ÚVOD DO PROGRAMOVÁNÍ V PROLOGU

RNDr. Jiří Matyska

1. Úvod

Slovo PROLOG je zkratkou od anglického termínu "Programming in Logic". Již z tohoto názvu je zřejmé, že jeho využití je především v oblastech jako jsou např. relační databáze, matematická logika, řešení abstraktních problémů, zpracování přirozeného jazyka, symbolické řešení rovnic a hlavně mnoho oblastí z umělé inteligence.

Na rozdíl od tradičních programovacích jazyků, jako např. FORTRAN, PL/I, PASCAL aj., které mají jistý souhrn příkazů, pomocí kterých provádějí různá dosazování hodnot do proměnných, pracuje PROLOG s uživatelem definovanou databází klauzulí. Po naplnění databáze klauzulemi komunikuje PROLOG s uživatelem, který mu klade dotazy týkající se obsahu databáze. Prolog odpovídá na naše dotazy buď stručně yes/no nebo zjišťuje, za jakých podmínek je odpověď yes. Veškerý další popis jazyka vychází z učebnice./1/.

2. Popis jazyka

2.1. Syntaxe

Základním stavebním kamenem pro zápis Prologovských programů jsou termy. Term je buď konstanta, proměnná nebo struktura. Každý term je tvořen posloupností znaků. Přípustné znaky v Prologu jsou velká písmena, malá písmena, číslice a speciální znaky jako např. + - ( ) atd.

Konstanty slouží k pojmenování určitých objektů nebo spe-

cifických vztahů mezi objekty. Dělíme je na dva druhy:
atomy a přirozená čísla.
Atomy jsou jednak jména použitá k označení určitých objektů a jednak speciální prologovské symboly, z nichž některé vysvětlíme později. Jména smí obsahovat pouze písmena, číslice a podtržítko a musí začínat malým písmenem. Příklady konstant jsou

eva ma_rada deti a kvetiny = -- ?- :- 123 0 16383

Následující výrazy nejsou konstanty:

123A ma_rada Konstanta _a

Pro zápis proměnných platí stejná pravidla jako pro zápis atomů s tím rozdílem, že proměnná začíná velkým písmenem nebo podtržítkem _. Příkladem mohou sloužit:

X Y Kdo__a

Struktura je jeden objekt, který je tvořen uspořádanou n-ticí jiných objektů. Každá struktura je určena funktorem a argumenty. Jednotlivé argumenty jsou uzavřeny do kulatých závorek a odděleny čárkami. Funktor se píše před závorku.
Jako příklad uveďme zápisy: *)

ma_rada(eva,kvetiny)	Eva má ráda květiny.
vypujcka(novak,kniha(capek,povetron))	Pan Novák si vypůjčil Čapkovu knihu Povětroň.
vypujcka(novak,kniha(Autor,Nazev)) .	Pan Novák si vypůjčil nějakou knihu.

Z historických důvodů lze struktury, kde funktory jsou symboly pro matematické operace * / + - mod zapisovat v běžné "matematické" formě. Tak lze psát x+y*z a ne +(x,*(y,z)).

Takovýmto funktorům potom říkáme operátory. Pro zajímavost uvádím, že Prolog má předdefinováno více jak 20 operátorů a uživatel má možnost si sám definovat další.

Speciálním případem struktur jsou seznamy. Seznam je

libovolně dlouhá posloupnost prvků. Prvek může být libovolná struktura. Konec seznamu je xžáy reprezentován prázdným seznamem. První prvek seznamu se nazývá hlava seznamu; ostatní prvky tvoří zbytek seznamu. Prvky seznamů se zapisují do hranatých závorek. Pro zápis seznamu s hlavou X a zbytkem Y je zaveden speciální zápis [X/Y]. Např.

[ ]	prázdný seznam (neobsahuje žádný prvek)
[a,b,c]	seznam o třech prvcích a b c
[a/ [b,c]]	stejný seznam jako v předchozím příkladě

2.2 Databáze a její prohledávání

Jak již bylo řečno v úvodu, je program v Prologu vlastně určitá databáze skutečností. Databáze je tvořena klauzulemi. Klauzule jsou fakta nebo pravidla. Fakta jsou obecné struktury následované ukončující tečkou; o hlavním funktoru takové struktury hovoříme jako o predikátu. Podívejme se teď, jak může vypadat jednoduchá databáze tvořená samými fakty a jak s námi Prolog může komunikovat. Mějme například databázi obsahující čtyři fakta:

miluje(jan,marie).	Jan miluje Marii.
miluje(eva,kvetiny).	Eva miluje květiny.
miluje(eva,vino).	Eva miluje víno.
miluje(marie,vino).	Marie miluje víno.

Prolog s námi komunikuje tak, že zobrazí dotazovací atom ?-. Na tuto výzvu odpovíme dotazem ve tvaru struktury ukončeným tečkou, např.

?- miluje(eva,kvetiny).	Je pravda, že Eva miluje květiny?

Po obdržení takového dotazu Prolog vyhledá v databázi první klauzuli se stejnojmenným predikátem a se stejným počtem argumentů, jaký je v našem dotazu. V našem případě to bude fakt miluje(jan,marie). Potom začne porovnávat jednotlivé argumenty na odpovídajících si místech. V našem případě zjistí, že kon-

stanty jan a eva nejsou totožné, takže zavrhne první vyhledanou klauzuli a přejde na porovnávání další. Ta je miluje(eva, kvetiny). Tato klauzule se shoduje s naším dotazem, takže Prolog odpoví yes a znovu předloží atom ?-.

Pro celý tento postup prohlédávání databáze budeme v dalším stručně říkat, že Prolog plní C; v našem případě je to cíl miluje(eva,kvetiny).

Všimněme si, že na dotaz

?- miluje(marie,jan). Je pravda, že Marie miluje Jana?

odpoví Prolog no, protože při porovnání první klauzule v databázi s naším dotazem se porovnávají konstanty jan s marie, ale nikdy ne jan s jan a marie s marie. Kdybychom byli odkázáni jen na takovéto dotazy, asi by nás Prolog brzy omrzel. Mnohem zajímavější odpovědi dostaneme, použijeme-li v dotazech proměnné. Položme např. dotaz:

?- miluje(eva,Co). Co miluje Eva?

Při prohledávání databáze dojde Prolog podobně jako v předchozím případě k porovnání dotazu s druhou klauzulí, kde zjistí, že se mu shodují, jak predikáty, tak i první argumenty eva. Na počátku prohledávání je proměnná z dotazu tzv. volná, nesvázaná, tj. není dosud spojena s žádnou konstantou. V okamžiku porovnávání Prolog provede navázání proměnné Co na konstantu květiny, a tak bude mít shodu mezi dotazem a klauzulí v databázi. Odpoví nám proto:

Co=kvetiny

a očekává náš další povel. Zmáčkneme-li nyní na terminálu return, prolog odpoví yes a znovu vydá výzvu ?-. Kromě toho máme ovšem možnost požádat Prolog o hledání dalšího řešení, což dosáhneme odesláním středníku. V tomto případě zruší Prolog vazbu mezi proměnnou Co a konstantou kvetiny, přejde v databázi na další klauzuli a celý proces s vyhledáváním a navazováním proměnné opakuje, takže teď odpoví:

Co=vino

Po opětovném stisknutí dalšího ; už Prolog odpoví no, neboť

v databázi už není žádná další klauzule, s kterou by mohl ještě kladně porovnat náš dotaz.

Podívejme se nyní na zápis pravidel v databázi. Každé pravidlo se skládá ze dvou částí - hlavy a těla - oddělených navzájem atomem :-. Hlava se skládá z obecné struktury a tělo je posloupnost obecných struktur oddělených navzájem čárkou nebo středníkem a ukončených tečkou. Tato čárka oddělující struktury má význam logické konjukce (and) a středník má význam logické disjunkce (or). Označme si pro jednoduchost $S, S_1, \ldots, S_n$ struktury. Potom zápis $S:-S_1,\ldots,S_n$ znamená: S platí, jestliže platí S_1 až S_n a podobně $S:-S_1;\ldots;S_n$ znamená S platí, jestliže platí S_1 nebo S_2 nebo ... S_n. Ukažme si nyní jednoduchý příklad. Přidejme do naší databáze toto pravidlo:

miluje(jan,X):-miluje(X,víno).	Jan miluje každého, kdo miluje víno.	(1)

a položme Prologu dotaz

?- miluje(jan,Koho).	Koho miluje Jan?

Při prohlížení databáze Prolog narazí na první fakt miluje (jan,marie). Prolog provede vazbu a odpoví nám Koho=marie. Předpokládejme, že jsme odpověděli středníkem, a tak Prolog pokračuje v prohledávání databáze. U faktů, kde je první argument eva nebo marie samozřejmě neuspěje, až při prohledávání dojde k pravidlu (1). Nejprve provede navázání Koho=X. Ale protože klauzule, kterou porovnává s dotazem není fakt nýbrž pravidlo, začne plnit cíl z těla pravidla, totiž miluje(X,víno). Tento cíl se podaří splnit, naváže-li X=eva. Protože již předtím svázal Koho=X, má v tomto okamžiku vazbu Koho=eva. Odpoví tedy

Koho=eva

Požádáme-li Prolog o další hledání odpovědi, bude se snažit nejprve vyhledat další řešení těla pravidla. To se mu podaří navázáním X=marie, takže opět odpoví Koho=marie. Prosím čtenáře, aby si laskavě uvědomil rozdílný postup Prologu při první a druhé odpovědi Koho=marie.

2.3. Navracení

Kromě dotazů ve tvaru struktury můžeme klást Prologu též dotazy složitější; sice dotazy, které svou formou připomínají tělo pravidla. Jsou to tedy opět obecné struktury, navzájem spojené čárkou nebo středníkem. Význam čárky i středníku je stejný jako u těla pravidla. Podívejme se na jednoduchý příklad:

```
?- miluje(eva,X),miluje(marie,X).   Co milují Eva a Marie
                                    společně?
```

Při vyhodnocování takového dotazu se Prolog snaží nejprve splnit cíl miluje(eva,X), což se mu podaří navázáním X=kvetiny. Potom přejde na plnění druhého cíle, kde ovšem X je už navázáno na konstantu kvetiny, takže Prolog plní cíl miluje (marie, květiny). Při plnění tohoto cíle Prolog neuspěje. V tomto okamžiku dojde k tzv. navracení. Prolog zapomene navázání X=kvetiny a začne hledat další řešení pro první cíl, které dostane vazbou X=vino. Potom se mu už podaří splnit cíl miluje(marie,vino), takže vydá odpověď na náš dotaz X=vino

Stejně probíhá mechanismus navracení i při plnění složitějších pravidel, které mají ve svém těle několik struktur. Neuspěje-li Prolog ve svém plnění některého z cílů, vrací se k hledání dalšího řešení předchozího cíle.

Zde bych rád upozornil na jeden důležitý rys proměnných v Prologu. Proměnná může být volná nebo vázaná. Jakmile je však jednou vázána, nemůžeme tuto vazbu změnit nějakým jiným navázáním, pouze při navracení se vazba zapomene.

2.4. Vestavěné predikáty true, fail, not a konstrukce cut

Prolog má některé vestavěné predikáty, což jsou predikáty, jejichž význam nemusí uživatel explicitně sám definovat. Predikát true uspěje vždy, predikát fail naopak nikdy neuspěje. Predikát

not má jeden argument. Jestliže Prolog narazí na predikát not s argumentem C, začne plnit cíl C.Jestliže cíl C uspěje, tak cíl not(C) neuspěje, naopak jestliže C neuspěje, tak not(C) uspěje. Upozorňuji čtenáře, že chceme-li použít not na konjunkci dvoucílů A,B, tak musíme použít zápis not((A,B)). Zápis not (A,B) by donutil Prolog k vyhledávání v databázi klauzule not s dvěma argumenty, zatímco předdefinované not má pouze jeden argument.

Konstrukce "cut" slouží v Prologu k samazení navrácení. Zapisuje se termem "!". Vyskytuje-li se v těle nějakého pravidla cut, a dochází-li k navrácení v místě, kde je cut zapsán, Prolog ukončí prohledávání databáze a plnění cíle pro toto pravidlo neuspěje.(přitom může ovšem dojít k navracení na místě, odkud bylo požádáno o plnění tohoto cíle).
Plní-li se např. cíl:

```
go:-a,b,!c,d.
```

a dojde-li k navracení mezi c a b, tak cut umístěný mezi nimi toto navracení "nepustí" a celý cíl go neuspěje.

2.5. Operátor is

Již jsme si řekli, že aritmetické operátory +,-,*,/ a mod tvoří spolu se svými argumenty obecné prologovské struktury. Ovšem neprovádějí matematický výpočet. Tak např. dotaz

```
?- X=2+3
```

bude zodpovězen

```
X=2+3
```

Chceme-li Prolog požádat o výpočet nějakého aritmetického výrazu, použijeme k tomu operátor is. Výpočet se zapisuje ve tvaru: X is výraz. Na levé straně od operátoru is musí být proměnná a naopak ve výrazu se musí vyskytovat pouze celá čísla a všechny eventuální proměnné musí být v okamžiku výpočtu navázány na celá čísla. Jako příklad si uveďme oblíbený faktoriál:

```
faktoriál(0,1):-!                    Faktoriál 0 je 1.
faktoriál(N,X):-N1 is N-1            Faktoriál N je faktoriál z N-1
                faktoriál(N1,Y),     vynásobený N
                X is Y*N.
```

2.6. Vstup, výstup

Pro čtení a psaní termů slouží předdefinované predikáty read a write s jedním argumentem. Pro odskok na novou řádku na výpisu slouží predikát nl bez argumentu.

Pro načítání dat do databáze slouží predikáty consult a reconsult, které mají jako argument jméno souboru, na němž je uložena naše databáze. Predikát consult přidává všechny načítané klauzule do databáze, zatímco predikát reconsult nahrazuje v databázi stejnojmenné klauzule načítanými. Speciální jméno souboru "user" umožní načítat databázi z terminálu.

2.7. Příklady na práci se seznamy

Definujme si predikát member, který bude zjišťovat, zda jeho první argument je prvkem seznamu na místě druhého argumentu.

```
member(X,[X/_]).                     X je prvek seznamu, je-li
                                     jeho hlavou.
member(X,[_/Y]):-member(X,Y).        X je prvkem seznamu, je-li
                                     prvkem zbytku seznamu.
```

Položíme-li nyní Prologu dotaz

```
?-member(2,[1,2,3]).                 Je 2 prvkem seznamu [1,2,3]?
```

odpoví yes. Zajímavý bude výpis na terminálu, položíme-li následující dotaz:

```
?-member(X,[1,2,3]),write(X),fail.
```

Potom se na terminálu objeví

123

no

?-

Při prvním plnění cíle member(X,[1,2,3]) tento uspěje s navázáním X=1, write vypíše 1. Následné fail způsobí navrácení. Write při navrácení podruhé neuspěje, takže se bude podruhé plnit cíl s predikátem member. Pro jeho splnění se použije druhá klauzule member v databázi. Ta však vede na plnění cíle member(X,[2,3]), kterýžto cíl uspěje s navázáním X=2, které write vypíše. Vše se opakuje ještě jednou, kdy se vypíše 3. Potom se Prolog pokouší znovu plnit member, který však už neuspěje, což má za následek vypsání no a atom ?-.

Jako druhý příklad si uveďme predikát append, který má tři argumenty - všechny jsou seznamy. Přitom třetí seznam je spojením prvních dvou seznamů.

```
append([],L,L,).
```

Spojením prázdného seznamu a seznamu L je seznam L.

```
append([X/Y],L,[X/Z]):-append(Y,L,Z).
```

Spojením seznamu s hlavou X a zbytkem Y se seznamem L vznikne seznam s hlavou X a zbytkem spojeným z Y a L.

Po takto zadefinovaném predikátu append se můžeme dotazovat nejen na výsledek spojení dvou seznamů, ale též obráceně, jak vypadají seznamy, jejichž spojením vznikne daný seznam. Tomuto dotazu odpovídá zápis:

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

Prolog postupně vrací řešení:

```
X=[],Y=[a,b,c];
X=[a],Y=[b,c];
X=[a,b],Y=[c];
X=[a,b,c],Y=[]
```

3. Závěr

Kromě učebnice [1] bych chtěl rád čtenáři doporučit též výbornou knihu [2], ve které najde řadu vyřešených problémů.

Pokud je mi známo, je Prolog v ČSSR implementován na řadě počítačů. Na počítačích EC je implementována tzv. marseilleská verze Prologu, která má poněkud jinou syntaxi. Na počítačích SMEP a 8-bitových mikropočítačích je implementována verze přesně podle [1]. Na počítačích ADT je Prolog dostupný v obou verzích.

Literatura

[1] W.F.Clocksin, C.S.Mellish: Programming in Prolog, Springer-Verlag, New York 1981

[2] H.Coelho, J.C.Cotta, L.M.Pereira: How to solve it with Prolog, Laboratorio Nacional de Engenharia Civil, Lisboa 1980.