

Logické programování I

PROLOG

Program popisuje "svět" Prologu

=

databáze faktů a pravidel (tzv. klauzulí).

fakta: predikát(arg1, arg2, ...argN).

cíle: ?- predikát(arg1, arg2, ...argN).

pravidla: hlava :- tělo.

 predikát(argumenty) :- fakta.

elementy programu:

 konstanty, proměnné, struktury

=

termy

Konstanty — čísla
 \ atomy a, ja_a_ty, “NOVY”, b1

Proměnné A, _X, _

Struktury muz(jan),
 cte(student(jan,novy),kniha(A,Nazev))

2+2

1) tvořené operátory a operandy – obr?

2) tvořené seznamy – výklad později

Př. dum.pro

ma(dum, dvere).

ma(dum, okno).

ma(dvere, klika).

rozbite(dvere).

?-ma(dum, dvere).

?-ma(Neco, klika).

?-ma(dum, Neco).

?-ma(X, Y), rozbite(Y).

Pravidla

`cil(argumenty)`:-logicka formule z podcilu.

$p \leftarrow q_1 \ \& \ q_2 \ \& \ \dots \ \& \ q_n.$

Pr. rodina.pro

`rodic(eva, petr). rodic(eva, jan).`

`rodic(jindrich, jan). zena(eva). muz(petr).`

`muz(jan). muz(jindrich).`

Jak definovat rodinné vztahy?

Predikáty se odlišují dle jména i arity

Anonymní proměnná = nechceme znát její hodnotu

Může jich být více v klauzuli.

Navzájem nesouvisí

Def.otce = někdo, kdo je rodičem a mužem

Def.syna

Databázi lze průběžně doplňovat

Rekurze

Definice českých panovníků – přemyslovců

Premyslovec je premysl_orac

Premyslovec je syn premyslovce

- Jak vytvořit dotaz na jména všech přemyslovců?
- Jak definovat potomka po meči?
- Jak definovat příbuznost osob X a Y po meči?
- Jak definovat příbuznost osob X a Y

Princip rezoluce

$a :- a_1, a_2, \dots, a_n.$

$b :- b_1, b_2, \dots, b_m.$

Necht' $b_i \equiv a$

Pak rezolucí je

$b :- b_1, b_2, \dots, b_{i-1}, a_1, a_2, \dots, a_n, b_{i+1}, \dots, b_m.$

Unifikace

- porovná-li se volná proměnná s konstantou, naváže se na tuto konstantu,
- porovnají-li se dvě volné (neinstalované) proměnné, stanou se synonymy,
- porovná-li se volná proměnná s termem, naváže se na tento term,
- porovnají-li se termy, které nejsou volnými proměnnými, musí být pro úspěšné porovnání stejné.

Př unifikace v dotazu $?- X = Y, Y = a.$

Příklad největšího společného dělitele

Největší společný dělitel A a A je A

Největší společný dělitel A a B je NSD jestliže

 při A větším než B platí: A_1 je $A-B$ a největší společný dělitel
 A_1 a B je NSD

 při A menším než B platí B_1 je $B-A$ a největší společný dělitel
 B_1 a A je NSD

Příklad faktoriálu

Faktoriál 1 je 1.

Faktoriál N je F , jestliže platí, že nějaké M má hodnotu $N-1$ a současně faktoriál M je G a současně F má hodnotu $G \cdot N$

Zásady při plnění cílů

- Dotaz může být složen z několika cílů.
- Při konjunkci cílů jsou cíle plněny postupně zleva.
- Pro každý cíl je při jeho plnění prohledávána databáze od začátku.
- Při úspěšném porovnání klauzule s cílem je její místo v databázi označeno ukazatelem. Každý z cílů má vlastní ukazatel.
- Při úspěšném porovnání cíle s hlavou pravidla, pokračuje výpočet plněním cílů zadaných tělem pravidla.
- Cíl je splněn, je-li úspěšně porovnán s faktem databáze, nebo s hlavou pravidla databáze a jsou splněny podcíle těla pravidla.
- Není-li během exekuce některý cíl splněn ani po prohlédnutí celé databáze, je aktivován mechanismus návratu.
- Splněním jednotlivých cílů dotazu je splněn globální cíl a systém vypíše hodnoty proměnných zadaných v dotazu.
- Zjistí-li se při výpočtu, že globální cíl nelze splnit, je výsledkem no.

Mechanismus návratu

- exekuce se vrací k předchozímu splněnému cíli, zruší se instalace proměnných a pokouší se opětovně splnit tento cíl prohledáváním databáze dále od ukazatele pro tento cíl,
- splní-li se opětovně tento cíl, pokračuje se plněním dalšího, (předtím nesplněného) vpravo stojícího cíle,
- nesplní-li se předchozí cíl, vrací se výpočet opětovně zpět.

Shrnutí základních principů

- **Program specifikuje množinou klauzulí. Klauzule mají podobu faktů, pravidel a dotazu. Prolog zná pouze to, co je definované programem.**
- **Fakt je jméno relace a argumenty (objekty) v daném uspořádání. Uspořádání je důležité.**
- **Pravidlo vyjadřuje vztahy, které platí jsou-li splněny podmínky z těla (cíle). Hlavu tvoří vždy jen jeden predikát.**
- **Dotaz může tvořit jeden nebo více cílů. Cíle mohou obsahovat proměnné i konstanty, Prolog najde tolik odpovědí kolik je požadováno (pokud existují).**
- **Proměnná je v klauzuli obecně kvantifikována. Její platnost je omezena na klauzuli.**

- **Definice predikátu je posloupnost klauzulí pro jednu relaci. Predikát může určovat vztah, databázovou relaci, typ, vlastnost, funkci. Jméno predikátu musí být atomem.**
- **Plnění cíle provádí Prolog pro nový cíl prohledáváním databáze od začátku, při opakovaném pokusu prohledáváním od naposled použité klauzule.**
- **Rekurzivní definice predikátu musí obsahovat ukončovací podmínku.**
- **Typ termu je rozpoznatelný syntaxí. Atomy a čísla jsou konstanty. Atomy a proměnné jsou jednoduchými termy. Anonymní proměnná představuje neznámý objekt, který nás nezajímá. Struktury jsou složené typy dat. Pravidlo je strukturou s funktorem :-**
- **Funktor je určen jménem a aritou**

Unifikace termů

Dva termy jsou úspěšně porovnány (podobné), pokud

- jsou totožné nebo
- proměnné v termech lze navázat na objekty tak, že po navázání proměnných jsou termy totožné.

Př. datum(D, M, 2003) datum(X, 12, R)

datum(D, M, 2003) datum(X, 12, 2004)

bod(X, Y, Z) datum(D, M, 2003)

- Prolog vybere vždy nejobecnější možnost porovnání
- Porovnání vyjadřuje operátor =

Při porovnání proměnné se strukturou je nutné vyloučit případ, kdy se tato proměnná vyskytuje ve výrazu

**Př. ?- X = f(X). %neproveditelné porovnání
Způsobí navázání X na f(X)**

na f(X)

na f(X) ...

!! Aritmetické výrazy jsou termy !!

$$?- X = +(2, 2).$$

$$?- X = 2 + 2 .$$

Pro vyhodnocení nutno použít is

$$?- X is +(2, 2).$$

$$?- X is 2 + 2 .$$

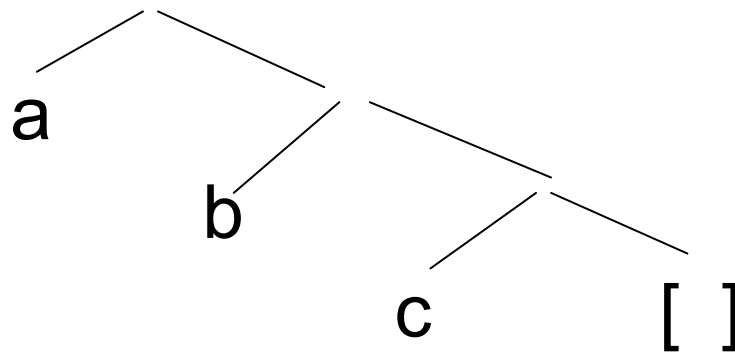
Seznamy

Seznam je rekurzivní datová struktura tvaru:

$[e_1, e_2, \dots, e_n]$

- [Hlava | Zbytek]
- [] prázdný seznam

Př. [a, b, c] dtto [a | [b, c]] dtto [a, b | [c]] dtto [a, b, c | []]



$$[X] = [a, b, c, d] \quad ?$$

$$[X | Y] = [a, b, c, d] \quad ?$$

$$[X, Y | Z] = [a, b, c, d] \quad ?$$

$$[X] = [[a, b, c, d]] \quad ?$$

$$[X | Y] = [[a, [b, c]], d] \quad ?$$

$$[X | Y] = [] \quad ?$$

$$[X | Y] = [d] \quad ?$$

Predikáty pro práci se seznamy

Zjištění, zda v nejvyšší úrovni seznamu S existuje prvek X

member(Prvek,Seznam)

Member platí, je-li prvek na začátku seznamu,
jinak platí pokud prvek je ve zbytku seznamu

?-member(a, [b,a,c,[d,a]]).

?-member(a, [b,c,[d,a]]).

Je mezi standardními

Nalezení posledního prvku seznamu

last(Seznam, Prvek)

Je-li v seznamu jen jeden prvek, tak je tím posledním, jinak je to poslední prvek ze zbytku seznamu

?- last([a,[b,c]],X).

Odstranění prvku ze seznamu

delete(Původní, Výsledný, Prvek)

delete([X|T],T,X).

delete([Y|T],[Y|T1],X) :- delete(T,T1,X).

Zkuste formulovat slovně

?- delete([a,b,a],L,a).

Přidání seznamu k seznamu

append(Seznam1,Seznam2,Výsledek)

append([],X,X).

append([A|B],X,[A|C]):-append(B,X,C).

Formulujte predikát slovně

?- append([a,b],[c],X).

X = [a,b,c]

yes

?-

append([],X,X).

append([A|B],X,[A|C]):-append(B,X,C).

?- append(X,Y,[a,b,c]).

argumenty

příklad

bbb **?-append([a, b], [c], [a, b, c]).**

bbf **?-append([a, b], [c], S3).**

bfb **?-append([a, b], S2, [a,b,c,d]).**

bff **?-append([a, b], S2, S3).**

fbf **?-append(S1, [c, d], [a,b,c,d]).**

fbf

?-append(S1, [c, d], S3).

ffb

?-append(S1, S2, [c, d]).

fff

?-append(S1, S2, S3).

Vícesměrnost dalších predikátů

member(X,[X|_]).

member(X,[_|Y]) :- member(X,Y).

?- member(X,[a,[b,c],d]).

Vícesměrnost dalších predikátů

delete([X|T],T,X).

delete([Y|T],[Y|T1],X) :- delete(T,T1,X).

?- delete(X,[a,b],c).

Seznamy znaků jsou řetězce

?- [X,Y|Z]="abcd".

X = 97

Y = 98

Z = [99,100]

yes

?-