



Umělá inteligence I

Roman Barták, KTIML

roman.bartak@mff.cuni.cz

<http://ktiml.mff.cuni.cz/~bartak>



Co je UI?

- Umělá inteligence je věda o vytváření strojů nebo systémů, které budou při řešení určitého úkolu užívat takového postupu, který – kdyby ho dělal člověk – bychom považovali za projev jeho inteligence.

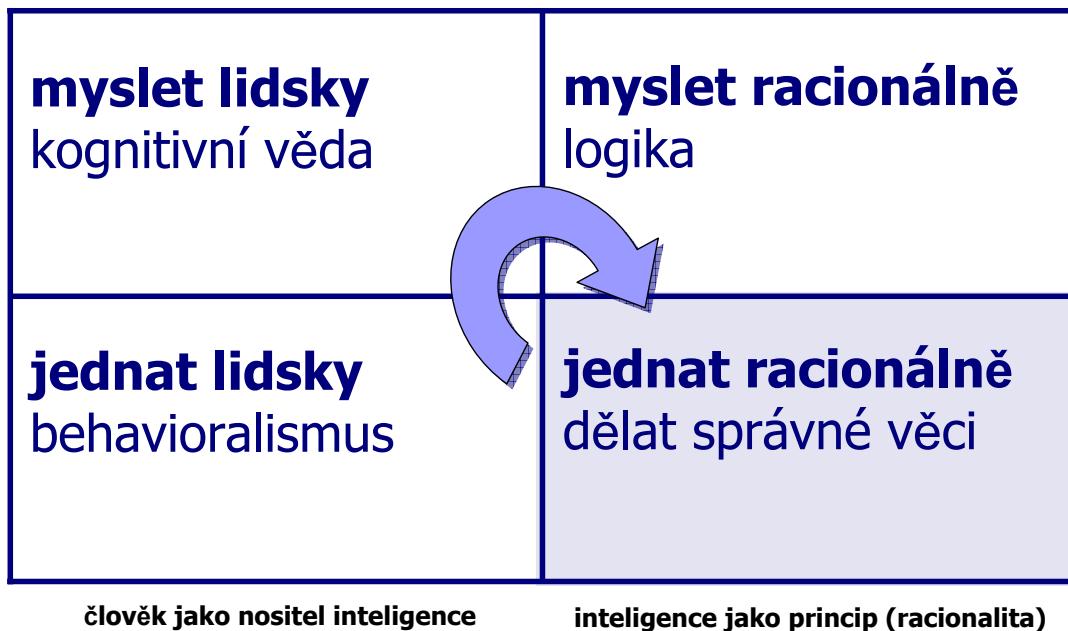
Marvin Minsky, 1967

... jako je třeba hraní fotbalu ☺





způsob řešení (cesta)



člověk jako nositel inteligence

inteligence jako princip (racionality)

Umělá inteligence I, Roman Barták



Jednat lidsky

- **Alan Turing** (1950) se pokusil nalézt operační definici inteligence.

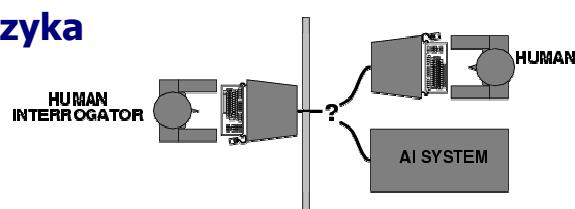
- „Mohou stroje myslet?“
 - ↳ „Mohou se stroje chovat *jako lidé* inteli>gentně?“

- Turingův test**

Stroj projde testem, pokud není člověk na základě písemné komunikace schopen rozpoznat, zda na druhé straně je člověk nebo stroj.

- Potřebné schopnosti:**

- zpracování přirozeného jazyka
- reprezentace znalostí
- automatické uvažování
- strojové učení



Umělá inteligence I, Roman Barták



A znáte tohle?

■ Reverzní Turingův test

aneb jak zjistit, zda u počítače sedí člověk
a ne robot

Security Check: A Reverse Turing Test

In order to eliminate cheaters using automated bots, please enter the following eight characters into the text box below and click OK. You must do this within 2 minutes. Sorry for the inconvenience.

CX82\$2\$A

Enter text here: OK

If the characters do not appear, you can try reloading: [Reload](#)

Mail Center Send a Message

Please enter the text from the image above:
The letters are not case-sensitive.
Do not type spaces between the numbers and letters.

Please enter the image text: Submit

Umělá inteligence I, Roman Barták



Myslet lidsky

■ Kognitivní modelování

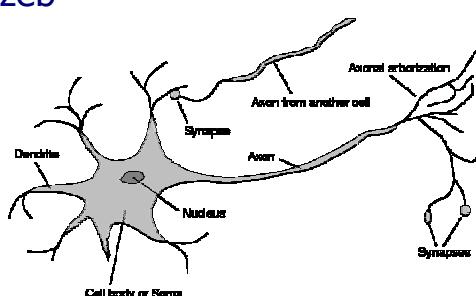
- modelujeme lidské myšlení
- potřebujeme vědět, jak mozek funguje

■ Přístup shora (**psychologie**)

- stejný mechanismus řešení problémů jaký používají lidé
- GPS – General Problem Solver (Newell & Simon, 1957)

■ Přístup zdola (**neurověda**)

- modelování neuronů a jejich vazeb
- konekcionismus
- „inteligentní chování se vynoří spojením velkého počtu jednoduchých elementů“



Umělá inteligence I, Roman Barták



Myslet racionálně

- Už od dob **Aristotela** (384 – 322 BC) se lidé zabývají hledáním „pravidel myšlení“.

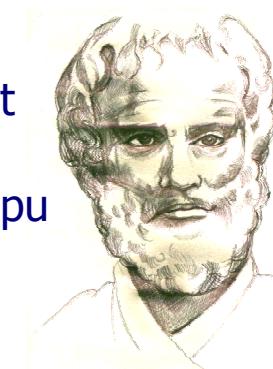
- Sylogismy

- vzory správného uvažování, které ze správných vstupů odvodí správně závěry
 - Sokrates je člověk, všichni lidé jsou smrtelní
⇒ Sokrates je smrtelný

- základy moderní **logiky** (a matematiky)

- **Problémy:**

- Jak pro logické odvozování formálně zachytit neformální znalost, která není 100% jistá?
 - Je velký rozdíl umět vyřešit problém v principu a problém skutečně vyřešit!



Umělá inteligence I, Roman Barták



Jednat racionálně

- **Racionální chování** = dělat „správné věci“
- **„správná věc“** = dosáhnout co nejlepší (očekávaný) výstup pro dané (i nejisté) vstupy
- Dělat správná odvození (myslet racionálně) je částí **racionálního agenta**, ale ne výlučnou.
 - Jsou situace, kde není žádná dokazatelná správná věc, kterou lze udělat, přesto je potřeba něco udělat.
 - Ne každé racionální chování obsahuje logické odvození (např. reflexy).
 - **Tato přednáška je o obecných principech racionálních agentů a o tom, jak takové agenty konstruovat.**



Umělá inteligence I, Roman Barták



Obsah přednášky

■ Úvod

- trochu historie, kontext, inteligentní agenti

■ Řešení problémů

- prohledávací algoritmy, splňování podmínek

■ Znalosti a uvažování

- logika a logické odvozování,
reprezentace znalostí

■ Plánování

- hledání plánů v umělém i reálném světě

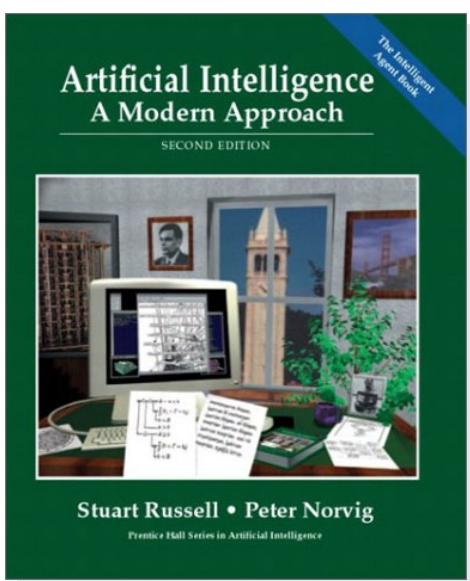


Umělá inteligence I, Roman Barták



Zdroje

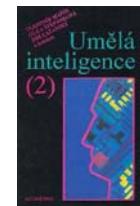
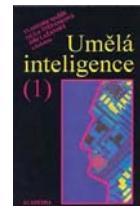
Artificial Intelligence: A Modern Approach



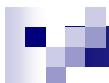
- S. Russell and P. Norvig
- Prentice Hall, 2003 (druhé v.)
- <http://aima.cs.berkeley.edu/>

Umělá inteligence 1-5

- Vladimír Mařík, Olga Štěpánková, Jiří Lažanský a kol.
- Academia



Umělá inteligence I, Roman Barták



Web přednášky

<http://ktiml.mff.cuni.cz/~bartak/ui>

The screenshot shows a Windows Internet Explorer window displaying the homepage of the course "Umělá inteligence I". The title bar reads "Umělá inteligence I - Windows Internet Explorer". The main content area includes:

- Course Information:** Umělá inteligence I, AII069, 2/0 ZK, zimní semestr, Roman Barták, KTIML.
- Navigation:** Zdroje | Přednáška | Zkouška | Kontakt.
- Text:** Umělá inteligence je věda o vytváření strojů nebo systémů, které budou při řešení určitého okruhu udržovat takový postup, který - kdyby ho dítal člověk - bychom považovali za provoz jeho inteligence.
- Image:** Marvin Minsky, 1967 (small portrait).
- Books:** Artificial Intelligence: A Modern Approach, Prentice Hall, 2002 (druhé vydání). Materials from the book are available at [this site](#).
- Topics:** Základy, Zdroje, Metody, Pracovní úlohy, Výpočetní architektury, Léčení, Pracovní úlohy, Výpočetní architektury, Léčení.
- Schedule:** Přednáška (ZS 2007/2008): čtvrtek 10:40 - 12:10, posluchačna S3 (Malá Strana, 3. patro). The schedule lists topics from October 4 to December 13, 2007.

Zde najdete:

- slajdy k přednášce**
- odkazy na zdroje**
- informace o zkoušce**
- ...

Umělá inteligence I, Roman Barták



Návaznosti

■ Seminář z umělé inteligence

- referativní seminář, kde téma určují účastníci

■ Programování s omezujícími podmínkami

- jak automaticky splňovat podmínky

■ Rozhodovací procedury a verifikace

- jak splňovat logické formule

■ Plánování a rozvrhování

- jak automaticky hledat libovolné plány

■ Strojové učení

- jak se počítače (sami či s pomocí) učí

■ ...

Umělá inteligence I, Roman Barták

Umělá inteligence čerpá myšlenky, ideje a techniky z mnoha oborů.

- **Filozofie** (428 BC -) jak v mozku vzniká mysl?
logika, metody uvažování
- **Matematika** (800 -) jak **formálně** odvodit platné závěry?
co lze spočítat?
- **Ekonomie** (1776 -) jak maximalizovat zisk?
tvorba **zisku**, rozhodovací procesy
- **Neurověda** (1861 -) jak **mozky** zpracovávají informaci?
fyzický substrát myšlení
- **Psychologie** (1879 -) jak lidé a zvířata myslí a konají?
behaviorismus
- **Počítačové inženýrství** (1940 -) jak postavit efektivní **počítač**?
stroje na zpracování informací
- **Teorie řízení** (1948 -) jak se mohou **artefakty** sami řídit?
systémy maximalizující efekt v čase
- **Lingvistika** (1957 -) jaký má **jazyk** vztah k myšlení?
reprezentace znalostí

Umělá inteligence I, Roman Barták

Historie UI

■ Před narozením (1943-1955)

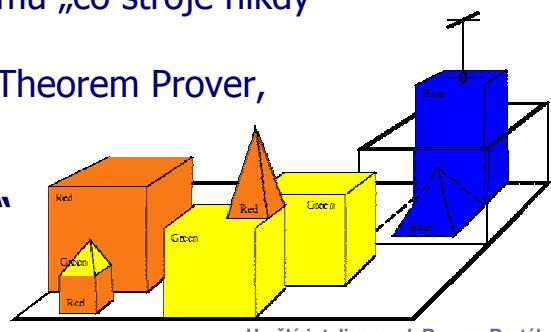
- W. McCulloch & W. Pitts: Booleovský model **neuronu**
- A. Turing: „**Computing Machinery and Intelligence**“
první kompletní vize UI

■ Narodila se UI (1956)

- Seminář v **Dartmouth** College, NH
- J. McCarthy: název **Artificial Intelligence**
- A. Newell & H. Simon: program **Logic Theorist**

■ Velká očekávání (1952-1969)

- postupné řešení problémů ze seznamu „co stroje nikdy nedokážou“
- General Problem Solver, Geometry Theorem Prover,
Lisp (1958), Analogy, svět kostek
- J. McCarthy: období typu
„Koukej mami, jedu bez rukou!“



Umělá inteligence I, Roman Barták

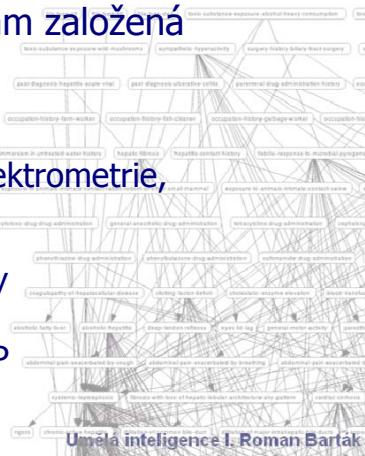
■ Poznání reality (1966-1973)

- stroje, které myslí, učí se a tvoří, ale jen na jednoduchých příkladech
 - Proč?
 - programy neobsahovaly **žádnou znalost problému**, dělaly jen syntaktické zpracování
 - **intractabilita** problémů (zkoušení všech kombinací kroků)
 - **fundamentální omezení** základních používaných struktur (perceptron se naučí vše, co umí reprezentovat, ale může toho reprezentovat hodně málo)

■ Znalostní systémy (1969-1979)

- alternativa ke „slabým“ obecným řešícím metodám založená na doménově specifických znalostech
 - expertní (znalostní) systémy:

- **DENDRAL** (Buchanan)
odvození molekulární struktury podle hmotové spektrometrie
zavedení odvozovacích pravidel na základě vzoru
 - **MYCIN** (Feigenbaum)
diagnostika infekcí krve, zavedení faktoru nejistoty
 - **PROLOG** (Colmerauer, 1972)
 - **rámce** (Minsky, 1975) – motivace pro dnešní OOP



UI dnes

■ UI jde do výroby (1980)

- systém **R1**, konfigurace počítačů DEC (\$40 mil./rok)
 - **pátá generace** počítačů (Japonsko, 1981)
 - cílem bylo vyvinout do deseti let inteligentní počítače běžící Prolog
 - **boom UI** průmyslu (miliardy US dolarů v 1988)
 - a pak náhle „**UI zima**“
 - společnosti nebyly schopny naplnit své sliby (jako dot.com bubble)

■ Návrat neuronů (1986)

- ## □ nové modely neuronových sítí

■ UI se stává vědou (1987)

- UI začíná používat **vědecké metody**, hypotézy musí být experimentálně ověřeny, statisticky validní, experimenty opakovatelné
 - nové přístupy: skryté Markovské modely, Bayesovské sítě, dolování dat
 - formalizace a specializace ale také vede k **fragmentaci**

■ Přichází agenti (1995)

- po úspěších v dílčích oblastech se znova vrací myšlenka „kompletního racionálního agenta“
 - SOAR (State, Operator and Result) – kompletní architektura racionálního agenta

Válka v zálivu 1991:

- Tradiční metoda:
 - Stovky lidských plánovačů
 - Měsíce na vytvoření plánů
- Metoda IP&S:
 - O-PLAN2 pomáhá plánovačům
- **Úspory:**
 - Rychlejší vytvoření zázemí
 - Méně leteckých misí
 - Finanční návratnost >> **veškerý výzkum AI sponzorovaný US vládou:**
 - od roku 1956
 - nejen výzkum IP&S, ale **veškerý výzkum AI!**



Contribution of On Time Systems

Umělá inteligence I, Roman Barták

Deep Space 1

Start: 24. října 1998

Cíl: Borrelliova kometa

Testování 12 nových technologií

- **autonomous remote agent**
 - plánuje, provádí a monitoruje akce kosmické lodi na základě obecných příkazů operátora
 - tři zkušební scénáře
 - 12 hodin nízké autonomie (provádění a monitorování)
 - 6 dní vysoké autonomie (snímání kamerou, simulace poruch)
 - 2 dny vysoké autonomie (udržení směru)
 - **pozor na backtracking!**
 - **pozor na deadlock v plánu!**



Contribution of NASA Ames

Umělá inteligence I, Roman Barták

- Cílem je vyvinout tým plně autonomních robotů, který do roku 2050 porazí tým mistrů světa ve fotbale.



- Simulace**
simulované utkání v počítači
- Malí roboti**
roboti do průměru 18 cm
- Střední roboti**
roboti do průměru 50 cm
všechny sensory
- Čtyřnozí roboti (standard platform)**
Sony Aibo
- Humanoidní roboti**
penalty a hra dva na dva



Umělá inteligence I, Roman Barták

Grand Challenge

- První závod **plně automatizovaných vozidel** na dlouhou vzdálenost sponzorovaný DARPA.
- Cílem je mít třetinu pozemních vojenských sil plně autonomní do roku 2015.



- 2004 Grand Challenge**
 - neúspěch - nikdo nedokončil (max. 11,78 km, CMU)
- 2005 Grand Challenge**
 - cíl splněn! vítěz Stanley (212.4 km za cca 7 hod., Stanford)
- ... a ted' vzhůru do města (2007)

Umělá inteligence I, Roman Barták



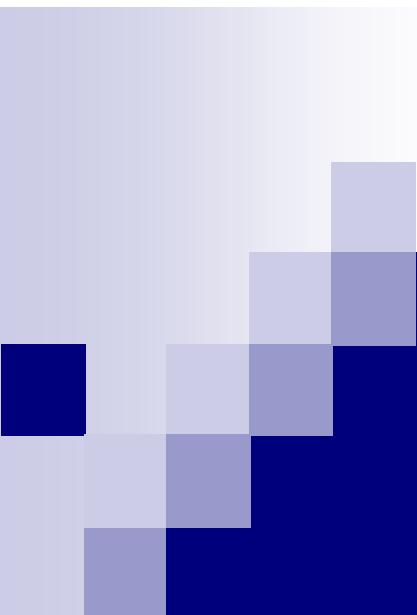
Unmanned Aerial Vehicle

- Cílem je plně autonomní let včetně vlastního racionální rozhodování o způsobu provedení mise.

- Patrick Doherty
Linköping universitet
- vyžaduje různé specializace:
 - teorie řízení
 - letecké inženýrství
 - zpracování signálů
 - umělá inteligence
 - software engineering
 - ...



Umělá inteligence I, Roman Barták



Umělá inteligence I

Roman Barták, KTI ML

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



2

Inteligentní agenti

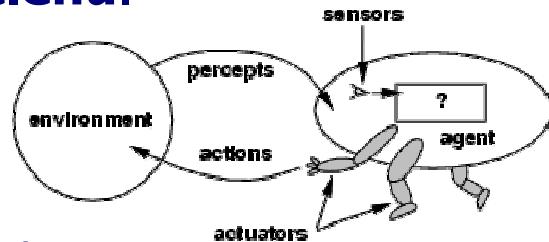
- Připomeňme, že z různých pohledů na UI bude tato přednáška pokrývat **racionální jednání**, tj.
 - zajímá nás výsledek řešení (ne postup řešení),
 - který by měl být co nejlepší (ne nutně lidský)
- Budeme konstruovat **racionální agenty**
 - Co je to **agent**?
 - A co je **racionální agent**?
 - V jakém **prostředí** se agent pohybuje?
 - Jaké **vlastnosti prostředí** nás zajímají?
 - Jaké **vnitřní struktury** agentů máme?



- **Agent** je cokoliv, co vnímá okolní **prostředí** prostřednictvím **senzorů** a ovlivňuje ho prostřednictvím **akčních členů**.

- **Příklady:**

- člověk
 - oči, uši, nos, ... → ruce, nohy, ústa,...
 - robot
 - kamera, IR detektor, ... → paže, kola, ...
 - software
 - klávesnice, pakety ze sítě ... → obrazovka, pakety do sítě,...



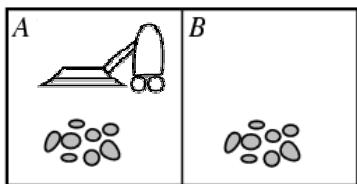
Umělá inteligence I, Roman Barták

Agent přesněji

- Agent přijímá **vjem** a jeho chování je plně určeno posloupností všech vjemů, které kdy přijal.
- Formálně tedy můžeme každého agenta popsát **agentovou funkcí** (tabulkou):
 - $V^* \rightarrow A$, kde V : množina vjemů, A : množina akcí
 - Na základě chování agenta, můžeme sestavit jeho agentovou funkci (vnější charakteristika).
 - tedy pokud můžeme agenta „restartovat“ a máme nekonečný prostor pro zápis tabulky a také hodně času
 - Tabulka je **abstraktní matematický popis** agenta.
- Interně ale bude chování agenta popsáno spíše **agentovým programem**.
 - Program je **konkrétní implementace** agentovy funkce.

Umělá inteligence I, Roman Barták

Příklad agenta



Vysavač

- vjemy: místo (A,B)
obsah (čisto,špína)
- akce: vysaj, doleva,
doprava, nic

Agentova funkce:

seznam vjemů	akce
(A,čisto)	doprava
(A,špína)	vysaj
(B,čisto)	doleva
(B,špína)	vysaj
(A,čisto), (A,čisto)	doprava
...	

Agentův program:

```
if obsah=špína then  
    vysaj  
else if místo=A then  
    doprava  
else if místo=B then  
    doleva
```



Umělá inteligence I, Roman Barták

Míra výkonu

- **Jak správně vyplnit agentovu tabulku? aneb
Co odlišujete dobrého agenta od špatného?**
- Potřebujeme měřit **míru výkonu** agenta, tj. jak úspěšné je jeho chování a to pokud možno **objektivně**.
- **Kdo určuje míru výkonu?**
 - tvůrce agenta
- **Jak určit míru výkonu?**
 - Podle toho, jak chceme aby vypadalo prostředí agenta, spíše než podle toho, jak si myslíme, že se má agent chovat.
- **Příklad** (vysavač)
 - míra výkonu: sesbírej co nejvíce špíny
 - možný výsledek: vysaj, vysyp, vysaj, vysyp, ...
 - míra výkonu (lépe): co největší čistá plocha





Racionální agent

- **Racionální agent** je takový agent, který pro každou možnou posloupnost vjemů zvolí akci maximalizující očekávanou míru výkonu a to na základě údajů daných tou posloupností vjemů a vnitřních znalostí agenta.
- Pozor, neplést se vševedoucností!
 - agent maximalizuje **očekávanou** míru výstupu
 - vševedoucí maximalizuje **skutečnou** míru výstupu
- Racionální agent by měl být **autonomní** – nespoléhá jen na znalosti dané tvůrcem, ale měl by se učit, aby kompenzoval předchozí částečné nebo špatné znalosti.

Umělá inteligence I, Roman Barták



Prostředí agenta

- Kromě **senzorů, akčních** členů a míry **výkonu** potřebujeme ještě **prostředí**, ve kterém se agent pohybuje.

Příklad: automatický řidič taxi



Agent	Výkon	Prostředí	Akční členy	Senzory
řidič taxi	bezpečnost rychlosť legálnost pohodlí zisk	silnice doprava chodci zákazník	brzda volant plyn směrovka houkačka	kamera sonar rychloměr GPS

Umělá inteligence I, Roman Barták



Vlastnosti prostředí

- **Plná (částečná) pozorovatelnost**
 - senzory poskytují úplný obraz prostředí
- **Deterministické (stochastické)**
 - další stav prostředí je plně určen současným stavem a akcí agenta
 - strategické = stav prostředí mění pouze další agenti
- **Epizodické (sekvenční)**
 - chování agenta lze rozdělit do epizod obsahujících po jedné akci, které jsou na sobě ale nezávislé (akce závisí pouze na dané epizodě)
- **Statické (dynamické)**
 - statické prostředí se nemění zatímco agent přemýšlí
 - semi-dynamické = prostředí se s časem nemění, ale míra výkonu ano
- **Diskrétní (spojité)**
 - rozlišení dle stavu prostředí, měření času, schopností snímačů a akčních prvků
- **Jeden agent (multi-agentní)**
 - Kdo musí být agent?
Objekt maximalizující míru kvality, které závisí na daném agentovi.
 - kompetitivní vs. kooperativní multi-agentní prostředí

Umělá inteligence I, Roman Barták



Příklady prostředí

Prostředí	Pozorovatel nost	Determinismus	Epizodické	Statické	Diskrétní	Agenti
Křížovka	úplná	deterministické	sekvenční	statické	diskrétní	jeden
Šachy s hodinami	úplná	strategické	sekvenční	semi	diskrétní	multi
Taxi	částečná	stochastické	sekvenční	dynamické	spojité	multi
Analýza obrazů	úplná	deterministické	epizodické	semi	spojité	jeden

■ Nejjednodušší prostředí

- plně pozorovatelné, deterministické, episodické, statické, diskrétní a jeden agent

■ Nejsložitější prostředí (běžné reálné)

- částečně pozorovatelné, stochastické, sekvenční, dynamické, spojité a multi-agentní

Umělá inteligence I, Roman Barták



agent = architektura + program

- **architektura** = výpočtové zařízení s fyzickými senzory a akčními členy
- **program** = implementace agentovy funkce
 - mapuje vjemy na akce
 - přesněji řečeno, vstupem je jeden vjem (jediné, co je přímo dostupné z prostředí prostřednictvím čidel)
 - pokud akce záleží na posloupnosti vjemů, musí si ji agent sám pamatovat
 - samozřejmě program musí být vhodný pro danou architekturu!

Umělá inteligence I, Roman Barták



Table-driven

Tabulkový agent

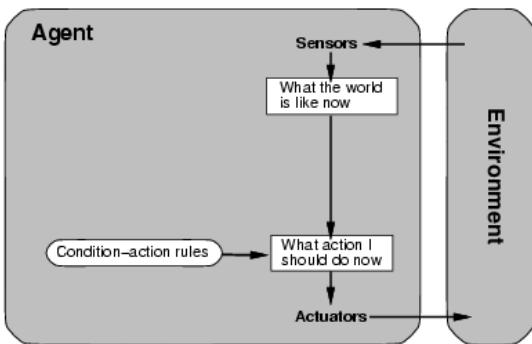
- Triviální agent, který udržuje seznam vjemů a používá ho jako index do tabulky s akcemi.

```
function TABLE-DRIVEN-AGENT(percept) returns an action
  static: percepts, a sequence initially empty
          table, a table of actions, indexed by percept sequence
  append percept to the end of percepts
  action  $\leftarrow$  LOOKUP(percepts, table)
  return action
```

Problémy:

- tabulka neúměrně velká (i pro agenta fungujícího omezený čas)
- tvůrce agenta nemá čas ji vyplnit
- agent se nemůže sám naučit celou správnou tabulku
- tvůrce agenta také neví jak tabulku vyplnit

Takto to asi nepůjde!



- Akce vybrána pouze na základě současného vjemu.
- Implementováno formou **pravidel** podmínka-akce (**if** obsah=špína **then** vysaj)
- Velká **redukce** možných situací vjem-akce (max. na počet vjemů)

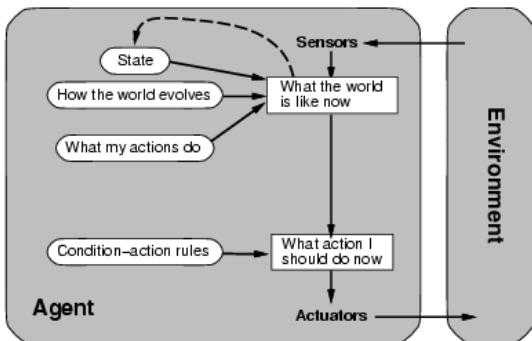
```
function SIMPLE-REFLEX-AGENT(percept) returns an action
```

static: *rules*, a set of condition-action rules

$$\begin{aligned} \text{state} &\leftarrow \text{INTERPRET-INPUT}(\text{percept}) \\ \text{rule} &\leftarrow \text{RULE-MATCH}(\text{state}, \text{rule}) \\ \text{action} &\leftarrow \text{RULE-ACTION}[\text{rule}] \\ \text{return } &\text{action} \end{aligned}$$

- Plně funguje pouze pro plně pozorovatelné prostředí, jinak může cyklot.
- Řešením může být randomizace výběru akcí.

Umělá inteligence I, Roman Barták



- Problémy s částečně pozorovatelným světem lze řešit udržováním vnitřního stavu agenta.
- Udržujeme dva typy informací
 - jak se svět vyvíjí (nezávisle na agentovi)
 - jak agentovy akce ovlivňují svět
- **Model světa**

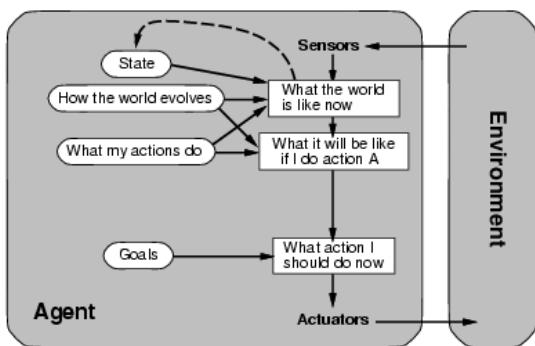
```
function REFLEX-AGENT-WITH-STATE(percept) returns an action
```

static: *rules*, a set of condition-action rules

state, a description of the current world state
action, the most recent action.

$$\begin{aligned} \text{state} &\leftarrow \text{UPDATE-STATE}(\text{state}, \text{action}, \text{percept}) \\ \text{rule} &\leftarrow \text{RULE-MATCH}(\text{state}, \text{rule}) \\ \text{action} &\leftarrow \text{RULE-ACTION}[\text{rule}] \\ \text{return } &\text{action} \end{aligned}$$

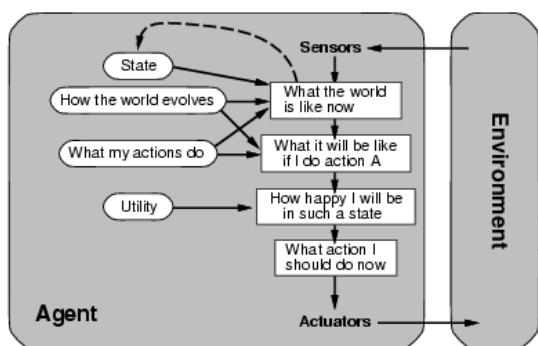
Umělá inteligence I, Roman Barták



- Zatím jsme se starali pouze o výběr další akce, ale ten také záleží na tom, co chce agent dosáhnout.
- Agent potřebuje **cíl** říkající, jaké situace jsou žádoucí.

- Základní změna je zahrnutí **uvažování o budoucnosti**.
- Hledáním (delší) posloupnosti akcí pro dosažení cíle se zabývá **plánování**.
- Řízení cílem je (zpravidla) **méně efektivní** než řízení reflexem, je ale **více flexibilní**.

Umělá inteligence I, Roman Barták



- Cíle sami o sobě nestačí pro „kvalitní“ chování (cíl splněn/nesplněn).
- Často je potřeba vyjádřit míru splnění či dosažení cíle nebo vybrat mezi různými i konfliktními cíli.

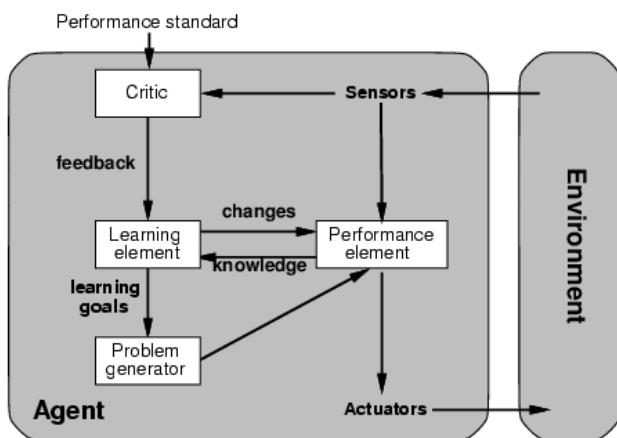
- Je možné mapovat stavy (nebo jejich posloupnosti) na reálná čísla určující **míru „žádoucnosti“** dosažení daného stavu.
- Agent se potom při výběru akce řídí i tím, jak je případně dosažitelný stav žádoucí.

Umělá inteligence I, Roman Barták



Agent učící se

- Zatím jsme se zabývali výběrem akcí, ale **odkud se berou programy pro výběr akcí?**
- Jedna z možností je vytvořit **stroje schopné se učit** a potom je učit místo podávání instrukci.
- Agent potom může pracovat i v původně neznámém prostředí.
- Všechny dosud zmíněné struktury agentů lze rozšířit na učící se agenty přidáním komponent pro učení:



□ výkonný prvek

- původní agent, který vybírá akce

□ učící se prvek

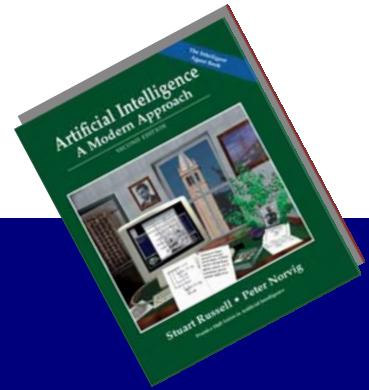
- vylepšuje program původního agenta

□ kritika

- dává zpětnou vazbu, jak dobře se agent chová (senzory na to nestačí)

□ generátor problémů

- navrhují akce, které by mohly vést k nové zkušenosti (trochu experimentování)



Umělá inteligence I

Roman Barták, KTI ML

3

roman.bartak@mff.cuni.cz

<http://ktiml.mff.cuni.cz/~bartak>



Na úvod

- **Agent s reflexy** „pouze“ převádí současný vjem na jednu akci.
- **Agent s cílem** umí plánovat několik akcí dopředu pro dosažení cíle.
 - Podívejme se na jeden typ agentů s cílem – **agenty řešící úlohy**.
 - Co je to **úloha** a co je její **řešení**?
 - Jak **nalézt řešení** úkolu?
 - **Prohledávací** algoritmy
BFS, DFS, DLS, IDS, BiS

Poznámka:

Budeme se zabývat „neinformovanými“ algoritmy, tj. algoritmy, které nevyužívají speciální znalost problému.



- Inteligentní agenti maximalizují míru svého výkonu, což si mohou zjednodušit formulací cílů a snahou jich dosáhnout.

Situace a možný postup řešení

- agent je v Rumunsku ve městě Arad
- míra výkonu se skládá z mnoha složek (chce být opálený, užít si noční život, prohlédnout okolí, ...), což komplikuje rozhodovací proces
- nechtě ale agent má na zítra nevratnou letenku z Bukurešti
- formulací cíle** – dostat se do zítřka do Bukurešti – se rozhodovací problém zjednoduší
- cíl určuje žádoucí stav světa, potřebuje ale další stavy světa, přes které se k cíli dostaneme, a akce, které nás mezi stavy budou posouvat – **formulace úlohy**
 - stavy mohou být města, přes která pojedeme (aktuální poloha na silnici je zbytečně jemná)
 - akce mohou být přejezdy mezi městy (akce typu zvedni a polož nohu jsou příliš jemné)
- Jak ale najdeme cestu do Bukurešti, když na rozcestí jsou jen tři nejbližší města a žádné není Bukurešť?
 - Uvažujme, že máme mapu Rumunska, můžeme tedy prozkoumat různé cesty (**prohledávání**), vybrat tu nejlepší a potom se po ní vydat (**exekuce**).



Umělá inteligence I, Roman Barták

Řešení úloh

Řešení úloh se skládá ze čtyř hlavních kroků:

- formulace cíle**
 - Jaké jsou žádoucí stavy světa?
- formulace úlohy**
 - Jaké akce a stavy je vhodné uvažovat pro dosažení cíle?
- vyřešení úlohy**
 - Jak najít nejlepší posloupnost stavů vedoucí k cíli?
- realizace řešení**
 - Známe-li požadované akce, jak je budeme realizovat?



```

function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  static: seq, an action sequence, initially empty
           state, some description of the current world state
           goal, a goal, initially null
           problem, a problem formulation

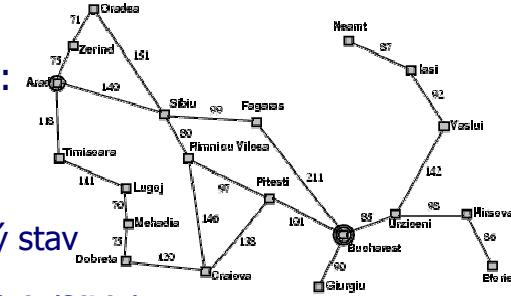
  state  $\leftarrow$  UPDATE-STATE(state, percept)
  if seq is empty then do
    goal  $\leftarrow$  FORMULATE-GOAL(state)
    problem  $\leftarrow$  FORMULATE-PROBLEM(state, goal)
    seq  $\leftarrow$  SEARCH(problem)
    action  $\leftarrow$  FIRST(seq)
    seq  $\leftarrow$  REST(seq)
  return action

```

Umělá inteligence I, Roman Barták

Formulace úloh

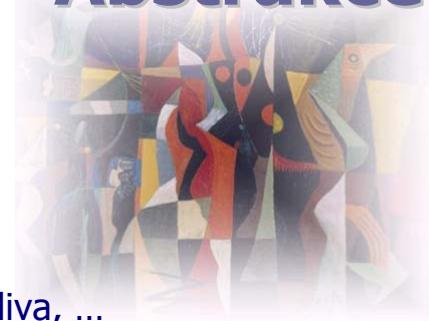
- Dobře formulovaná úloha se skládá z:
 - počátečního stavu
 - *in(Arad)*
 - popisu možných akcí
 - typicky funkcií následníka, který pro každý stav vrátí akci a stav, kam nás akce převede
 - $SUCCESSOR-FN(in(Arad)) = \langle go(Sibiu), in(Sibiu) \rangle$
 - implicitně definuje **stavový prostor** (množina stavů dosažitelných z počátečního stavu)
 - **cesta** je potom posloupnost stavů propojených akcemi
 - **testu cíle**
 - funkce určující zda daný stav je cílový nebo ne, může být třeba výčet cílových stavů – $\{ in(Bucharest) \}$
 - **ceny cest**
 - numerická cena každé cesty, určuje míru výkonu agenta
 - Budeme předpokládat, že cena cesty se skládá z nezáporných cen akcí po cestě.
- **Řešením úlohy** je cesta z počátečního stavu do stavu cílového.
- **Optimální řešení** je řešení z nejmenší cenou cesty mezi všemi řešeními.



Umělá inteligence I, Roman Barták

Abstrakce

- V naší formulaci jsme použili
 - **abstrakci stavu světa**
 - ignorujeme počasí, stav silnice, ...
 - **abstrakci akcí**
 - neuvažujeme zapnutí rádia, čerpání paliva, ...
- Jaká je **správná abstrakce**?
 - abstraktní řešení lze rozvést na konkrétní (**validita**)
 - z libovolného místa v Aradu najdeme cestu do libovolného místa v Sibiu
 - provádění akcí z řešení je snazší než původní úloha (**užitečnost**)
 - cestu z Aradu do Sibiu můžeme svěřit „řidiči“ bez dalšího plánování



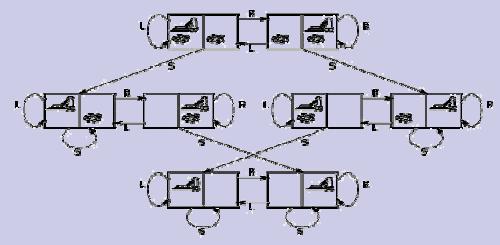
Bez možnosti pracovat s abstrakcemi by nás reálný svět zahiltil.

Umělá inteligence I, Roman Barták

- Hračky (**toy problems**) slouží pro ilustraci a porovnání řešících technik.

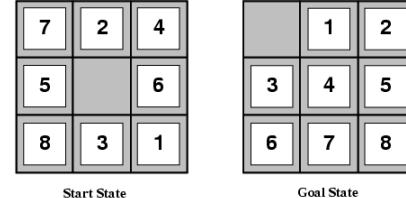
Svět vysavače

stavy (robot \times smetí)
počátek, cíl
následník (L, R, S)



Loydova osmička

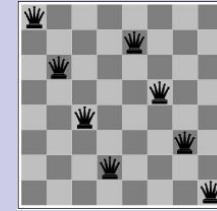
stavy (pozice kostiček)
počátek, cíl
následník (L, R, U, D)



8-královen

umístit na šachovnici 8 královen bez ohrožení

- inkrementální reprezentace (přidáváme královny)
- reprezentace úplných stavů (posouváme královny)



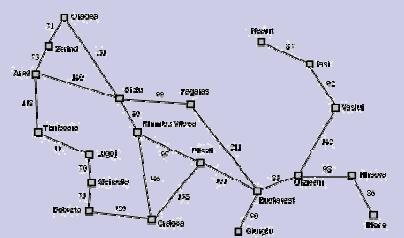
Umělá inteligence I, Roman Barták

Reálné úlohy

- To je to, co lidi zajímá.

Hledání cest

stavy (reprezentace míst)
počátek (ted' a tady), cíl (tam a včas)
následník
cenová funkce (čas, cena,...)

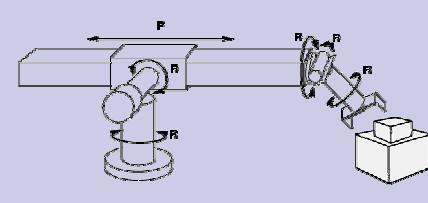


Okružní cesty

cíl (navštívit každé město alespoň jednou)
stavy (aktuální a navštívený místo)
TSP (Travelling Salesman Problem), NP-těžký

Sestavování produktů

stav (pozice robotické ruky \times součástky)
cíl (sestavený produkt)
následník (pohyby „kloubů“)
cena (celkový čas sestavení)
sestavení proteinů z posloupnosti aminokyselin



Umělá inteligence I, Roman Barták

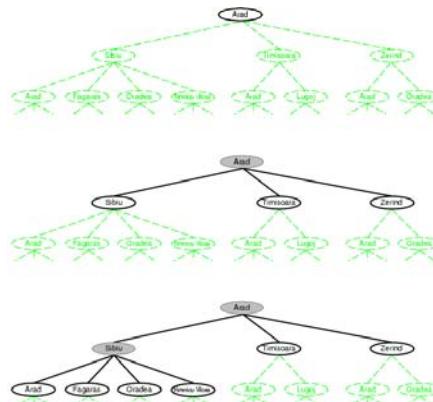


Hledáme řešení

- Když už máme formulované úlohy, jak je výřešíme?

■ Prohledávání stavového prostoru

- začneme v **počátečním stavu** (kořenový uzel)
- **otestujeme**, zda počáteční stav není zároveň **cílový**
- pokud není cílový, provedeme **expanzi stavu**, čímž generujeme novou množinu stavů
- **vybereme další stav** pro pokračování pomocí **prohledávací strategie**



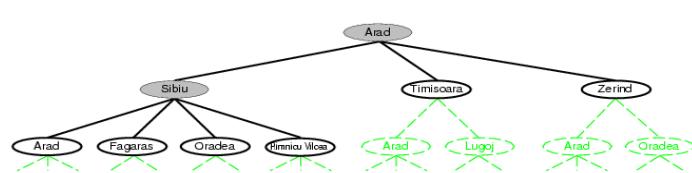
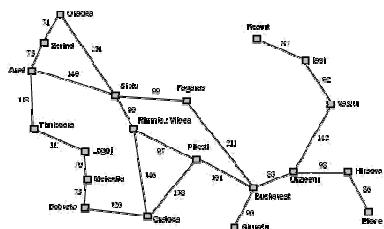
```
function TREE-SEARCH(problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
```

Umělá inteligence I, Roman Barták



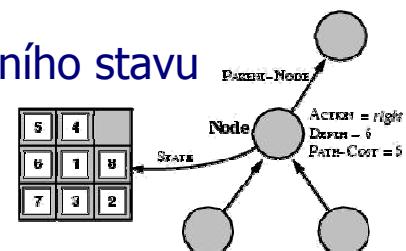
Uzel vs. stav

- Stavový prostor se liší prohledávacího stromu stejně jako stav není uzel stromu!



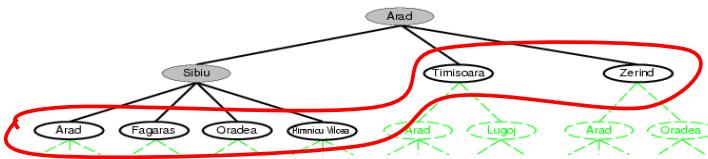
- Uzel prohledávacího stromu obsahuje

- aktuální **stav**
- odkaz na **rodiče**
- **akci**, která vedla od rodiče do aktuálního stavu
- **cenu** cesty (z kořene), $g(n)$
- **hloubku** (počet kroků od kořene)



Umělá inteligence I, Roman Barták

- Při prohledávání potřebujeme také reprezentovat soubor uzlů pro expandování – **okraj** (fringe).



- Uzly z okraje jsou nazývány **listy**.
- Abychom usnadnili výběr uzlu z okraje, budeme okraj reprezentovat **frontou** s následujícími operacemi:
 - MAKE-QUEUE(element,...)
 - EMPTY?(queue)
 - FIRST(queue)
 - REMOVE-FIRST(queue)
 - INSERT(element, queue)
 - INSERT-ALL(elements, queue)



Umělá inteligence I, Roman Barták

Stromové prohledávání

```

function TREE-SEARCH( problem, fringe) returns a solution, or failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
    fringe ← INSERTALL(EXPAND(node, problem), fringe)

function EXPAND( node, problem) returns a set of nodes
  successors ← the empty set
  for each action, result in SUCCESSOR-FN[problem](STATE[node]) do
    s ← a new NODE
    PARENT-NODE[s] ← node; ACTION[s] ← action; STATE[s] ← result
    PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
    DEPTH[s] ← DEPTH[node] + 1
    add s to successors
  return successors
  
```

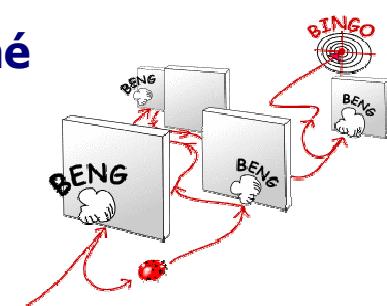
- Prohledávací algoritmus vrátí **řešení** nebo **neúspěch**.
- **Výkon algoritmu** budeme posuzovat ze čtyř hledisek:
 - **úplnost**
 - Pokud existuje řešení, najde ho?
 - **optimalita**
 - Najde algoritmus optimální řešení?
 - **časová složitost**
 - Jak dlouho trvá nalezení řešení?
 - **prostorová složitost**
 - Kolik paměti pro prohledávání potřebujeme?
 - Čas a prostor typicky měříme vzhledem k nějaké míře složitosti problému.
 - **větvící faktor b** (maximální počet následníků) – vhodné, když je stavový prostor reprezentován implicitně (počátek a následníci)
 - **hloubka d** (hloubka nejméně zanořeného cílového uzlu)
 - **cesta m** (délka maximální cesty ve stavovém prostoru)
- **cena prohledávání** (kolik času a prostoru potřebujeme)
- **celková cena** (kombinuje cenu prohledávání a cenu nalezeného řešení)



Umělá inteligence I, Roman Barták

Informovanost

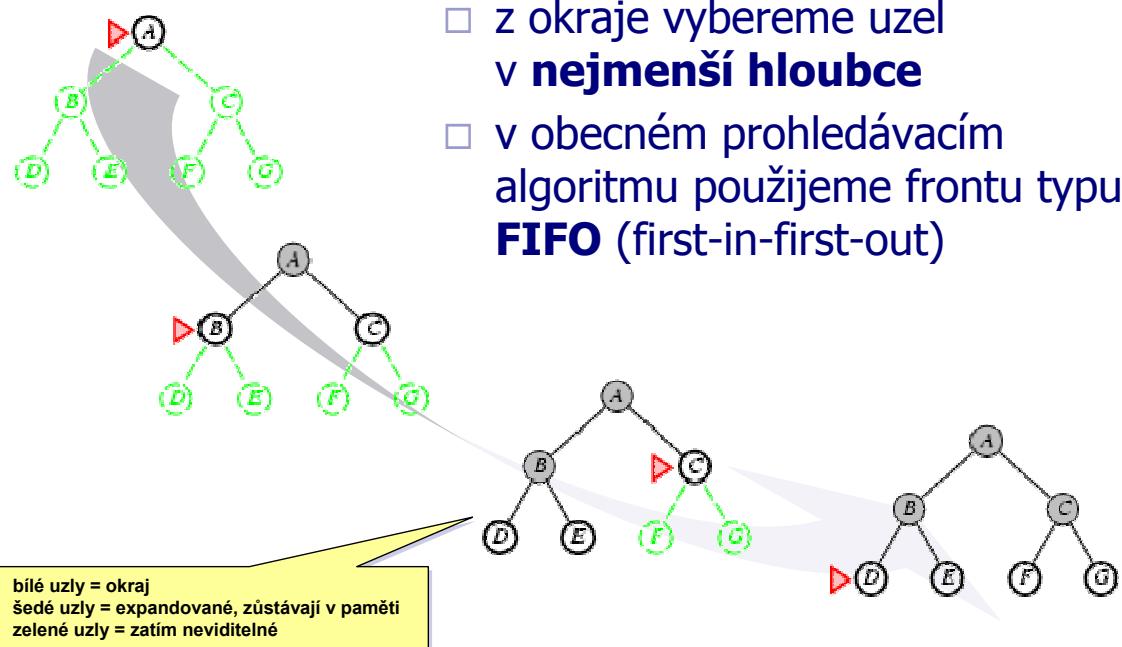
- Dnes se podíváme na tzv. **neinformované (slepé)** prohledávání
 - nemá **žádnou dodatečnou informaci** o stavech, kromě popisu úlohy
 - tj. umí rozpoznat cílový stav od necílového a generovat následníky
- Příště doplníme **informované (heuristické)** prohledávání
 - u stavů dokáže rozlišit, jak jsou **slibné** na cestě k řešení
 - např. prostřednictvím odhadu ceny zbývající do cíle



Umělá inteligence I, Roman Barták



- Vždy nejprve **expandujeme celou vrstvu** prohledávacího stromu než postoupíme do další



Umělá inteligence I, Roman Barták



- Je to metoda **úplná** (za předpokladu konečného větvení).
- Nejdříve nalezený cíl **nemusí být optimální!**
 - optimum lze garantovat, pokud je cena neklesající funkcí hloubky
- **Časová složitost** (počet navštívených uzlů pro cíl v hloubce d , který je na konci vrstvy)
 - $1+b+b^2+b^3+\dots+b^d + b(b^d-1) = O(b^{d+1})$
- **Paměťová složitost**
 - v paměti drží všechny navštívené uzly, tj. stejná jako časová složitost
 - $O(b^{d+1})$
- **Paměť je větší problém než čas ale ani čas není zanedbatelný.**

b = 10
10.000 uzlů/sec.
1000 byte/uzel

hloubka	uzly	čas	paměť
2	1100	0.11 sec.	1 megabyte
4	111100	11 sekund	106 megabytes
6	10^7	19 minut	10 gigabytes
8	10^9	31 hodin	1 terabyte
10	10^{11}	129 dní	101 terabytes
12	10^{13}	35 let	10 petabytes
14	10^{15}	3523 let	1 exabyte

Umělá inteligence I, Roman Barták



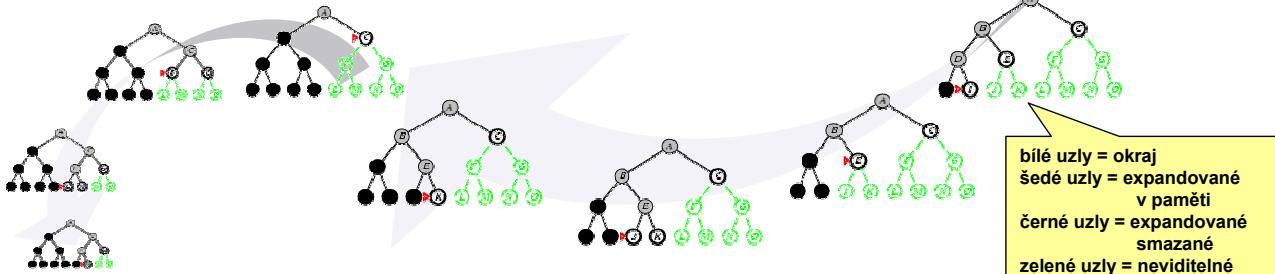
- Modifikace BFS pro **hledání optimálního řešení**.
- Jako první expandujeme uzel s **nejmenší cenou** cesty od kořene.
- Pozor na kroky s nulovou cenou, mohou vést k cyklům!
- **Úplnost** (a nalezení optima) lze garantovat, pokud je cena každé akce větší či rovna konstantě ϵ .
- **Časová (a paměťová) složitost**
 - záleží na ceně cesty spíše než na hloubce
 - $O(b^{1+\lfloor C^*/\epsilon \rfloor})$, kde C^* je cena optimálního řešení
 - může být **mnohem horší než u BFS**, protože se často prohledávají dlouhé cesty s „krátkými“ kroky než se zkusí cesty s delšími a možná vhodnějšími kroky



Umělá inteligence I, Roman Barták



- Vždy **expandujeme nejhļubší uzel**
- až se dostaneme do vrstvy, kde nelze expandovat, a pak se **vrátíme** k uzlu s menší hloubkou
- v obecném prohledávacím algoritmu použijeme frontu typu **LIFO** (last-in-first-out), tj. **zásobník**
- často se implementuje rekurzivně



Umělá inteligence I, Roman Barták



- Pokud se vydá špatnou cestou, nemusí najít řešení (**není úplný**) a samozřejmě **nemusí najít optimum**.

■ Časová složitost

- O(b^m)**, kde m je maximální hloubka
- může se stát, že d << m, kde d je hloubka řešení

■ Paměťová složitost

- stačí si pomatovat jednu cestu z kořene společně se sousedními uzly
- expandovaný uzel s prozkoumanými potomky lze odstranit
- O(bm)**, kde m je maximální navštívená hloubka
- Lze ještě zlepšit technikou známou jako **backtracking**!
 - generuje jednoho následníka (místo všech) → **O(m)** stavů
 - mění stav na místě (místo kopírování), pokud tedy umí obnovit původní stav na základě akce → **O(1)** stavů a **O(m)** akcí

Umělá inteligence I, Roman Barták



■ Problém s neomezenými stromy můžeme řešit přidáním **limitu l na hloubku prohledávání**.

- uzly v hloubce l se považují jako uzly bez následníků
- algoritmus vrací řešení, neúspěch nebo vyčerpání limitu (cut-off)
- časová složitost **O(b^l)**, prostorová složitost **O(bl)**
- pokud l < d, pak řešení nenajdeme (d je hloubka řešení)
- pokud d << l, pak často prohledáváme zbytečně mnoho

Jak určit vhodný limit?

```

function DEPTH-LIMITED-SEARCH( problem, limit) returns soln/fail/cutoff
  RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)

function RECURSIVE-DLS(node, problem, lmit) returns soln/fail/cutoff
  cutoff-occurred? ← false
  if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
  else if DEPTH[node] = limit then return cutoff
  else for each successor in EXPAND(node, problem) do
    result ← RECURSIVE-DLS(successor, problem, limit)
    if result = cutoff then cutoff-occurred? ← true
    else if result ≠ failure then return result
  if cutoff-occurred? then return cutoff else return failure
  
```

- použitím informací o problému
- např. pro hledání cesty ve světě 20-ti měst, můžeme použít limit 19
- studiem mapy můžeme získat ještě přesnější limit – mezi libovolnými dvěma městy se lze dostat do 9-ti kroků (tzv. **průměr grafu**)

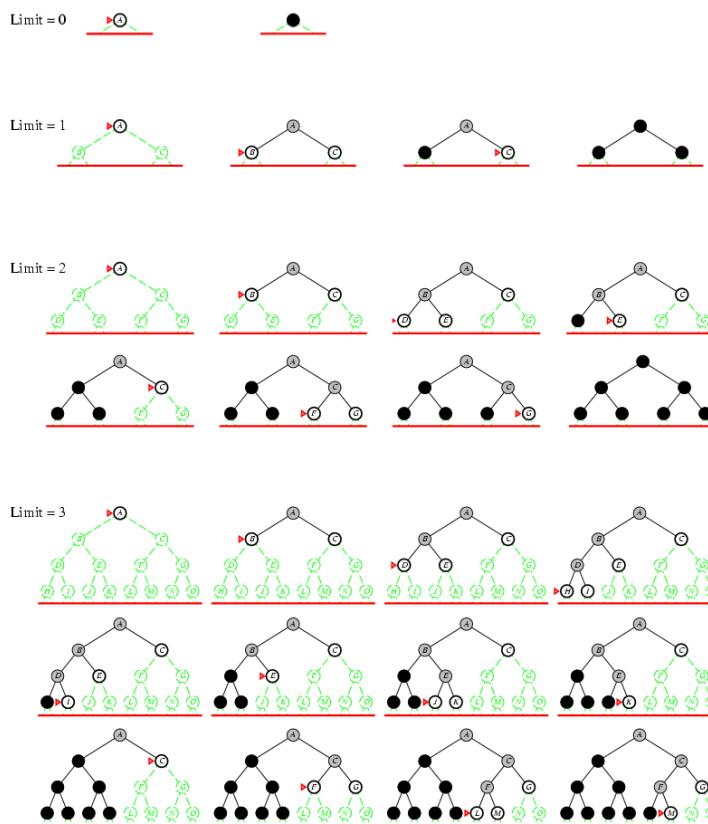
Umělá inteligence I, Roman Barták

Jak zúplnit prohledávání s omezenou hloubkou?

- budeme postupně zvětšovat limit l
- spojíme tak výhody BFS a DFS
 - **úplnost** (pro konečné větvení)
 - garance **optimality** (pro cenu závislou na hloubce)
 - malé **paměťové** nároky **O(bd)**
 - a co **časová** složitost?
 - $d b^1 + (d-1)b^2 + \dots + 3b^{d-2} + 2b^{d-1} + 1b^d = O(b^d)$
 - fakticky lepší než BFS, které často přidá ještě jednu vrstvu
 - Př. ($b = 10, d = 5$): BFS = 1.111.100, DFS = 111.110, IDS = 123.450
- **IDS je preferovaná metoda slepého prohledávání ve velkém prostoru a s neznámou hloubkou řešení.**

```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution, or failure
  inputs: problem, a problem
  for depth ← 0 to ∞ do
    result ← DEPTH-LIMITED-SEARCH( problem, depth)
    if result ≠ cutoff then return result
```

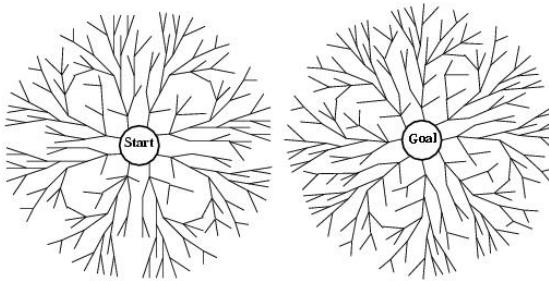
Umělá inteligence I, Roman Barták



Umělá inteligence I, Roman Barták



- Můžeme také **kombinovat prohledávání od počátku s prohledáváním od cíle** (dokud se někde nepotkají).



- **Proč?**

- $b^{d/2} + b^{d/2} \ll b^d$
 - Př. (b= 10, d = 6): BFS = 11.111.100, BiS = 22.200

- **Jak?**

- Před expanzí uzlu vždy zkонтrolujeme, zda není v okraji druhého prohledávacího stromu.
 - Potřebujeme mít v **paměti** alespoň jeden ze stromů **O(b^{d/2})**, test přítomnosti uzlu lze provést v konstantním čase (hašování).
 - Použijeme-li z obou stran prohledávání do šířky, dostaneme **úplný algoritmus** garantující nalezení **optima**.

Umělá inteligence I, Roman Barták



Pozpátku?

- Při prohledávání od počátku jdeme přes stav, **co ale prohledáváme při cestě od cíle?**
 - Je-li cíl jediný stav, můžeme jít také přes stav (cesta Arad → Bucharest).
 - Je-li cílem více explicitně daných stavů (svět vysavače), můžeme zavést **umělý stav**, jehož jsou dané cílové stavы předchůdci
 - nebo můžeme skupinu stavů vidět jako **jediný meta-stav**.
 - Nejtěžším případem jsou **implicitně dané cílové stavы** (8-královen), kdy je potřeba najít kompaktní popis umožňující test přítomnosti stavu z dopředné fáze.
- Jiný problém je **jak definovat předchůdce stavu**.
 - Předchůdcem n je každý stav, jehož je n následníkem.
 - Nejtěžším případem jsou implicitně definované přechody pomocí nějaké funkce.

Umělá inteligence I, Roman Barták

Kritérium	Breadth-First	Uniform-cost	Depth-First	Depth-limited	Iterative deepening	Bidirectional search
Úplnost?	ANO*	ANO*	NE	ANO, if $l \geq d$	ANO	ANO*
Čas	b^{d+1}	$b^{C^*/\epsilon}$	b^m	b^l	b^d	$b^{d/2}$
Prostor	b^{d+1}	$b^{C^*/\epsilon}$	bm	bl	bd	$b^{d/2}$
Optimalita?	ANO*	ANO*	NE	NE	ANO	ANO

b – větvící faktor

d – nejmenší hloubka (optimálního) řešení

C^* – cena optimálního řešení

ϵ – minimální nárůst ceny (minimální cena akce)

m – maximální navštívená hloubka

l – limit hloubky

* úplnost při omezeném větvení (BFS)

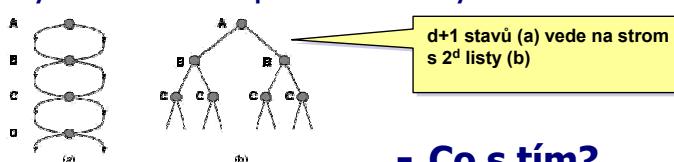
optimalita při minimálním nárůstu ceny resp. jednotkové ceně (BFS)

Umělá inteligence I, Roman Barták



Závěrečné poznámky opakující se stavý

- Zatím jsme ignorovali jednu z typických obtíží prohledávání – expanzi již navštívených stavů.
 - ne vždy je to problém (vhodná formulace problému 8-královen)
 - někdy ale výrazně zvětší prohledávaný strom



■ Co s tím?

```
function GRAPH-SEARCH(problem, fringe) returns a solution, or failure
  closed ← an empty set
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
    if STATE[node] is not in closed then
      add STATE[node] to closed
      fringe ← INSERT-ALL(EXPAND(node, problem), fringe)
```

- můžeme se pomatovat již expandované stavý – tzv. **uzavřené stavý**
- pokud se do uzavřeného stavu dostaneme znova, dále ho v prohledávání neexpandujeme

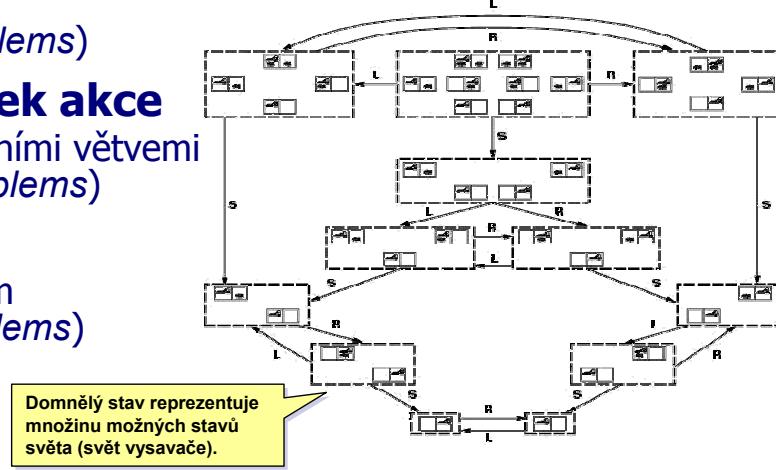
Umělá inteligence I, Roman Barták



Závěrečné poznámky

částečná informace

- Zatím jsme uvažovali statické, plně pozorovatelné, diskrétní a deterministické prostředí.
- Co dělat v případě, že agentovi **chybí informace?**
 - **zádné senzory** (nevíme v jakém jsme stavu)
 - pracujeme s **domnělým (belief) stavem** – množina reálných stavů – a hledáme domnělý stav obsahující pouze cílové stavy
(*conformant problems*)
 - **neurčitý výsledek akce**
 - plány s alternativními větvemi
(*contingency problems*)
 - **neznámé akce**
 - řešení průzkumem
(*exploration problems*)



Umělá inteligence I, Roman Barták



Umělá inteligence I

Roman Barták, KTI ML

4

roman.bartak@mff.cuni.cz

<http://ktiml.mff.cuni.cz/~bartak>



Na úvod

- **Neinformované (slepé)** prohledávání umí najít (optimální) řešení problému, ale ve většině případů je hodně neefektivní.
- Naopak **informované** prohledávání, které používá problémově závislou informaci, může řešení nalézt mnohem rychleji.
 - **Jak využívat heuristiky v prohledávání?**
 - BFS, A*, IDA*, RBFS, SMA*
 - **Jak konstruovat heuristiky?**
 - relaxace, databáze vzorů





Informace v prohledávání

- Připomeňme, že hledáme (nejkratší) cestu z počátku do nějakého cílového stavu ve stavovém prostoru.
- Jaká informace může pomoci prohledávacímu algoritmu?
 - Například délka cesty do nějakého cílového stavu.
 - Tento údaj ale zpravidla nebývá známý (pokud ano, potom nemusíme nic prohledávat), proto se místo něj používá **evaluační funkce $f(n)$** ohodnocující uzel n na základě délky cesty do cíle.
 - **prohledávání dle ceny (best-first search)**
 - po expanzi pokračujeme s uzlem s nejmenší hodnotou $f(n)$.
 - Existují různé prohledávací algoritmy lišící se funkcí $f(n)$, její složkou ale skoro vždy bývá **heuristická funkce $h(n)$** odhadující délku nejkratší (nejlevnější) cesty do cíle.
 - Heuristická funkce je nejběžnější formou dodatečné informace pro prohledávání.
 - Nadále budeme uvažovat, že **$h(n) = 0 \Leftrightarrow n$ je cílový stav.**

Umělá inteligence I, Roman Barták



Greedy best-first search

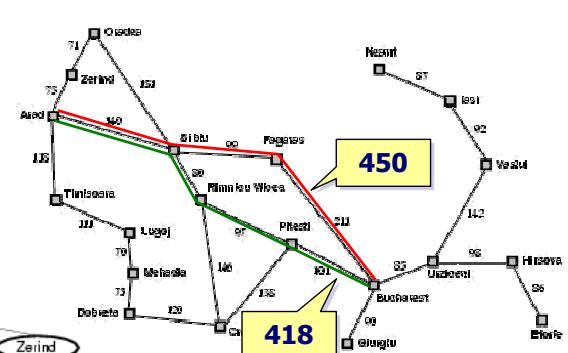
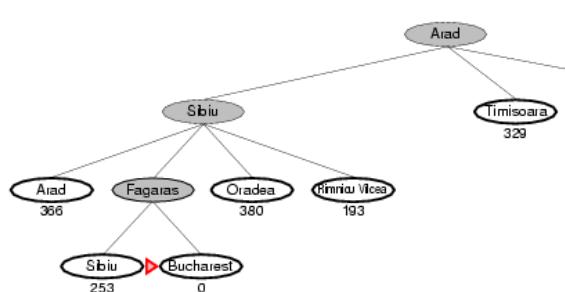
Hedání s cenou

- Zkusme při prohledávání preferovat uzel, který je nejblíže cíli, tj. $f(n) = h(n)$.
 - **hladový algoritmus prohledávání s cenou**

Příklad (cesta Arad → Bukurešť):

- Máme k dispozici tabulku přímých vzdáleností do Bukurešti.
- Pozn.: tato informace není součástí původního problému!

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



Nejkratší cesta?

Umělá inteligence I, Roman Barták

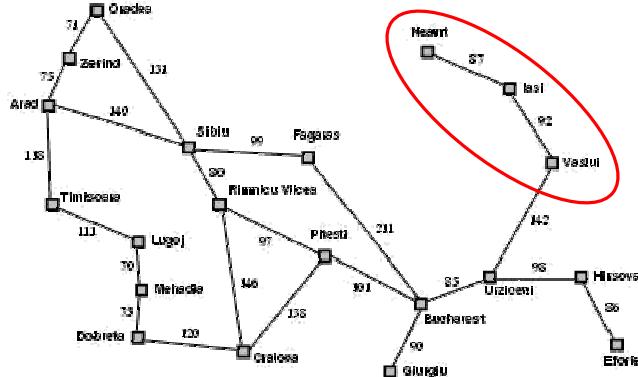


- Již víme, že hladový algoritmus s cenou **nemusí najít optimální řešení.**
- **Garantuje alespoň nalezení řešení?**

- Pokud vždy bude pro expanzi brát uzel s nejmenší cenou, tak **řešení najít nemusí.**

Příklad: cesta Iasi → Fagaras

- půjde do Neamt, potom opět Iasi, Neamt, ...
- je potřeba detektovat opakování stavu!



Umělá inteligence I, Roman Barták

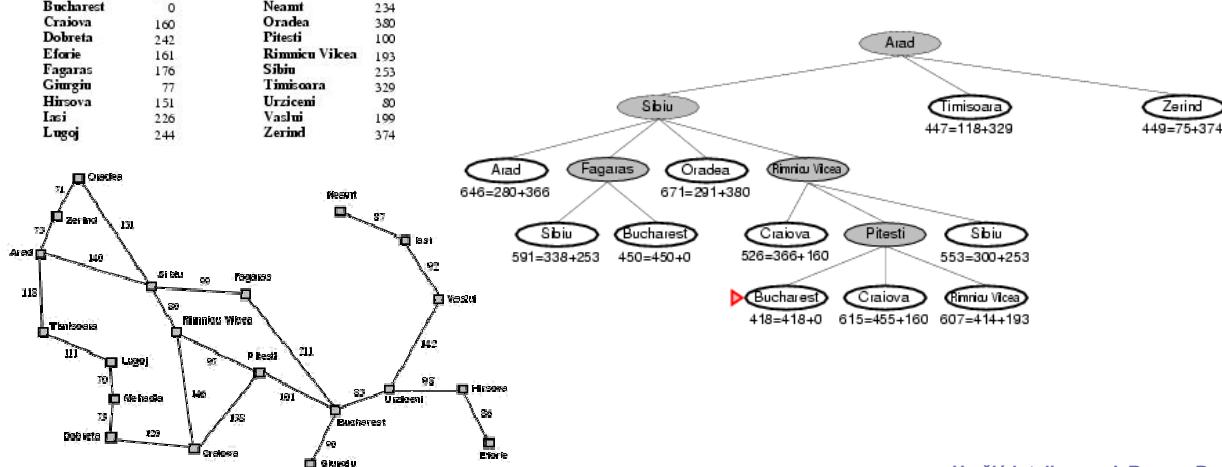
- **Čas $O(b^m)$,** kde m je maximální hloubka
- **Paměť $O(b^m)$**
- Dobrá heuristická funkce může složitost výrazně redukovat.



Algoritmus A*

- Zkusme nyní vzít $f(n) = g(n) + h(n)$
 - připomeňme, že $g(n)$ je cena cesty z kořene do n
 - asi nejznámější algoritmus prohledávání s cenou
 - $f(n)$ reprezentuje cenu cesty přes n
 - algoritmus tedy neprodlužuje cesty, které už jsou dlouhé

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesi	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iaci	226	Vaslui	199
Lugoj	244	Zerind	374



Umělá inteligence I, Roman Barták



Vlastnosti heuristik

Jak je to s úplností a optimalitou A*?

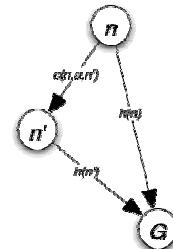
Nejprve pár definic:

přípustná heuristika $h(n)$

- $h(n) \leq$ „nejmenší cena cesty z n do cílového uzlu“
- optimistický pohled (uvažuje lepší cenu, než jaká skutečně je)
- funkce $f(n)$ u A* tedy také zdola odhaduje cenu řešící cesty přes n

monotónní (konzistentní) heuristika $h(n)$

- nechť n' je následník n přes akci a , $c(n,a,n')$ je cena přechodu
- $h(n) \leq c(n,a,n') + h(n')$
- jedná se o formu trojúhelníkové nerovnosti



Monotónní heuristika je přípustná.

nechť n_1, n_2, \dots, n_k je optimální cesta z n_1 do n_k , potom

$$h(n_i) - h(n_{i+1}) \leq c(n_i, a_i, n_{i+1}), \text{ dle monotonie}$$

$$h(n_1) \leq \sum_{i=1, \dots, k-1} c(n_i, a_i, n_{i+1}), \text{ po „sečtení“}$$

Pro monotónní heuristiku jsou hodnoty $f(n)$ podél libovolné cesty neklesající.

nechť n' je následník n , tj. $g(n') = g(n) + c(n, a, n')$, potom

$$f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') \geq g(n) + h(n) = f(n)$$

Umělá inteligence I, Roman Barták



Algoritmus A*

optimalita

■ Je-li $h(n)$ přípustná heuristika, potom je algoritmus A* v rámci TREE-SEARCH optimální.

jinými slovy, první expandovaný cílový uzel je optimální

nechť G_2 je sub-optimální cílový uzel z okraje a C^* je optimální cena

$$\blacksquare f(G_2) = g(G_2) + h(G_2) = g(G_2) > C^*, \text{ protože } h(G_2) = 0$$

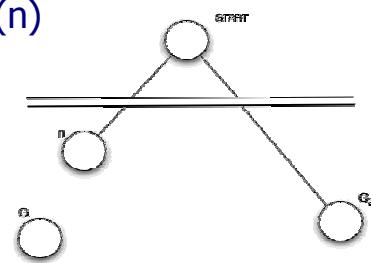
nechť n je uzel z okraje, který je na optimální cestě

$$\blacksquare f(n) = g(n) + h(n) \leq C^*, \text{ dle přípustnosti } h(n)$$

dohromady

$$\blacksquare f(n) \leq C^* < f(G_2),$$

tj. algoritmus musí vždy expandovat n dříve než G_2 a tím se dostane do optimálního cílového uzlu.



Umělá inteligence I, Roman Barták



Algoritmus A*

optimalita

■ Je-li $h(n)$ monotónní heuristika, potom je algoritmus A* v rámci GRAPH-SEARCH optimální.

- Problém nastane, pokud se do stejného stavu podruhé dostaneme lepší cestou – standardní GRAPH-SEARCH tuto druhou cestu zahodí!
- Řešením může být vybrání lepší z cest už zavřeného uzlu (extra bookkeeping) nebo použití monotónní heuristiky.
 - pro monotónní heuristiku jsou hodnoty $f(n)$ podél libovolné cesty neklesající
 - pro expanzi (zavření) vybírá A* vždy uzel s nejmenším $f(n)$, tj. ostatní otevřené uzly nemají menší $f(m)$, tj. mezi „otevřenými“ cestami nemůže být žádná cesta vedoucí do n lepší než ta právě nalezená (cesta se nezkrátí)
 - proto první zavřený cílový uzel musí být optimální

Umělá inteligence I, Roman Barták

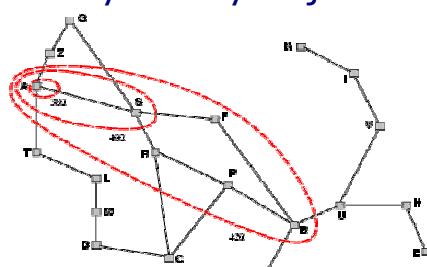


Algoritmus A*

vlastnosti

■ Pro neklesající $f(n)$ je možné ve stavovém grafu nakreslit **vrstevnice** tak, že v dané vrstvě jsou vždy všechny vrcholy mající $f(n)$ do dané meze.

- pro $h(n) = 0$ dostaneme soustředné okruhy
- pro přesnější $h(n)$ se budou okruhy „protahovat“ podél optimální cesty k cíli.



- A* expanduje všechny uzly, kde $f(n) < C^*$, a to po vrstevnicích
- A* může expandovat některé uzly, kde $f(n) = C^*$
- uzly, kde $f(n) > C^*$, nejsou expandovány
- algoritmus tedy **vždy najde optimální řešení**

Časová složitost:

A* může expandovat exponenciálně mnoho uzlů

- lze tomu zabránit, pokud $|h(n)-h^*(n)| \leq O(\log h^*(n))$, kde $h^*(n)$ je cena optimální cesty z n do cíle

Prostorová složitost:

A* drží v paměti všechny vygenerované uzly

A* zpravidla dříve vyčerpá paměť než nastanou problémy s časem

Umělá inteligence I, Roman Barták

- Jednoduchý způsob zmenšení paměťových nároků je použití iterovaného prohlubování.

■ Algoritmus IDA*

```

function IDA*(problem) returns a solution sequence
  inputs: problem, a problem
  static: f-limit, the current f- COST limit
         root, a node

  root ← MAKE-NODE([INITIAL-STATE[problem]])
  f-limit ← f- COST(root)
  loop do
    solution, f-limit ← DFS-CONTOUR(root, f-limit)
    if solution is non-null then return solution
    if f-limit = ∞ then return failure; end

function DFS-CONTOUR(node, f-limit) returns a solution sequence and a new f- COST limit
  inputs: node, a node
  f-limit, the current f- COST limit
  static: next-f, the f- COST limit for the next contour, initially ∞

  if f- COST[node] > f-limit then return null, f- COST[node]
  if GOAL-TEST[problem][STATE[node]] then return node, f-limit
  for each node s in SUCCESSORS(node) do
    solution, new-f ← DFS-CONTOUR(s, f-limit)
    if solution is non-null then return solution, f-limit
    next-f ← MIN(next-f, new-f); end
  return null, next-f

```

- pro limit prohledávání se místo hloubky použije cena $f(n)$
- pro následující iteraci se použije limit odpovídající nejmenšímu $f(n)$ uzlu n , který v předchozí iteraci překročil limit
- často používaný algoritmus

Umělá inteligence I, Roman Barták

- Zkusme **rekurzivní verzi prohledávání s cenou** za použití lineárního prostoru
 - algoritmus přeruší prohledávání, pokud je v paralelní větví lepší cena $f(n)$
 - pokud se algoritmus vrací do uzlu n , upřesní jeho cenu $f(n)$ dle prozkoumaných potomků (paměť již prozkoumané části)
- **Je-li $h(n)$ přípustná, je algoritmus optimální.**
- **Paměťová složitost $O(bd)$**
- **Časová složitost stále exponenciální** (řada uzel se může procházet opakováně)

```

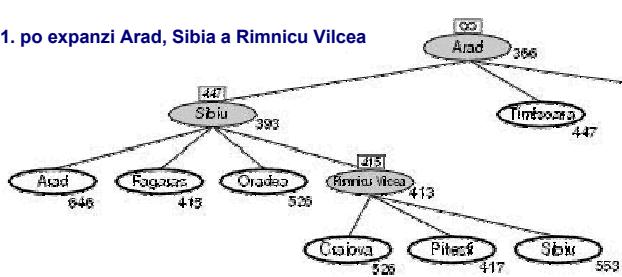
function RECURSIVE-BEST-FIRST-SEARCH(problem) returns a solution, or failure
  RBFS(problem, MAKE-NODE([INITIAL-STATE[problem]], ∞))

function RBFS(problem, node, f_limit) returns a solution, or failure and a new f-cost limit
  if GOAL-TEST[problem][STATE[node]] then return node
  successors ← EXPAND(node, problem)
  if successors is empty then return failure, ∞
  for each s in successors do
    f[s] ← max(g(s) + h(s), f[node])
  repeat
    best ← the lowest f-value node in successors
    if f[best] > f_limit then return failure, f[best]
    alternative ← the second-lowest f-value among successors
    result, f[best] ← RBFS(problem, best, min(f_limit, alternative))
    if result ≠ failure then return result

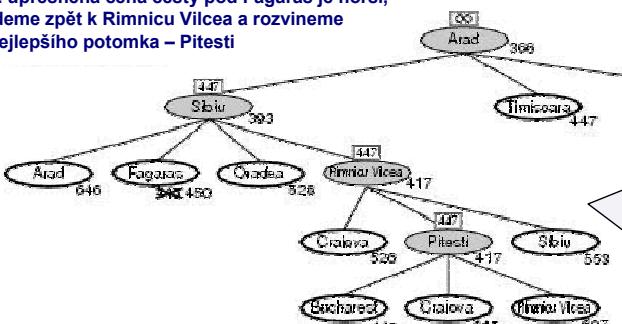
```

Umělá inteligence I, Roman Barták

1. po expanzi Arad, Sibia a Rimnicu Vilcea



3. upřesněná cena cesty pod Fagaras je horší, jdeme zpět k Rimnicu Vilcea a rozvineme nejlepšího potomka – Pitesti



2. cesta pod Rimnicu Vilcea se ukázala moc drahá, vracíme se k nejlepšímu sousedovi – Fagaras u Rimnicu Vilcea si ale pamatujeme přesné cenu

Umělá inteligence I, Roman Barták

Zjednodušené MA*

- IDA* i RBFS nevyužívají dostupnou paměť!
- To je škoda, protože se dříve expandované uzly znova procházejí (ztráta času)
- Zkusme vzít standardní algoritmus A*

```

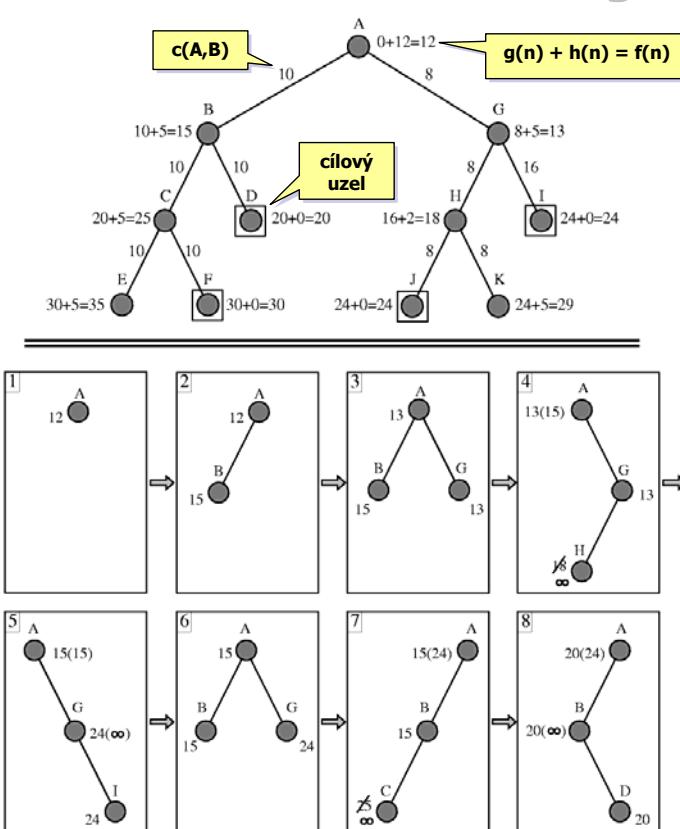
function SMA*(problem) returns a solution sequence
  inputs: problem, a problem
  static: Queue, a queue of nodes ordered by f-cost
  Queue ← MAKE-QUEUE({MAKE-NODE(INITIAL-STATE[problem])})
  loop do
    if Queue is empty then return failure
    n ← deepest least-f-cost node in Queue
    if GOAL-TEST(n) then return success
    s ← NEXT-SUCCESSOR(n)
    if s is not a goal and is at maximum depth then
      f(s) ← ∞
    else
      f(s) ← MAX(f(n), g(s)+h(s))
      if all of n's successors have been generated then
        update n's f-cost and those of its ancestors if necessary
      if SUCCESSORS(n) all in memory then remove n from Queue
      if memory is full then
        delete shallowest, highest-f-cost node in Queue
        remove it from its parent's successor list
        insert its parent on Queue if necessary
      insert s on Queue
    end
  end

```

Cesta z kořene k tomuto
necelověmu listu se
nevejde do paměti, a proto
přes tento uzel nemůžeme
najít optimální řešení!

- když vyčerpá paměť, vyhodíme nejméně slibný list, tj. ten s největší cenou $f(n)$
- podobně jako u RBFS si ale tuto cenu zapamatujeme u rodiče

Umělá inteligence I, Roman Barták



- Uvažujme paměť pouze pro **tři uzly**.

- Pokud je paměť dostatečná pro (optimální) cestu, SMA* najde (optimální) řešení.

- Jinak najde nejlepší možné řešení s dostupnou pamětí.

- Pokud by cena J byla 19, tak to je optimální řešení, ale cesta k němu se nevejde do paměti!

Umělá inteligence I, Roman Barták

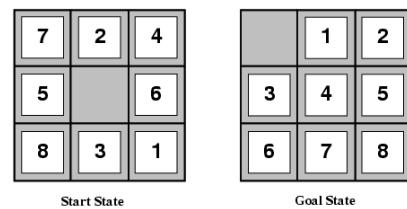


Hledáme heuristiky

Jak hledat přípustné heuristiky?

Příklad: Loydova osmička

- průměrně 22 kroků k nalezení řešení
- větvící faktor je kolem 3
- (plný) prohledávací strom: $3^{22} \approx 3,1 \times 10^{10}$ uzlů
- počet stavů ale jen $9!/2 = 181\,440$
- pro Loyodovu patnáctku už 10^{13} stavů
- potřebujeme heuristiku, pokud možno přípustnou
 - h_1 = „počet špatně umístěných kostiček“
= 8
 - h_2 = „součet vzdáleností kostiček od cílových pozic“
= $3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$
tzv. Manhattanská heuristika
 - skutečné řešení potřebuje 26 kroků



Jak měřit kvalitu heuristik? Efektivní větvící faktor b^*

- nechť algoritmus potřebuje N uzelů na nalezení řešení v hloubce d .
- b^* je větvící faktor rovnoměrného stromu hloubky d , který obsahuje $N+1$ uzelů, tj.

$$N+1 = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$$

Příklad:

- Loydova patnáctka
- průměr ze 100 náhodných problémů pro každou délku řešení

d	Search Cost			Effective Branching Factor		
	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	–	539	113	–	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Umělá inteligence I, Roman Barták



Dominance

- Je h_2 (z Loydovy osmičky) **vždy lepší než h_1 a jak to snadno poznat?**
 - všimněme si, že $\forall n h_2(n) \geq h_1(n)$
 - říkáme, že h_2 **dominuje** h_1
 - A* s h_2 nikdy neexpanduje více uzelů než A* s h_1
 - A* expanduje všechny uzly, kde $f(n) < C^*$, tj. $h(n) < C^* - g(n)$
 - tedy, pokud expanduje uzel dle h_2 , potom expanduje stejný uzel i dle h_1
- Vždy je lepší použít heuristickou funkci s většími hodnotami.
 - pokud nepřekročímez $C^* - g(n)$ (to by nebyla přípustná)
 - pokud se nepočítá příliš dlouho

Umělá inteligence I, Roman Barták

Může agent sám najít přípustné heuristiky pro obecný problém?

Ano, pomocí **relaxace problému!**

- relaxace je zjednodušení problému takové, aby řešení původního problému bylo řešením i relaxovaného problému (i když třeba neoptimální)
- relaxujeme do takové míry, že relaxovaný problém umíme rychle vyřešit
- cena optimálního řešení relaxovaného problému je potom dolním odhadem ceny originálního problému, tj. přípustnou (a také monotónní) heuristikou

■ Příklad (Loyd)

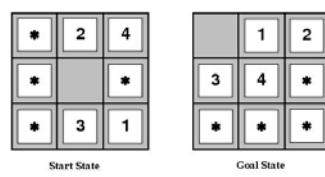
- kostičku lze posunou z místa A na místo B pokud:
 - A je horizontálně i vertikálně sousedem B
 - B je prázdné políčko
- možné relaxace (vyřazení podmínek z pravidla pro posun):
 - posun A na B je možný, pokud A je sousedem B (Manhattanská heuristika)
 - posun A na B je možný, pokud B je prázdné
 - posun A na B je možný vždy (heuristika h_1)

Umělá inteligence I, Roman Barták

Databáze vzorů

Jiný přístup k hledání přístupných heuristik je přes **databázi vzorů** (pattern database)

- založeno na **řešení podproblémů** (typových příkladů)
- prohledáním od cíle najdeme různá optimální řešení, ze kterých uděláme **vzory**
- heuristiku sestrojíme tak, že vezmeme nejhorší optimální řešení pro vzory, které najdeme v aktuálním stavu
- Pozor! Součet řešení vzorů nemusí být přípustný (v postupu pro řešení jednoho vzoru můžeme použít kroky z řešení jiných vzorů)



Máme-li **více heuristik**, můžeme z nich vzít **maximum** (dominuje každé z nich).

Umělá inteligence I, Roman Barták



Umělá inteligence I



Roman Barták, KTI ML

roman.bartak@mff.cuni.cz

<http://ktiml.mff.cuni.cz/~bartak>



Na úvod

- **Minule** jsme si řekli, jak využívat heuristiky v prohledávání a jak konstruovat heuristiky
 - BFS, A*, IDA*, RBFS, SMA*
 - relaxace problému, databáze vzorů
- **Dnes:**
 - **Co když na cestě nezáleží?**
 - lokální prohledávání: HC, SA, BS, GA
 - **Co když se svět mění?**
 - on-line prohledávání, LRTA*





Lokální prohledávání

- Dosud jsme systematicky prohledávali cesty vedoucí do cílového stavu a nalezená cesta potom byla součástí řešení.
- U některých problémů ale není cesta relevantní, důležitý je nalezený cíl (např. 8-královen).
- V takovém případě můžeme zkusit tzv. **lokální prohledávání**.
 - pracuje s jedním stavem (konstantní paměť)
 - v každém kroku tento stav „trochu“ změní na jiný stav
 - zpravidla si nepamatuje prošlou cestu
 - umí hledat i **optimální stav**, kde optimalita je definovaná objektivní funkcí (na stavech)
 - Např. u problému 8-královen může být objektivní funkcí (pro minimalizaci) počet konfliktních dvojic – informace o problému.

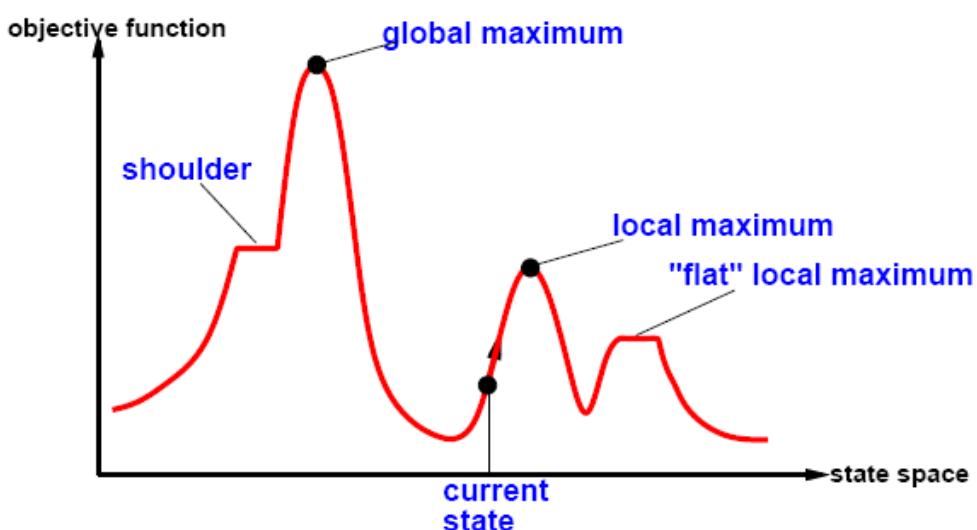
Umělá inteligence I, Roman Barták



Lokální prohledávání

základní pojmy

- Lokální prohledávání je často definováno pohybem v **krajině**, kde souřadnice pozice určují stav a výška definuje hodnotu objektivní funkce v daném stavu.



Umělá inteligence I, Roman Barták



- Z okolí daného stavu vždy vybereme stav, který má nejlepší objektivní funkci a do tohoto stavu přejdeme (**metoda největšího stoupání**).
 - vidí pouze okolí daného stavu
 - předchozí stav okamžitě zapomíná

- počet konfliktů = 17
- změna stavu = změna řádku jedné královny
- náhodný výběr mezi více „nejlepšími“ soused

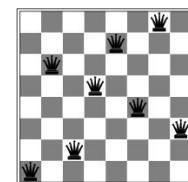
```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  inputs: problem, a problem
  local variables: current, a node
                  neighbor, a node
  current  $\leftarrow$  MAKE-NODE(INITIAL-STATE[problem])
  loop do
    neighbor  $\leftarrow$  a highest-valued successor of current
    if VALUE[neighbor]  $\leq$  VALUE[current] then return STATE[current]
    current  $\leftarrow$  neighbor
```

Umělá inteligence I, Roman Barták



HC je **hladový algoritmus** – jde za nejlepším sousedem bez pohledu dále dopředu

- **lokální optimum** (stav, kde každý pohyb vede do horšího stavu)
 - HC z něho neumí uniknout
- **hřebeny** (posloupnost lokálních optim)
 - velmi komplikované pro hladové algoritmy
- **plošiny** (plateau - oblast se stejně dobrými stavy)
 - rameno – typ plošiny, ze které lze uniknout
 - HC se nemusí podařit najít cestu (cyklus)



Umělá inteligence I, Roman Barták



■ stochastický HC

- mezi zlepšujícími sousedy vybírá náhodně s pravděpodobností danou velikostí zlepšení
- nemusí se vydat do nejlepšího souseda (pomalejší konvergence)
- pro některé problémy dává lepší řešení

■ HC první volby

- náhodně generuje sousedy dokud nenajde lepší stav a do něj se vydá
- vhodné, pokud je velké množství sousedů

■ HC s náhodnými restarty

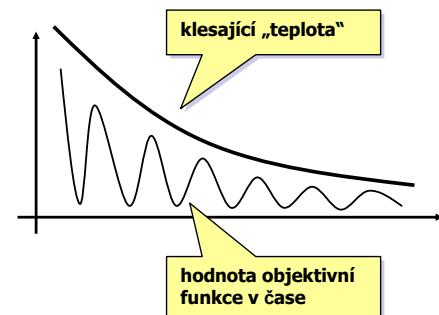
- ocitne-li se v lokálním optimu, začne prohledávat znova z náhodného stavu (restart)
- umožňuje únik z lokálního optima
- při pravděpodobnosti p , že cesta vede k řešení, je očekávaný počet restartů $1/p$
- velmi efektivní metoda pro řešení N-královen ($p \approx 0,14$ tj. 7 iterací)

Umělá inteligence I, Roman Barták



- **HC** se nikdy nevydá dolů (proti objektivní funkci), proto není úplný (nepřekoná lokální optima).
- **Náhodná procházka** (random walk), která vybírá stav z okolí zcela náhodně, je úplná, ale pomalá.
- **Simulované žíhání** kombinuje výhody obou metod
 - motivace v metalurgii – postupné ochlazování umožňuje lepší usazení atomů do krystalické mřížky
 - algoritmus náhodně volí stav z okolí, do které se vydá pokud:
 - je lepší než aktuální stav
 - je horší než aktuální stav, ale zhoršení je povoleno s určitou mírou pravděpodobnosti odvozenou od aktuální teploty; teplota se přitom postupně snižuje podle „ochlazovacího“ schématu
 - stav se v krajině pohybuje podobně jako skákající míček

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"
  local variables: current, a node
                    next, a node
                    T, a "temperature" controlling prob. of downward steps
  current ← MAKE-NODE(INITIAL-STATE[problem])
  for t ← 1 to ∞ do
    T ← schedule[t]
    if T = 0 then return current
    next ← a randomly selected successor of current
    ΔE ← VALUE[next] - VALUE[current]
    if ΔE > 0 then current ← next
    else current ← next only with probability e^Δ E/T
```



Umělá inteligence I, Roman Barták



Představené algoritmy lokálního prohledávání mají extrémně malé paměťové nároky (jeden stav).

Nešlo by využít dostupnou paměť lépe?



■ Algoritmus lokálních paprsků

- začíná s k náhodnými stavy
- v každém kroku najde všechny jejich sousedy
 - je-li mezi nimi cílový stav, potom končí
- ze sousedů vybere k nejlepších pro další krok

■ Pozor! To není paralelní simulace k restartů HC!

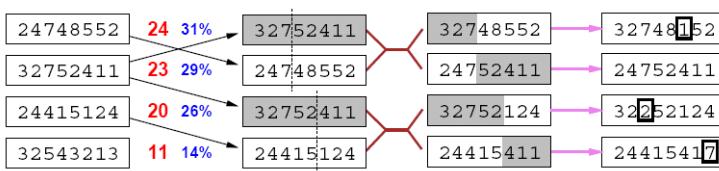
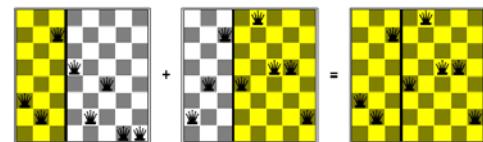
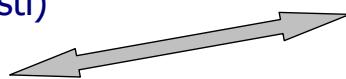
- zde se předává informace mezi jednotlivými paprsky
- vybraná k-tice stavů mohou být např. sousedé jediného stavu
- algoritmus se tak soustředí na prozkoumání slibné oblasti
- někdy tak ale může přijít o rozmanitost prozkoumaných stavů
 - lze řešit stochastickou verzí (náhodný výběr stavů s pravděpodobností danou kvalitou stavu)
 - trochu připomíná přirozený výběr druhů dle Darwina



Genetické algoritmy

■ Varianta stochastického prohledávání s paprsky – kombinují se dva stavy (sexuální reprodukce)

- začneme s k náhodnými stavami – **populace**
 - stav je reprezentován řetězcem symbolů v konečné abecedě (jako DNA)
 - **fitness** funkce určuje kvalitu stavu (dle objektivní funkce)
- vyberou se dvojice stavů pro reprodukci (pravděpodobnost výběru dána fitness funkcí)
- pro každý pár se určí bod přechodu (**crossover**), který rozdělí řetězce popisující stavы
- komplementární řetězce se spojí do nového stavu
- u nového stavu se provede **mutace** (náhodná změna písmenka s malou pravděpodobností)





Genetické algoritmy

struktura

```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
    new_population ← empty set
    loop for i from 1 to SIZE(population) do
      x ← RANDOM-SELECTION(population, FITNESS-FN)
      y ← RANDOM-SELECTION(population, FITNESS-FN)
      child ← REPRODUCE(x, y)
      if (small random probability) then child ← MUTATE(child)
      add child to new_population
    population ← new_population
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to FITNESS-FN

function REPRODUCE(x, y) returns an individual
  inputs: x, y, parent individuals

  n ← LENGTH(x)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
```

Umělá inteligence I, Roman Barták



Offline vs. online



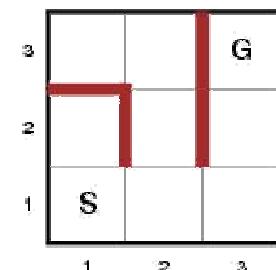
- Dosud probírané **offline prohledávání**
 - nejprve najde kompletní řešení
 - potom aplikuje řešení aniž by uvažovalo vjemů
- **Online prohledávání** naopak
 - prolíná hledání řešení a provádění akcí
 - vybere akci
 - provede akci
 - zjistí jak vypadá svět
 - vybere další akci
 - vhodné pro dynamické a semi-dynamické prostředí
 - vhodné pro neurčité výsledky akcí a pro neznámé akce

Umělá inteligence I, Roman Barták



Online prohledávání

- Online prohledávání má smysl pro **agenty, kteří provádějí akce** (nemá smysl pro čistě „výpočtové“ agenty).
- Agent má k dispozici následující **informace**
 - **Actions(s)** – seznam akcí aplikovatelných na stav S
 - **c(s,a,s')** – cena provedení jednoho kroku (může být použita až když agent zná stav s')
 - **Goal-Test(s)** – stav s je cílový
- Dále budeme předpokládat, že agent
 - umí **rozpoznat již navštívený stav**
 - (agent může budovat mapu světa)
 - používá **deterministické akce**
 - má k dispozici **přípustnou heuristiku** $h(s)$



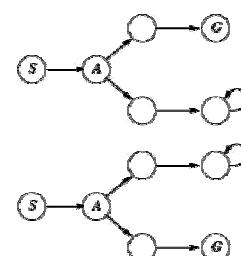
Umělá inteligence I, Roman Barták



Kvalita algoritmů

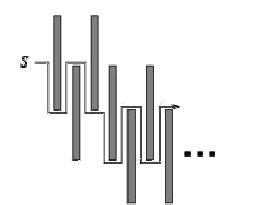
Kvalita online algoritmů se typicky měří **porovnáním s offline řešením**.

- **Kompetitivní poměr**
= kvalita online řešení / kvalita nejlepšího řešení
 - Může být ∞ , např. při vkročení do **slepé uličky** (pokud nemáme reverzní akce pro krok zpět).
 - **Tvrzení:** Žádný online algoritmus se umí vyhnout slepým uličkám ve všech problémech.
 - Důkaz (metodou protivníka)
Agent, který navštívil S a A se musí v obou příkladech rozhodnout stejně a v jednom případě tedy skončí ve slepé uličce.



Budeme předpokládat **bezpečně prozkoumatelné světy** (z každého stavu vede cesta do cíle).

- Ani zde ale nelze garantovat kompetitivní poměr, pokud máme cesty neomezené ceny.
- Metodou protivníka můžeme stavět do cesty překážky, které cestu libovolně prodlouží.

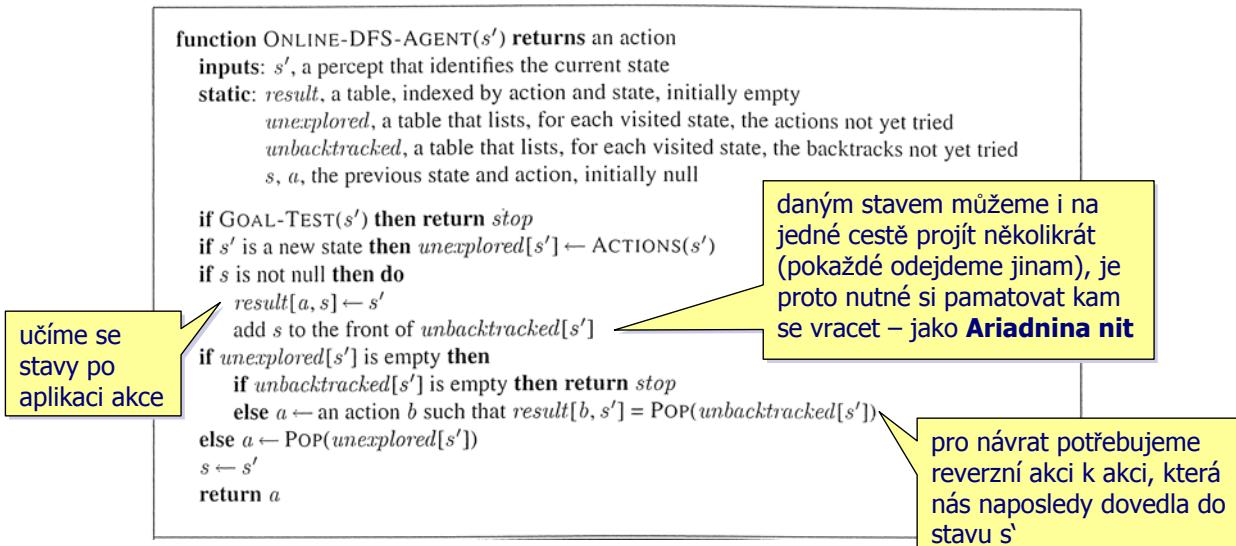


Kvalita online algoritmů se proto často měří **vzhledem k velikosti celého stavového prostoru** (místo délky nejkratší cesty).

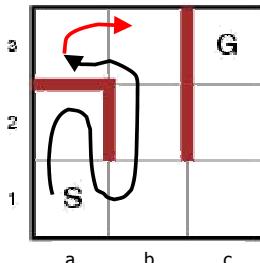
Umělá inteligence I, Roman Barták



- Uvědomme si, že na rozdíl od offline algoritmů typu A* **nemohou on-line algoritmy přeskočit do zcela jiného stavu** aniž by absolvovaly příslušnou cestu.
- Je proto vhodné expandovat uzly v **lokálním** pořadí, jako to například dělá DFS.



Umělá inteligence I, Roman Barták



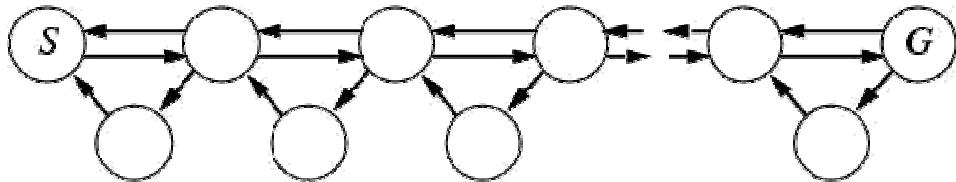
stav	unEX	unBT	rUP	rDN	rLF	rRG
(1,a)	{}	(2,a)	(2,a)	-	-	(1,b)
(2,a)	{}	(1,a)	-	(1,a)	-	-
(1,b)	LF,RG	(1,a)	(2,b)	-		
(2,b)	DW	(1,b)	(3,b)		-	-
(3,b)	DW	(3,a), (2,b)	-		(3,a)	-
(3,a)		(3,b)	-	-	-	(3,b)

- V nejhorším případě přejde po každém spoji dvakrát (tam a zpátky).
- To je optimální pro průzkum světa, ale cesta i do blízkého cíle může být dlouhá.
 - on-line verze iterování hloubky to řeší
- On-line prohledávání do hloubky lze uplatnit pouze tehdy, pokud jsou **akce oboustranné** (může se vrátit o krok zpět).

Umělá inteligence I, Roman Barták

■ Horolezecká metoda je on-line algoritmus.

- drží v paměti **jediný stav** (kde se nachází agent)
- dělá pouze **lokální kroky** do „sousedních“ stavů
- v nejjednodušší podobě se ale **neumí dostat z lokálního optima**
 - Pozor! **Nelze vyřešit náhodným restartem!**
 - Můžeme ale použít **náhodnou procházku**.
 - Za předpokladu konečného stavového prostoru **nakonec najde řešení** nebo prozkoumá celý stavový prostor.
 - Obecně ale může **projít exponenciálně dlouhou cestu**.
 - V následujícím stavovém prostoru je pravděpodobnost kroku zpět dvakrát větší než pravděpodobnost kroku dopředu

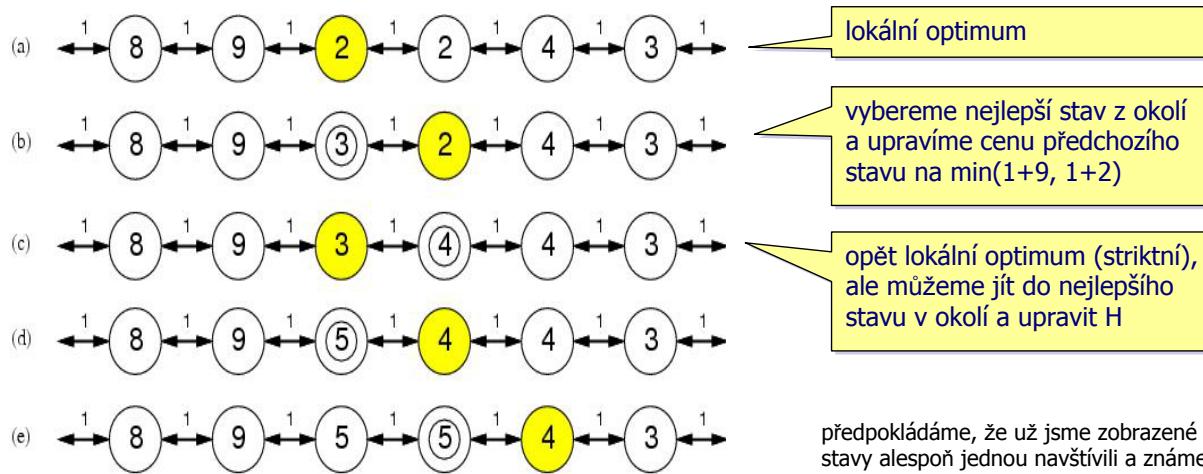


Umělá inteligence I, Roman Barták

Lokálně s učením

■ Jiná (efektivnější) metoda cesty z lokálního optima je **využití paměti**.

- **H(s)** – současný nejlepší odhad délky cesty ze stavu s do cíle (na začátku $h(s)$)



Umělá inteligence I, Roman Barták



- Algoritmus LRTA* dělá lokální kroky a učí se, kam ho daná akce doveze (result) a jaká je vzdálenost do cíle (H).

```
function LRTA*-AGENT( $s'$ ) returns an action
  inputs:  $s'$ , a percept that identifies the current state
  static: result, a table, indexed by action and state, initially empty
         H, a table of cost estimates indexed by state, initially empty
          $s, a$ , the previous state and action, initially null

  if GOAL-TEST( $s'$ ) then return stop
  if  $s'$  is a new state (not in H) then  $H[s'] \leftarrow h(s')$ 
  unless  $s$  is null
    result[ $a, s$ ]  $\leftarrow s'$ 
     $H[s] \leftarrow \min_{b \in \text{ACTIONS}(s)} \text{LRTA}^*-\text{COST}(s, b, \text{result}[b, s], H)$ 
     $a \leftarrow$  an action  $b$  in ACTIONS( $s'$ ) that minimizes LRTA $^*$ -COST( $s', b, \text{result}[b, s'], H$ )
     $s \leftarrow s'$ 
  return  $a$ 

function LRTA $^*$ -COST( $s, a, s', H$ ) returns a cost estimate
  if  $s'$  is undefined then return  $h(s)$ 
  else return  $c(s, a, s') + H[s']$ 
```

upřesníme odhad v předchozím stavu

vybereme další akci s nejlepší cenou do
cíle (může vést i do předchozího stavu);
dává přednost novým cestám

pokud jsme akci na daný stav
ještě nepoužili, optimisticky
předpokládáme, že nás
dovede do cíle s nejmenší
možnou cenou, tj. $h(s)$



Umělá inteligence I

Roman Barták, KTI ML

roman.bartak@mff.cuni.cz

<http://ktiml.mff.cuni.cz/~bartak>



6

Na úvod

- Zatím pro nás byl model světa **černou skříňkou**, ke které přistupujeme pouze přes:
 - funkci následníka
 - test cílového stavu
 - heuristickou funkci (vzdálenost do cíle)
- **Dnes** si ukážeme
 - jednu z možných obecných reprezentací problémů – **problém splňování podmínek (CSP)**
 - má vnitřní strukturu, kterou lze využít při řešení
 - obecné metody pro **řešení CSP**
 - kombinace prohledávání a odvozovacích technik





Sudoku?

- **Logická hádanka**, jejímž cílem je doplnit chybějící čísla 1-9 do tabulky 9×9 tak, aby se čísla neopakovala v žádném řádku, sloupcu ani v malých čtvercích 3×3.

9	6	3	1	7	4	2	5	8
1	7	8	3	2	5	6	4	9
2	5	4	6	8	9	7	3	1
8	2	1	4	3	7	5	9	6
4	9	6	8	5	2	3	1	7
7	3	5	9	6	1	8	2	4
5	8	9	7	1	3	4	6	2
3	1	7	2	4	6	9	8	5
6	4	2	5	9	8	1	7	3

Trochu historie

1979: poprvé publikováno v New Yorku

„Number Place“ – „Umísti číslice“

1986: populární v Japonsku

Sudoku – zkráceno z japonštiny "Súdži wa dokushin ni kagiru"

„Čísla musí být jedinečná“

2005: populární mezinárodně

Umělá inteligence I, Roman Barták



Začínáme se Sudoku

Jak poznáme, kam které číslo patří?

x	x	6	①	3				
3	9	x						
2	1	8			4			

- Využijeme informace, že každé číslo je v řádku maximálně jedenkrát.

A co když to nestací?

- Podíváme se do sloupců resp. zkombinujeme informaci ze sloupců a řádků.

6	x	1	3					
3	9	x	x	2			1	
②	1	8	x	x	x	4		
8	7		②					
			8	6	1			
				7		4	9	
					7		8	
						2	5	
			9	②		3		

Umělá inteligence I, Roman Barták



Sudoku o krok dál

		6		1	3	2	x	2
3	9			(2)	x	1	x	
(2)	1	8			4	x	x	
8	7	2						
		8	6	1				
			7		4	9		
	3			7	8			
4				(2)	5			
		9	2	3				

- Pokud řádky ani sloupce neposkytnout přesnou informaci, alespoň si můžeme poznamenat, kde dané číslo může ležet.

- Poloha čísla může být odvozena i z přítomnosti jiných čísel a z vlastnosti, že každé číslo se musí objevit alespoň jednou.

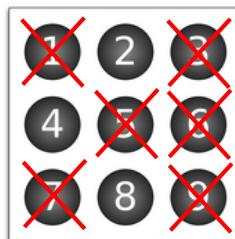
5	6		1	3				
3	9			2		1		
2	1	8				4		
8	(7)		2		6	1		
		8	6	1				
			7		4	9		
	3			7	9	8		
4				1	2	(5)		
		9	2	3	6	4		

Umělá inteligence I/Roman Barták



Sudoku obecně

5	3		7					
6			1	9	5			
9	8				6			
8			6			3		
4		8	3				1	
7			2			6		
6				2	8			
	4	1	9			5		
	8			7	9			



Na každé políčko se podíváme jako na **proměnnou** nabývající hodnoty z **domény** {1,...,9}.

Mezi všemi dvojicemi proměnných v řádku, sloupci či malém čtverci je **podmínka** nerovnosti.

Hodnoty nesplňující podmínky **škrtáme**.

Takto zadaný problém se nazývá **problém splňování omezujících podmínek**.

Škrtání hodnot – **filtrace domén** – se provádí tak dlouho, dokud se nějaká doména mění.

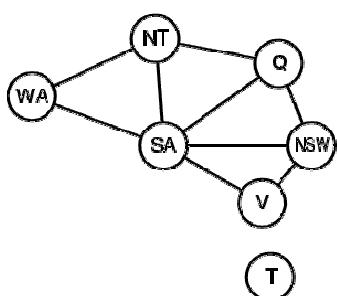
Problém splňování podmínek se skládá z:

- konečné množiny **proměnných**
 - popisují „atributy“ řešení, o kterých je potřeba rozhodnout například pozici dámy na šachovnici
- domén** – konečné množiny hodnot pro proměnné
 - popisují možnosti, které máme při rozhodování např. čísla řádků pro umístění dámy
 - někdy se definuje jedna superdoména společná pro všechny proměnné a individuální domény se v ní vytyčí přes unární podmínky
- konečné množiny **podmínek**
 - podmínka je libovolná relace nad množinou proměnných například poziceA \neq poziceB
 - může být definována extenzionálně (jako množina kompatibilních n-tic) nebo intenzionálně (formulí)

Umělá inteligence I, Roman Barták

Příklad CSP

- Najděte obarvení států (červená, modrá, zelená) takové, že sousední státy nemají stejnou barvu.



Reprezentace formou CSP

- proměnné: {WA, NT, Q, NSW, V, SA, T}
- superdoména: {č, m, z}
- podmínky: WA \neq NT atd.

- Lze také popsat formou **sítě podmínek** (vrcholy=proměnné, hrany=podmínky)

Řešení problému

$$\begin{aligned} WA &= \text{č}, & NT &= z, & Q &= \text{č}, & NSW &= z, \\ V &= \text{č}, & SA &= m, & T &= z \end{aligned}$$



Umělá inteligence I, Roman Barták

Stav odpovídá přiřazení hodnot do (některých) proměnných.

Konzistentní stav (přiřazení) neporušuje žádnou z podmínek.

V **úplném stavu** (přiřazení) mají všechny proměnné přiřazenu hodnotu.

Cílem je najít úplný konzistentní stav (přiřazení).

Někdy se k definici problému přidává **objektivní funkce**, tj. funkce definovaná na (některých) proměnných.

Potom je **optimálním cílovým stavem** úplný konzistentní stav (přiřazení), který maximalizuje/minimalizuje hodnotu objektivní funkce.

Umělá inteligence I, Roman Barták

Jak řešit CSP?

- Zatím jsme se naučili **prohledávací algoritmy**, tak je zkusíme i pro CSP.
 - **počáteční stav**: prázdné přiřazení
 - **možné akce**: přiřazení hodnoty do volné proměnné takové, že neporuší žádnou podmínu
 - **cíl**: úplné konzistentní přiřazení

Poznámky:

- řešící postup je stejný pro všechny CSP
- řešení je vždy v hloubce n , kde n je počet proměnných
 - můžeme bez problémů použít DFS (bez kontroly cyklů)
- pořadí akcí nemá žádný vliv na řešení (problém je tzv. **komutativní**)
 - $\langle WA=\check{c}, NT=z \rangle$ je stejné jako $\langle NT=z, WA=\check{c} \rangle$
- pro řešení CSP lze používat i jiná typy větvení než enumeraci (například půlení domén)

Umělá inteligence I, Roman Barták



Backtracking

rekurzivní algoritmus

- Základní neinformovaný algoritmus pro řešení CSP
 - postupně přiřazujeme hodnoty do volných proměnných
 - pokud žádná hodnota přiřadit nelze, potom u poslední přiřazené proměnné zkusíme jinou hodnotu

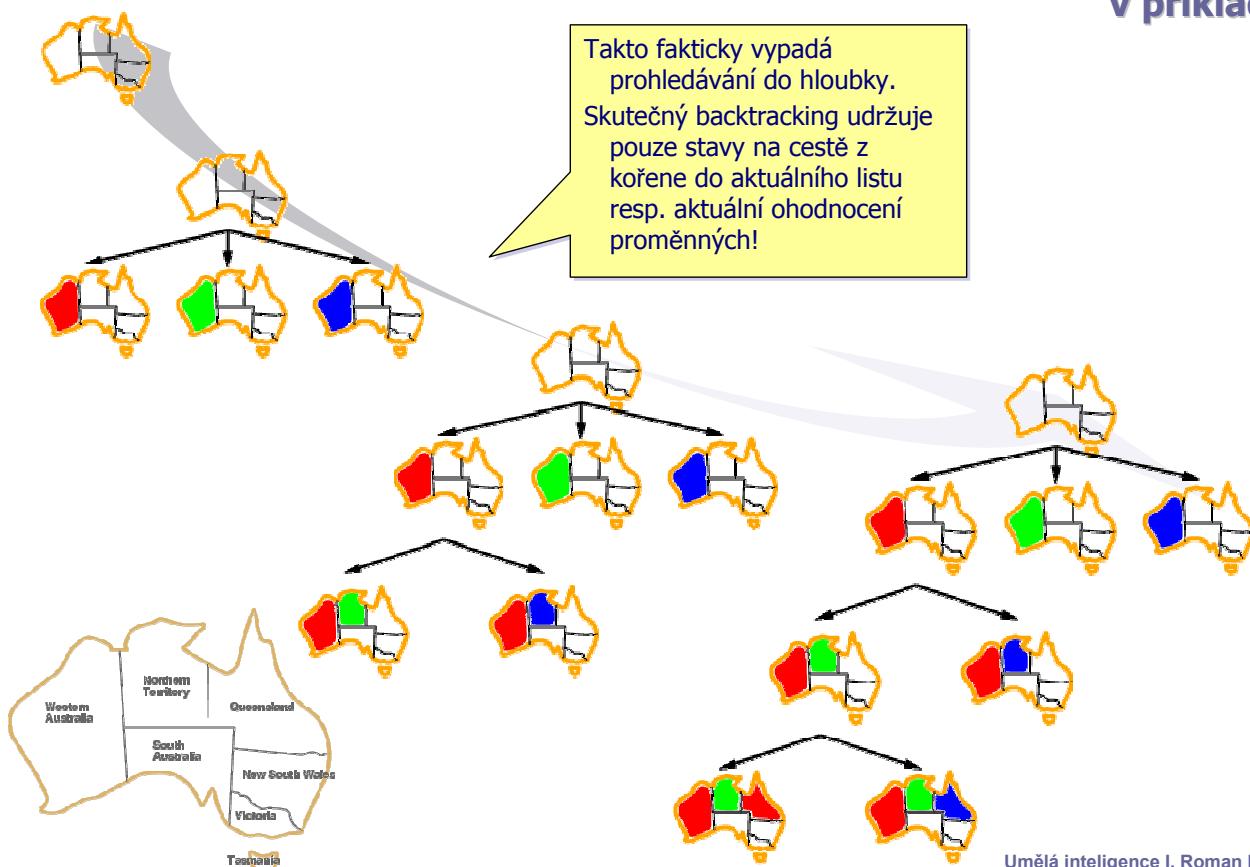
```
function BACKTRACKING-SEARCH( csp) returns a solution, or failure
    return RECURSIVE-BACKTRACKING( {}, csp)
function RECURSIVE-BACKTRACKING( assignment, csp) returns a solution, or failure
    if assignment is complete then return assignment
    var  $\leftarrow$  SELECT-UNASSIGNED-VARIABLE( Variables[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment according to Constraints[csp] then
            add { var = value } to assignment
            result  $\leftarrow$  RECURSIVE-BACKTRACKING(assignment, csp)
            if result  $\neq$  failure then return result
            remove { var = value } from assignment
    return failure
```

Umělá inteligence I, Roman Barták



Backtracking

v příkladě



Umělá inteligence I, Roman Barták



Jak ovlivnit efektivitu prohledávání?

volbou „správné“ hodnoty

- zpravidla problémově závislé

volbou proměnné pro ohodnocení

- i když nakonec musíme přiřadit hodnoty do všech proměnných, může pořadí proměnných ovlivnit velikost prohledávaného prostoru
- úspěšné problémově nezávislé heuristiky

včasné detekci slepých větví

- odvozením dodatečné informace

využitím struktury problému

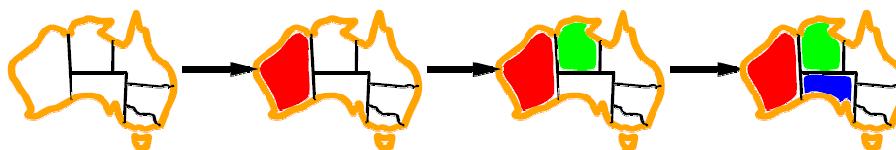
- některé problémy (např. CSP se stromovým grafem podmínek) lze vyřešit bez navracení

Umělá inteligence I, Roman Barták



Nejvíce omezená proměnná

- tj. proměnná s nejmenším počtem možných akcí
- tj. proměnná s nejmenší doménou
- tzv. **dom heuristika**



Proměnná s nejvíce podmínkami s volnými proměnnými

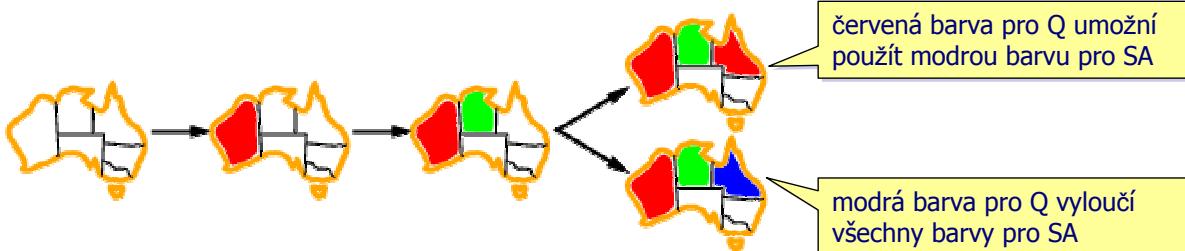
- tzv. **deg heuristika**
- používá se pro dodatečný výběr pokud dom heuristika neurčí jedinou proměnnou (**dom+deg heuristika**)



V obou případech se jedná o instanci **pravidla prvního neúspěchu** (fail-first), které doporučuje vybrat proměnnou, jejíž ohodnocení nejspíše povede k neúspěchu.



- Pří výběru hodnoty naopak preferujeme hodnotu, která nejspíše patří do řešení – **pravidlo prvního úspěchu** (succeed-first).
- Jak ji ale obecně poznáme?
 - například hodnota, která **nejméně omezí zbývající volné proměnné** (nechá v problému největší flexibilitu)



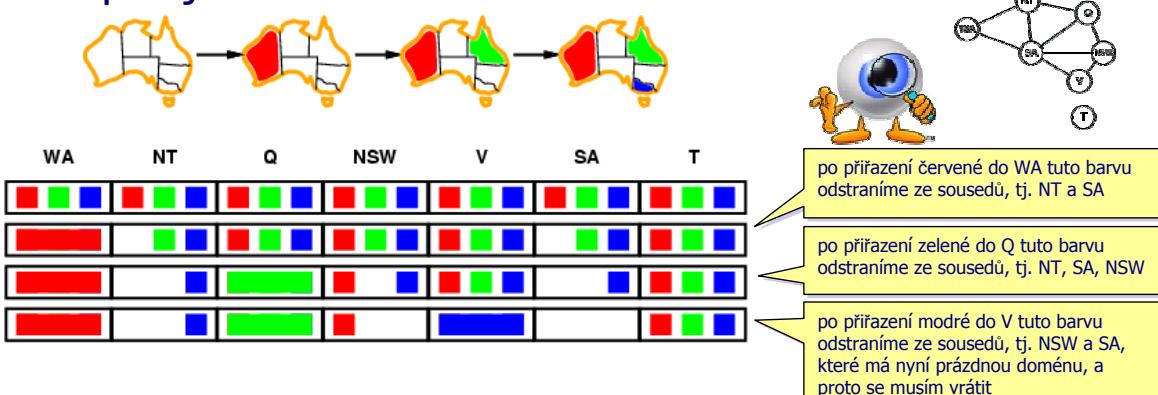
- „přesnější“ hodnotu lze najít například zjištěním počtu řešení relaxovaného problému, kde je daná hodnota použita
- výpočtově náročné, proto se často používají problémově závislé heuristiky

Umělá inteligence I, Roman Barták



- Nemohli bychom „uhodnout“ dopředu, že daná cesta nevede do cíle?

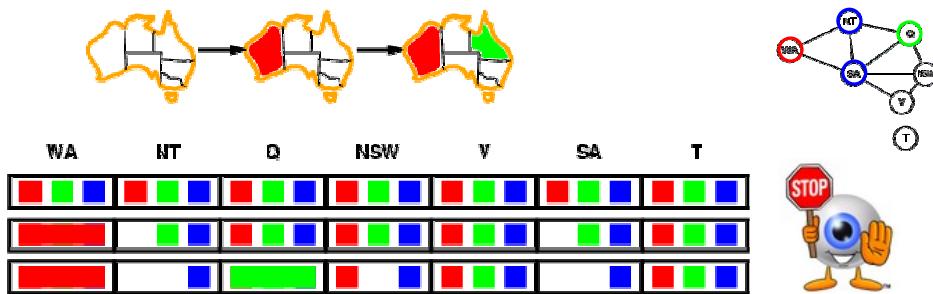
- po přiřazení hodnoty můžeme dopředu zkontovalovat podmínky, které vedou z aktuální proměnné do volných proměnných – **kontrola dopředu**
- zkontovalovat = vyřadit hodnoty, které podmínu nesplňují



Umělá inteligence I, Roman Barták

Propagace podmínek

- Nemohli bychom z podmínek odvodit ještě více informací?

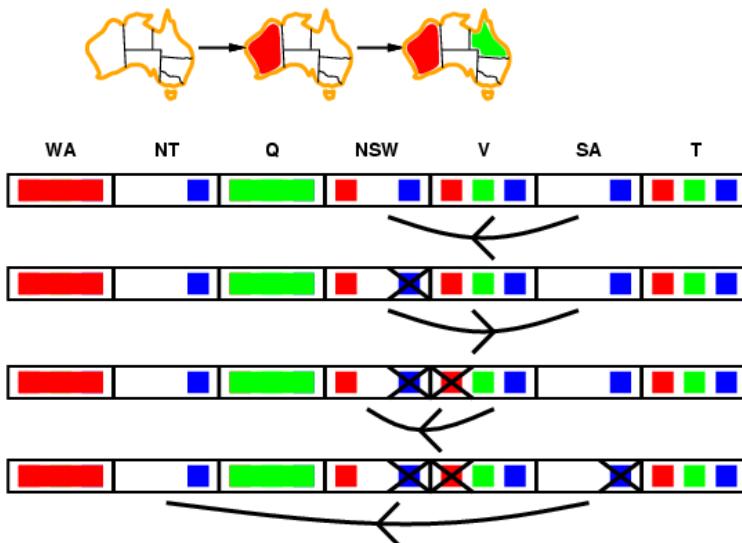


- pokud bychom se při kontrole dopředu podívali i na podmínky mezi dosud neohodnocenými proměnnými, zjistíme, že sousedé NT a SA nemohou být najednou modré, což jsou jediné hodnoty v jejich doménách
- protože se přiřazená hodnota propaguje přes všechny podmínky do dalších proměnných, hovoříme o **propagaci podmínek** nebo také o **pohledu dopředu** (look ahead)
- realizuje se metodou udržování **konzistence podmínek**

Umělá inteligence I, Roman Barták

Hranová konzistence

- každá podmínka odfiltrujte z domén „svých“ proměnných hodnoty, které ji nesplňují
- často se realizuje směrově odstraněním hodnot z domény proměnné X, které nemají podporu v doméně proměnné Y z (binární) podmínky (X,Y) a naopak



- filtraci je třeba zavolat kdykoliv se změní doména proměnné Y
- filtrace se tak opakuje, dokud se zmenšují domény až do dosažení pevného bodu nebo vyprázdnění nějaké domény

Umělá inteligence I, Roman Barták



Algoritmus AC-3

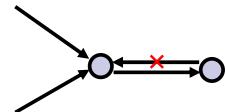
```
function AC-3(csp) returns the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables  $\{X_1, X_2, \dots, X_n\}$ 
  local variables: queue, a queue of arcs, initially all the arcs in csp
```

Algoritmus lze volat inkrementálně v rámci prohledávání - pokud se instanciovala proměnná X_i , potom se fronta inicializuje všemi hranami (podmínkami), které do ní vedou.

```
  while queue is not empty do
     $(X_i, X_j) \leftarrow \text{REMOVE-FIRST}(\text{queue})$ 
    if RM-INCONSISTENT-VALUES( $X_i, X_j$ ) then
      for each  $X_k$  in NEIGHBORS[ $X_i$ ] do
        add  $(X_k, X_i)$  to queue
```

Pokud se změnila doména proměnné X_i , potom je potřeba zkontrolovat všechny hrany (podmínky), které do ní vedou s výjimkou X_j .

```
function RM-INCONSISTENT-VALUES( $X_i, X_j$ ) returns true iff remove a value
  removed  $\leftarrow$  false
  for each  $x$  in DOMAIN[ $X_i$ ] do
    if no value  $y$  in DOMAIN[ $X_j$ ] allows  $(x, y)$  to satisfy constraint( $X_i, X_j$ )
      then delete  $x$  from DOMAIN[ $X_i$ ]; removed  $\leftarrow$  true
  return removed
```



Filtrace domény proměnné X_i odstraní hodnoty, které nemají kompatibilní hodnotu v proměnné X_j a zároveň oznámí, zda se něco smazalo.
Pokud známe sémantiku podmínky, nemusíme kontrolovat hodnotu po hodnotě (např. u $X < Y$)

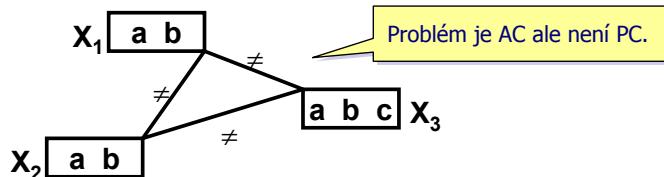
Časová složitost algoritmu $O(ed^3)$, kde je počet podmínek a d velikost domény, není optimální, protože se opakovaně testují stejné dvojice hodnot. Pamatováním si výsledků testů lze udělat optimálně v $O(ed^2)$, AC-4, AC-3.1, AC-2001

Umělá inteligence I, Roman Barták



Silnější konzistence

- Obecně lze definovat **k-konzistenci**, jako zajištění, že pro konzistentní hodnoty ($k-1$) různých proměnných lze najít konzistentní hodnotu libovolné další proměnné
 - hranová konzistence (AC) = 2-konzistence
 - konzistence po cestě (PC) = 3-konzistence



- Je-li problém i-konzistentní $\forall i=1,\dots,n$ (n je počet proměnných), potom umíme řešit bez navracení.
 - v DFS vždy najdeme hodnotu další proměnné, která je kompatibilní s i předchůdci
- Bohužel, složitost k-konzistence je exponenciální v k.

Umělá inteligence I, Roman Barták

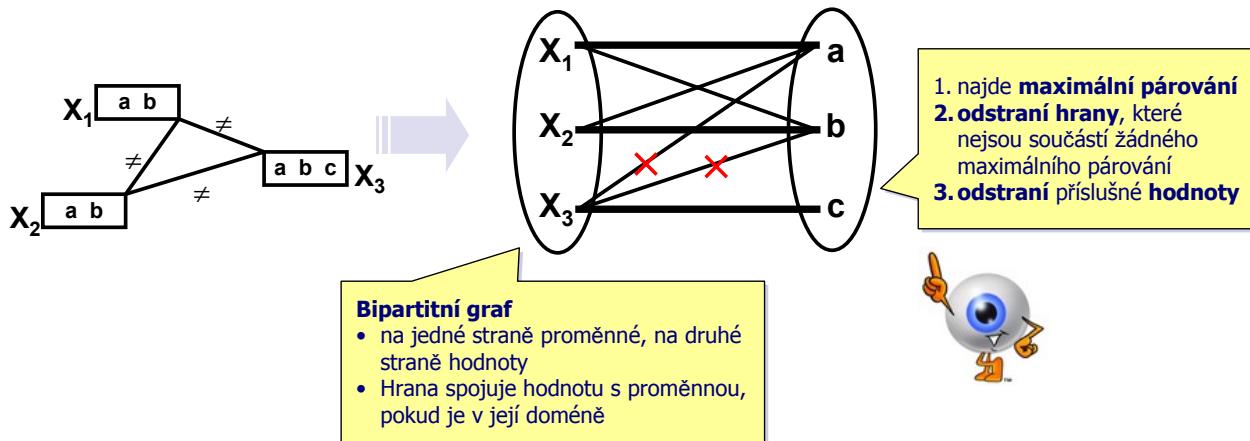
Globální podmínky

- Místo silnějších typů konzistencí se v praxi spíše používají tzv. **globální podmínky**, které v sobě sdružují několik jednoduchých podmínek a speciálním filtračním algoritmem zajišťují pro tuto množinu podmínek globální konzistenci.

Příklad:

podmínka **all_different({X₁,..., X_k}**)

- nahrazuje binární nerovnosti $X_1 \neq X_2, X_1 \neq X_3, \dots, X_{k-1} \neq X_k$
- **all_different({X₁,..., X_k}**) = { $(d_1, \dots, d_k) \mid \forall i \in D_i \text{ & } \forall i \neq j \text{ } d_i \neq d_j$ }
- filtrační algoritmus založen na párování v bipartitních grafech



Umělá inteligence I, Roman Barták

Finální poznámky

- **Deklarativní přístup** k řešení problémů
 - vytvoříme **model** (proměnné, domény, podmínky)
 - použijeme **obecný řešič** podmínek
- Možná rozšíření
 - **optimalizační problémy**
 - řešeno metodou větví a mezi (branch-and-bound)
 - **měkké podmínky**
 - podmínka určuje pouze preferenci, kterou je dobré respektovat
 - Řeší se podobně jako optimalizační problémy
- Jiné řešící techniky
 - **lokální prohledávání** (na cestě nezáleží)
 - celočíselné programování

Umělá inteligence I, Roman Barták

Oblasti aplikací



Bioinformatika

- sekvencování DNA
- hledání 3D struktury proteinů



Plánování

- autonomní plánování operací kosmických lodí (Deep Space 1)



Rozvrhování výroby

- úspory po aplikaci CSP: denně US\$ 0.2-1 milión

Umělá inteligence I, Roman Barták

Pro zájemce

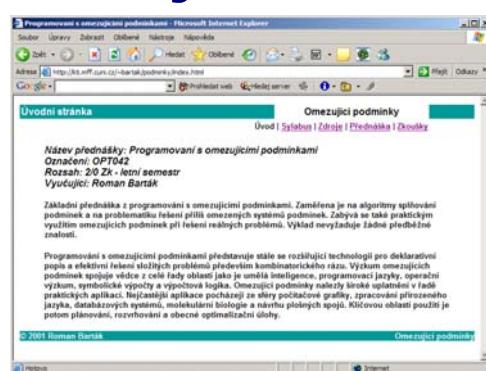
■ Systémy pro řešení podmínek

- SICStus Prolog (v našich laboratořích)
- ECLiPSe (Open Source, <http://eclipse.crosscoreop.com/>)
- GECODE (Open Source C++, <http://www.gecode.org/>)
- ...

■ Přednáška Programování s omezujícími podmínkami

□ LS 2/0 Zk

- <http://kti.mff.cuni.cz/~bartak/podminky/>



Umělá inteligence I, Roman Barták



Umělá inteligence I

Roman Barták, KTI ML

roman.bartak@mff.cuni.cz

<http://ktiml.mff.cuni.cz/~bartak>



Dnes

- Dosud popisované algoritmy nepředpokládaly přítomnost dalších agentů v prostředí, zvlášť ne agentů, kteří hrají proti nám.
- Dnes se podíváme na **hraní her**, jako na příklad **kompetitivního multi-agentního prostředí**
 - hry dvou hráčů s nulovým součtem a úplnou informací (piškvorky, šachy)
 - optimální (dokonalé) strategie (minimax, alfa-beta)
 - „nedokonalé“ strategie („uříznuté“ prohledávání)
 - hry s náhodným vstupem (vrhcáby – backgammon)

- **Matematická teorie her** (oblast ekonomie) vidí multi-agentní prostředí jako hru, kde má každý agent důležitý vliv na ostatní agenty.

je-li agentů moc, hovoříme o ekonomice (spíše než o hře)

- Umělá inteligence se zpravidla omezuje na **hry dvou hráčů**, kteří se **střídají** a mají **nulový součet** (zisk jednoho znamená ztrátu druhého).

deterministické hry vs. náhoda

úplná vs. částečná informace

Proč hry a UI? Protože hry jsou

- těžké
- dobře reprezentovatelné (a agenti mají málo akcí)
- zábavné



perfect information	deterministic
chess, checkers, go, othello	backgammon, monopoly
	bridge, poker, scrabble, nuclear war

imperfect Information

deterministic	chance
chess, checkers, go, othello	backgammon, monopoly
	bridge, poker, scrabble, nuclear war

Umělá inteligence I, Roman Barták

Základní nastavení

- Máme **dva hráče MAX a MIN**

MAX začíná a potom se hráči střídají dokud hra neskončí

hledáme strategii pro hráče MAX

- Opět se na hraní her podíváme jako na prohledávání:

počáteční stav: stav „desky“ a kdo je na tahu

funkce následníka: seznam dvojic (tah, stav)

- počáteční stav a funkce následníka určují **strom hry**

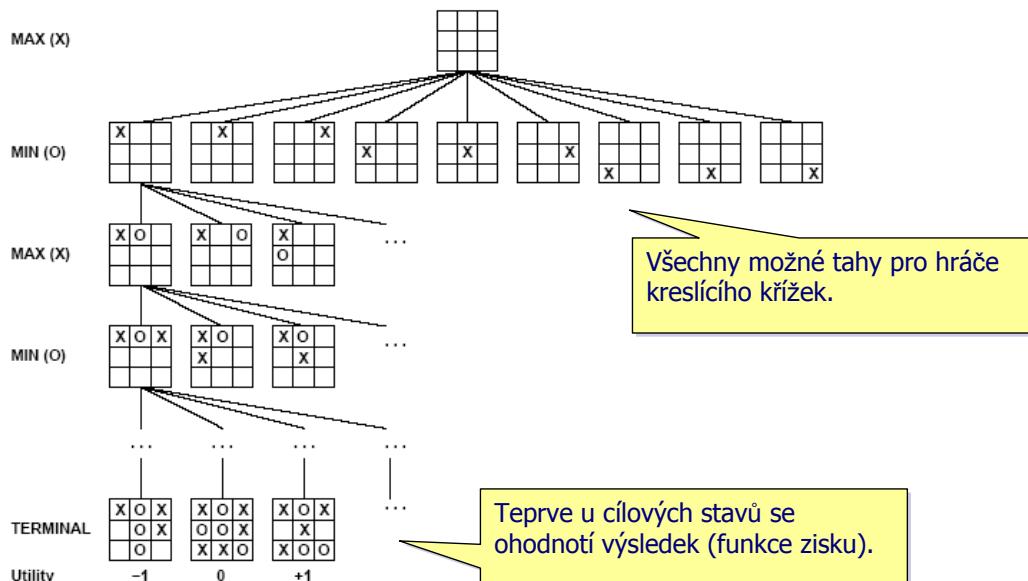
test ukončení: kdy hra končí (cílový stav)

funkce zisku (utility): numerické ohodnocení koncového stavu (výhra, remíza, prohra je +1, 0, -1)

- vysoké hodnoty jsou dobré pro MAXe, nízké naopak pro MINa



- Dva hráči střídavě kreslí křížky a kolečka dokud jeden nemá řadu tří svých symbolů nebo je vše obsazeno.



Umělá inteligence I, Roman Barták



Optimální strategie

- Při klasickém prohledávání **hledáme nějakou (nejkratší) cestu do cíle.**
- U her také **hledáme cestu do koncového stavu s největším ziskem**, ale brání nám protivník MIN (určuje každý druhý krok na cestě), který hledá přesně opačné koncové stavy.
- MAX proto musí najít **podmíněnou strategii**, která určuje
 - počáteční tah
 - tah jako odpověď na každý tah MINa
 - **optimální strategie** dá takový výsledek jako libovolná jiná strategie, která je hrána proti neomylnému protivníkovi

Umělá inteligence I, Roman Barták

Hodnota minimax

- Optimální strategii budeme hledat pomocí hodnoty minimax vypočtené v každém uzlu stromu hry takto:

$\text{MINIMAX-VALUE}(n) =$

$\text{UTILITY}(n)$

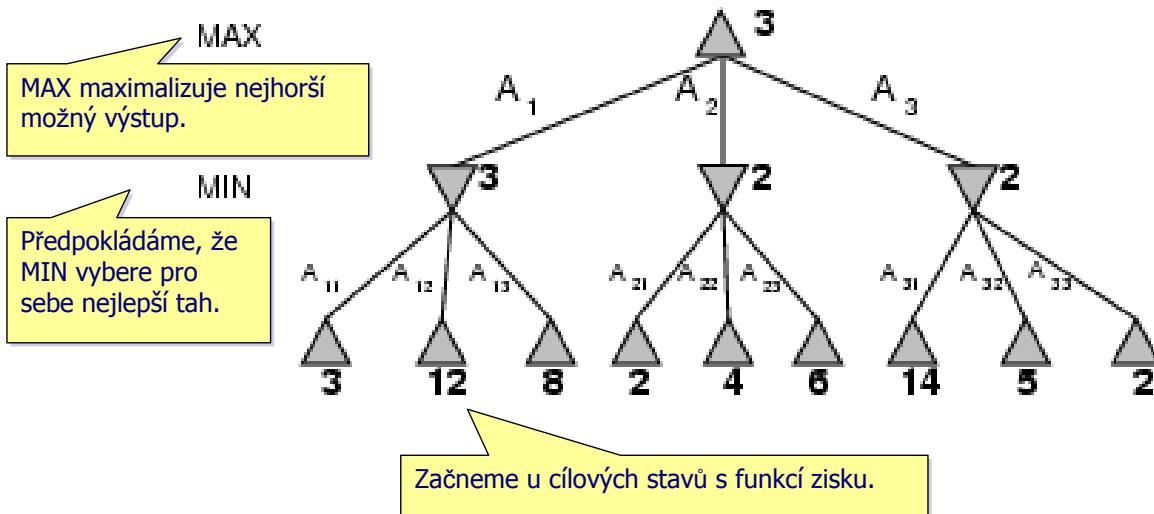
$\max_{s \in \text{successors}(n)} \text{MINIMAX-VALUE}(s)$

$\min_{s \in \text{successors}(n)} \text{MINIMAX-VALUE}(s)$

pokud je n cílový

pokud v n hraje MAX

pokud v n hraje MIN



Umělá inteligence I, Roman Barták

Algoritmus minimax

```
function MINIMAX-DECISION(state) returns an action
```

```
    v ← MAX-VALUE(state)
```

```
    return the action in SUCCESSORS(state) with value v
```

```
function MAX-VALUE(state) returns a utility value
```

```
if TERMINAL-TEST(state) then return UTILITY(state)
```

```
v ← -∞
```

```
for a, s in SUCCESSORS(state) do
```

```
    v ← MAX(v, MIN-VALUE(s))
```

```
return v
```

```
function MIN-VALUE(state) returns a utility value
```

```
if TERMINAL-TEST(state) then return UTILITY(state)
```

```
v ← ∞
```

```
for a, s in SUCCESSORS(state) do
```

```
    v ← MIN(v, MAX-VALUE(s))
```

```
return v
```

Algoritmus předpokládá, že protihráč hraje optimálně. Pokud ne, je zisk ještě větší.

•Časová složitost $O(b^m)$
•Prostorová složitost $O(bm)$

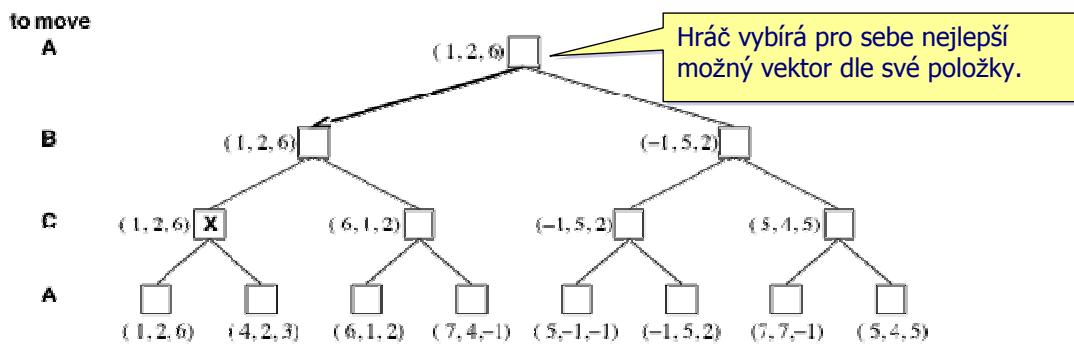
(b je počet akcí na uzel, m je délka hry)

Umělá inteligence I, Roman Barták



Minimax a více hráčů

- Pro hru více hráčů použijeme **vektor hodnot** a každý hráč (na svém tahu) optimalizuje svoji položku.



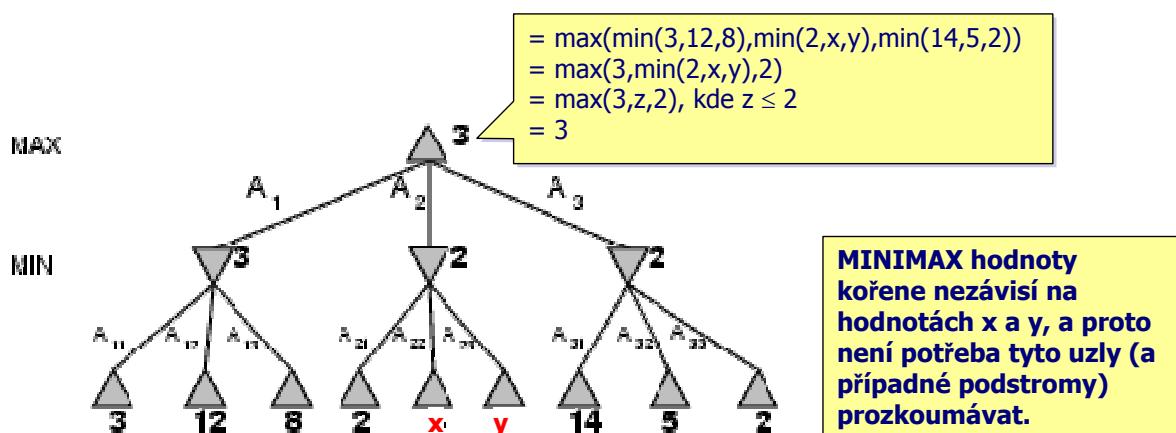
- Hry více hráčů přinášejí další možnosti jako je třeba **vytváření aliancí**.
 - Aliance se zdají být přirozeným důsledkem optimálních strategií každého hráče.
 - Jsou-li A a B slabí a C silný, je pro A i B zpravidla optimální neútočit proti sobě, ale zaměřit se na C.
Je-li potom C oslaben, aliance se často rozpadá.

Umělá inteligence I, Roman Barták



Lepší minimax?

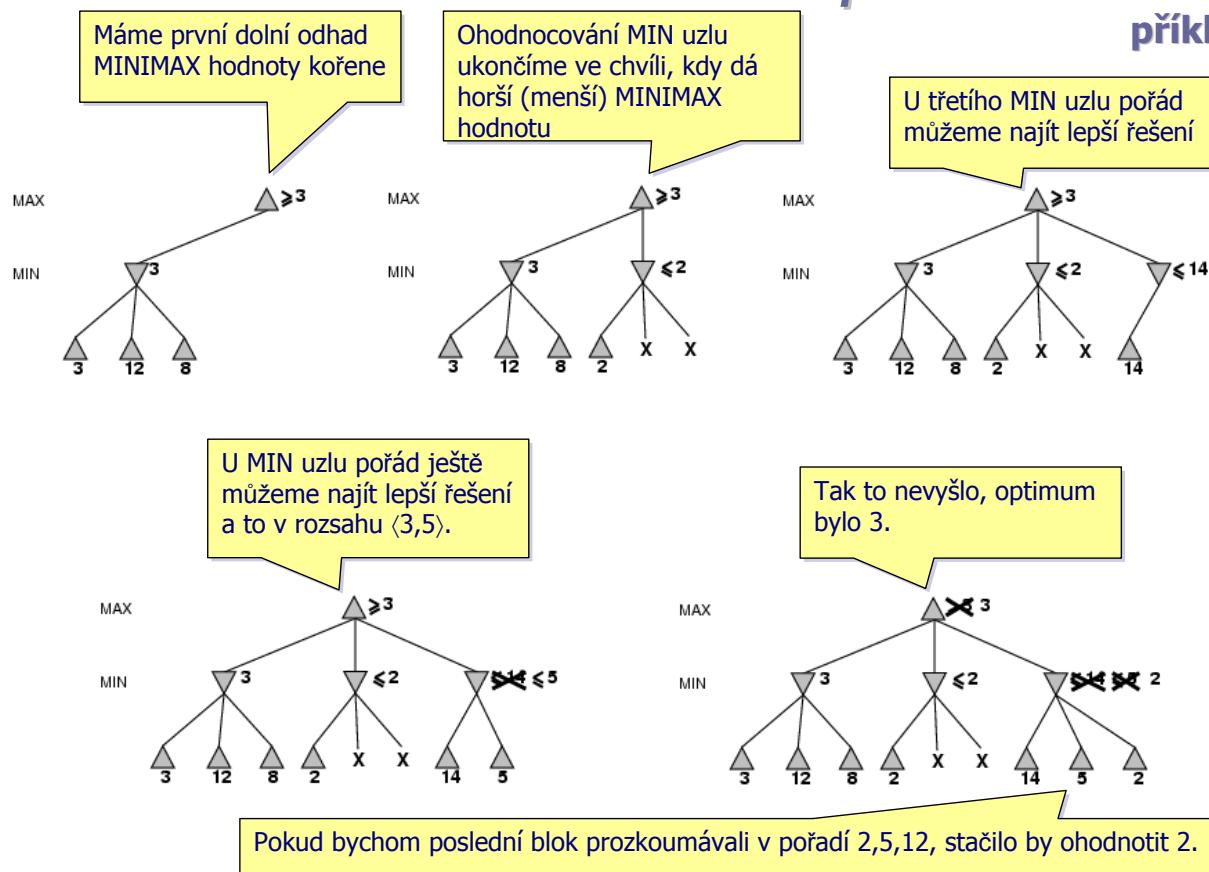
- Algoritmus minimax vždy najde optimální strategii, ale musí prozkoumat celý strom hry (všechny možné partie).
- Nešlo by dosáhnout stejného výsledku rychleji?
 - ANO!
Nemusíme prozkoumávat všechny uzly, pokud jsou „špatné“.
 - Tzv. α - β ořezávání eliminuje podstromy, kam se při hraní nedostaneme, protože tam nevede optimální strategie.



Umělá inteligence I, Roman Barták

α - β ořezávání

příklad



Umělá inteligence I, Roman Barták

Algoritmus α - β

```

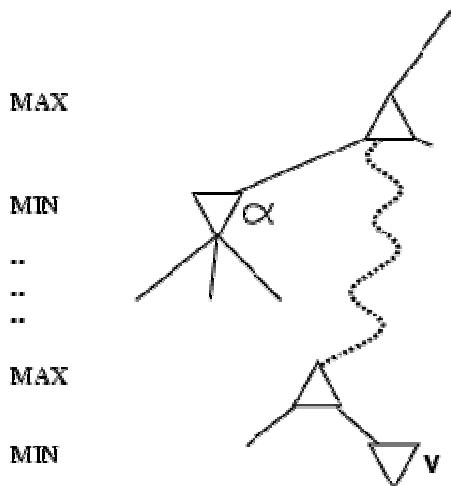
function ALPHA-BETA-SEARCH(state) returns an action
    inputs: state, current state in game
    v  $\leftarrow$  MAX-VALUE(state,  $-\infty$ ,  $+\infty$ )
    return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
    inputs: state, current state in game
         $\alpha$ , the value of the best alternative for MAX along the path to state
         $\beta$ , the value of the best alternative for MIN along the path to state
    if TERMINAL-TEST(state) then return UTILITY(state)
    v  $\leftarrow -\infty$ 
    for a,s in SUCCESSORS(state) do
        v  $\leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta))$ 
        if v  $\geq \beta$  then return v
         $\alpha \leftarrow \text{MAX}(\alpha, v)$ 
    return v

function MIN-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
    inputs: state, current state in game
         $\alpha$ , the value of the best alternative for MAX along the path to state
         $\beta$ , the value of the best alternative for MIN along the path to state
    if TERMINAL-TEST(state) then return UTILITY(state)
    v  $\leftarrow +\infty$ 
    for a,s in SUCCESSORS(state) do
        v  $\leftarrow \text{MIN}(v, \text{MAX-VALUE}(s, \alpha, \beta))$ 
        if v  $\leq \alpha$  then return v
         $\beta \leftarrow \text{MIN}(\beta, v)$ 
    return v

```

Umělá inteligence I, Roman Barták



- α je hodnota nejlepší volby (tj. ta největší), kterou jsme dosud nalezli pro hráče MAX
 - pokud α není horší (není menší) než v , nebude MAX nikde hrát směrem k v , proto není potřeba strom pod v prozkoumávat
- β je hodnota nejlepší volby (tj. ta nejmenší), kterou jsme dosud nalezli pro hráče MIN
 - podobně můžeme odříznout podstromy hráče MIN

Vlastnosti:

- Oříznutím nepřijdeme o optimální řešení.
- Při „perfektním uspořádání“ srazíme časovou složitost na $O(b^{m/2})$, což dává větvící faktor \sqrt{b} (minimax má b), takže lze prozkoumat uzly do dvojnásobné hloubky.

Umělá inteligence I, Roman Barták

„Nedokonalé“ strategie

- Metody minimax i α-β musí **prohledat strom hry až do listů pro** získání dokonalé strategie.
 - Není praktické při větší hloubce (hloubka = počet tahů pro dohrání hry).
- Můžeme jednoduše **prohledávání větve ukončit dříve** a použít heuristický odhad hodnoty MINIMAX.
 - negarantuje nalezení optimální strategie, ale
 - dokončí prohledávání v požadovaném čase
- **Realizace:**
 - test ukončení → test přerušení (cutoff)
 - funkce zisku → heuristická evaluační funkce EVAL

Umělá inteligence I, Roman Barták



Evaluacní funkce

- Poskytuje odhad zisku v daném podstomě (podobně jako funkce h v prohledávání).
- Kvalita výsledného algoritmu silně závisí na kvalitě heuristické evaluační funkce.

Vlastnosti:

- koncové uzly musí uspořádat stejně jako funkce zisku
- výpočet nesmí trvat moc dlouho
- pro nekoncové uzly by měla být silně svázaná se skutečnou šancí vyhrát
 - šance místo jistoty je zde z výpočtových důvodů (není čas prozkoumat vše)
- Jak takovou funkci ale najít?



Umělá inteligence I, Roman Barták



Evaluacní funkce

Očekávaná hodnota

příklady

- na základě vybraných vlastností, které umíme rychle rozpoznat, roztrídíme stavy do kategorií
- kategorie ohodnotíme podle podílu vítězných a prohrávajících stavů
 - $\text{EVAL} = (0.72 \times +1) + (0.20 \times -1) + (0.08 \times 0) = 0.52$

„Materiální“ hodnota

- odhadneme numerický příspěvek každé vlastnosti
 - šachy: pěšák = 1, střelec = kůň = 3, věž = 5, královna = 9
- tyto příspěvky zkombinujeme (např. vážený součet)
 - $\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$
 - Součet předpokládá nezávislost vlastností!
 - Možno použít nelineární kombinace.



Umělá inteligence I, Roman Barták

Problémy s ořezáním

- Situace se dramaticky změní uvažováním dalšího tahu za limitní hloubkou.

Pro obě šachovnice je stejná materiální hodnota (lepší pro černou), ale **vpravo vítězí bílý** sebráním královny bez náhrady.

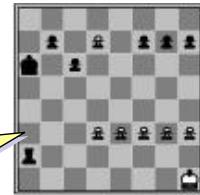


- **uklidnění** (quiescent): pokud zrovna protivník může brát, potom je odhad nestabilní a je dobré prozkoumat ještě pár tahů dopředu (například jen vybrané tahy)

■ Efekt horizontu

- špatnou situaci tak odsunout až za horizont, tj. není rozpoznána, ale stejně nastane.

Materiálně má více černý, ale pokud si bílý udělá z pěšáka královnu, tak **vyhraje bílý**. Černý může opakovaně dávat věží šach, takže několik tahů to nevypadá tak špatně.



Umělá inteligence I, Roman Barták

Možná vylepšení

■ Singulární prodloužení

- prozkoumáme sekvenci tahů, které jsou v dané pozici „jasně lepší“ než ostatní
- rychlý způsob jak prozkoumat oblast za limitní hloubkou (uklidnění je speciální případ)

■ Dopředné prořezání

- některé tahy v dané pozici nejsou vůbec uvažovány (lidský přístup)
- nebezpečné, protože může minout optimální strategii
- bezpečné, pokud jsou například tahy symetrické

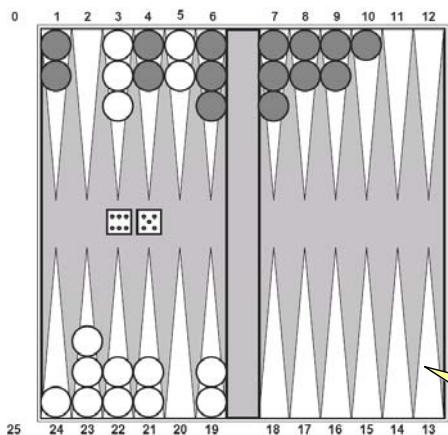
■ Transpoziční tabulky

- podobně jako u pohledávání je možné si pamatovat jednou ohodnocené stav, protože se do nich můžeme dostat jinou posloupností tahů

Umělá inteligence I, Roman Barták

- Reálný svět často obsahuje nepředvídatelné vnější události, které vedou k neočekávaným situacím.
- Hry simulují **nepředvídatelnost** zahrnutím prvku náhodnosti, jako je třeba hození kostkou.

Vrhcáby (backgammon)



- cílem je přesunout vlastní kameny z počáteční pozice do koncové
- vítězí ten, kdo to udělá první
- hod kostkami určí, jaké tahy jsou v dané situaci povolené
 - vzdálenost skoku

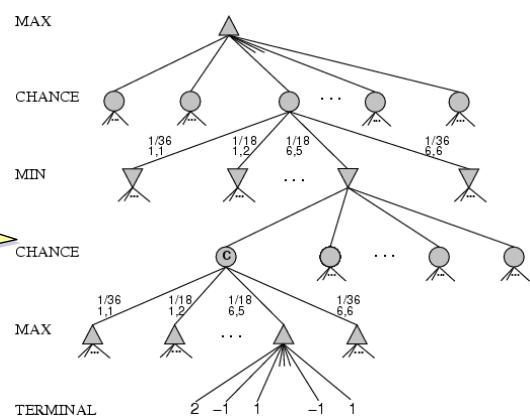
Bílý může udělat čtyři legální tahy:
(5-10,5-11), (5-11,19-24), (5-10,10-16), (5-11,11-16)

Umělá inteligence I, Roman Barták

Hry s náhodou

- Do stromu hry s MAX a MIN uzly přidáme **uzly náhody**, kde větve popisují všechny možné výsledky „hodu kostkou“.
 - 36 výsledků pro dvě kostky, 21 po odstranění symetrií (5-6 a 6-5)
 - šance na double je $1/36$, ostatní výsledky $1/18$

Uzly náhody jsou vloženy do každé vrstvy resp. všude, kde je tah ovlivněn náhodou.
V této vrstvě hází MAX.



- Místo MINIMAX hodnoty počítáme **očekávanou hodnotu** (dle rozložení šancí):

$$\text{EXPECTIMINIMAX-VALUE}(n) =$$

$$\text{UTILITY}(n)$$

$$\max_{s \in \text{successors}(n)} \text{MINIMAX-VALUE}(s)$$

$$\min_{s \in \text{successors}(n)} \text{MINIMAX-VALUE}(s)$$

$$\sum_{s \in \text{successors}(n)} P(s) \cdot \text{EXPECTIMINIMAX}(s)$$

pokud je n cílový

pokud v n hraje MAX

pokud v n hraje MIN

pokud je n uzel náhody

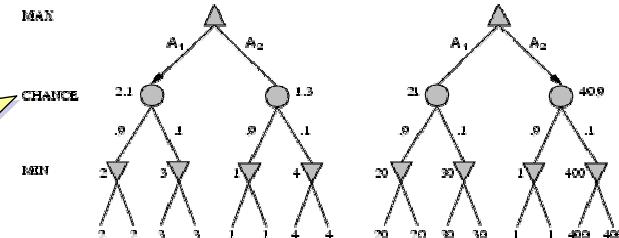


Umělá inteligence I, Roman Barták

■ Pozor na evaluační funkci (při cutoff)

- roli nehráje jen pořadí uzlů, ale i absolutní hodnoty
- měly by být lineární transformací očekávaného zisku v uzlu

Vlevo vychází lépe A_1 a vpravo A_2 ,
přestože evaluační hodnoty uspořádají
listy stejně.

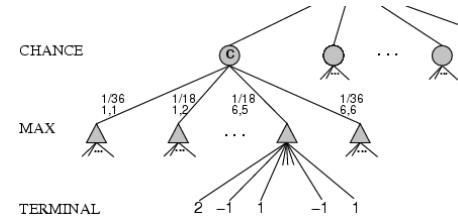


■ Časová složitost $O(b^m n^m)$, kde je n počet různých náhodných výsledků

- není realistické dostat se do větší hloubky, zvlášť pro větší náhodné větvení

■ Použití ořezání à la α - β

- můžeme ořezat i u uzlů náhody, pokud máme omezenou funkci zisku
- pro výpočet očekávané hodnoty použijeme meze tam, kde hodnotu ještě nemáme



Umělá inteligence I, Roman Barták

Karty



- Na první pohled vypadají karty jako hra s náhodou, ale kostky jsou zde vrženy hned na začátku!
- Zajímavé na kartách je to, že (někdy) nevidíme všechny karty protihráče (tj. máme částečnou informaci).

Příklad: karetní hra „větší bere“ s otevřenými kartami

Situace 1: MAX: ♡6 ♦6 ♣9 8 MIN: ♡4 ♠2 ♣10 5

1. MAX dá ♣9, MIN potvrdí barvu ♣10 vítěz MIN
 2. MIN dá ♠2, MAX dá ♦6 vítěz MIN
 3. MAX dá ♡6, MIN potvrdí barvu ♡4 vítěz MAX
 4. MIN dá ♣5, MAX potvrdí barvu ♣8 vítěz MAX
- ♣9 je pro MAXe optimální první volba

Situace 2: MAX: ♡6 ♦6 ♣9 8 MIN: ♦4 ♠2 ♣10 5

- symetrický případ, ♣9 je opět pro MAXe optimální první volba

Situace 3: MIN skryl první kartu ($\heartsuit 4$ nebo $\diamondsuit 4$), jaká je optimální první volba pro MAX?

- Nezávisle na $\heartsuit 4$ nebo $\diamondsuit 4$ byl v předchozích situacích optimální první tah ♣9, tedy i teď je ♣9 optimální první tah.
- Opravdu?

Umělá inteligence I, Roman Barták



Neúplná informace



Příklad: cesta k bohatství (aneb karty jinak)

- **Situace 1:** Cesta A vede k hromadě zlata a cesta B vede na rozcestí. Vydejte se doleva a narazíte na mohylu drahokamů, ale vydejte se doprava a přejede Vás autobus. Kam půjdete (když drahokamy jsou cennější než zlato)?
 - B a doleva
- **Situace 2:** Cesta A vede k hromadě zlata a cesta B vede na rozcestí. Vydejte se doprava a narazíte na mohylu drahokamů, ale vydejte se doleva a přejede Vás autobus. Kam půjdete?
 - B a doprava
- **Situace 3:** Cesta A vede k hromadě zlata a cesta B vede na rozcestí. Vydejte se správně a narazíte na mohylu drahokamů, ale vydejte se špatně a přejede Vás autobus. Kam půjdete?
 - rozumný člověk (ten kdo neriskuje ;-) jede A
- To je přesně stejný případ jako u příkladu s kartami. Na rozcestí B neznáme, která ze situací 1 a 2 nastává, stejně jako u karet nakonec také nevíme, zda MIN drží ♥4 nebo ♦4, takže je 50% šance neúspěchu.
- **Poučení:** Je potřeba uvažovat informaci, jakou budeme mít v každém stavu hry (chyba volby ♣9 je v tom, že MAX uvažoval jakoby v každém kroku viděl všechny karty).

Umělá inteligence I, Roman Barták



Hry – jak si stojí UI?

■ Šachy

- 1997 superpočítač Deep Blue porází Kasparova 3.5 – 2.5
- 2002 „běžné“ PC (program FRITZ) remízuje s Kramníkem

■ Dáma

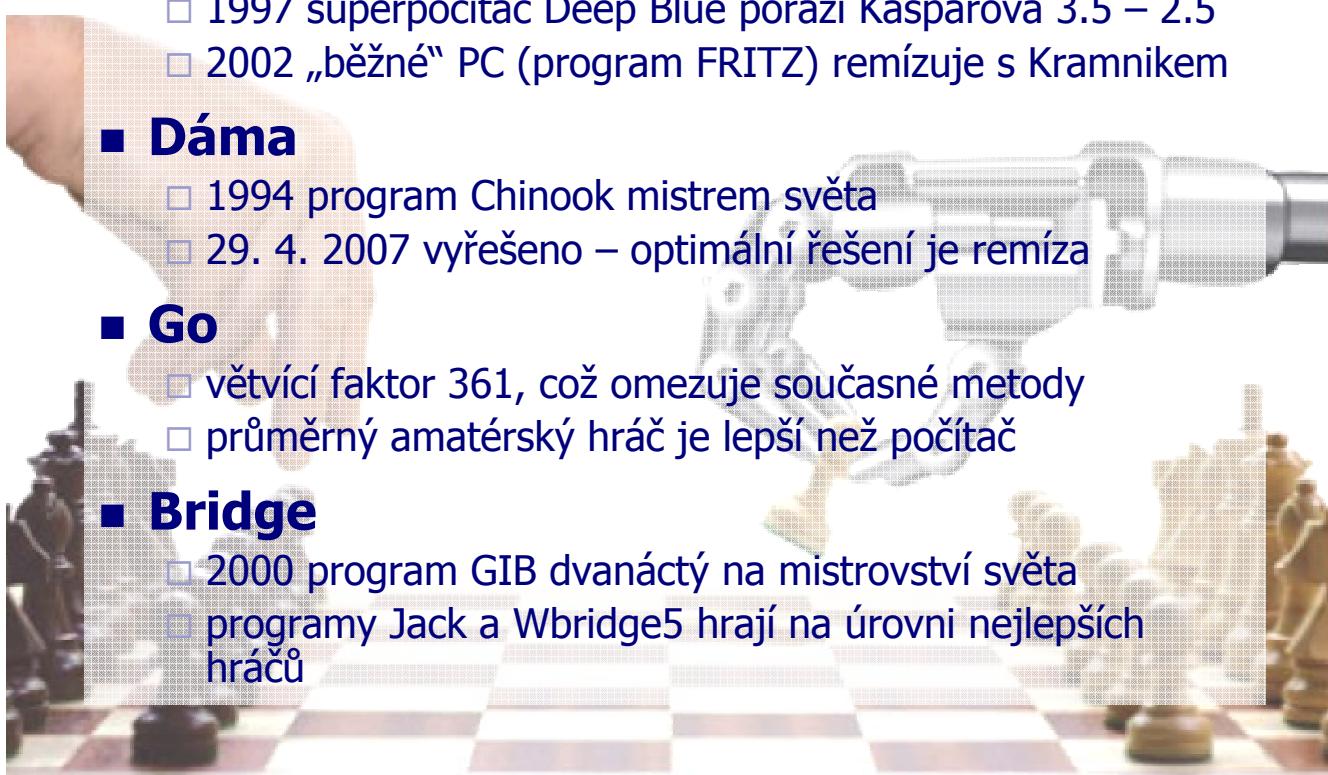
- 1994 program Chinook mistrem světa
- 29. 4. 2007 vyřešeno – optimální řešení je remíza

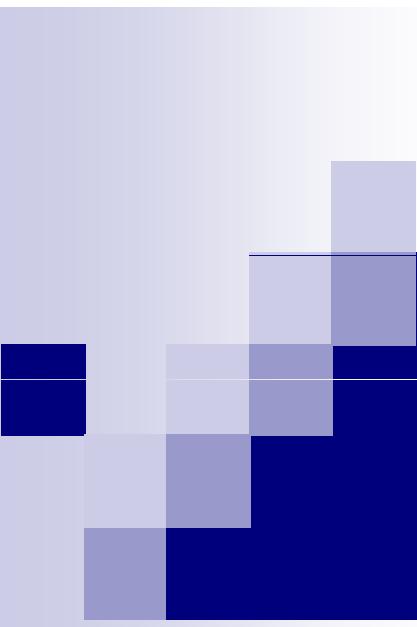
■ Go

- větvící faktor 361, což omezuje současné metody
- průměrný amatérský hráč je lepší než počítač

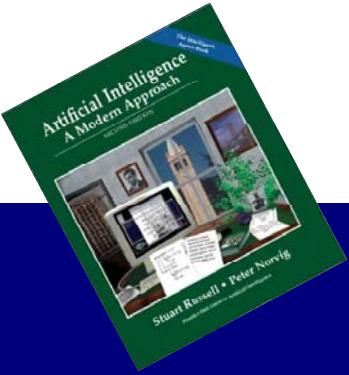
■ Bridge

- 2000 program GIB dvanáctý na mistrovství světa
- programy Jack a Wbridge5 hrají na úrovni nejlepších hráčů





Umělá inteligence I



Roman Barták, KTI ML

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



8



Dnes

- Dnes se podíváme na agenty, kteří si vytvářejí **reprezentaci** světa, **odvozují** z ní novou informaci, kterou potom použijí k rozhodnutí, co dělat dál.
- Řeč budu o **logických (znalostních) agentech**, kteří kombinují obecné znalosti s aktuálním pozorováním světa, aby odhalili skryté aspekty současné stavu a použili je při výběru další akce.
- K tomu potřebujeme znát:
 - **jak reprezentovat znalosti**
 - jak přivést znalosti k životu pomocí **uvažování**



Znalostní agent

- Znalostní agent používá **bázi znalostí** – množina vět v daném jazyce – do které přidává další znalosti operací TELL a získává z ní informace operací ASK.
- Odpovědi na operace ASK jsou **odvozeny** od toho, co bylo do báze znalostí uloženo operacemi TELL.
 - odvození může vytvářet nové věty, které nebyly do báze znalostí explicitně zadány (tím se báze znalostí liší od databáze)

```
function KB-AGENT( percept) returns an action
  static: KB, a knowledge base
           t, a counter, initially 0, indicating time
  TELL(KB, MAKE-PERCEPT-SENTENCE( percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

do báze znalostí ukládáme informace o pozorování světa i o vlastních akcích

odvozování nám umožní vybrat nejvhodnější akci i v případě neúplné informace o světě

Umělá inteligence I, Roman Barták

Svět Wumpus

- Jeskynní bludiště, kde žije zapáchající **příšera** Wampus, která každého sežere, jsou tam hluboké **díry**, kde každý uvízne, a jedna hrouda **zlata**, kterou hledáme.

4	Stench		Breeze	PIT
3	Wumpus	Breeze Stench Gold	PIT	Breeze
2	Stench		Breeze	
1	START	Breeze	PIT	Breeze
	1 2 3 4			

- Wampus je cítit v nejbližších okolních „místnostech“ (ne po diagonále).
- Díry se projevují vánkem v nejbližších okolních místnostech.
- Zlato se třptytí v místnosti, kde se nachází.
- Přes stěny nelze procházet a náraz je cítit.
- Wumpuse lze zastřelit lukem, ale k dispozici je pouze jeden šíp (proletí všechny místnosti ve směru pohledu agenta až ke stěně nebo k zásahu Wumpuse).
 - Zašažený Wampus vydá skřek, který je slyšet všude, a pak hned zemře.
 - Mrtvý Wampus stále zapáchá, ale už nikoho nesežere.



Umělá inteligence I, Roman Barták



Svět Wumpus

z pohledu agenta

■ Výkon

- +1000 bodů za sebrané zlato
- 1000 bodů za spadnutí do jámy nebo sežrání Wumpusem
- 1 bod za provedení libovolné akce
- 10 bodů za vystřelený šíp

■ Prostředí

- 4 × 4 místnosti, agent začíná na [1,1] pohledem doprava

■ Akční členy

- krok dopředu, otočení o 90° vlevo/vpravo,
vystřelení, uchopení

■ Senzory

- západ ano/ne, vánek ano/ne, třpyt ano/ne,
náraz ano/ne, skřek ano/ne



Umělá inteligence I, Roman Barták



Svět Wumpus

z pohledu prostředí

■ Plná pozorovatelnost?

- NE, vidíme pouze své okolí (částečná pozorovatelnost)

■ Deterministické?

- ANO, výsledek akce je jasně daný

■ Episodické?

- NE, záleží na pořadí akcí (sekvenční)

■ Statické?

- ANO, Wumpus ani díry se nehýbají

■ Diskrétní?

- ANO

■ Jeden agent?

- ANO, Wumpus se (zatím) nechová jako agent,
ale jako vlastnost prostředí



Umělá inteligence I, Roman Barták

Svět Wumpus

hledáme zlato



nic necítím, nikde nefouká ⇒ jsem v pohodě a můžu jít kam chci

1.

1,4	2,4	3,4	4,4	
1,3	2,3	3,3	4,3	
1,2	2,2	3,2	4,2	
OK				
1,1	A OK	2,1 OK	3,1 OK	4,1

A = Agent
B = Breeze
G = Glitter, Gold
OK = Safe square
P = Pit
S = Stench
V = Visited
W = Wumpus



tady nějak fouká ⇒ někde tady bude díra, raději se vrátím

2.

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2 P?	3,2 OK	4,2
OK			
1,1	V OK	A B OK	3,1 P? 4,1

tady se to pěkně třptytí ⇒ jsem za vodou ☺



tady něco smrdí ⇒ někde bude Wumpus

3.

1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 A S OK	2,2 OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P! OK	4,1

na [1,1] to nebude, tam už jsem byl

na [2,2] to také nebude, to bych ho cítil, když jsem byl na [2,1]

Wumpus je na [1,3]

necítím vítr ⇒ [2,2] bude bezpečná, jdu tam (díra je na [3,1])



5.

1,4	2,4 P?	3,4	4,4
1,3 W!	2,3 A S G B	3,3 P?	4,3
1,2 S V OK	2,2 V OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

Umělá inteligence I, Roman Barták

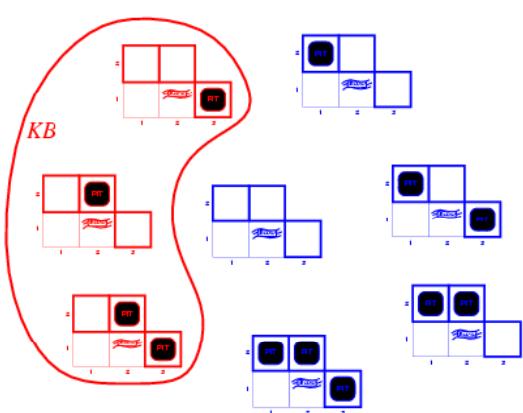


Svět Wumpus

modely báze znalostí

- Uvažujme situaci, kdy jsme na políčku [1,1] nic nedetekovali, přešli jsme doprava na [2,1] a tam cítíme vánek.

?	?		
	B	A	?
A	→	A	



- Pokud uvažujeme pouze detekci děr, máme $8 (=2^3)$ možných modelů (stavů světa).

- Pouze tři modely odpovídají naší bázi znalostí, protože ostatní modely jsou ve sporu s pozorováními:

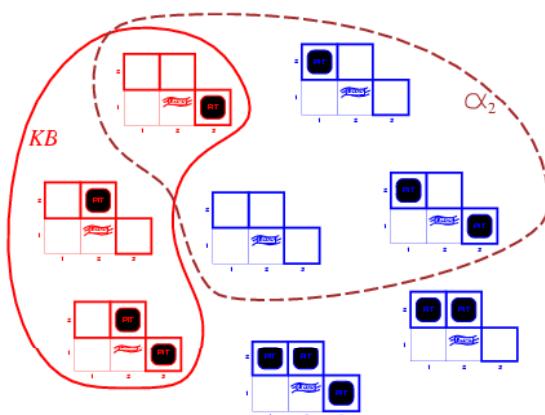
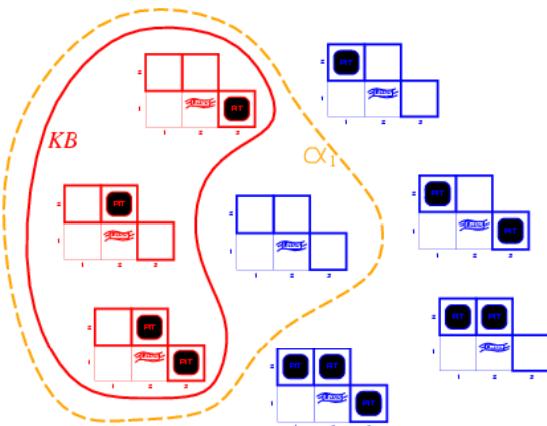
- žádná detekce na [1,1]
- vánek na [2,1]

Umělá inteligence I, Roman Barták

Svět Wumpus

důsledky báze znalostí

- zeptejme se nyní, zda je pozice [1,2] bezpečná
- platí, že $\alpha_1 = "[1,2]"$ je bezpečné je důsledkem naší báze znalostí?
- porovnáme modely pro KB a α_1
- každý model KB je modelem α_1 tj. α_1 je důsledkem KB
- a co bezpečnost pozice [2,2]?
- porovnáme modely pro KB a α_2
- některé modely pro KB nejsou modely pro α_2
- α_2 tedy není důsledkem KB a nemáte tak jistotu, že [2,2] je bezpečná pozice



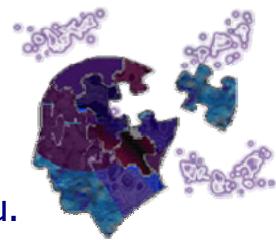
Umělá inteligence I, Roman Barták

Obecné odvozování

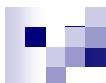
Jak budeme realizovat odvozování z báze znalostí obecně?

Použijeme **výrokovou logiku**,
tj. věty jsou výrokové formule a báze znalostí je konjunkcí těchto formulí.

- **Výrokové proměnné** popisují „mapu světa“
 - $P_{i,j} = \text{true}$ právě když je na pozici $[i, j]$ díra
 - $B_{i,j} = \text{true}$ právě když na pozici $[i, j]$ cítíme vánek
- **Výrokové formule** popisují
 - známé informace o světě
 - $\neg P_{1,1}$ na pozici $[1, 1]$ není díra (protože tam stojíme)
 - obecné znalosti o vlastnostech světa (např. pokud cítíme vánek, musí být na některé z okolních pozic díra)
 - $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
 - $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
 - ...
 - získaná pozorování
 - $\neg B_{1,1}$ na pozici $[1, 1]$ nefouká
 - $B_{2,1}$ na pozici $[2, 1]$ fouká
- Použijeme **odvozovací postupy** pro výrokovou logiku.



Umělá inteligence I, Roman Barták



Výroková logika

v kostce

■ Syntax definuje povolené věty.

- výroková proměnná (případně konstanty true a false) je (atomická) věta
- spojení vět pomocí logických spojek \neg , \wedge , \vee , \Rightarrow , \Leftrightarrow je (složená) věta

■ Sémantika definuje pravdivost každé věty vzhledem k libovolnému světu (modelu).

- **model** je přiřazení hodnot true/false do všech výrokových proměnných
- P je pravda v každém modelu obsahujícím $P=true$
- sémantika složených vět je určena pravdivostní tabulkou

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	false	true	false	false	true	true
false	true	true	false	true	true	false
true	false	false	false	true	false	false
true	true	false	true	true	true	true

Umělá inteligence I, Roman Barták



Výroková logika

důsledek a odvození

■ M je **modelem věty** α , pokud je α pravda v M.

- Množinu všech modelů pro α značíme $M(\alpha)$.

■ **logický důsledek: KB $\models \alpha$**

říká, že α logicky plyne z KB

- nastává právě tehdy, když $M(KB) \subseteq M(\alpha)$

■ Nás budou zajímat **odvozovací metody**, které hledají/ověřují důsledky KB.

- KB $\vdash_i \alpha$ říká, že algoritmus i odvozuje větu α z KB
- algoritmus je **korektní** pokud KB $\vdash_i \alpha$ implikuje KB $\models \alpha$
- algoritmus je **úplný** pokud KB $\models \alpha$ implikuje KB $\vdash_i \alpha$

Umělá inteligence I, Roman Barták



Odvozovací metody

- Existující dvě základní třídy odvozovacích algoritmů.

□ ověřování modelů (model checking)

- enumerace pravdivostní tabulky
- Davis-Putnam-Logemann-Loveland (DPLL)
- lokální prohledávání (minimalizace konfliktů)

□ odvozovací pravidla (inference rules)

- hledání důkazu aplikací odvozovacích pravidel
- rezoluční metoda

Umělá inteligence I, Roman Barták



Enumerace

```

function TT-ENTAILS?(KB, α) returns true or false
    symbols ← a list of the proposition symbols in KB and α
    return TT-CHECK-ALL(KB, α, symbols, [])

```

```

function TT-CHECK-ALL(KB, α, symbols, model) returns true or false
    if EMPTY?(symbols) then
        if PL-TRUE?(KB, model) then return PL-TRUE?(α, model)
        else return true
    else do
        P ← FIRST(symbols); rest ← REST(symbols)
        return TT-CHECK-ALL(KB, α, rest, EXTEND(P, true, model)) and
               TT-CHECK-ALL(KB, α, rest, EXTEND(P, false, model))

```

Svět Wumpus
 $\alpha_1 = "[1,2]"$ je bezpečné =
 $\neg P_{1,2}$ je logickým
důsledkem KB, protože $P_{1,2}$ je
pro modely KB vždy false, tj.
 $[1,2]$ neobsahuje díru

- Jednoduše prozkoumáme všechny modely metodou **generuj a testuj**.
- Pro každý model, kde je pravda KB, musí být pravda také α .

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	KB	α_1
false	false	true						
false	false	false	false	false	false	true	false	true
:	:	:	:	:	:	:	:	:
false	true	false	false	false	false	false	false	true
false	true	false	false	false	false	true	true	true
false	true	false	false	false	true	true	true	true
false	true	false	false	true	false	false	false	true
:	:	:	:	:	:	:	:	:
true	false	false						

Umělá inteligence I, Roman Barták

Trochu logiky

- Věta (formule) je **splnitelná**, pokud je pravdivá v **nějakém** modelu.
Příklad: $A \vee B, C$
- Věta (formule) je **nesplnitelná**, pokud není pravdivá v **žádném** modelu.
Příklad: $A \wedge \neg A$
- Logický důsledek lze převést na testování splnitelnosti následujícím způsobem: **$KB \models \alpha$ právě tehdy když $(KB \wedge \neg \alpha)$ je nesplnitelná.**
 - důkaz pomocí zamítnutí
 - důkaz sporem
- Ověřování, zda α je logickým důsledkem KB tedy můžeme převést na problém testování splnitelnosti logické formule $(KB \wedge \neg \alpha)$.
Typicky se používají formule v **konjunktivním normálním tvaru** (CNF)
 - literál** je výroková proměnná nebo její negace
 - klauzule** je disjunkce literálů
 - formule** v CNF je konjunkce klauzulíPříklad: $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$
Každou větu (formuli) lze převést na CNF.

$$\begin{aligned} B_{1,1} &\Leftrightarrow (P_{1,2} \vee P_{2,1}) \\ (B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}) \\ (\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1}) \\ (\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1}) \\ (\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1}) \end{aligned}$$

Umělá inteligence I, Roman Barták

DPLL

- Davis, Putnam, Logemann, Loveland
 - úplný a korektní algoritmus pro testování splnitelnosti CNF formule (najde její model)

```
function DPLL-SATISFIABLE?(s) returns true or false
    inputs: s, a sentence in propositional logic
    clauses ← the set of clauses in the CNF representation of s
    symbols ← a list of the proposition symbols in s
    return DPLL(clauses, symbols, [])

function DPLL(clauses, symbols, model) returns true or false
    if every clause in clauses is true in model then return true
    if some clause in clauses is false in model then return false
    P, value ← FIND-PURE-SYMBOL(symbols, clauses, model)
    if P is non-null then return DPLL(clauses, symbols-P, [P = value | model])
    P, value ← FIND-UNIT-CLAUSE(clauses, model)
    if P is non-null then return DPLL(clauses, symbols-P, [P = value | model])
    P ← FIRST(symbols); rest ← REST(symbols)
    return DPLL(clauses, rest, [P = true | model]) or
           DPLL(clauses, rest, [P = false | model])
```

Brzká terminace pro částečný model

- klauzule je pravdivá, pokud je libovolný literál pravdivý
- formule je nepravdivá, pokud je libovolná klauzule nepravdivá

Heuristika ryzí proměnné

- proměnná je ryzí, pokud jsou všechny její výskytu se stejným „znaménkem“
- odpovídající literál je nastaven na true

Heuristika jednotkové klauzule

- klauzule je jednotková, pokud obsahuje jedený literál
- tento literál je nastaven na true

Větvení pro backtracking

Umělá inteligence I, Roman Barták

■ Horolezecká metoda kombinovaná s náhodnou procházkou

- minimalizujeme počet konfliktních (nesplněných) klauzulí
- krok je realizován překlopením hodnoty vybrané proměnné
- **korektní, neúplný algoritmus**

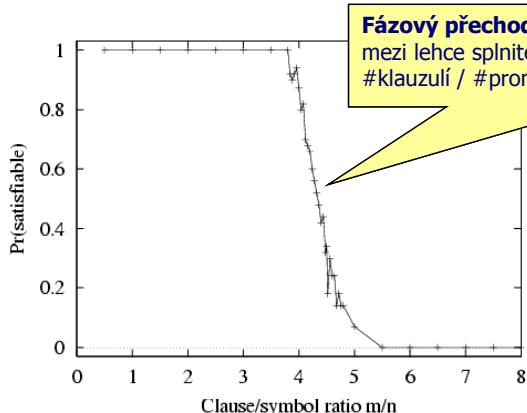
```

function WALKSAT(clauses, p, max-flips) returns a satisfying model or failure
  inputs: clauses, a set of clauses in propositional logic
           p, the probability of choosing to do a “random walk” move
           max-flips, number of flips allowed before giving up
  model  $\leftarrow$  a random assignment of true/false to the symbols in clauses
  for i = 1 to max-flips do
    if model satisfies clauses then return model
    clause  $\leftarrow$  a randomly selected clause from clauses that is false in model
    with probability p flip the value in model of a randomly selected symbol
      from clause
    else flip whichever symbol in clause maximizes the number of satisfied clauses
  return failure

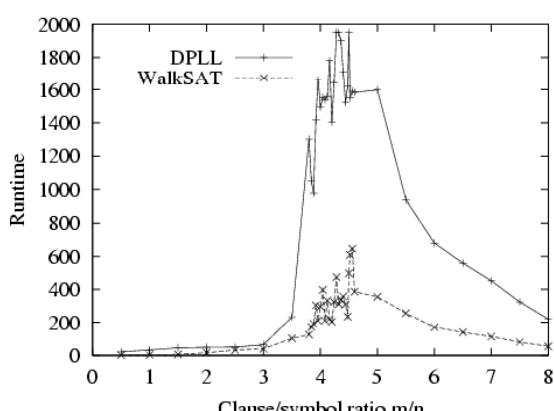
```

Umělá inteligence I, Roman Barták

WalkSAT vs. DPLL



- Náhodné 3-SAT problémy pro 50 proměnných
 - klauzule obsahuje 3 různé proměnné
 - proměnná v klauzuli je v negované podobě s 50% pravděpodobností



- Medián času na vyřešení problému (pro 100 splnitelných problémů)
 - DPLL je docela efektivní
 - WalkSAT je ještě rychlejší

Umělá inteligence I, Roman Barták

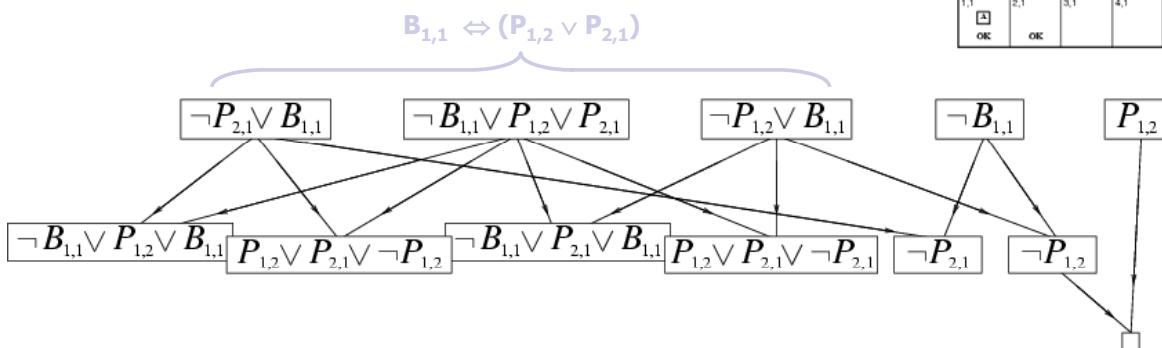
Obecná rezoluce

- Rezoluční algoritmus dokazuje nesplnitelnost formule ($KB \wedge \neg\alpha$) převedené na CNF opakovánou aplikací **rezolučního pravidla**, které ze dvou klauzulí sdílejících komplementární literaly (P a $\neg P$) udělá novou klauzuli:

$$\frac{\ell_1 \vee \dots \vee \ell_k \quad m_1 \vee \dots \vee m_n}{\ell_1 \vee \dots \vee \ell_{i-1} \vee \ell_{i+1} \vee \dots \vee \ell_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n}$$

kde ℓ_i a m_j jsou komplementární literály

- Algoritmus končí pokud
 - nelze přidat žádnou další klauzuli (potom $\neg KB \models \alpha$)
 - vznikla prázdná klauzule (potom $KB \models \alpha$)
- **úplný a korektní algoritmus**



Umělá inteligence I, Roman Barták

Obecná rezoluce algoritmus

- Pro každé dvě klauzule vytvoří všechny možné rezolventy, které přidá do množiny klauzulí pro další rezoluci.
 - prázdná klauzule odpovídá false (prázdná disjunkce)
 - ↳ formule je nesplnitelná
 - dosažen pevný bod (nevznikly nové klauzule)
 - ↳ formule je splnitelná, najdeme model
 - postupně bereme výrokové proměnné P_i
 - máme-li klauzuli s $\neg P_i$ takovou, že ostatní literály jsou nepravda při ohodnocení zvoleném pro P_1, \dots, P_{i-1} , potom do P_i přiřad' false
 - jinak do P_i přiřad' true

```

function PL-RESOLUTION( $KB, \alpha$ ) returns true or false
  clauses ← the set of clauses in the CNF representation of  $KB \wedge \neg\alpha$ 
  new ← {}
  loop do
    for each  $C_i, C_j$  in clauses do
      resolvents ← PL-RESOLVE( $C_i, C_j$ )
      if resolvents contains the empty clause then return true
      new ← new ∪ resolvents
      if new ⊆ clauses then return false
      clauses ← clauses ∪ new
  
```

Umělá inteligence I, Roman Barták



Hornovské klauzule

- Řada bází znalostí obsahuje pouze klauzule, které mají speciální tvar, tzv. **Hornovské klauzule**.
 - obsahují maximálně jeden pozitivní literál

Příklad: $C \wedge (\neg B \vee A) \wedge (\neg C \vee \neg D \vee B)$

 - typicky vznikají z bází dat, kde jsou všechny věty ve tvaru implikace (například Prolog bez proměnných)

Příklad: $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$
- Budeme řešit problém, zda je daná výroková proměnná – **dotaz** – odvoditelná z báze znalostí skládající se pouze z Hornovských klauzulí.
 - je možné použít speciální variantu rezolučního principu, která běží v lineárním čase vzhledem k velikosti KB
 - **dopředné řetězení** (od faktů ke všem závěrům)
 - **zpětné řetězení** (od dotazu k faktům)

Umělá inteligence I, Roman Barták



Dopředné řetězení

- Z dostupných faktů odvozujeme pomocí Hornovských klauzulí všechny možné závěry dokud vznikají nové fakty resp. nedokážeme platnost dotazu.
- Je to **metoda řízená daty**.

```
function PL-FC-ENTAILS?(KB, q) returns true or false
    local variables: count, a table, indexed by clause, initially the number of premises
                  inferred, a table, indexed by symbol, each entry initially false
                  agenda, a list of symbols, initially the symbols known to be true

    while agenda is not empty do
        p ← POP(agenda)
        unless inferred[p] do
            inferred[p] ← true
            for each Horn clause c in whose premise p appears do
                decrement count[c]
                if count[c] = 0 then do
                    if HEAD[c] = q then return true
                    PUSH(HEAD[c], agenda)
            end
        end
    end
    return false
```

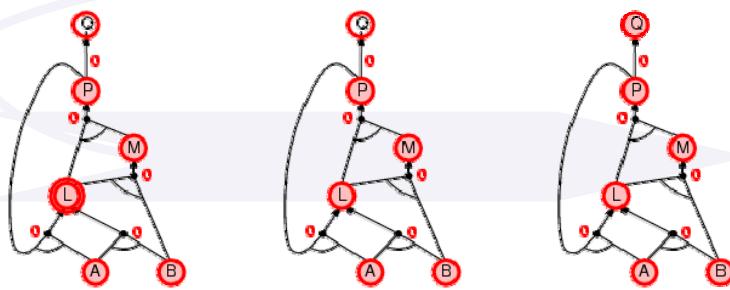
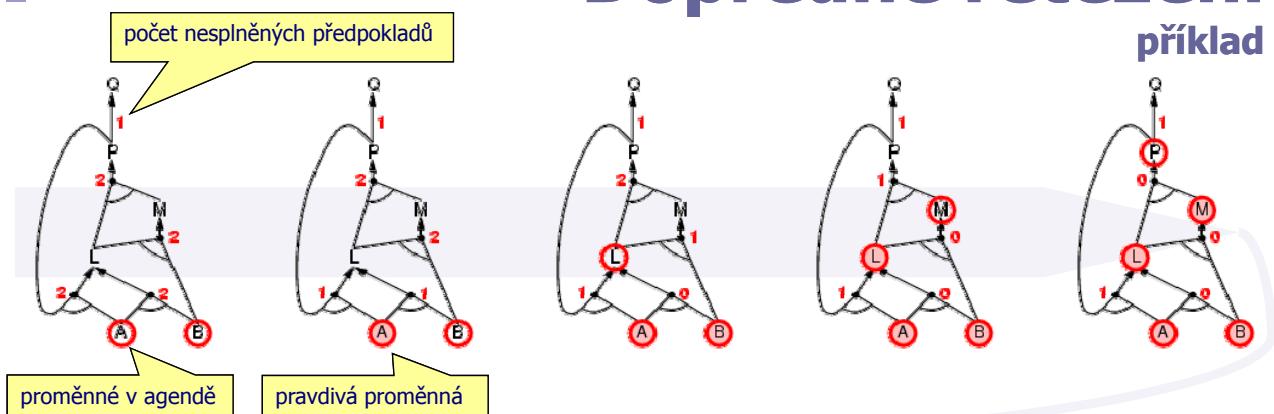
U každé klauzule si pamatujeme počet předpokladů, který zmenšujeme, když je předpoklad dokázán.
Klauzule s nula (zbývajícími) předpoklady slouží k dokázání hlavy klauzule

- **korektní a úplný** algoritmus pro Hornovské klauzule
- **lineární časová složitost** v rozměru báze znalostí

Umělá inteligence I, Roman Barták

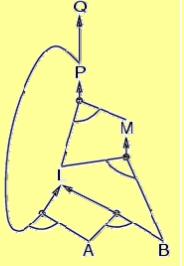
Dopředné řetězení

příklad



Báze znalostí a její grafové znázornění

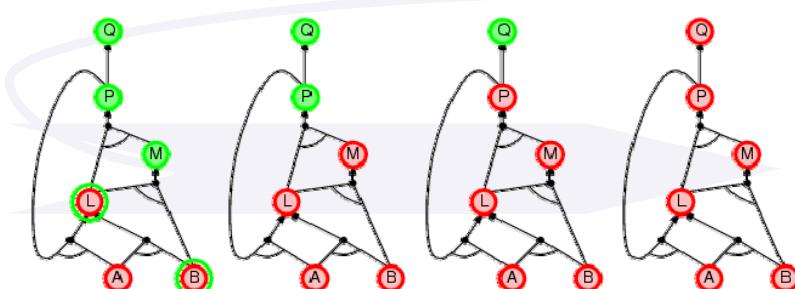
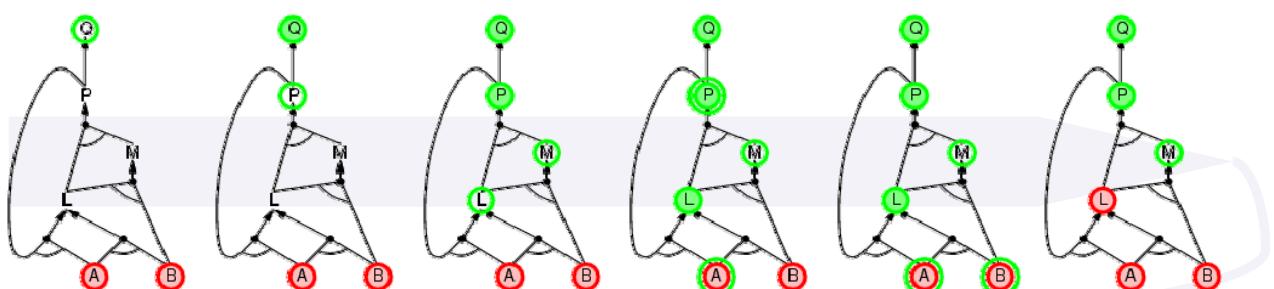
$P \Rightarrow Q$
 $L \wedge M \Rightarrow P$
 $B \wedge L \Rightarrow M$
 $A \wedge P \Rightarrow L$
 $A \wedge B \Rightarrow L$
 A
 B



Umělá inteligence I, Roman Barták

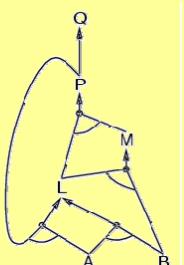
Zpětné řetězení

- Pro daný dotaz hledáme rozklad pomocí Hornovské klauzule na pod-dotazy, které řešíme stejným způsobem.
- **Metoda řízená dotazem.**



Báze znalostí a její grafové znázornění

$P \Rightarrow Q$
 $L \wedge M \Rightarrow P$
 $B \wedge L \Rightarrow M$
 $A \wedge P \Rightarrow L$
 $A \wedge B \Rightarrow L$
 A
 B



Umělá inteligence I, Roman Barták

Agent Wumpus

báze znalostí

- Pro jednoduchost budeme v bázi znalostí zatím reprezentovat jen „**fyziku**“ světa Wumpus.
 - na úvod víme, že
 - $\neg P_{1,1}$
 - $\neg W_{1,1}$
 - také víme, kde a proč vzniká vánek,
 - $B_{x,y} \Leftrightarrow (P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y})$
 - jak se v místnostech objevuje zápach
 - $S_{x,y} \Leftrightarrow (W_{x,y+1} \vee W_{x,y-1} \vee W_{x+1,y} \vee W_{x-1,y})$
 - a na závěr tady máme „skrytou“ informaci, že Wumpus je právě jeden
 - $W_{1,1} \vee W_{1,2} \vee \dots \vee W_{4,4}$
 - $\neg W_{1,1} \vee \neg W_{1,2}$
 - $\neg W_{1,1} \vee \neg W_{1,3}$
 - ...
- Správně bychom měli také sledovat **agenta**.
 - kde se agent přesně nachází
 - $L_{1,1}$
 - FacingRight¹
 - a co se stane provedením různých akcí
 - $L_{x,y}^t \wedge \text{FacingRight}^t \wedge \text{Forward}^t \Rightarrow L_{x+1,y}^{t+1}$
 - potřebujeme horní limitu na počet časových kroků a i tak dostaneme obrovské množství formulí

Umělá inteligence I, Roman Barták

Agent Wumpus

algoritmus

function PL-WUMPUS-AGENT(*percept*) **returns** an *action*

inputs: *percept*, a list, [*stench*, *breeze*, *glitter*]

static: *KB*, initially containing the “physics” of the wumpus world

x, *y*, *orientation*, the agent’s position (init. [1,1]) and orient. (init. right)

visited, an array indicating which squares have been visited, initially false

action, the agent’s most recent action, initially null

plan, an action sequence, initially empty

update *x*, *y*, *orientation*, *visited* based on *action*

if *stench* **then** TELL(*KB*, $S_{x,y}$) **else** TELL(*KB*, $\neg S_{x,y}$)

if *breeze* **then** TELL(*KB*, $B_{x,y}$) **else** TELL(*KB*, $\neg B_{x,y}$)

if *glitter* **then** *action* \leftarrow *grab*

else if *plan* is nonempty **then** *action* \leftarrow POP(*plan*)

else if for some fringe square $[i,j]$, ASK(*KB*, $(\neg P_{i,j} \wedge \neg W_{i,j})$) is true or

 for some fringe square $[i,j]$, ASK(*KB*, $(P_{i,j} \vee W_{i,j})$) is false **then do**

plan \leftarrow A*-GRAPH-SEARCH(ROUTE-PB([*x*,*y*], *orientation*, [*i*,*j*], *visited*))

action \leftarrow POP(*plan*)

else *action* \leftarrow a randomly chosen move

return *action*

Informace o výsledku pozorování pro bázi znalostí

Najdeme buňku z okraje, která je bezpečná.

Nebo alespoň buňku, která není prokazatelně nebezpečná.

Najdeme posloupnost akcí, která nás dovede k této buňce přes již navštívené stavby.



Umělá inteligence I, Roman Barták

9

Umělá inteligence I



Roman Barták, KTI ML

roman.bartak@mff.cuni.cz

<http://ktiml.mff.cuni.cz/~bartak>



Dnes

- Už umíme používat výrokovou logiku pro reprezentaci znalostí a odvozování důsledků.
- Dnes zopakujeme stejný postup jako minule, ale použijeme obecnější nástroj – **predikátovou logiku.**
 - **reprezentace znalostí**
 - **odvozování v predikátové logice**
 - převodem na výrokovou logiku





Reprezentace znalostí

Hledáme formální jazyk, který nám umožní

- reprezentovat znalosti
- pracovat se znalostmi



Co třeba **programovací jazyky** (C++, Java,...)?

- asi nejrozšířenější třída formálních jazyků
- fakta lze popisovat **datovými strukturami**
 - array svět [4,4]
- **programy** popisují výpočtové procesy (změny datových struktur)
 - svět[2,2] ← díra
- Jak ale obecně odvozovat další fakta z faktů známých?
 - změny datových struktur jsou dělány ad-hoc procedurami → **procedurální přístup**
 - **deklarativní přístup** naproti tomu odděluje znalost a mechanismus pro práci se znalostí (ten je navíc obecný, nezávislý na problému)
- Jak reprezentovat znalosti typu „na pozicích [2,2] nebo [3,1] je díra“?
 - proměnné v programech nabývají jediné hodnoty

Umělá inteligence I, Roman Barták



Přirozené jazyky

- Nemohli bychom pro reprezentaci znalostí použít **jazyk přirozený**, kde můžeme vyjádřit „vše“?
 - Bylo by to krásné, ale zatím se nepodařilo najít přesnou formální sémantiku!
 - Současný pohled na přirozený jazyk je, že slouží spíše jako **médium pro komunikaci** než médium pro čistou reprezentaci.
 - Věta sama nekóduje informaci, záleží také na **kontextu**, ve kterém je vyslovena.
 - „Podívejte!“
 - Jazyk **není kompozicionální** – význam jedné věty může záviset na významu vět okolních.
 - “Pak jsem to uviděl.“
 - Problémem přirozeného jazyka je jeho **nejednoznačnost**.
 - koleje, stopky, ...

Umělá inteligence I, Roman Barták



Zpět k logice

- **Výroková logika** je deklarativní, má kompoziciální sémantiku, která nezáleží na kontextu a je jednoznačná.
- Některé vlastnosti světa ale vyjadřuje těžkopádně (ne moc kompaktně).
 - Wumpus: v okolí každá buňka s dírou je vánek
 - $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
 - $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
 - ...
- Poučme se v přirozeném jazyce:
 - máme zde podstatná jména, která představují **objekty** (díra, buňka,...)
 - slovesa vyjadřují **vztahy** mezi objekty (sousedí s, ...)
 - některé relace jsou vlastně **funkce** (je otcem od)
- Místo prostých faktů (výroková logika) začneme pracovat s objekty, relacemi a funkcemi a navíc budeme vyjadřovat fakta o některých nebo o všech objektech světa (**predikátová logika**).

Umělá inteligence I, Roman Barták



Logiky všeobecně

Výroková logika	fakta, která platí nebo ne
Predikátová logika	objekty a relace mezi nimi platí nebo ne
Temporální logika	fakta a časy, kdy daná fakta platí nebo ne
Logika vyšších řádů	relace mezi objekty jsou také objekty (lze dělat tvrzení o všech relacích)

Umělá inteligence I, Roman Barták



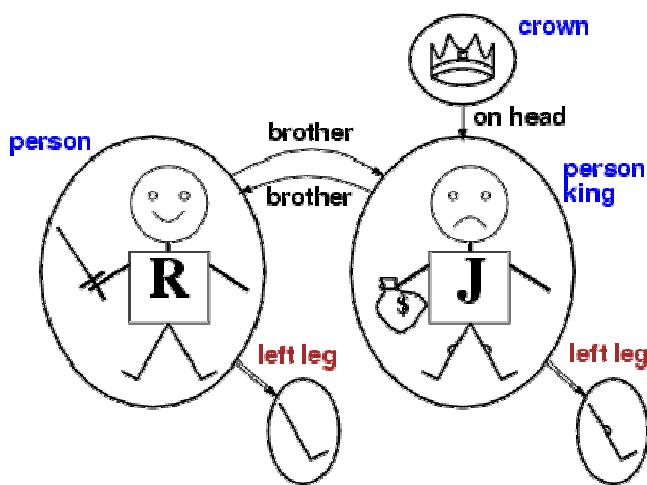
Predikátová logika

syntax

- konstanty John, 2, Crown,...
- predikáty Brother, >, ...
- funkce Sqrt, LeftLeg, ...
- proměnné x, y, a, b, ...
- spojky \neg , \Rightarrow , \wedge , \vee , \Leftrightarrow
- rovnost =
- kvantifikátory \forall , \exists

Umělá inteligence I, Roman Barták

Příklad



- **konstanty** (jména objektů):
 - Richard, John, TheCrown
- **funkční symbol:**
 - LeftLeg
- **termy** (jiný způsob pojmenování objektu)
 - LeftLeg(John)
- **predikátové symboly:**
 - Brother, OnHead, Person, King, Crown
- **atomická tvrzení** (popis relace mezi objekty):
 - Brother(Richard, John)
- **složená tvrzení:**
 - King(Richard) \vee King(John)
 - \neg King(Richard) \Rightarrow King(John)
- **tvrzení o výběru objektů** (kvantifikátory):
 - $\forall x \text{ King}(x) \Rightarrow \text{Person}(x)$
 - Pozor: $\forall x \text{ King}(x) \wedge \text{Person}(x)$!!!
 - $\exists x \text{ Crown}(x) \wedge \text{OnHead}(x, John)$
 - Pozor: $\exists x \text{ Crown}(x) \Rightarrow \text{OnHead}(x, John)$!!!

- $\forall x, y \text{ Brother}(x, y) \Rightarrow \text{Brother}(y, x)$
- $\exists x, y \text{ Brother}(x, \text{Richard}) \wedge \text{Brother}(y, \text{Richard})$
- $\exists x, y \text{ Brother}(x, \text{Richard}) \wedge \text{Brother}(y, \text{Richard}) \wedge \neg(x=y)$

Rovnost říká, že dva termy odkazují na stejný objekt ($\text{Father}(\text{John}) = \text{Henry}$).

Umělá inteligence I, Roman Barták

■ Univerzální kvantifikátor $\forall x P$

- P je pravda pro každý možný objekt x
- zhruba odpovídá konjunkci všech instaciovaných P
 - $P(John) \wedge P(Richard) \wedge P(TheCrown) \wedge P(LeftLeg(John)) \wedge \dots$
- typicky se váže s implikací (výběr objektů, pro které je něco pravda)
 - $\forall x \text{King}(x) \Rightarrow \text{Person}(x)$

■ Existenční kvantifikátor $\exists x P$

- existuje objekt x , pro kterou je P pravda
- zhruba odpovídá disjunkci všech instaciovaných P
 - $P(John) \vee P(Richard) \vee P(TheCrown) \vee P(LeftLeg(John)) \vee \dots$

■ Vazby mezi kvantifikátory

- $\forall x \forall y$ je to samé jako $\forall y \forall x$
 $\exists x \exists y$ je to samé jako $\exists y \exists x$
- $\exists x \forall y$ není to samé jako $\forall y \exists x$ ($\exists x \forall y \text{Loves}(x,y)$ vs. $\forall y \exists x \text{Loves}(x,y)$)
- $\forall x P$ je to samé jako $\neg \exists x \neg P$
 $\exists x P$ je to samé jako $\neg \forall x \neg P$

Umělá inteligence I, Roman Barták

PL a báze znalostí

- ## ■ Podobně jako u výrokové logiky použijeme operace **TELL** pro přidávání tvrzení do báze znalostí:
- $\text{TELL}(\text{KB}, \text{King}(John))$
 - $\text{TELL}(\text{KB}, \forall x \text{King}(x) \Rightarrow \text{Person}(x))$
 - Přidáváme **axiomy (fakta)** jako atomická tvrzení, **definice** s použitím \Leftrightarrow a jiná složená tvrzení) a někdy i **teorémy** (lze je logicky odvodit z axiomů, ale „zrychlují“ odvozování)

- ## ■ a operace **ASK** pro dotazování se na logické důsledky KB:

- $\text{ASK}(\text{KB}, \text{King}(John))$ jako databázový dotaz
- $\text{ASK}(\text{KB}, \text{Person}(John))$ tady už je potřeba odvození
- $\text{ASK}(\text{KB}, \exists x \text{Person}(x))$ zde budeme chtít nejen odpověď true, ale také hodnotu x , pro kterou to platí – substituce { $x/John$ }

Umělá inteligence I, Roman Barták



Odvozování v PL

- Již umíme odvozovat ve výrokové logice a teď se podíváme na odvozování v logice predikátové.
- Základní odlišnosti:
 - kvantifikátory → **skolemizace**
 - funkce a proměnné → **unifikace**
- Zbytek už je nám známý:
 - dopředné řetězení** (dedukční databáze, produkční systémy)
 - zpětné řetězení** (logické programování)
 - rezoluce** (dokazování vět)



Umělá inteligence I, Roman Barták



Redukce PL na VL

- **Odvozování v predikátové logice může udělat převedením do logiky výrokové a použitím jejích odvozovacích nástrojů.**
 - „uzemnění“ (grounding)
 - za všechny proměnné dosadíme všechny možné termy
 - atomická tvrzení potom odpovídají výrokovým proměnným
 - A co kvantifikátory?
 - univerzální kvantifikátor: proměnná nahrazena termem
 - existenciální kvantifikátor: skolemizace (proměnné nahrazena novou konstantou)

Umělá inteligence I, Roman Barták



Redukce PL na VL

odstranění kvantifikátorů

Univerzální instanciace

$$\frac{\forall v \alpha}{\text{Subst}(\{v/g\}, \alpha)}$$

Pro proměnnou a instanciovaný term g aplikujeme substituci g za x.

Aplikujeme opakováně pro různé instanciace g.

- Příklad: $\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$ vede na:

$\text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$

$\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})$

$\text{King}(\text{LeftLeg}(\text{John})) \wedge \text{Greedy}(\text{LeftLeg}(\text{John})) \Rightarrow \text{Evil}(\text{LeftLeg}(\text{John}))$

...

Existenciální instanciace

$$\frac{\exists v \alpha}{\text{Subst}(\{v/k\}, \alpha)}$$

Pro proměnnou a novou konstantu k aplikujeme substituci k za x.

Aplikujeme jedenkrát s novou konstantou, která se nevyskytuje v bázi znalostí (Skolemova konstanta)

- Příklad: $\exists x \text{ Crown}(x) \wedge \text{OnHead}(x, \text{John})$ vede na:

$\text{Crown}(\text{C}_1) \wedge \text{OnHead}(\text{C}_1, \text{John})$

Umělá inteligence I, Roman Barták



Redukce PL na VL

příklad

- Ze znalostní báze v predikátové logice (zatím bez funkcí):

$\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$

$\text{King}(\text{John})$

$\text{Greedy}(\text{John})$

$\text{Brother}(\text{Richard}, \text{John})$

- vytvoříme znalostní bázi ve výrokové logice všemi možnými instanciacemi:

$\text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$

$\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})$

$\text{King}(\text{John})$

$\text{Greedy}(\text{John})$

$\text{Brother}(\text{Richard}, \text{John})$

- Odvození provedeme ve výrokové logice.

Problém: funkční symboly vedou k nekonečně mnoha termům
 $\text{LeftLeg}(\text{John}), \text{LeftLeg}(\text{LeftLeg}(\text{John})), \dots$

- Herbrand: odvození v PL z dané KB existuje, pokud existuje odvození ve VL v konečné podmnožině instanciované KB
- postupně přidáváme větší a větší termy do KB, dokud nenajdeme odvození
- když ale odvození neexistuje, neskončíme ☺

Umělá inteligence I, Roman Barták

10

Umělá inteligence I



Roman Barták, KTI ML

roman.bartak@mff.cuni.cz

<http://ktiml.mff.cuni.cz/~bartak>



Odvozování v PL

- Již umíme odvozovat ve výrokové logice a také umíme odvozovat v PL převodem na VL:
 - „uzemnění“ výrazů (dosazení všech možných termů)
 - skolemizace pro odstranění existenčního kvantifikátoru
- Odvození převodem na VL ale není moc efektivní
 - generuje velké množství irrelevantních tvrzení typu $\text{King}(\text{LeftLeg}(\text{John})) \wedge \text{Greedy}(\text{LeftLeg}(\text{John})) \Rightarrow \text{Evil}(\text{LeftLeg}(\text{John}))$
- Zkusíme odvozovat přímo v PL:
 - **liftování**
 - **unifikace**
 - **standardizace stranou**
- Zbytek už je nám známý:
 - **dopředné řetězení** (dedukční databáze, produkční systémy)
 - **zpětné řetězení** (logické programování)
 - **rezoluce** (dokazování vět)





Odvodování v PL

- Upravíme odvozovací pravidla pro práci v PL:
 - **liftování** – uděláme jen takové substituce, které potřebujeme pro odvozování
 - **zobecněné (liftované) pravidlo Modus Ponens**

$$\frac{p_1, p_2, \dots, p_n, q_1 \wedge q_2 \wedge \dots \wedge q_n \Rightarrow q}{\text{Subst}(\theta, q)}$$

kde θ je substituce taková, že $\text{Subst}(\theta, p_i) = \text{Subst}(\theta, q_i)$
(pro **definitní klauzule** s právě jedním pozitivním literálem - **pravidla**)

Umělá inteligence I, Roman Barták



Unifikace

- Jak najít substituci θ takovou, že dva logické výrazy p a q po její aplikaci vypadají identicky?

- $\text{Unify}(p,q) = \theta$, kde $\text{Subst}(\theta,p) = \text{Subst}(\theta,q)$

p	q	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}
Knows(John,x)	Knows(y,OJ)	{x/OJ,y/John}
Knows(John,x)	Knows(y,Mother(y))	{y/John,x/Mother(John)}{}
Knows(John,x)	Knows(x,OJ)	{fail}

- ## ☐ Co když ale existuje více takových substitucí?

Knows(John,x) Knows(v,z),

↳ $\theta = \{y/John, x/z\}$ nebo $\theta = \{y/John, x/John, z/John\}$

- První **substituce je obecnější** než druhá substituce (druhý výsledek lze získat dodatečnou aplikací substituce $\{z/John\}$).

- Existuje jednoznačná (až na přejmenování proměnných) substituce, která je obecnější než všechny ostatní – **nejobecnější unifikátor** (mgu – most general unifier).

Hledáme unifikátor

```
function UNIFY( $x, y, \theta$ ) returns a substitution to make  $x$  and  $y$  identical
  inputs:  $x$ , a variable, constant, list, or compound
           $y$ , a variable, constant, list, or compound
           $\theta$ , the substitution built up so far

  if  $\theta = \text{failure}$  then return failure
  else if  $x = y$  then return  $\theta$ 
  else if VARIABLE?( $x$ ) then return UNIFY-VAR( $x, y, \theta$ )
  else if VARIABLE?( $y$ ) then return UNIFY-VAR( $y, x, \theta$ )
  else if COMPOUND?( $x$ ) and COMPOUND?( $y$ ) then
    return UNIFY(ARGS[ $x$ ], ARGS[ $y$ ], UNIFY(OP[ $x$ ], OP[ $y$ ],  $\theta$ ))
  else if LIST?( $x$ ) and LIST?( $y$ ) then
    return UNIFY(REST[ $x$ ], REST[ $y$ ], UNIFY(FIRST[ $x$ ], FIRST[ $y$ ],  $\theta$ ))
  else return failure
```

výrazy se procházejí současně použitím rekurze a staví se mgu dokud se struktury „nevyprázdní“ resp. nezjistí se, že se liší

složené termíny musí mít stejný funkční symbol a unifikovatelné argumenty

seznamy se řeší odděleně, aby nevzniknul cyklus při jejich reprezentaci strukturou .(First,Rest)

```
function UNIFY-VAR( $var, x, \theta$ ) returns a substitution
  inputs:  $var$ , a variable
           $x$ , any expression
           $\theta$ , the substitution built up so far

  if  $\{var/val\} \in \theta$  then return UNIFY( $val, x, \theta$ )
  else if  $\{x/val\} \in \theta$  then return UNIFY( $var, val, \theta$ )
  else if OCCUR-CHECK?( $var, x$ ) then return failure
  else return add  $\{var/x\}$  to  $\theta$ 
```

kontrola výskytu proměnné var v termu x

- x a $f(x)$ nejsou unifikovatelné
- vede ke kvadratické složitosti algoritmu
- existují i lineární algoritmy
- někdy se neprovádí (Prolog)

Umělá inteligence I, Roman Barták

Standardizace stranou

- Představme si, že **dotaz** na bázi znalostí je Knows(John, x)
- Odpovědi můžeme získat **nalezením tvrzení** v bázi znalostí, která jsou **unifikovatelná s dotazem**:

Knows(John, Jane)	→ {x/Jane}
Knows(y, Mother(y))	→ {x/Mother(John)}
Knows(x, Elizabeth)	→ fail

???

- Knows(x,Elizabeth) v bázi znalostí znamená, že každý zná Elizabeth (předpokládáme univerzální kvantifikátor), tedy i John.
- Problém je, že oba výrazy obsahují proměnnou x a nelze je tedy unifikovat.
- $\forall x \text{ Knows}(x, \text{Elizabeth})$ je ale to samé jako $\forall y \text{ Knows}(y, \text{Elizabeth})$
- Před každým použitím tvrzení z báze znalostí v něm přejmenujeme všechny proměnné na úplně nové proměnné, které se ještě nikde nepoužily – **standardizace stranou**.

Umělá inteligence I, Roman Barták

- Podle práva v USA je zločincem každý Američan, který prodává zbraně nepřátelským státům. Země Nono je nepřítelem USA vlastnícím rakety a všechny tyto rakety jim byly prodány plukovníkem Westem, který je Američanem.
- Dokažte, že West je zločinec.

... je zločincem každý Američan, který prodává zbraně nepřátelským státům:
 $American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$

Nono ... vlastní rakety, tj. $\exists x \text{ Owns}(Nono,x) \wedge \text{Missile}(x)$:
 $\text{Owns}(Nono,M_1) \text{ and } \text{Missile}(M_1)$

... všechny tyto rakety jim byly prodány plukovníkem Westem
 $\text{Missile}(x) \wedge \text{Owns}(Nono,x) \Rightarrow \text{Sells}(West,x,Nono)$

Rakety jsou zbraně.

$\text{Missile}(x) \Rightarrow \text{Weapon}(x)$

Nepřítel USA je „nepřátelský stát“.

$\text{Enemy}(x,America) \Rightarrow \text{Hostile}(x)$

West je Američan ...

$\text{American}(West)$

Země Nono je nepřítelem USA ...

$\text{Enemy}(Nono,America)$



Umělá inteligence I, Roman Barták

Odvozovací techniky

Všechna tvrzení v příkladě mají tvar definitních klauzulí a nemáme tam žádné funkční symboly.

Pro řešení příkladu můžeme použít:

■ dopředné řetězení

- pomocí Modus Ponens odvodíme z daných faktů všechna možná tvrzení
- typický způsob odvozování v deduktivních databázích (Datalog) a produkčních systémech

■ zpětné řetězení

- půjdeme naopak od dotazu $Criminal(West)$ a budeme hledat fakta, která ho potvrzují
- typický způsob odvozování v logickém programování

Dopředné řetězení

algoritmus

```

function FOL-FC-ASK( $KB, \alpha$ ) returns a substitution or false
repeat until  $new$  is empty
     $new \leftarrow \{\}$ 
    for each sentence  $r$  in  $KB$  do
         $(p_1 \wedge \dots \wedge p_n \Rightarrow q) \leftarrow \text{STANDARDIZE-APART}(r)$ 
        for each  $\theta$  such that  $(p_1 \wedge \dots \wedge p_n)\theta = (p'_1 \wedge \dots \wedge p'_n)\theta$ 
            for some  $p'_1, \dots, p'_n$  in  $KB$ 
                 $q' \leftarrow \text{SUBST}(\theta, q)$ 
                if  $q'$  is not a renaming of a sentence already in  $KB$  or  $new$  then do
                    add  $q'$  to  $new$ 
                     $\phi \leftarrow \text{UNIFY}(q', \alpha)$ 
                    if  $\phi$  is not fail then return  $\phi$ 
            add  $new$  to  $KB$ 
return false

```

vezmeme pravidlo z databáze,
kde přejmenujeme proměnné
(standardizace stranou)

odvodíme všechny důsledky,
které ještě v KB nejsou

pokud už jsme odvodili tvrzení
unifikatelné s dotazem, potom
vrátíme příslušnou substituci

získaná tvrzení přidáme do
báze znalostí
a vše opakujeme dokud
přidáváme další tvrzení

Dopředné řetězení je **korektní** a **úplný** algoritmus pro nalezení odvození.

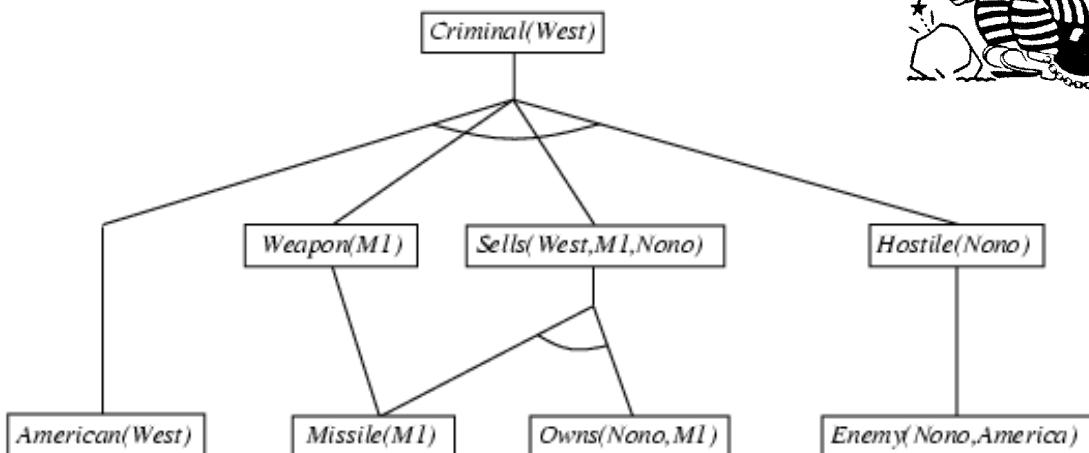
- Pozor! Pokud odvození neexistuje, nemusí skončit (máme-li v jazyce funkční symboly).

Umělá inteligence I, Roman Barták

Dopředné řetězení

příklad

$\text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x,y,z) \wedge \text{Hostile}(z) \Rightarrow \text{Criminal}(x)$
 $\text{Owns}(\text{Nono}, \text{M1}) \wedge \text{Missile}(\text{M1})$ (vzniklo z $\exists x \text{ Owns}(\text{Nono}, x) \wedge \text{Missile}(x)$)
 $\text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})$
 $\text{Missile}(x) \Rightarrow \text{Weapon}(x)$
 $\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x)$
 $\text{American}(\text{West})$
 $\text{Enemy}(\text{Nono}, \text{America})$



Umělá inteligence I, Roman Barták

Dopředné řetězení

hledání vzorů

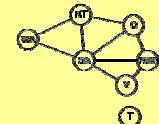
for each θ such that $(p_1 \wedge \dots \wedge p_n)\theta = (p'_1 \wedge \dots \wedge p'_n)\theta$
for some p'_1, \dots, p'_n in KB

- Jak najdeme (rychle) množinu faktů p'_1, \dots, p'_n , aby šla unifikovat s tělem pravidla?
 - tzv. **hledání vzorů** (pattern matching)
 - Příklad 1: *Missile(x) \Rightarrow Weapon(x)*
 - můžeme **indexovat** množinu **faktů podle jména** predikátu, čímž se vyhneme pokusům typu *Unify(Missile(x), Enemy(Nono, America))*
 - Příklad 2: *Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)*
 1. můžeme hledat objekty, které vlastní Nono, a jsou to rakety ...
 2. nebo můžeme hledat rakety a zjišťovat, zda je vlastní Nono
- Co je lepší?
 - Začni tam, kde je méně možností (pokud existují dvě rakety, ale Nono vlastní hodně věcí, je lepší alternativa 2) – **připomeňme pravidlo prvního neúspěchu** z omezujících podmínek



Hledání vzorů je NP-úplný problém.

$Diff(wa,nt) \wedge Diff(wa,sa) \wedge Diff(nt,q) \wedge Diff(nt,sa) \wedge Diff(q,nsw) \wedge$
 $Diff(q,sa) \wedge Diff(nsw,v) \wedge Diff(nsw,sa) \wedge Diff(v,sa) \Rightarrow Colorable()$



Umělá inteligence I, Roman Barták

Dopředné řetězení

inkrementálně

- Příklad: *Missile(x) \Rightarrow Weapon(x)*
 - při první iteraci dopředného řetězení odvodí, že všechny známé rakety jsou zbraně
 - při druhé (a každé další) iteraci odvodí to samé takže bázi znalostí už nemění
- Kdy má cenu pravidlo znova zkoušet?
 - pokud se v KB objeví nový fakt z těla pravidla
- **Inkrementální dopředné řetězení**
 - pravidlo se spustí v iteraci t pouze pokud byl v iteraci t-1 odvozen nový fakt unifikovatelný s faktom v těle pravidla
 - když do KB přidáme nový fakt, podíváme se na všechna pravidla, která obsahují s ním unifikovatelný výraz ve svém těle
 - **Rete algoritmus**
 - předzpracuje pravidla do podoby sítě závislostí, kde rychleji vyhledá, jaké pravidlo se má zkuskit po přidání nového faktu

Umělá inteligence I, Roman Barták



Dopředné řetězení

magická množina

- Dopředné řetězení odvozuje všechna fakta, i když nejsou relevantní pro zjištění odpovědi.
 - jedním z řešení je **zpětné řetězení**
 - jiná možnost je upravit pravidla tak, aby umožňovaly pouze relevantní navázání proměnných z tzv. **magické množiny**

Příklad: dotaz na Criminal(West)

*Magic(x) \wedge American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z)
 \Rightarrow Criminal(x)*
Magic(West)

- Magickou množinu lze získat zpětným průchodem přes pravidla.



Umělá inteligence I, Roman Barták



Produkční systémy

TESTY

Podnikoví správci pravidel

Máte-li možnost dosáhnout flexibilitu, výkonnosti a snadné údržby vašich firemních aplikací díky implementaci nákladového ekstraktivního produktu, jako je JBoss Rules nebo Jess, můžete se otázka, v čem se tyto systémy liší od klasické funkčních sil. Několik rozdílů se můžete stnit příce najít. Strana 10

Řízení obchodních pravidel levně a jednoduše

JAMES OWEN

Uvážme-li, že high-endové systémy pro řízení obchodních pravidel, Business Rule Management Systems, vás výjdu na zhruba 50 000 dolarů, je při zprovoznění a za roční udržbu, provozu poplatky a profesionální služby mohou celkově náklady výhodnou téměř až k půl milionu nebo více, mají organizace s těsnějším rozpočtem velmi dobrou motivaci pochlednut se po alternativách. Dobré volby naštěstí existují – JBoss Rules a Jess představují solidní nástroje pro řízení pravidel a respektu hodný výkon za sympatickou cenu.

D výšce a rychlostí B2BMS náležejí však alespoň částečně do Jess a společnosti Sandia National Laboratories a JBoss Rules firmy JBoss, divize společnosti Red Hat. Stejně jako poslovce systémů Jess firmy Jess, rozšířil Jessa David Ferrai, který JBoss odkrýval obchodní logiky komplexních Javaových aplikací jako sadu pravidel, která mohou být rychle a snadno změněna bez směry v základním Java kódu. Nicméně na rendl od této systémů třídy Expertise ani Jess, ani JBoss Rules neposkytují uživatelsky

Servery s architekturou x86

Servery postavené na architektuře x86, letitěm a takřka nesnesitelném standardu, představují velmi ekstraktivní řešení „hardwarového problému“ pro mnoho firem a širokou šířku aplikací. Jejich výkon lze díky stálé leptání komponentů aktualizovat až do maximální výšky a pěvce se zde zalyžují i grafické technologie. Strana 20

plivitivitou možnost (vlastní editor, diagramy toků či tabulkové GUI), jež dodávají běžným firemním/obchodním uživatelům stejně jako programátorem vkládat, měnit a mazat pravidla.

Na rozdíl od systémů B2BMS a JBoss patříme do JBoss Rules novinku i přinášející atraktivní pravidel (rule repository). Jsa o JBoss Rules mohou být integrovány s CVS systémem pro kontrolu verzí, takové řešení však daleko zaujímá za možností firemního cyklu, garantuje kontroly příslušného pravidloho repozitoria, poskytuje jednotné rozhraní pro všechny pravidly a využívání repozitorium může být klíčem ke spolu práci mezi mnoha vývojáři a obchodními analytiky a hojně možností reprezentace, mnoho by nepostředované pravidlo pro ladění a optimizaci.

Sandia Labs je společnost, která má open source, který reprezentuje Jess a JBoss, má také své výhody. Jen JBoss vyvíjí výrobají z celého světa, kteří neplatí žádají a opravují chybky, nové funkce, požadavky, kód a ve skutečnosti vlastní fungují jako součásti vývojářské skupiny startující o tyto produkty. Všichni IT personál když JBoss Rules nebo konzultantům četí stran – využívají uživatelsky plivitivní tabulkové GUI, vizualizaci pravidel a vývojového prostředí, které si budete přát. Takové výhody ale brázdou klasické nároky na personál, školení a investice po dobu několika měsíců až let, zattimco problem, který potřebuje řešit, existuje právě nyní.

V kontextu se zde JBoss a JBoss Rules jsou nejvhodnější metou pro řízení a kód anachiv pravidel či množství množství reportingu když neplatíte výkonné požadavky a kde tvorba a údržba pravidel mohou být svěřeny jednotnou nebo několika zavěřeným programátorkám.

Sandia Labs Jess

Jess, systém společnosti Sandia Labs a Ernesta Friedmann-Hilla, byl, pokud je nám známo, první implementací na světě založeného systému pravidel. Stal se první vývojovou až doby zrujšených částeček CLIPS (na jazyce C založených rozhraní k Production Systems), projektu organizace NASA. Poté se začala objevovat řada high-endových systémů, jako je zminěný JBoss od IfLang a Blaze Advisor od Faiz Isaac. V následujících letech trval Fried-

- založené na rete algoritmu

XCON (R1)

- konfigurace počítačů DEC

OPS-5

- jazyk založený na dopředném řetězení

CLIPS

- nástroj pro tvorbu expertním systémů vyvinutý v NASA

Jess, JBoss Rules,...

- řízení obchodních pravidel

Umělá inteligence I, Roman Barták

Zpětné řetězení algoritmus

```

function FOL-BC-ASK( $KB, goals, \theta$ ) returns a set of substitutions
  inputs:  $KB$ , a knowledge base
          $goals$ , a list of conjuncts forming a query
          $\theta$ , the current substitution, initially the empty substitution  $\{ \}$ 
  local variables:  $ans$ , a set of substitutions, initially empty
  if  $goals$  is empty then return  $\{ \theta \}$  {vezmeme první z cílů a aplikujeme na něj dosud nalezenou substituci}
   $q' \leftarrow \text{SUBST}(\theta, \text{FIRST}(goals))$ 
  for each  $r$  in  $KB$  where  $\text{STANDARDIZE-APART}(r) = (p_1 \wedge \dots \wedge p_n \Rightarrow q)$ 
    and  $\theta' \leftarrow \text{UNIFY}(q, q')$  succeeds
     $ans \leftarrow \text{FOL-BC-Ask}(KB, [p_1, \dots, p_n | REST(goals)], \text{COMPOSE}(\theta, \theta')) \cup ans$ 
  return  $ans$  {najdeme pravidlo, jehož hlava je unifikovatelná s cílem}

  tělo pravidla přidáme mezi cíle, upravíme substituci a rekursivně pokračujeme v redukci cílů dokud nejsou prázdné
  skládání substitucí Subst(Compose( $\theta, \theta'$ ),  $p$ ) = Subs( $\theta'$ , Subst( $\theta, p$ ))

```

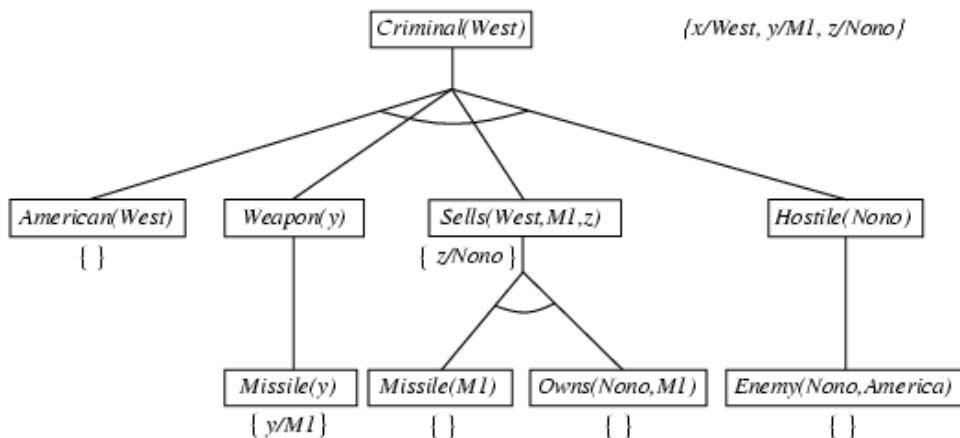


Algoritmus **FOL-BC-Ask** metodou **prohledávání do hloubky** hledá **všechna řešení** (všechny substituce) daného dotazu.
 Pro běh stačí **lineární prostor** (v délce důkazu).
 Na rozdíl od dopředného řetězení **není úplné**, kvůli opakovaným stavům (vytvoří se stejný cíl).

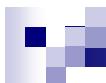
Umělá inteligence I, Roman Barták

Zpětné řetězení příklad

$\text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x,y,z) \wedge \text{Hostile}(z) \Rightarrow \text{Criminal}(x)$
 $\text{Owns}(\text{Nono}, \text{M1}) \wedge \text{Missile}(\text{M1})$ (vzniklo z $\exists x \text{ Owns}(\text{Nono}, x) \wedge \text{Missile}(x)$)
 $\text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})$
 $\text{Missile}(x) \Rightarrow \text{Weapon}(x)$
 $\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x)$
 $\text{American}(\text{West})$
 $\text{Enemy}(\text{Nono}, \text{America})$



Umělá inteligence I, Roman Barták



Logické programování

- Zpětné řetězení v podstatě odpovídá **logickému programování** (Prolog).

```

hlava pravidla      tělo pravidla
      ↗             ↘

criminal(X) :-  

    american(X), weapon(Y), sells(X,Y,Z), hostile(Z).  

owns(nono,m1).  

missile(m1).  

sells(west,X,nono) :-  

    missile(X), owns(nono,X).  

weapon(X) :-  

    missile(X).  

hostile(X) :-  

    enemy(X,america).  

american(west).  

enemy(nono,america).  

?- criminal(west).

```

```

?- criminal(west).  

?- american(west), weapon(Y),  

   sells(west,Y,Z), hostile(Z).  

?- weapon(Y), sells(west,Y,Z),  

   hostile(Z).  

?- missile(Y), sells(west,Y,Z),  

   hostile(Z).  

?- sells(west,m1,Z), hostile(Z).  

?- missile(m1), owns(nono,m1),  

   hostile(nono).  

?- owns(nono,m1), hostile(nono).  

?- hostile(nono).  

?- enemy(nono,america).  

?- true.

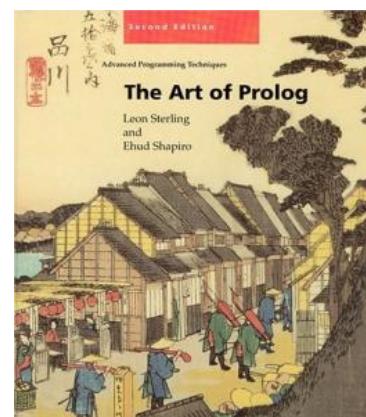
```

Umělá inteligence I, Roman Barták

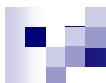


Logické programování vlastnosti

- pevně daný výpočtový mechanismus**
 - podcíle se řeší zleva doprava
 - pravidla se berou v pořadí shora dolu
- vrací jediné řešení**, další řešení na žádost
 - možnost zacyklení (`brother(X,Y) :- brother(Y,X)`)
- obsahuje aritmetiku**
 - `X is 1+2.`
 - (aritmeticky) vyhodnotí výraz nalevo a unifikuje s výrazem napravo
- rovnost** slouží pro explicitní přístup k unifikaci
 - `1+Y = 3.`
 - možnost použít omezující podmínky
(CLP – Constraint Logic Programming)
- negace jako neúspěch** (negation as failure)
 - `alive(X) :- not dead(X).`
 - „Každý je živý, pokud nelze dokázat, že je mrtvý“
 - $\neg \text{Dead}(x) \Rightarrow \text{Alive}(x)$ není definitní klauzule!
 - $\text{Alive}(x) \vee \text{Dead}(x)$
 - „Každý je živý nebo mrtvý“



Umělá inteligence I, Roman Barták



Rezoluční metoda

konjunktivní normální forma

Odvozovací metoda pro formule v **konjunktivní normálně formě**.

- $\forall x [\forall y \text{Animal}(y) \Rightarrow \text{Loves}(x,y)] \Rightarrow [\exists y \text{Loves}(y,x)]$
- odstraníme implikace**
 - $\forall x [\neg \forall y \neg \text{Animal}(y) \vee \text{Loves}(x,y)] \vee [\exists y \text{Loves}(y,x)]$
- negaci posuneme dovnitř** ($\neg \forall x p \equiv \exists x \neg p$, $\neg \exists x p \equiv \forall x \neg p$)
 - $\forall x [\exists y \neg (\neg \text{Animal}(y) \vee \text{Loves}(x,y))] \vee [\exists y \text{Loves}(y,x)]$
 - $\forall x [\exists y \neg \neg \text{Animal}(y) \wedge \neg \text{Loves}(x,y)] \vee [\exists y \text{Loves}(y,x)]$
 - $\forall x [\exists y \text{Animal}(y) \wedge \neg \text{Loves}(x,y)] \vee [\exists y \text{Loves}(y,x)]$
- standardizace proměnných**
 - $\forall x [\exists y \text{Animal}(y) \wedge \neg \text{Loves}(x,y)] \vee [\exists z \text{Loves}(z,x)]$
- Skolemizace** (Skolemovy funkce)
 - $\forall x [\text{Animal}(F(x)) \wedge \neg \text{Loves}(x,F(x))] \vee [\text{Loves}(G(x),x)]$
- odstranění univerzálních kvantifikátorů**
 - $[\text{Animal}(F(x)) \wedge \neg \text{Loves}(x,F(x))] \vee [\text{Loves}(G(x),x)]$
- distribuce \vee a \wedge**
 - $[\text{Animal}(F(x)) \vee \text{Loves}(G(x),x)] \wedge [\neg \text{Loves}(x,F(x)) \vee \text{Loves}(G(x),x)]$

Umělá inteligence I, Roman Barták



Rezoluční metoda

odvozovací pravidlo

- Liftovaná verze rezolučního pravidla pro výrokovou logiku:

$$\frac{\ell_1 \vee \dots \vee \ell_k \quad m_1 \vee \dots \vee m_n}{(\ell_1 \vee \dots \vee \ell_{i-1} \vee \ell_{i+1} \vee \dots \vee \ell_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n)\theta}$$

kde $\text{Unify}(\ell_i, \neg m_j) = \theta$.

- U klauzulí se předpokládá standardizace stranou, aby nesdílely žádné proměnné.
- Pro úplnost je potřeba bud':
 - rozšířit binární rezoluci na více literálů
 - použít **factoring** pro odstranění redundantních literálů (ty, které jsou unifikovatelné)

Příklad:

$$\frac{[\text{Animal}(F(x)) \vee \text{Loves}(G(x),x)], \quad [\neg \text{Loves}(u,v) \vee \neg \text{Kills}(u,v)]}{[\text{Animal}(F(x)) \vee \neg \text{Kills}(G(x),x)]}$$

kde $\theta = \{u/G(x), v/x\}$

- Dotaz α nad bází znalostí KB řešíme aplikací rezolučních kroků na formulí $\text{CNF}(\text{KB} \wedge \neg \alpha)$.
 - Pokud vede k prázdné formuli, je $\text{KB} \wedge \neg \alpha$ nesplnitelné a tedy $\text{KB} \models \alpha$.
- Jedná se o **korektní a úplnou** odvozovací metodu pro predikátovou logiku

Umělá inteligence I, Roman Barták

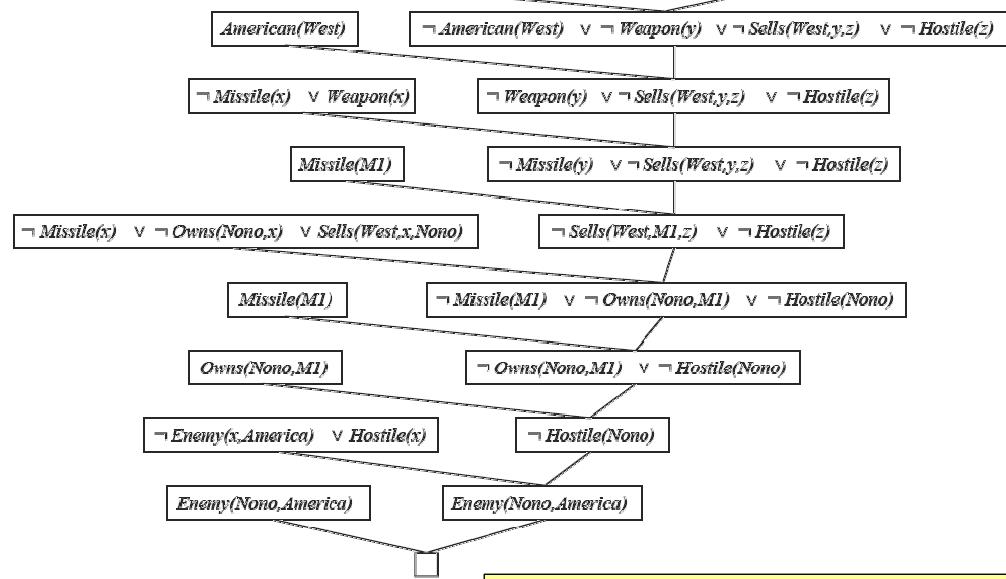


Rezoluční metoda

příklad

$\neg American(x) \vee \neg Weapon(y) \vee \neg Sells(x,y,z) \vee \neg Hostile(z) \vee Criminal(x)$

$\neg Criminal(West)$



! Resoluční metoda aplikovaná na definitní klauzule je vlastně aplikace **zpětného řetězení**, které určuje jaké rezoluční pravidlo se má použít.

American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)
 Owns(Nono,M1) a Missile(M1) (vzniklo z $\exists x$ Owns(Nono,x) \wedge Missile(x))
 Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)
 Missile(x) \Rightarrow Weapon(x)
 Enemy(x,America) \Rightarrow Hostile(x)
 American(West)
 Enemy(Nono,America)

Umělá inteligence I, Roman Barták



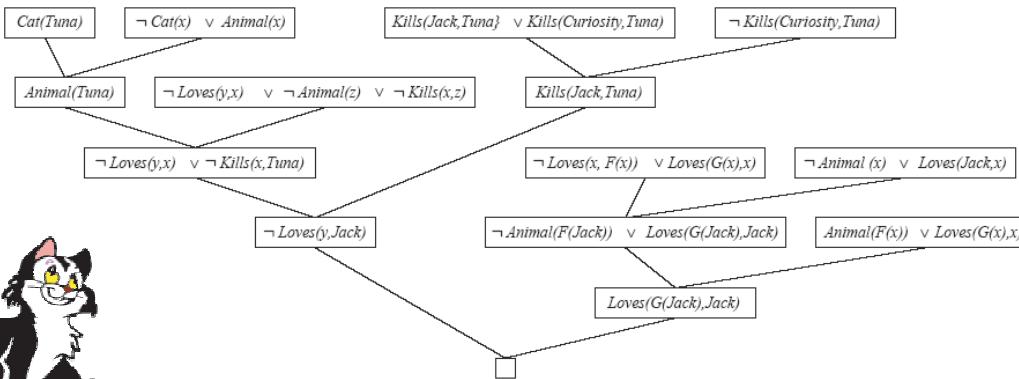
Rezoluční metoda

složitější příklad

- Každého, kdo má rad všechna zvířata, má někdo rád. Každého, kdo zabíjí zvířata, nemá rád nikdo. Jack má rád všechny zvířata. Bud' Jack nebo Curiosity zabili kočku jménem Tuna. Každá kočka je zvíře.
- Zabila Curiosity Tunu?

$\forall x [\forall y Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y Loves(y,x)]$
 $\forall x [\exists y Animal(y) \wedge Kills(x,y)] \Rightarrow [\forall z \neg Loves(z,x)]$
 $\forall x Animal(x) \Rightarrow Loves(Jack,x)$
 Kills(Jack,Tuna) \vee Kills(Curiosity, Tuna)
 Cat(Tuna)
 $\forall x Cat(x) \Rightarrow Animal(x)$
 $\neg Kills(Curiosity,Tuna)$

Animal(F(x)) \vee Loves(G(x),x)
 $\neg Loves(x,F(x)) \vee Loves(G(x),x)$
 $\neg Animal(y) \vee \neg Kills(x,y) \vee \neg Loves(z,x)$
 $\neg Animal(x) \vee Loves(Jack,x)$
 Kills(Jack,Tuna) \vee Kills(Curiosity,Tuna)
 Cat(Tuna)
 $\neg Cat(x) \vee Animal(x)$
 $\neg Kills(Curiosity,Tuna)$



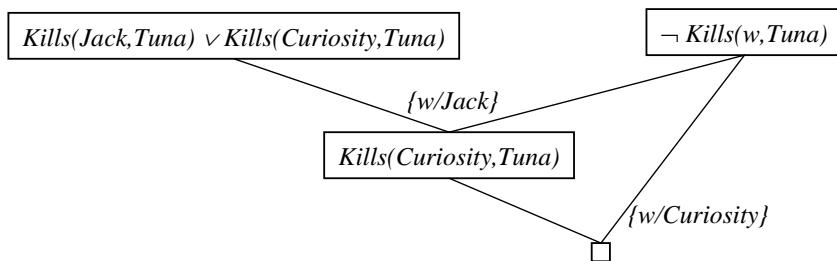
Umělá inteligence I, Roman Barták



Rezoluční metoda

nekonstruktivní důkaz

- Co když bude dotaz formulován jako „**Kdo zabil Tunu?**“



- Odpověď tedy je „**Ano, někdo zabil Tunu**“.
- Můžeme do dotazu přidat speciální **literál pro získání odpovědi**.
 - $\neg \text{Kills}(w, \text{Tuna}) \vee \text{Answer}(w)$
 - předchozí nekonstruktivní důkaz by skončil u $\text{Answer}(\text{Curiosity}) \vee \text{Answer}(\text{Jack})$
 - je tedy vynucen důkaz se strukturou podobnou pro dotaz
→ $\neg \text{Kills}(\text{Curiosity}, \text{Tuna})$

Umělá inteligence I, Roman Barták



Rezoluční metoda

strategie

Jak **efektivně** hledat rezoluční důkazy?

- **jednotková rezoluce**
 - snahou je získat prázdnou klauzuli, proto je dobré, pokud se nově odvozené klauzule zkracují
 - preferujeme rezoluční krok, kde je jedna z klauzulí jednotková (obsahuje jeden literál)
 - obecně se nelze omezit pouze na jednotkové rezoluce, ale pro Hornovské klauzule je to úplná metoda (vlastně dopředné řetězení)
- **množina podpor**
 - vybrané klauzule, z nichž alespoň jedna se vždy účastní rezoluce (výsledek rezoluce se přidává do množiny podpor)
 - počáteční množina podpor typicky obsahuje negovaný dotaz
- **vstupní rezoluce**
 - každého rezolučního kroku se účastní alespoň jedna klauzule ze vstupu – počáteční KB nebo dotaz
 - není to úplná metoda
- **subsumce**
 - eliminuje klauzule, které jsou zahrnuty jinými klauzulemi
 - máme-li v bázi znalostí $P(x)$, nemá cenu přidávat $P(A) \vee Q(B)$

Umělá inteligence I, Roman Barták

11

Umělá inteligence I



Roman Barták, KTI ML

roman.bartak@mff.cuni.cz

<http://ktiml.mff.cuni.cz/~bartak>



Dnes

Umíme **reprezentovat znalosti** v logice a
odvozovat nové znalosti.

Jak ale reprezentace znalostí vypadá v praxi?

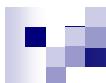
■ **Znalostní inženýrství**

- postup tvorby báze znalostí

■ **Reprezentace znalostí**

- kategorie a taxonomie
- akce a plány
 - situační kalkulus





- **Znalostní inženýrství** se zabývá procesem, jak obecně budovat znalostní báze.
- **Znalostní inženýr** musí:
 - **pochopit** příslušnou problémovou **oblast** (doménu)
 - Jak příslušná oblast funguje?
 - typicky ve spolupráci s expertem v dané oblasti
 - **určit** jaké **koncepty** jsou v ní důležité pro řešení problémů
 - Jaké otázky budeme klást a co potřebujeme znát pro nalezení odpovědi?
 - **navrhnut** formální **reprezentaci** objektů a relací
 - Jak vše formálně zakódovat, aby si s tím počítač poradil?



Umělá inteligence I, Roman Barták



1. **identifikace úlohy**
 - Jaké otázky budeme klást do znalostní báze?
 - Wumpus: vybíráme akce nebo se jen ptáme na vlastnosti prostředí?
2. **sestavení (získání) relevantních znalostí** (knowledge acquisition)
 - Jak daná doména skutečně funguje?
 - Wumpus: co znamená vítr a zápach?
3. **rozhodnutí o slovníku predikátů, funkcí a konstant**
 - Jak přeložit koncepty světa na logické termíny?
 - Wumpus: je díra elementární fakt nebo funkce buňky?
 - Výsledkem je **ontologie** dané domény (slovník používaných pojmu).
4. **zakódování obecných informací o doméně**
 - Jaké axiomy v doméně platí?
 - Wumpus: vítr znamená díru v okolní buňce
5. **zakódování specifické problémové instance**
 - Co aktuálně víme o stavu domény?
 - Wumpus: stojíme na buňce (1,1) s pohledem doprava.
6. **kladení otázek inferenčnímu mechanismu a získání odpovědí**
 - Jak funguje obecný inferenční mechanismus s naší bází znalostí?
 - Wumpus: je buňka (2,2) opravdu bezpečná?
7. **ladění znalostní báze**
 - Co (jaké axiomy) jsme zapomněli uvést v bázi znalostí?
 - Wumpus: ve světě žije jediný Wumpus



Umělá inteligence I, Roman Barták

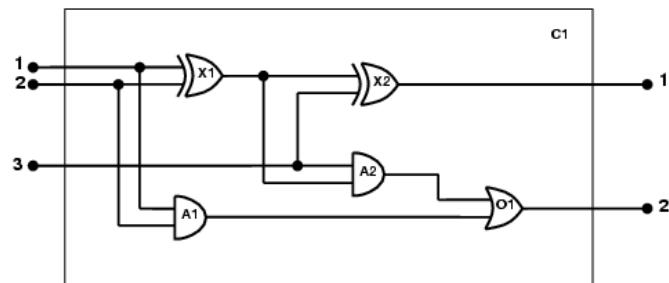


Elektronické obvody

identifikace úlohy (1/7)

Bitová sčítačka

- 1 a 2 jsou vstupní bity,
3 je vstupní přenosový bit
- 1 je výstupní bit pro součet,
2 je výstupní přenosový bit



Co nás bude zajímat?

- Sčítá správně?
- Jak vypadá výstup pro daný vstup?
- Jak vypadá vstup pro požadovaný výstup?
- Jiný typ otázek může vyžadovat jiný typ znalostí.**
 - Kolik takový čip stojí?
 - Kolik plochy zabere?
 - Jakou má spotřebu?



Umělá inteligence I, Roman Barták



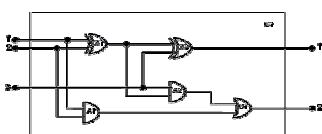
knowledge acquisition

Elektronické obvody

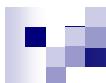
získání znalostí (2/7)

Co víme o elektronických obvodech?

- obvody se skládají z drátů a hradel
- mezi hradly putují signály 0 a 1 přes dráty
- dráty přivádí signál na vstup(y) hradla
- každé hradlo má jeden výstup,
jehož hodnota je dána vstupy a typem hradla
- máme čtyři typy hradel: AND, OR, XOR, NOT
- obvod má také vstupy a výstupy
- dráty nás zde zajímají pouze jako spojnice vstupů a výstupů
- neuvažujeme zpoždění signálu, spotřebu a tvar hradel



Umělá inteligence I, Roman Barták

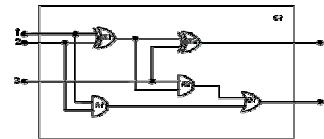


Elektronické obvody

slovník pojmu (3/7)

Jaké budou konstanty, funkce, a predikáty?

- potřebujeme popisovat obvody, hradla, vstupy, výstupy, signály a spojnice
 - **hradla** můžeme označit **konstantami X_1, X_2, A_1, \dots**
 - není potřeba popisovat chování každého hradla zvlášť, chování záleží jen na **typu hradla**
 - zavedeme konstanty AND, OR, XOR, NOT
 - typ hradla určíme **funkcí $Type(X_1) = XOR$**
 - můžeme použít i predikáty typu $Type(X_1, XOR)$ nebo $XOR(X_1)$
 - Pozor! Budeme potřebovat axiomy, že typ hradla je jedinečný.
 - **vstupy a výstupy** hradel můžeme také pojmenovat konstantami ($X_1 In_1, \dots$), ale stejně je musím nějak navázat na konkrétní hradlo
 - asi lepší bude opět použít **funkci $In(1, X_1), \dots$**
 - **spojnice** můžeme popisovat **predikáty**
 - **Connected($Out(1, X_1), In(1, X_2)$)**, ...
 - Pozor! Spojujeme vstupy a výstupy (ne hradla).
 - **signály** na vstupech a výstupech určíme **funkcí**
 - **Signal(g) = 1**



Umělá inteligence I, Roman Barták



Elektronické obvody

kódování obecných znalostí (4/7)

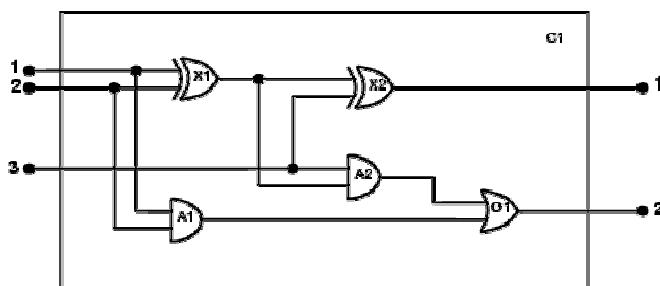
- **Signály na koncích spojnice jsou totožné**
 - $\forall t_1, t_2 \text{ Connected}(t_1, t_2) \Rightarrow \text{Signal}(t_1) = \text{Signal}(t_2)$
- **Signál je pouze tvaru 0 nebo 1, ale ne obojí**
 - $\forall t \text{ Signal}(t) = 1 \vee \text{Signal}(t) = 0$
 - $1 \neq 0$
- **Spojnice je komutativní**
 - $\forall t_1, t_2 \text{ Connected}(t_1, t_2) \Rightarrow \text{Connected}(t_2, t_1)$
- **Chování hradla je určeno jeho typem**
 - $\forall g \text{ Type}(g) = \text{OR} \Rightarrow \text{Signal}(\text{Out}(1, g)) = 1 \Leftrightarrow \exists n \text{ Signal}(\text{In}(n, g)) = 1$
 - $\forall g \text{ Type}(g) = \text{AND} \Rightarrow \text{Signal}(\text{Out}(1, g)) = 0 \Leftrightarrow \exists n \text{ Signal}(\text{In}(n, g)) = 0$
 - $\forall g \text{ Type}(g) = \text{XOR} \Rightarrow \text{Signal}(\text{Out}(1, g)) = 1 \Leftrightarrow \text{Signal}(\text{In}(1, g)) \neq \text{Signal}(\text{In}(2, g))$
 - $\forall g \text{ Type}(g) = \text{NOT} \Rightarrow \text{Signal}(\text{Out}(1, g)) \neq \text{Signal}(\text{In}(1, g))$

Umělá inteligence I, Roman Barták



Elektronické obvody

kódování instance problému (5/7)



Type(X_1) = XOR
Type(X_2) = XOR
Type(A_1) = AND
Type(A_2) = AND
Type(O_1) = OR

Connected(Out(1, X_1),In(1, X_2))
Connected(Out(1, X_1),In(2, A_2))
Connected(Out(1, A_2),In(1, O_1))
Connected(Out(1, A_1),In(2, O_1))
Connected(Out(1, X_2),Out(1, C_1))
Connected(Out(1, O_1),Out(2, C_1))

Connected(In(1, C_1),In(1, X_1))
Connected(In(1, C_1),In(1, A_1))
Connected(In(2, C_1),In(2, X_1))
Connected(In(2, C_1),In(2, A_1))
Connected(In(3, C_1),In(2, X_2))
Connected(In(3, C_1),In(1, A_2))

Umělá inteligence I, Roman Barták



Elektronické obvody

dotazy a ladění (6,7/7)

Dotazy klademe v podobě **logické formule**.

- Co musí být na vstupu, abychom dostali výstup 0 s přenosem 1?

$\exists i_1, i_2, i_3 \text{ Signal}(In(1,C_1)) = i_1 \wedge \text{Signal}(In(2,C_1)) = i_2 \wedge \text{Signal}(In(3,C_1)) = i_3 \wedge \text{Signal}(Out(1,C_1)) = 0 \wedge \text{Signal}(Out(2,C_1)) = 1$

Odpověď je v podobě **substituce proměnných** i_1, i_2, i_3 .

$\{i_1/1, i_2/1, i_3/0\}, \{i_1/1, i_2/0, i_3/1\}, \{i_1/0, i_2/1, i_3/1\}$

Ladění báze znalostí

- Dotazy, které dávají nečekanou (špatnou) odpověď, indikují nějaký problém v bázi znalosti (špatný axiom).
 - Typickým příkladem je chybějící axiom říkající, že dvě různé konstanty označují různé objekty.
 - $1 \neq 0$

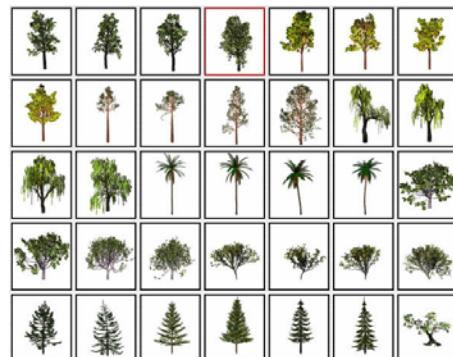


Umělá inteligence I, Roman Barták



Objekty a kategorie

- Všimněme si, že
 - agent **pracuje** s reálnými **objekty**
 - ale **uvažování** provádí na úrovni **kategorií** objektů
 - Agent z pozorování světa odvodí (na základě vnímaných vlastností) pro daný objekt jeho příslušnost do dané kategorie a potom používá informaci o této kategorii k dělání předpovědí o objektu.
- **Kategorie**
 - = množina svých členů
 - = komplexní objekt s relacemi
 - být členem (MemberOf)
 - být podmnožinou (SubsetOf)



Umělá inteligence I, Roman Barták



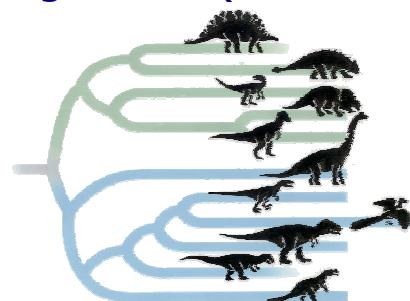
Kategorie a logika

Jak reprezentovat kategorie logickým způsobem?

- objekt je **členem** kategorie
 - MemberOf(BB₁₂,Basketballs)
- kategorie je **podtřídou** jiné kategorie
 - SubsetOf(Basketballs,Balls)
- všichni členové kategorie mají nějakou **vlastnost**
 - $\forall x (\text{MemberOf}(x,\text{Basketballs}) \Rightarrow \text{Round}(x))$
- všichni členové kategorie jsou **rozpoznatelní** na základě společných vlastností
 - $\forall x (\text{Orange}(x) \wedge \text{Round}(x) \wedge \text{Diameter}(x)=9.5\text{in} \wedge \text{MemberOf}(x,\text{Balls}) \Rightarrow \text{MemberOf}(x,\text{BasketBalls}))$
- kategorie jako celek může mít nějakou vlastnost
 - MemberOf(Dogs,DomesticatedSpecies)

Umělá inteligence I, Roman Barták

- Kategorie organizují a zjednodušují bázi znalostí prostřednictvím **dědění vlastností**.
 - vlastnost definujeme pro kategorii, ale dědí ji všichni členové kategorie
 - potrava je jedlá, ovoce ke potrava, jablka jsou ovoce, tudíž všechna jablka jsou jedlá
- Podtřídy organizují kategorie do **taxonomie**
 - hierarchická struktura sloužící pro kategorizaci objektů
 - původně kategorizace všech živých organismů (alfa taxonomie)
 - kategorizace veškerého vědění
 - klasifikace v knihovnictví
 - Dewey Decimal Classification
 - 330.94 European economy



Umělá inteligence I, Roman Barták

Akce a situace

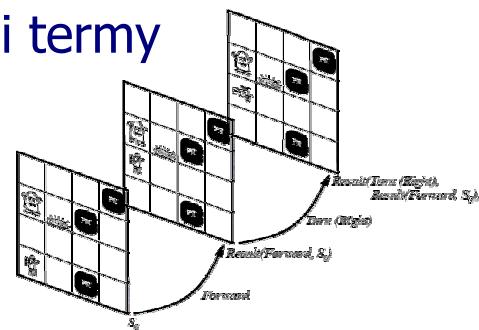
- Zatím jsme se soustředili na popis znalostí o statickém světě.
- **Jak ale uvažovat o akcích a jejich důsledcích?**
- Ve výrokové logice potřebujeme kopii každé akce pro každý čas (situaci):
 - $L_{x,y}^t \wedge \text{FacingRight}^t \wedge \text{Forward}^t \Rightarrow L_{x+1,y}^{t+1}$
 - potřebujeme horní limit na počet časových kroků a i tak dostaneme obrovské množství formulí
- Můžeme akce reprezentovat lépe v logice predikátové?
 - kopiím axiomů popisujícím akce se můžeme vyhnout univerzální kvantifikací přes čas (situace)
 - $\forall t P$ je výsledkem v čase $t+1$ provedení akce A



Situační kalkulus

■ akce reprezentujeme logickými termíny

- Go(x,y)
- Grab(g)
- Release(g)



■ situace jsou logické termíny

- počáteční situace: S_0
- situace po aplikaci akce a na situaci s: Result(a,s)

■ flexibilní predikáty a funkce (fluents), které se mění s časem

- situace je v posledním argumentu
- Holding(G, S_0)

■ neměnné (rigid, eternal) predikáty a funkce

- Gold(G)
- Adjacent(x,y)

Umělá inteligence I, Roman Barták

Plány

■ Je užitečné uvažovat také o posloupnostech (seznamech) akcí – **plánech**.

- $\text{Result}([], s) = s$
- $\text{Result}([a|seq], s) = \text{Result}(seq, \text{Result}(a, s))$

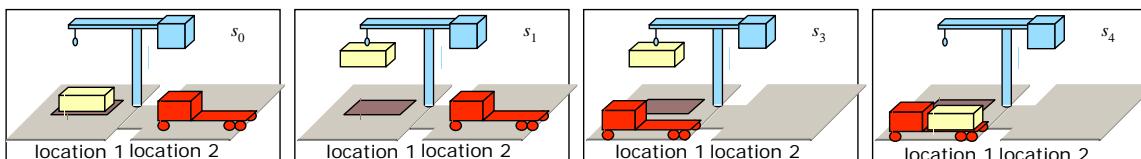
■ Jaké úlohy s plány bude agent řešit?

- **projekční úloha** – jaká je výsledná situace po aplikování dané posloupnosti akcí?

- At(Agent, [1,1], S_0) \wedge At(G, [1,1], S_0) \wedge \neg Holding(o, S_0)
 - At(G, [1,1], $\text{Result}([\text{Go}([1,1],[1,2]), \text{Grab}(G), \text{Go}([1,2],[1,1])], S_0)$)

- **plánovací úloha** – jaká posloupnost akcí vede k dané situaci?

- $\exists seq \ At(G, [1,1], \text{Result}(seq, } S_0))$



Umělá inteligence I, Roman Barták



Reprezentace akce

- Akci můžeme popsat dvěma axiomy:
 - **axiom použitelnosti** $\text{Preconditions} \Rightarrow \text{Poss}(a,s)$
 - $\text{At}(\text{Agent},x,s) \wedge \text{Adjacent}(x,y) \Rightarrow \text{Poss}(\text{Go}(x,y),s)$
 - $\text{Gold}(g) \wedge \text{At}(\text{Agent},x,s) \wedge \text{At}(g,x,s) \Rightarrow \text{Poss}(\text{Grab}(g),s)$
 - $\text{Holding}(g,s) \Rightarrow \text{Poss}(\text{Release}(g),s)$
 - **axiom efektu** $\text{Poss}(a,s) \Rightarrow \text{Changes}$
 - $\text{Poss}(\text{Go}(x,y),s) \Rightarrow \text{At}(\text{Agent},y,\text{Result}(\text{Go}(x,y),s))$
 - $\text{Poss}(\text{Grab}(g),s) \Rightarrow \text{Holding}(g,\text{Result}(\text{Grab}(g),s))$
 - $\text{Poss}(\text{Release}(g),s) \Rightarrow \neg \text{Holding}(g,\text{Result}(\text{Release}(g),s))$
- Pozor! To nám ještě nestačí, abychom mohli odvodit, že plán vede k cíli.
 - Axiomy efektu popisují, co se ve světě mění, ale neříkají, že vše ostatní se nemění!
 - odvodíme $\text{At}(\text{Agent}, [1,2], \text{Result}(\text{Go}([1,1],[1,2]), S_0))$
 - ale neodvodíme $\text{At}(G, [1,2], \text{Result}(\text{Go}([1,1],[1,2]), S_0))$
 - **problém rámce** (frame problem)

Umělá inteligence I, Roman Barták



frame problem

Problém rámce

- Reprezentace všeho, co se po provedení akce nemění.
- Jednoduchý **axiom rámce**, který říká, co se nemění:
$$\text{At}(o,x,s) \wedge o \neq \text{Agent} \wedge \neg \text{Holding}(o,s) \Rightarrow \text{At}(o,x,\text{Result}(\text{Go}(y,z),s))$$
 - pro F flexibilních predikátů a A akcí potřebujeme O(FA) axiomů rámce
 - To je hodně, zvlášť když si uvědomíme, že akce většinu predikátů nemění.



Umělá inteligence I, Roman Barták

Problém rámce

efektivní reprezentace

Lze řešit problém rámce efektivněji (menší počet axiomů)?

■ Axiom následujícího stavu

$\text{Poss}(a,s) \Rightarrow$

(fluent platí v $\text{Result}(a,s)$) \Leftrightarrow

fluent je efektem a \vee (fluent platí v s \wedge a fluent nemění)

- dostaneme F axiomů (F je počet flexibilních predikátů) s celkovým počtem literálů O(AE) (A je počet akcí, E je počet efektů na akci)

Příklady:

$\text{Poss}(a,s) \Rightarrow$

($\text{At}(\text{Agent},y,\text{Result}(a,s)) \Leftrightarrow a=\text{Go}(x,y) \vee (\text{At}(\text{Agent},y,s) \wedge a \neq \text{Go}(y,z))$)

$\text{Poss}(a,s) \Rightarrow$

($\text{Holding}(g,\text{Result}(a,s)) \Leftrightarrow a=\text{Grab}(g) \vee (\text{Holding}(g,s) \wedge a \neq \text{Release}(g))$)

□ Pozor na implicitní efekty!

- Pokud agent něco drží a přesune se jinam, potom se tam přesune také dotyčný objekt.

■ problém důsledku (ramification problem)

$\text{Poss}(a,s) \Rightarrow$

($\text{At}(o,y,\text{Result}(a,s)) \Leftrightarrow$

($a=\text{Go}(x,y) \wedge (o=\text{Agent} \vee \text{Holding}(o,s)) \vee$

($\text{At}(o,y,s) \wedge \neg \exists z (y \neq z \wedge a=\text{Go}(y,z) \wedge (o=\text{Agent} \vee \text{Holding}(o,s)))$))

Umělá inteligence I, Roman Barták

Problém rámce

efektivní projekce

■ Axiom následující stavu je pořád veliký, v průměru má O(AE/F) literálů.

- Čas řešení projekční úlohy délky t (kam se dostaneme danou posloupností akcí) tedy záleží nejen na t, ale i na počtu akcí – O(AEt).
- Pokud v každém kroku známe danou akci, nešlo by to rychleji, třeba O(Et)?

■ Klasický axiom následujícího stavu:

$\text{Poss}(a,s) \Rightarrow$

($F_i(\text{Result}(a,s)) \Leftrightarrow (a=A_1 \vee a=A_2 \vee \dots) \vee (F_i(s) \wedge a \neq A_3 \wedge a \neq A_4 \dots)$)

akce, které mají F_i jako svůj efekt

akce, které mají $\neg F_i$ jako svůj efekt

■ Můžeme zavést pozitivní a negativní efekty akcí

- **PosEffect(a, F_i)** akce a způsobí, že F_i bude pravda

- **NegEffect(a, F_i)** akce a způsobí, že F_i nebude pravda

■ Upravený axiom následujícího stavu:

$\text{Poss}(a,s) \Rightarrow (F_i(\text{Result}(a,s)) \Leftrightarrow \text{PosEffect}(a, F_i) \vee (F_i(s) \wedge \neg \text{NegEffect}(a, F_i)))$

$\text{PosEffect}(A_1, F_i)$

$\text{PosEffect}(A_2, F_i)$

$\text{NegEffect}(A_3, F_i)$

$\text{NegEffect}(A_4, F_i)$

Umělá inteligence I, Roman Barták



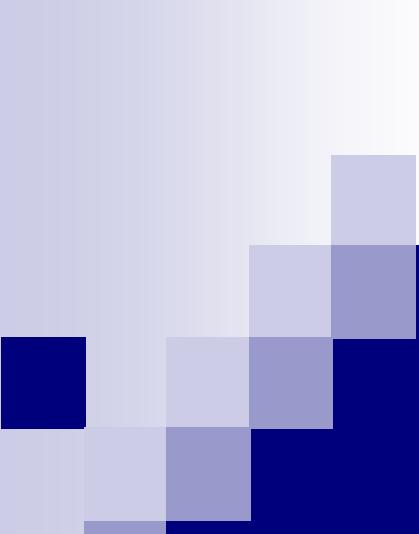
Skryté předpoklady

Příklad:

- Předpokládejme, že máme k dispozici následující tvrzení:
 - „V létě budou vyučovány kurzy CS101, CS102, CS106 a EE101“
 - v řeči predikátové logiky máme fakta
 - Course(CS,101), Course(CS, 102), Course(CS,106), Course(EE,101)
- Kolik bude v létě vyučováno kurzů?
 - někde mezi jedním a nekonečnem!!

Proč?

- obecně předpokládáme, že poskytnutá informace je úplná, tj. že atomická tvrzení, která nejsou uvedena, nejsou pravdivá – **předpoklad uzavřeného světa** (CWA – Closed World Assumption)
- predikátová logika ale takový předpoklad nemá, bázi znalostí je potřeba zúplnit:
$$\text{Course}(d,n) \Leftrightarrow [d,n] = [\text{CS},101] \vee [d,n] = [\text{CS},102] \vee [d,n] = [\text{CS},206] \vee [d,n] = [\text{EE},101]$$
- také jsme předpokládali, že různá jména reprezentují různé objekty – **předpoklad jednoznačnosti jmen** (UNA – Unique Name Assumption)
- opět je potřeba explicitně uvést, že se jedná o různé objekty
 - $[\text{CS},101] \neq [\text{CS},102]$, ...



Umělá inteligence I



Roman Barták, KTI ML

roman.bartak@mff.cuni.cz

<http://ktiml.mff.cuni.cz/~bartak>



12

Plánování

Klasickou plánovací úlohu tedy můžeme řešit odvozovacími metodami predikátové logiky.

Nešlo by to lépe?

■ Reprezentace plánovacího problému

- množiny predikátů (místo formulí)

■ Plánovací algoritmy

- plánování v prostoru stavů

- odvozovací mechanismy z logiky
přizpůsobíme nové reprezentaci
- dopředné a zpětné plánování

- plánování v prostoru plánů

- stavíme plán po kouskách





Klasická reprezentace

- **Klasická reprezentace** plánovacího problému používá **predikátovou logiku**:

- **Stavy** jsou množiny logických atomů, které jsou v dané interpretaci buď pravda nebo nepravda.
- **Akce** jsou reprezentovány plánovacími operátory, které mění pravdivostní hodnotu těchto atomů.

Přesněji:

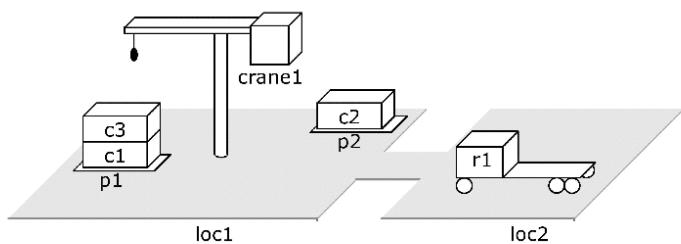
- L (**jazyk**) je konečná množina predikátových symbolů a konstant (nemáme funkce!).
- **Atom** je predikátový symbol s argumenty, např. $\text{on}(c3, c1)$.
- Můžeme používat **proměnné**, např. $\text{on}(x, y)$.

Umělá inteligence I, Roman Barták



Reprezentace stavů

- **Stav je množina instanciovaných atomů** (bez proměnných). Je jich konečně mnoho!



$\{\text{attached}(p1, \text{loc1}), \text{in}(c1, p1), \text{in}(c3, p1),$
 $\text{top}(c3, p1), \text{on}(c3, c1), \text{on}(c1, \text{pallet}), \text{attached}(p2, \text{loc1}), \text{in}(c2, p2), \text{top}(c2, p2),$
 $\text{on}(c2, \text{pallet}), \text{belong}(\text{crane1}, \text{loc1}), \text{empty}(\text{crane1}), \text{adjacent}(\text{loc1}, \text{loc2}), \text{adjacent}(\text{loc2}, \text{loc1}), \text{at}(r1, \text{loc2}), \text{occupied}(\text{loc2}), \text{unloaded}(r1)\}$.

- pravdivostní hodnota některých atomů se mění
 - **flexibilní atomy** (fluent)
 - např. $\text{at}(r1, \text{loc2})$
- některé atomy nemění svojí pravdivostní hodnotu s různými stavami
 - **neměnné atomy** (rigid)
 - např. $\text{adjacent}(\text{loc1}, \text{loc2})$

- **Předpoklad uzavřeného světa** (closed world assumption)
Atom, který není ve stavu explicitně uveden, neplatí!

Umělá inteligence I, Roman Barták



Plánovací operátory

operátor o je trojice:

(name(o), precond(o), effects(o))

□ **name(o): jméno operátoru** ve tvaru $n(x_1, \dots, x_k)$

- n : symbol operátoru (jednoznačný pro každý operátor)
- x_1, \dots, x_k : symboly proměnných (parametry operátoru)
 - musí obsahovat všechny symboly proměnných v operátoru!

□ **precond(o): předpoklady**

- literály, které musí být splnitelné, aby šlo operátor použít

□ **effects(o): efekty**

- literály, které se stanou pravdivými aplikací operátoru (nesmí to být neměnné atomy!)

`take(k, l, c, d, p)`

;; crane k at location l takes c off of d in pile p

precond: `belong(k, l)`, `attached(p, l)`, `empty(k)`, `top(c, p)`, `on(c, d)`

effects: `holding(k, c)`, $\neg \text{empty}(k)$, $\neg \text{in}(c, p)$, $\neg \text{top}(c, p)$, $\neg \text{on}(c, d)$, `top(d, p)`

Umělá inteligence I, Roman Barták

Akce

Akce jsou plně instanciované operátory

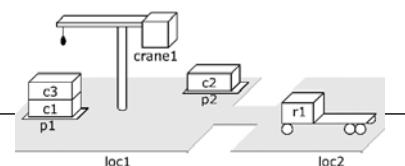
– za proměnné jsou dosazeny konstanty

`take(k, l, c, d, p)`

;; crane k at location l takes c off of d in pile p

precond: `belong(k, l)`, `attached(p, l)`, `empty(k)`, `top(c, p)`, `on(c, d)`

effects: `holding(k, c)`, $\neg \text{empty}(k)$, $\neg \text{in}(c, p)$, $\neg \text{top}(c, p)$, $\neg \text{on}(c, d)$, `top(d, p)`



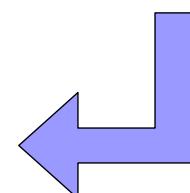
`take(crane1, loc1, c3, c1, p1)`

akce

;; crane $crane1$ at location $loc1$ takes $c3$ off $c1$ in pile $p1$

precond: `belong(crane1, loc1)`, `attached(p1, loc1)`,
 $\text{empty}(\text{crane1})$, `top(c3, p1)`, `on(c3, c1)`

effects: `holding(crane1, c3)`, $\neg \text{empty}(\text{crane1})$, $\neg \text{in}(c3, p1)$,
 $\neg \text{top}(c3, p1)$, $\neg \text{on}(c3, c1)$, `top(c1, p1)`



Umělá inteligence I, Roman Barták

Notace:

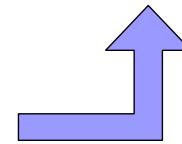
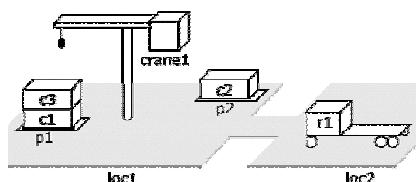
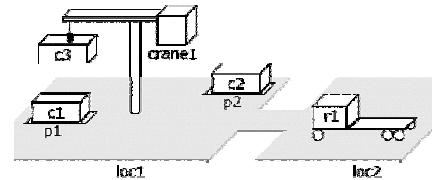
- $S^+ = \{\text{pozitivní atomy v } S\}$
- $S^- = \{\text{atomy, jejichž negace je v } S\}$

Akce a je **použitelná** na stav s právě když
 $\text{precond}^+(a) \subseteq s \quad \& \quad \text{precond}^-(a) \cap s = \emptyset$

Výsledkem aplikace akce a na s je
 $\gamma(s,a) = (s - \text{effects}^-(a)) \cup \text{effects}^+(a)$

Implicitně řeší problém rámce

```
take(crane1,loc1,c3,c1,p1)
;; crane crane1 at location loc1 takes c3 off c1 in pile p1
precond: belong(crane1,loc1), attached(p1,loc1),
empty(crane1), top(c3,p1), on(c3,c1)
effects: holding(crane1,c3), !empty(crane1), !in(c3,p1),
!top(c3,p1), !on(c3,c1), top(c1,p1)
```



Umělá inteligence I, Roman Barták

Plánovací doména

Nechť L je jazyk a O je množina operátorů.

Plánovací doména Σ nad jazykem L a s operátory O
je trojice (S,A,γ) :

- stavy** $S \subseteq P(\{\text{všechny instanciované atomy nad } L\})$
- akce** $A = \{\text{všechny instanciované operátory z } O \text{ nad } L\}$
 - akce a je **použitelná** na stav s , pokud
 $\text{precond}^+(a) \subseteq s \quad \& \quad \text{precond}^-(a) \cap s = \emptyset$
- přechodová funkce** γ :
 - $\gamma(s,a) = (s - \text{effects}^-(a)) \cup \text{effects}^+(a)$, je-li a použitelná na s
 - S je uzavřená vzhledem ke γ (je-li $s \in S$, potom pro každou akci a aplikovatelnou na s platí $\gamma(s,a) \in S$)



Plánovací problém

Plánovací problém P je trojice (Σ, s_0, g) :

- $\Sigma = (S, A, \gamma)$ je plánovací doména
- s_0 je počáteční stav, $s_0 \in S$
- g je množina instanciovaných literálů
 - stav s splňuje g právě tehdy, když $g^+ \subseteq s$ & $g^- \cap s = \emptyset$
 - $S_g = \{s \in S \mid s \text{ splňuje } g\}$ - množina cílových stavů

Zápis plánovacího problému je trojice (O, s_0, g) .

Umělá inteligence I, Roman Barták



Plány a řešení

Plán π je posloupnost akcí $\langle a_1, a_2, \dots, a_k \rangle$.

Plán π je **řešením** P právě když $\gamma(s_0, \pi)$ splňuje g.

■ **Dopředné ověření existence plánu**

Přímí následníci stavu s: $\Gamma(s) = \{\gamma(s, a) \mid a \in A \text{ je aplikovatelná na } s\}$

Dosažitelné stavy: $\Gamma_\infty(s) = \Gamma(s) \cup \Gamma^2(s) \cup \dots$

Plánovací problém má řešení právě když $S_g \cap \Gamma_\infty(s_0) \neq \emptyset$.

■ **Zpětné ověření existence plánu**

Akce a je relevantní pro cíl g právě když:

akce přispívá do g: $g \cap \text{effects}(a) \neq \emptyset$

efekty akce nejsou v konfliktu s g: $g^- \cap \text{effects}^+(a) = \emptyset \wedge g^+ \cap \text{effects}^-(a) = \emptyset$

Regresní (zpětná) množina cíle g pro (relevantní) akci a:

$\gamma^{-1}(g, a) = (g - \text{effects}(a)) \cup \text{precond}(a)$

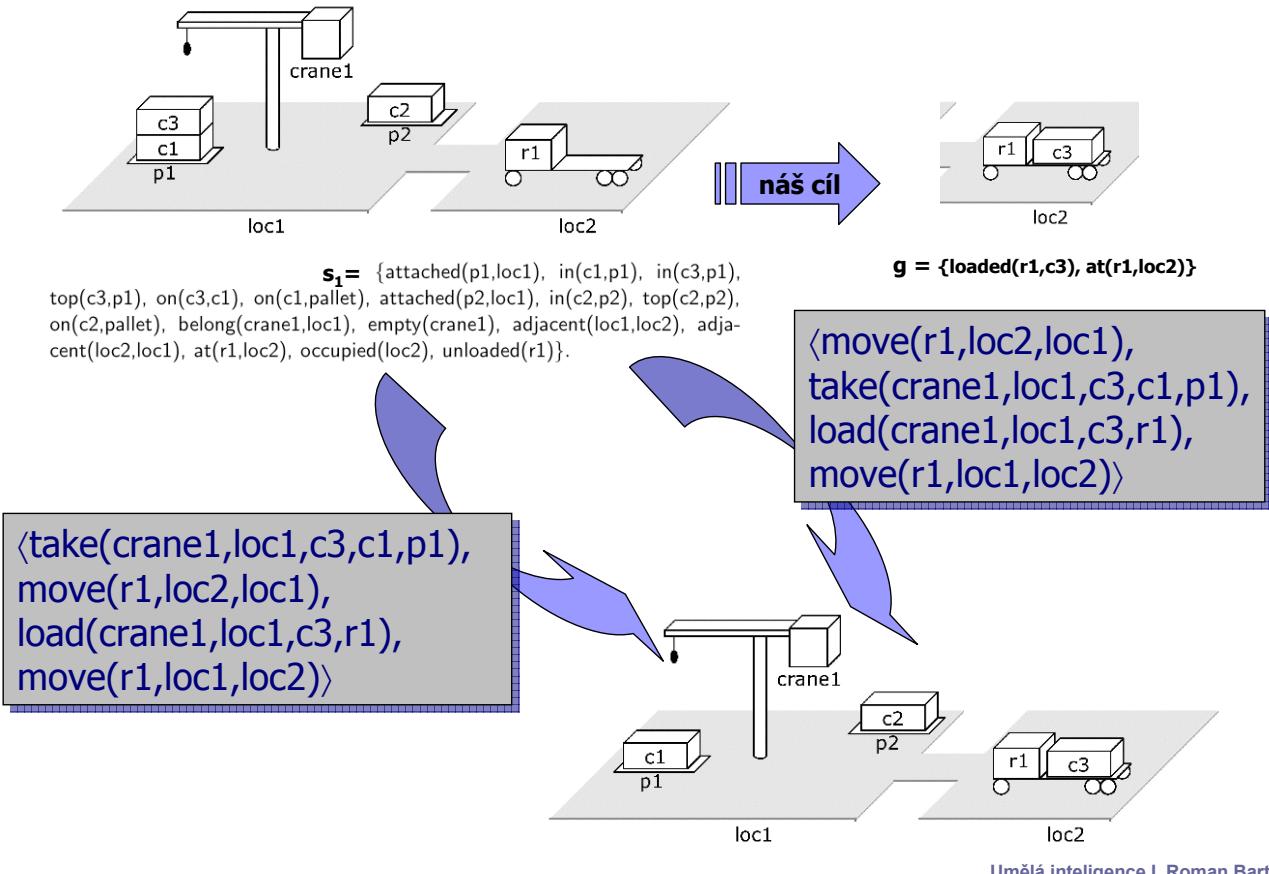
$\Gamma^{-1}(g) = \{\gamma^{-1}(g, a) \mid a \in A \text{ je relevantní pro } g\}$

$\Gamma_\infty^{-1}(g) = \Gamma^{-1}(g) \cup \Gamma^{-2}(g) \cup \dots$

Plánovací problém má řešení právě když s_0 je nadmnožinou nějakého prvku z $\Gamma_\infty^{-1}(g)$.

Umělá inteligence I, Roman Barták

Ukázka plánu



Umělá inteligence I, Roman Barták

Plánování se stavý

- Prohledávaný prostor odpovídá stavovému prostoru plánovacího problému.
 - uzly odpovídají stavům
 - hrany odpovídají stavovým přechodům pomocí akcí
 - cílem je najít cestu mezi počátečním stavem a některým koncovým stavem
- Typy prohledávání
 - dopředné (forward search, progression)
 - začínáme v počátečním stavu a jdeme k některému stavu cílovému
 - zpětné (backward search, regression)
 - začínáme s cílem (pozor to není stav, ale reprezentace množiny stavů!) a jdeme přes podcíle k počátečním stavu
 - liftovaná verze (částečné instanciování akcí, tj. použijeme proměnné)

Umělá inteligence I, Roman Barták

Dopředné plánování

Forward-search(O, s_0, g)

$s \leftarrow s_0$

$\pi \leftarrow$ the empty plan

loop

if s satisfies g then return π

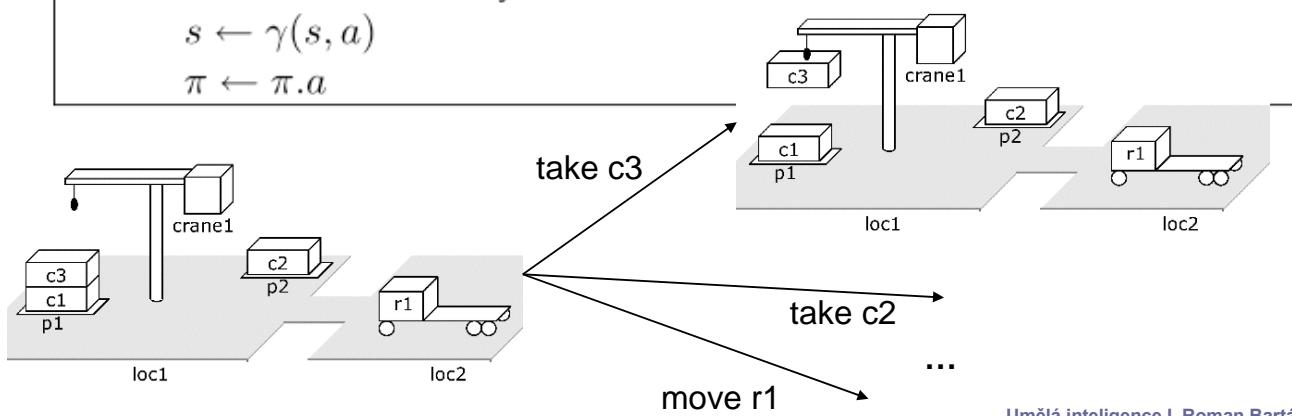
$E \leftarrow \{a | a \text{ is a ground instance an operator in } O, \text{ and precond}(a) \text{ is true in } s\}$

if $E = \emptyset$ then return failure

nondeterministically choose an action $a \in E$

$s \leftarrow \gamma(s, a)$

$\pi \leftarrow \pi.a$

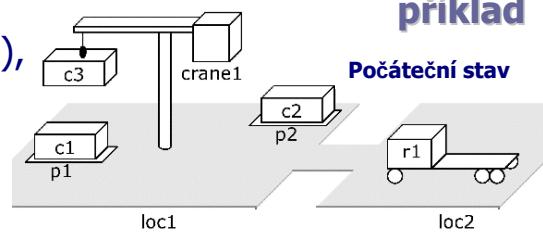


Umělá inteligence I, Roman Barták

Dopředné plánování

příklad

{belong(crane1,loc1), adjacent(loc2,loc1),
holding(crane1,c3), unloaded(r1),
at(r1,loc2), \neg occupied(loc1),
...}



move(r1,loc2,loc1)

move(r, l, m)
;; robot r moves from location l to location m
precond: adjacent(l, m), at(r, l), \neg occupied(l)
effects: at(r, m), occupied(m), \neg occupied(l), \neg at(r, l)

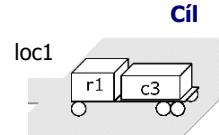
{belong(crane1,loc1),
adjacent(loc2,loc1), holding(crane1,c3), unloaded(r1),
at(r1,loc1), occupied(loc1), \neg occupied(loc2), ...}

load(crane1,loc1,c3,r1)

load(k, l, c, r)
;; crane k at location l loads container c onto 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)

{belong(crane1,loc1), adjacent(loc2,loc1),
empty(crane1), loaded(r1,c3),
at(r1,loc1), occupied(loc1), \neg occupied(loc2), ...}

Cíl = {at(r1,loc1), loaded(r1,c3)}



Umělá inteligence I, Roman Barták

Zpětné plánování

$\text{Backward-search}(O, s_0, g)$

$\pi \leftarrow$ the empty plan

loop

if s_0 satisfies g then return π

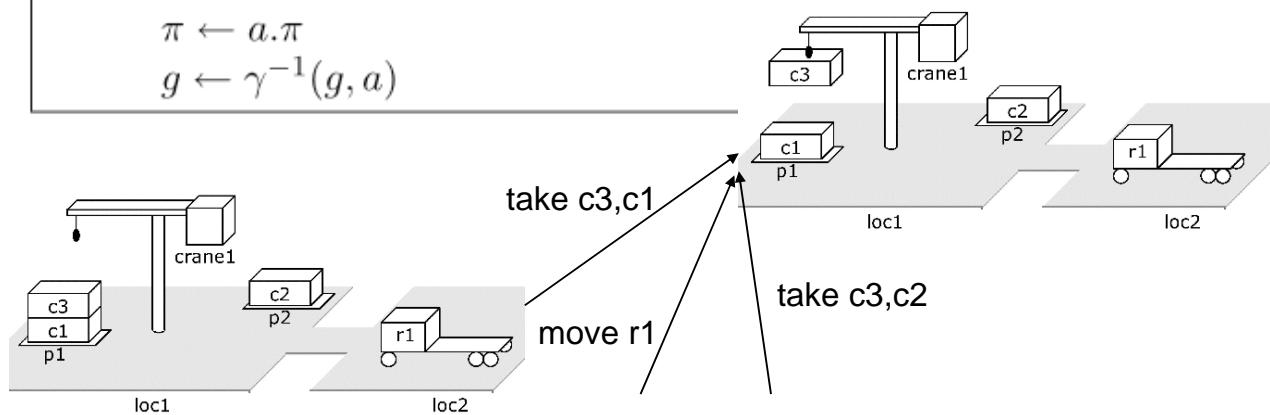
$A \leftarrow \{a | a \text{ is a ground instance of an operator in } O \text{ and } \gamma^{-1}(g, a) \text{ is defined}\}$

if $A = \emptyset$ then return failure

nondeterministically choose an action $a \in A$

$\pi \leftarrow a.\pi$

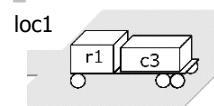
$g \leftarrow \gamma^{-1}(g, a)$



Umělá inteligence I, Roman Barták

Zpětné plánování příklad

Cíl = $\{\text{at}(r1, \text{loc1}), \text{loaded}(r1, c3)\}$



load(crane1, loc1, c3, r1)

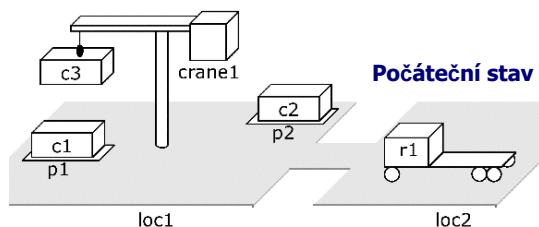
$\text{load}(k, l, c, r)$
;; crane k at location l loads container c onto robot r
precond: $\text{belong}(k, l), \text{holding}(k, c), \text{at}(r, l), \text{unloaded}(r)$
effects: $\text{empty}(k), \neg \text{holding}(k, c), \text{loaded}(r, c), \neg \text{unloaded}(r)$

$\{\text{at}(r1, \text{loc1}), \text{belong}(\text{crane1}, \text{loc1}),$
 $\text{holding}(\text{crane1}, c3), \text{unloaded}(r1)\}$

move(r1, loc2, loc1)

$\text{move}(r, l, m)$
;; robot r moves from location l to location m
precond: $\text{adjacent}(l, m), \text{at}(r, l), \neg \text{occupied}(m)$
effects: $\text{at}(r, m), \text{occupied}(m), \neg \text{occupied}(l), \neg \text{at}(r, l)$

$\{\text{belong}(\text{crane1}, \text{loc1}), \text{holding}(\text{crane1}, c3),$
 $\text{unloaded}(r1),$
 $\text{adjacent}(\text{loc2}, \text{loc1}),$
 $\text{at}(r1, \text{loc2}),$
 $\neg \text{occupied}(\text{loc1})\}$



Umělá inteligence I, Roman Barták



Zpětné plánování-liftování

Lifted-backward-search(O, s_0, g)

$\pi \leftarrow$ the empty plan

loop

if s_0 satisfies g then return π

$A \leftarrow \{(o, \theta) | o \text{ is a standardization of an operator in } O,$
 $\theta \text{ is an mgu for an atom of } g \text{ and an atom of effects } (o),$
 $\text{and } \gamma^{-1}(\theta(g), \theta(o)) \text{ is defined}\}$

if $A = \emptyset$ then return failure

nondeterministically choose a pair $(o, \theta) \in A$

$\pi \leftarrow$ the concatenation of $\theta(o)$ and $\theta(\pi)$

$g \leftarrow \gamma^{-1}(\theta(g), \theta(o))$

- standardizace = kopie s novými proměnnými
- mgu = most general unifier (nejobecnější unifikace)
- použití volných proměnných zmenšuje větvení, ale komplikuje detekci cyklu

Umělá inteligence I, Roman Barták



Plánování v prostoru plánů

- 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

Umělá inteligence I, Roman Barták

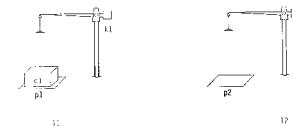


Plánování v prostoru plánů

příklad

- Předpokládejme, že v částečném plánu zatím máme akce:

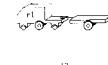
- take(k1,c1,p1,l1)
 - load(k1,c1,r1,l1)



- Možné úpravy plánu:

- Přidání akce**

- aby šlo použít **load**, musí být robot r1 na místě l1
 - přesuň robota r1 na místo l1 **move(r1,l1)**



12

- 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

Umělá inteligence I, Roman Barták



Principy PSP

- 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.

Umělá inteligence I, Roman Barták

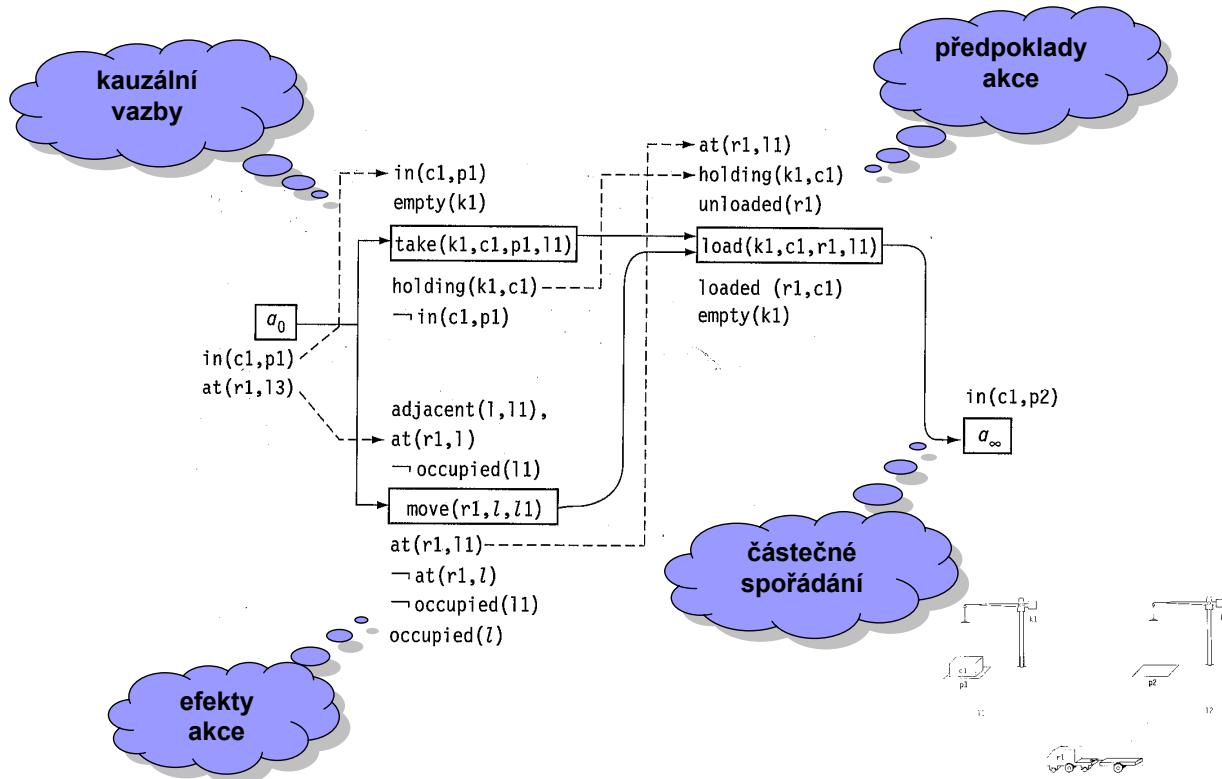
Uzly prohledávaného prostoru jsou tvořeny částečnými plány.

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

- A je množina částečně instanciovaných plánovacích operátorů $\{a_1, \dots, a_k\}$
- $<$ je částečné uspořádání na A ($a_i < a_j$)
- B je množina vazeb tvaru $x=y$, $x \neq y$ 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

Umělá inteligence I, Roman Barták

Částečný plán: příklad



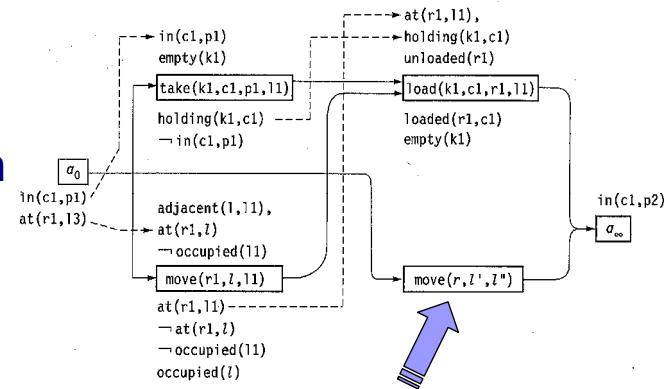
Umělá inteligence I, Roman Barták

- **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 (bud' 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

Umělá inteligence I, Roman Barták

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



Umělá inteligence I, Roman Barták

- **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!

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

Umělá inteligence I, Roman Barták

Procedura PSP

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

```
PSP( $\pi$ )
  flaws  $\leftarrow$  OpenGoals( $\pi$ )  $\cup$  Threats( $\pi$ )
  if flaws =  $\emptyset$  then return( $\pi$ )
    select any flaw  $\phi \in$  flaws
    resolvers  $\leftarrow$  Resolve( $\phi, \pi$ )
    if resolvers =  $\emptyset$  then return(failure)
    nondeterministically choose a resolver  $\rho \in$  resolvers
     $\pi' \leftarrow$  Refine( $\rho, \pi$ )
    return(PSP( $\pi'$ ))
  end
```

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

Umělá inteligence I, Roman Barták

- 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 odstraněné otevřených cílů a hrozeb se používají pouze **konzistentní** zjednodušení plánu.
 - konzistence uspořádání bud' 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

Umělá inteligence I, Roman Barták

Více o plánování

■ Přednáška Plánování a rozvrhování

- <http://kti.mff.cuni.cz/~bartak/planovani/>

Název přednášky: Plánování a rozvrhování
Označení: AIL071
Rozsah: 2/0 Zk - zimní semestr
Vyučující: Roman Barták

Přednáška podává úvod do plánování a rozvrhování. Zaměřena je především na algoritmy pro řešení plánovacích a rozvrhovacích problémů s důrazem na použití technik splňování omezuječích podmínek. Výklad nevyžaduje žádné předběžné znalosti.

Osnova

1. Úvod: definice problému plánování a rozvrhování, příklady použití.
2. Omezuječí podmínky: stručný úvod do technik splňování omezuječích podmínek, globální podmínky, prohledávací algoritmy.
3. Řešení plánovacích problémů: STRIPS reprezentace problému, plánovací graf a jeho použití, heuristiky a strategie řízení, reprezentace plánovacího grafu pomocí omezuječích podmínek, hierarchické sítě úloh.
4. Řešení rozvrhovacích problémů: klasické rozvrhování, Grahamova klasifikace problémů, rozvrhování jako problém splňování omezuječích podmínek, podmínky popisující zdroje a precedence, rozvrhovací strategie.
5. Problémová studie.

Umělá inteligence I, Roman Barták