

Základní algoritmy Prologu

1

Program, který zjistí, zda se v seznamu vyskytují pouze zvolené symboly n(0), ..

```

n(0). n(1). n(3). n(2). n(4). n(5).
n_seznam([]).
n_seznam([H|T]):-n(H),n_seznam(T).
?- n_seznam([5,1,3,2]).           Yes
?- n_seznam([5,1,3,2,9]).        No

?- n_seznam(L).
[], [0], [0,0], [0,0,0], [0,0,0,0], ...

```

Umělá inteligence I.

Jiný program, který zjistí, zda se v seznamu vyskytují pouze zvolené symboly n(0), ..

```

n(0). n(1). n(3). n(2). n(4). n(5).
n_seznamJ([]).
n_seznamJ([H|T]):-n_seznamJ(T),n(H).
?- n_seznamJ([5,1,3,2]).           Yes
?- n_seznamJ([5,1,3,2,9]).        No
?- n_seznamJ(L).
[ ], [0], [1], [3], [2], [4], [5], [0,0], [1,0], [3,0], [2,0], [4,0], [5,0],
[0, 1], [1,1], [3,1], [2,1], [4,1], [5,1],
[0,3], ...

```

Umělá inteligence I.

Principy jazyka Prolog (Programming in Logic):

Charakteristické vlastnosti a základní výpočetní postupy:

- **srovnávání vzorů** (pattern matching) prostřednictvím **UNIFIKACE**
- **resoluce**
- automatický zpětný chod
- využití stromových datových struktur (seznamy) pro **popis objektů**: objekt má tvar složené funkce a lze jej sestavit pouze jedním z následujících 2 způsobů:
 - > Nejjednodušší objekt je libovolná konstanta nebo proměnná.
 - > Jsou-li o_1, \dots, o_k objekty a f k -ární funkce uvažovaného jazyka, pak $f(o_1, \dots, o_k)$ je složený objekt s vedoucím (principal) funktorem f

Umělá inteligence I.

Unifikace

Vstup: 2 objekty (termy) **S, T** **Výstup:** substituce σ (pokud existuje)

*Hledá odpověď na otázku, zda existuje substituce za proměnné vyskytující se v **S** a **T** tak, aby oba výrazy byly totožné (match) – pokud ano, výsledkem unifikace je právě nejobecnější substituce, jinak unifikace neexistuje*

- Jsou-li **S, T konstanty** a pokud jde o totožné objekty, pak σ existuje a je prázdná
- Je-li **S proměnná** a **T libovolný objekt**, pak σ je **S/T** (za **S** dosadit **T**) – podobně je-li **T** proměnná.
- Jsou-li **S i T strukturované objekty**, pak je lze unifikovat pouze tehdy, pokud
 - mají totožný hlavní (principal) funktor,
 - všechny vzájemně si odpovídající komponenty v **S** a **T** lze unifikovat (stejnou substitucí).

Umělá inteligence I.

Se kterým z následujících objektů lze unifikovat term [1,2,3,4]?

Term	Unifikace (odpov. substituce)
1. S	1. S/[1,2,3,4]
2. a	2. neexistuje
3. []	3. neexistuje
4. [H T]	4. H/1, T/[2,3,4]
5. f(1,2,3,4)	5. neexistuje
6. [2 L]	6. neexistuje
7. [X1,X2 Y]	7. X1/1, X2/2, Y/[3,4]
8. 1+2+3	8. neexistuje

Umělá inteligence I.

Úloha unifikace v Prologu

- Předávání parametrů
- Přiřazení hodnoty proměnné
- Konstrukce dat

Program { $p(f(X),X)$. }

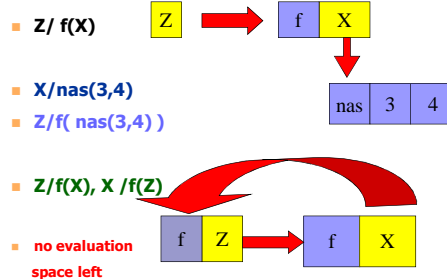
?- $p(f(2),2)$. Yes
 ?- $p(2,2)$. No ?- $p(1,2)$. No
 ?- $p(f(a),a)$. Yes ?- $p(W, f(a))$. $W=f(f(a))$
 ?- $p(Z, Z)$.
 $Z = f(f(f(f(f(f(f(f(f(f(...))))))))))))$ Yes

Umělá inteligence I.

Substituce

jako odkaz v paměti

$p(f(X),X)$.
 ?- $p(Z,nas(3,4))$.



Umělá inteligence I.

Způsoby ukončení logického programu

Zpracování log. programu = vytváření derivačního grafu podle *SLD resoluce* obsahuje 2 zdroje nedeterminismu:

- které pravidlo použít? *prohledávání derivačního grafu do hloubky*
- který podcíl rozvíjet? výběrové pravidlo – „leva-do-prava“

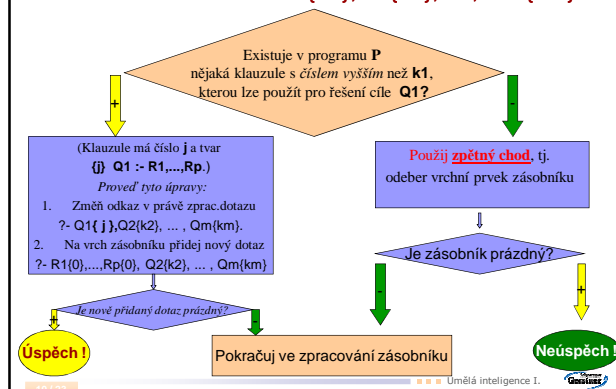
Ukončení:

- výchozí cíl byl splněn ... **ÚSPĚCH (success)**
- všechny možnosti, jak řešit výchozí cíl byly použity a selhaly ... **KONEČNÉ SELHÁNÍ (finite failure)**,
- neukončený výpočet ...

Umělá inteligence I.

Zpracování zásobníku s vrchním prvkem

?- $Q1\{k1\}, Q2\{k2\}, \dots, Qm\{km\}$.



Umělá inteligence I.

Stačí čistý Prolog ?

Napište predikát, který z daného seznamu vynechá všechny výskyt prvku X.

1. vse_pryc (X, [], []).
 2. vse_pryc (X, [X|L1], L2) :- vse_pryc (X, L1, L2).
 3. vse_pryc (X, [Y|L1], [Y|L2]) :- vse_pryc (X, L1, L2).
 ?- vse_pryc (1, [1,2,1,1,4], M). M = [2,4]; ?
 ?- vse_pryc (1, [1,2,1,1,4], M), prvek(1, M). M = [2,1,4]; [2,1,4], [2,1,1,4].

Důvod? Klauzule 3 je „obecnější než klauzule 2“. Algoritmus „*vykonej program*“ jí docela určitě použije při hledání alternativy !!! Abychom se tomu mohli vyhnout, zavádíme **mimologický predikát REZ**, jehož úkolem je **potvrdit jednou přijaté rozhodnutí jako jediné možné**.

Predikát ! (řez) se používá, pokud nám jde o **DETERMINISTICKÉ řešení**

Umělá inteligence I.

Příklad vysvětlující funkci řezu !

- Mějme program P obsahující následující 3 klauzule
 1. $b(X,Y) :- e(X,Y), f(Y), !, g(X), h(Y,Z)$.
 2. $b(X,Y) :- k(X,Y)$.
 3. $p(X) :- a(X), b(X,Y), c(Y), d$.
- Řez fixuje přijaté „částečné řešení“ – omezuje splnění podcílů **vlevo od řezu** (t.j. e, f a b) na **jedinou možnost**.
- **Překročení řezu** zamezí využití ostatních pravidel pro b. Uvažme např., že platí $e(o1,o2)$ & $f(o2)$ pro nějaké konstanty $o1$ a $o2$. V takovém případě se při dotazu ?- $b(X,Y)$, nikdy nedostaneme ke zjišťování, zda platí $k(X,Y)$.
- Řez neovlivňuje zpětný chod vpravo od svého výskytu, t.j. mezi cíli g a h.
- Při řešení složeného cíle ?- $p(X)$, se postupuje takto:
 Pokud podcíl, ve kterém se vyskytuje řez ! **neuspěje** (v našem případě jde o $b(X,Y)$), zpětný chod se vrací k jeho předchůdci (tedy k $a(X)$)

Umělá inteligence I.

Ilustrace použití řezu

```

b(X,Y):- e(X,Y), f(Y), l, g(Y).
b(X,Y):- k(X,Y).

p(X):- a(X), b(X,Y).      r(X,Y):- a(X),l, b(X,Y).

a(o1).  a(o2).  a(o3).
e(o1,v1). e(o2,v2). e(o2,v3). e(o3,v2). e(o3,o3).
g(v3).
f(v1).  f(v3).
k(o1,Z). k(X,X).
    
```

```

?- p(o1).           No.

?- b(o2,Y).        Yes, Y=o2

?- p(X).
    
```

Ilustrace použití řezu

```

b(X,Y):- e(X,Y), f(Y), l, g(Y).
b(X,Y):- k(X,Y).      r(X):- a(X),l, b(X,Y).

a(o1).  a(o2).  a(o3).
e(o1,v1). e(o2,v2). e(o2,v3). e(o3,v2). e(o3,o3).
g(v3).
f(v1).  f(v3).
k(o1,Z). k(X,X).
    
```

```

?- p(X).           ?- a(X), b(X,Y).      ?- b(o1,Y).
                  ?- e(o1,Y), f(Y), l, g(Y). ... ?- g(v1).
    
```

% pozor neplatí g(v1).

/ V tomto okamžiku se již nehledá nová možnost splnit e(o1,Y) ani b(o1,Y) a zpětný chod začíná od a(X) – hledá se další vhodná substituce za X (jiná než konstanta o1) ... */*

X= o2, X=o3

```

?- r(X)           No
    
```

Postup řešení dotazu v Prologu s řezem !

Mějme program **R**, t.j. očíslovaný seznam faktů a klauzulí.

Číslovány jsou nejen klauzule programu, ale i dotazy v zásobníku.

Zatímco klasické dotazy v zásobníku jsou v závorce { } doplněny o číselný parametr, který odpovídá číslu další klauzule programu **R**,

řez je doplněn o údaj **a** označující číslo dotazu v zásobníku, kam je nutné se vrátit při zpětném chodu přes řez (t.j. dotazy s čísly vyššími než **a** jsou pak ze zásobníku odebrány = vymazány).

dotaz **?- D1,D2, ..., Dn**.

Jediný rozdíl ve zpracování zásobníku je při práci s dotazy a klauzulemi, které obsahují řez.

Zásobník dotazů a řez

Založení zásobníku. Dej první prvek do zásobníku, a to

```
{0} ?- D1{0},D2{0}, ..., Dn{0}.
```

Práce s vrchním prvkem zásobníku (jedná se o právě řešený dotaz s číslem **d**)

```
{d} ?- Q1{k1},Q2{k2}, ..., Qm{km}.
```

pokud Q1 není řez, postupuje jako v případě čistého Prologu až na situaci, že existuje v programu **P** klauzule s číslem vyšším než **k1**, kterou lze použít pro řešení cíle **Q1** a tato klauzule má tvar

```
{j} Q1 :- R1{!},...,Rp.
```

V takovém případě proved' následující 2 kroky:

1. uprav odkaz v právě zpracovávaném dotazu takto
?- Q1{ j },Q2{k2}, ..., Qm{km}.

2. a do zásobníku přidej nový dotaz s číslem **d+1**

```
{d+1} ?- R1{0}, ! {d-1}, ...,Rp{0}, Q2{k2},...,Qm{km}.
```

Zpracování zásobníku s vrchním prvkem {c} ?- l{k1},Q2{k2}, ..., Qm{km}.

Je k1 konstanta?

1. uprav právě zpracovávaný dotaz takto
{c} ?- l{jump(k1)},Q2{k2}, ..., Qm{km}
2. a do zásobníku přidej nový dotaz
{c+1} ?- Q2{k2},...,Qm{km}.

Má-li k1 tvar jump(i),
vymaž ze zásobníku
všechny dotazy s číslem vyšším než je **i**
Tato situace může nastat pouze
při zpětném chodu

Pokračuj ve zpracování zásobníku

Příklad: databáze údajů o studentech

```

domaci ('Antoš').           ...
zahranicni ('Akimoto').     zahranicni ('Cozreck').
zahranicni ('Kim').         zahranicni ('Nowak'). ...
jazykovy_test ('Akimoto', 60).
jazykovy_test ('Cozreck', 65).
jazykovy_test ('Kim', 75).
test_odborny ('Antoš', 85).  ...
prijmout(X):- zahranicni(X),
               jazykovy_test(X, ZnamkaX), ZnamkaX >= 65.
    
```

• **Výčet všech kandidátů na přijetí ?**

```
?- prijmout(X).
```

```
X = 'Cozreck'; 'Kim'; ...
```

POZOR na rychlost! jazykovy_test je prohledávan zbytečně, ačkoliv pro každého studenta je zaznamenána jediná známka

Příklad: databáze údajů o studentech - pokračování

```
domaci('Antoš'). ...
zahrancnici('Akimoto'). ...
jazykovy_test('Akimoto', 60). ...
test_odborny('Antoš', 85). ...
prijmout1(X):- zahrancnici(X),
               jazykovy_test(X,ZnamkaX),!,ZnamkaX >= 65.
• Vychet vsech kandidatu na prijati
?- prijmout1(X).
X = No

jazykovy_test1('Akimoto', 60) :- !.
jazykovy_test1('Cozreck', 65) :- !.
jazykovy_test1('Kim', 75) :- !.

prijmout2(X):- zahrancnici(X),
               jazykovy_test1(X,ZnamkaX),ZnamkaX >= 65.
?- prijmout2(X).
X = 'Cozreck': 'Kim' ...
```

```
zahr('Akimoto').
zahr('Cozreck').
zahr('Kim').
zahr('Nowak').
...
j_t('Akimoto', 60).
j_t('Cozreck', 65).
j_t('Kim', 75).
...

```

```
jazykovy_test1('Akimoto', 60) :- !.
jazykovy_test1('Cozreck', 65) :- !.
jazykovy_test1('Kim', 75) :- !.
prijmout2(X):- zahrancnici(X),
               jazykovy_test1(X, ZnamkaX), ZnamkaX >= 65.
?- prijmout2(X).
X = 'Hans Cozreck': 'Kim' ...
• Výsledky jazykových testů všech kandidátů?
?- jazykovy_test1(X, ZnamkaX).
X='Sumi Akimoto', ZnamkaX = 60; no
```

Metapredikát **once (P):- P, !.**

```
prijmout3(X):- zahrancnici(X),
               once(jazykovy_test(X, ZnamkaX),ZnamkaX >= 65).
```

Můžeme najít zahraničního studenta, který potřebuje vyzkoušet?

tj. studenta, který *ještě nedělal jazykový test*
 vyzkouset_jazyk(X)
 :- zahrancnici(X), jazykovy_test(X,_),!, fail.
 vyzkouset_jazyk(X).

Předpoklad uzavřenosti světa:

Pokud fakt není explicitně uveden, předpokládá se, že neplatí.

```
zahr('Akimoto').
zahr('Cozreck').
zahr('Kim').
zahr('Nowak').
...
j_t('Akimoto', 60).
j_t('Cozreck', 65).
j_t('Kim', 75).
...

```

Mimologický predikát not

Definice metapredikátu not(_) pomocí řezu

not (P) :- P, !, fail.
not (P).

Příklad použití:

vyzkouset_jazyk1(X) :- zahrancnici(X), **not** (jazykovy_test(X,_)).

Program P0: student(jan). student(petr). **ženatý(petr).**

svobodný_student1(X) :- **not** (ženatý (X)), student(X).

svobodný_student2(X) :- student(X), **not** (ženatý (X)).

?- svobodný_student1(X). **No** **?- svobodný_student2(X).** X=jan

Not se chová „korektně“ JEN při použití na podcil BEZ VOLNÝCH PROMĚNNÝCH

Role mimologických predikátů při srovnání Prologu a logiky 1. řádu

Program P1:
 student(jan). student(petr). **ženatý(petr).**
 svobodný_student(X) :- **not** (ženatý (X)),student(X).

P1: ?- svobodný_student(jan). **Yes.**

Program P2:
P1 + {ženatý(jan).}

P2: ?- svobodný_student(jan). **No.**

Prolog s not není monotónní, tj. ačkoliv je v programu P dokazatelné tvrzení ϕ , nemusí ϕ být dokazatelné v $P' \supset P$, kde P' je P rozšířené o nějaká další pravidla