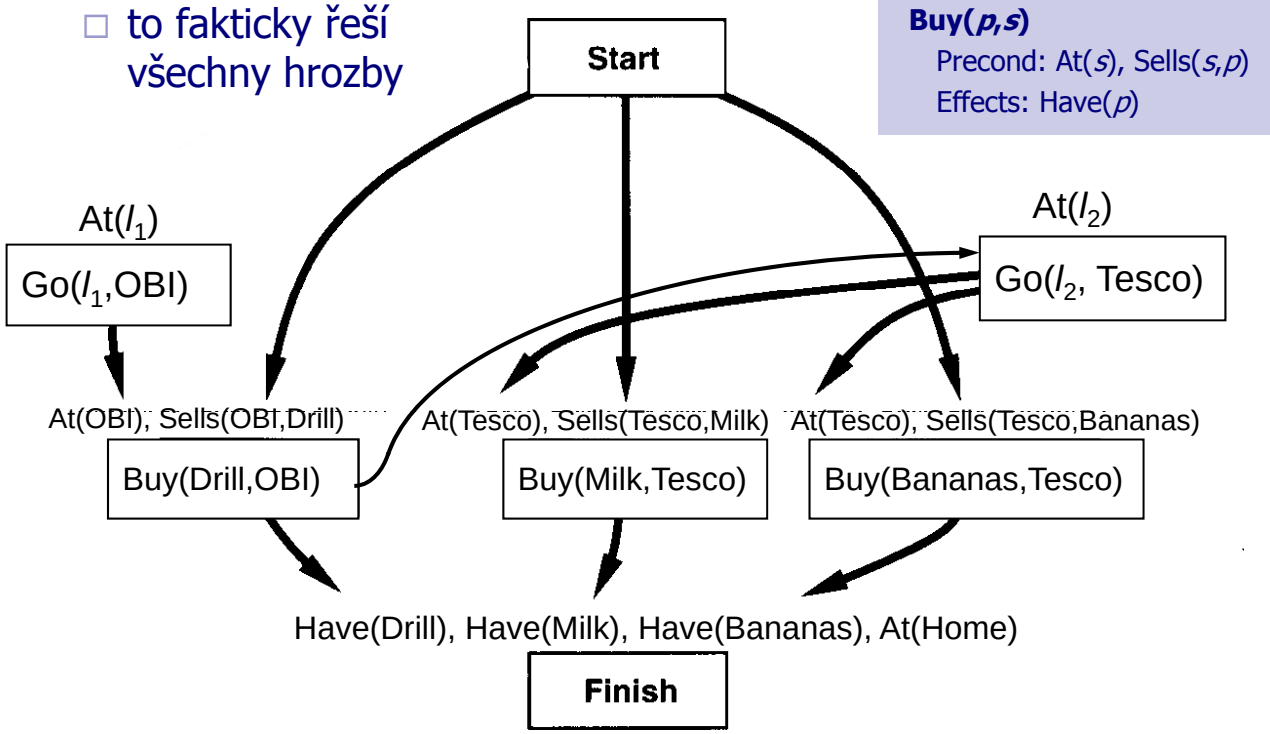
Příklad na závěr

- **třetí hrozbu** můžeme odstranit **uspořádáním akce** Buy(Drill) před Go(Tesco)
  - to fakticky řeší všechny hrozby

Operátory

Go(*l*,*m*)
Precond: At(*l*)
Effects: At(*m*), ¬At(*l*)

Buy(*p*,*s*)
Precond: At(*s*), Sells(*s*,*p*)
Effects: Have(*p*)



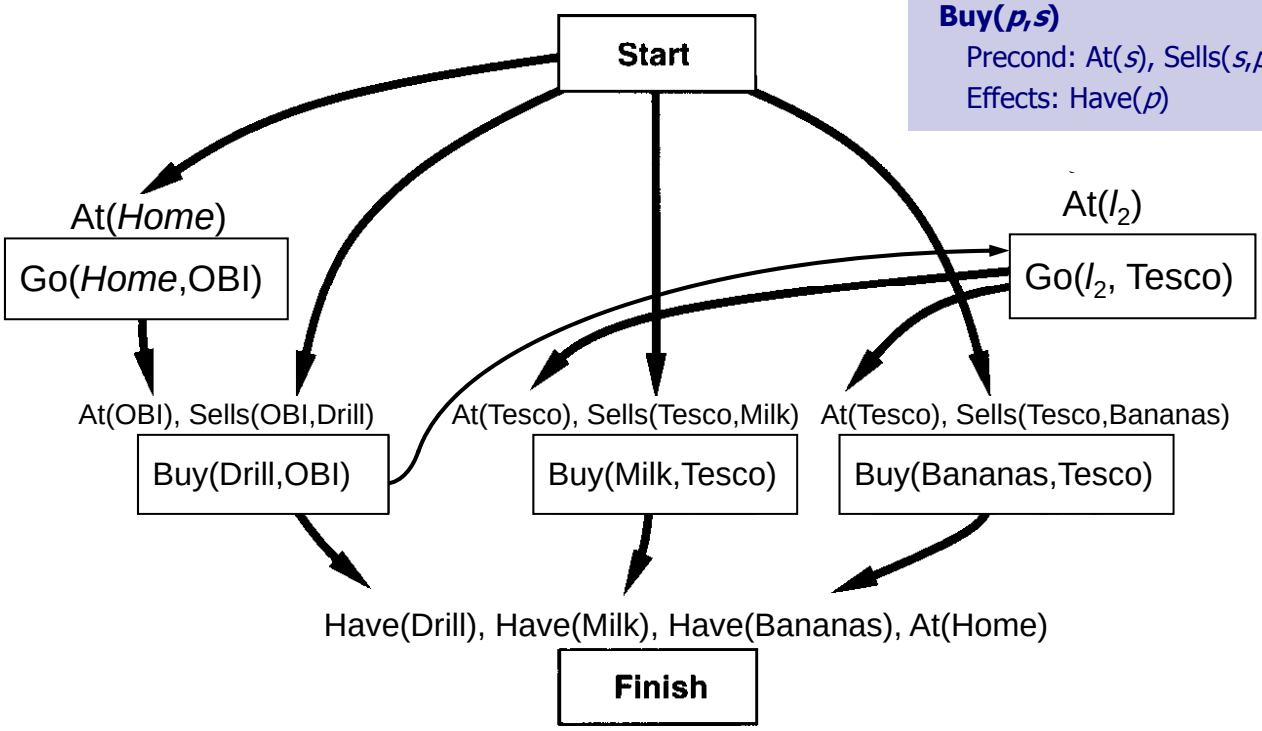
Příklad na závěr

- **otevřený cíl At(l_1)** můžeme splnit **přiřazením** l_1=Home z akce Start

Operátory

Go(*l*,*m*)
Precond: At(*l*)
Effects: At(*m*), ¬At(*l*)

Buy(*p*,*s*)
Precond: At(*s*), Sells(*s*,*p*)
Effects: Have(*p*)



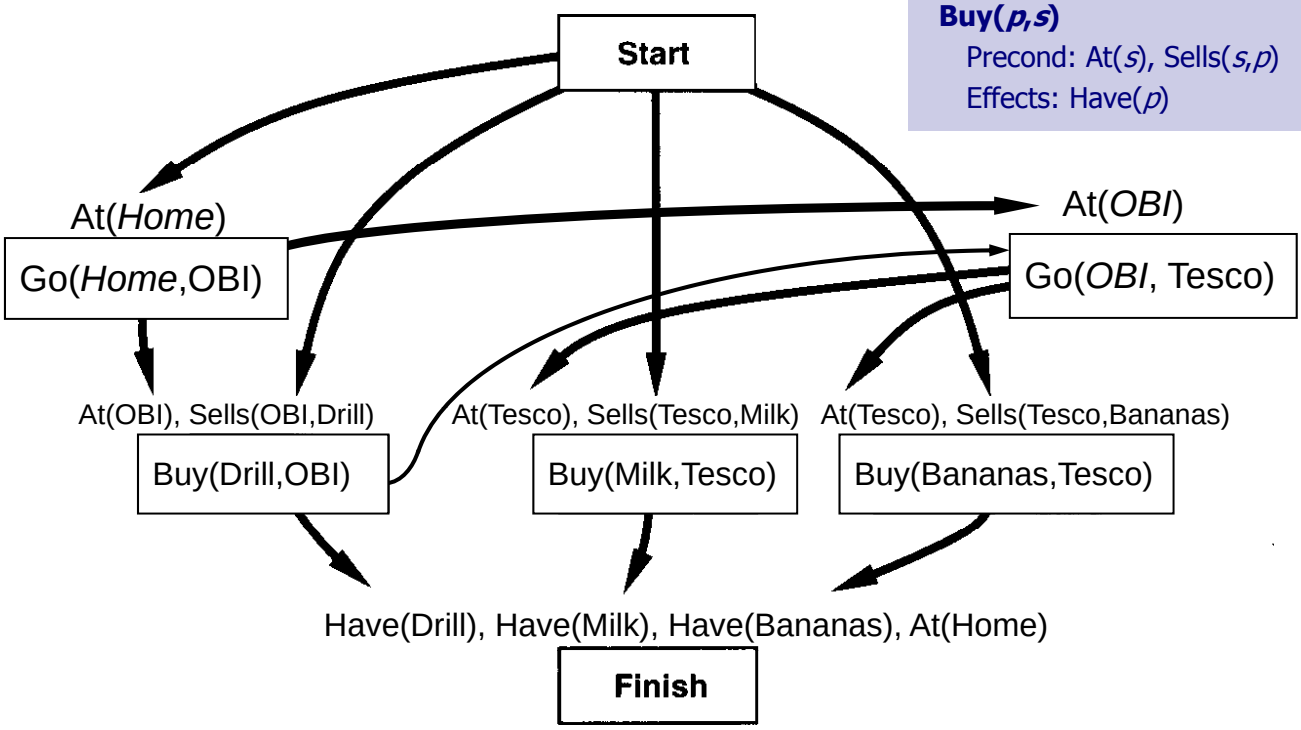
Příklad na závěr

- **otevřený cíl At(l_2)** můžeme splnit **přiřazením** l_2=OBI z akce Go(Home, OBI)

Operátory

Go(*l*,*m*)
Precond: At(*l*)
Effects: At(*m*), ¬At(*l*)

Buy(*p*,*s*)
Precond: At(*s*), Sells(*s*,*p*)
Effects: Have(*p*)



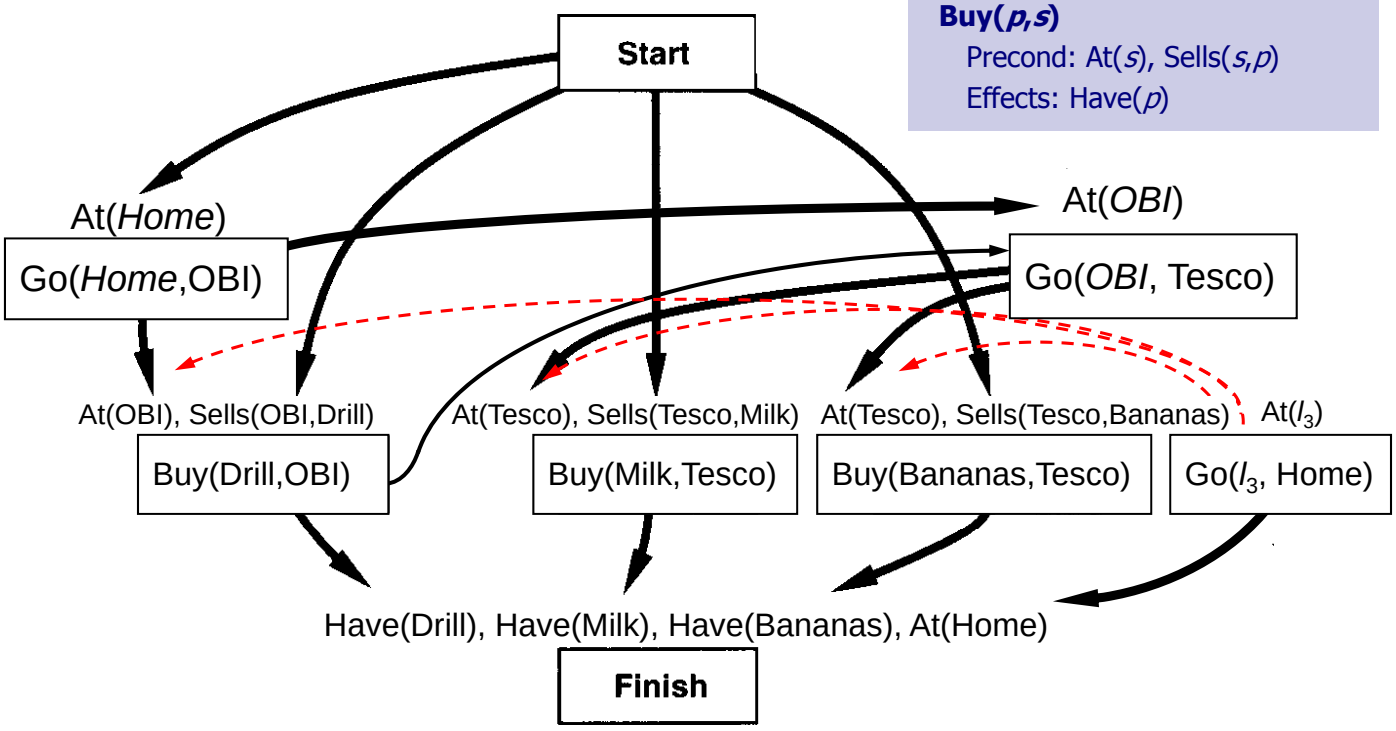
Příklad na závěr

- **otevřený cíl** At(Home) z Finish splníme **akcí** Go
  - vzniknou nové hrozby

Operátory

Go(*l*,*m*)
Precond: At(*l*)
Effects: At(*m*), ¬At(*l*)

Buy(*p*,*s*)
Precond: At(*s*), Sells(*s*,*p*)
Effects: Have(*p*)



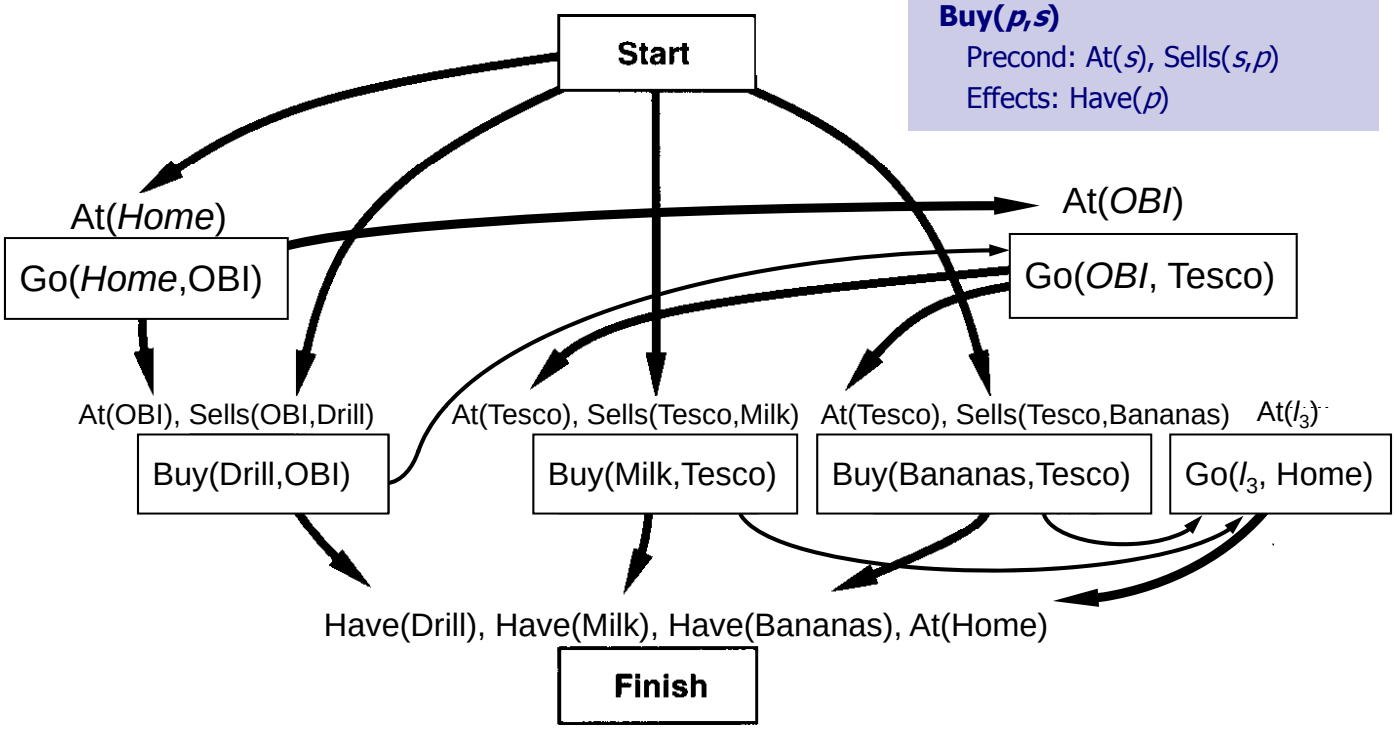
Příklad na závěr

- **hrozby** na At(Tesco) odstraníme **uspořádáním** akce Go(Home) za obě akce Buy

Operátory

Go(*l*,*m*)
Precond: At(*l*)
Effects: At(*m*), ¬At(*l*)

Buy(*p*,*s*)
Precond: At(*s*), Sells(*s*,*p*)
Effects: Have(*p*)



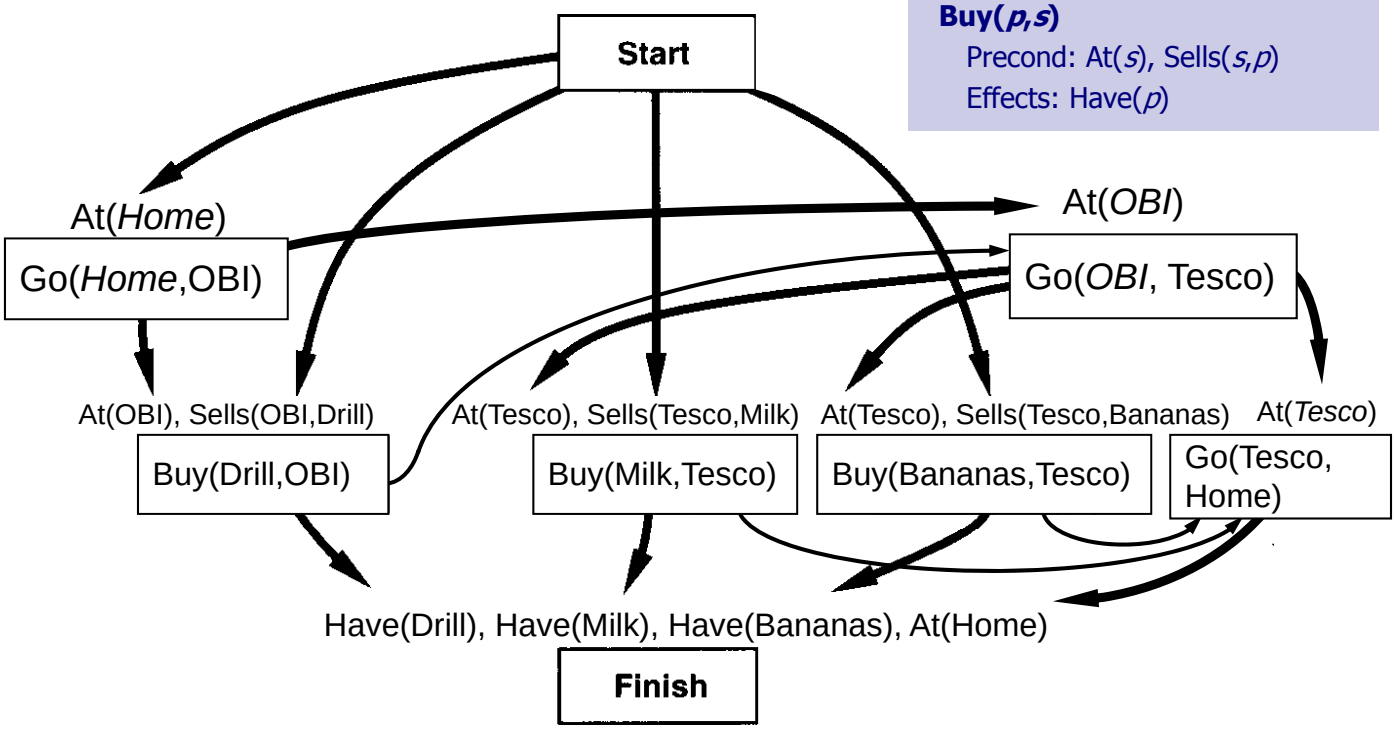
Finále

- **otevřený cíl** At(l_3) splníme **přiřazením** l_3=Tesco z akce Go(OBI,Tesco)

Operátory

Go(*l*,*m*)
Precond: At(*l*)
Effects: At(*m*), ¬At(*l*)

Buy(*p*,*s*)
Precond: At(*s*), Sells(*s*,*p*)
Effects: Have(*p*)



Srovnání technik

	Plánování se stavy	Plánování s plány
prohled. prostor	konečný	nekonečný
uzly	jednoduché (stavy světa)	komplikovanější (částečné plány)
stavy světa	explicitní	nejsou
částečný plán	uspořádání a volba akcí se dělá najednou	volba akcí a jejich pořadí oddělené
struktura plánu	lineární bez vazeb	kauzální vazby

- Díky doménově specifickým heuristikám je dnes **plánování se stavy výrazně rychlejší**.
- Ale, **plánování v prostoru plánů**:
  - vytváří **flexibilnější plány** díky částečnému uspořádání
  - umožňuje další **rozšíření**, např. přidání času a zdrojů