PRINCIPY LOGICKEHO PROGRAMOVANI A POUZIVANI PROLOGU

Ing. Miloš K o c h
Brno, VUT FS, katedra ekonomiky a řízení strojírenské výroby

V příspěvku jsou předloženy základní informace o logickém programování a jazyku PROLOG, který z něj vychází. Příklady, na nichž je PROLOG demonstrován byly voleny s úmyslem přiblížit možnosti jeho použití v praxi.

1 LOGICKE PROGRAMOVANI

Základem logického programování je Hornova logika, která je fragmentem predikátové logiky prvního řádu. Hornovy klauzule můžeme zapsat (zjednodušeně) ve tvaru

$$P \leftarrow C_1 \,\&\, C_2 \,\&\, \ldots \,\&\, C_n \qquad (1)$$

ve smyslu "P platí, jestliže platí všechny klauzule C spojené vztahem logické konjunkce & (a současně)". Z (1) můžeme odvodit tři základní prvky logického programování - fakta, pravidla a otázky.

fakta

Je-li n = 0, vystupuje symbol P samostatně v roli tzv. faktu, tedy konstatování nějaké skutečnosti.

PP Petr krade

pravidla

Pro n > 0 přepisujeme (1) do tvaru

$$P \text{ :- } C_1 , C_2 , \ldots , C_n \qquad (2)$$

ve stejném významu, kde P nazýváme hlavou pravidla a konjunkci C tělem pravidla.

Př. je zločinec :- krade,vraždí,plení

otázky

Nahradíme-li symbol P otazníkem (pro n >= 0), chápeme zápis jako posloupnost dotazů na platnost objektů C.

Př. ?- je zločinec

Hledáme buď potvrzení či vyvrácení dotazu, nebo konkrétní zjištění objektů splňujících požadované podmínky.

2 PROLOG

Programovací jazyk PROLOG je dotazovací, neprocedurální jazyk vhodný především pro symbolickou manipulaci s daty. Méně vhodný je pro aritmetické výpočty a hromadné zpracování dat, proto nalézá uplatnění především v úlohách tzv. umělé inteligence, při zpracování přirozeného jazyka a v expertních systémech.

PROLOG není úplnou implementací logického programování, protože odvozovací mechanismus je deterministický a výsledek je obecně závislý na tvaru programu. Z toho mohou plynout problémy s korektností a ukončitelností PROLOGovských programů.

Programování v PROLOGu spočívá ve vytvoření databáze faktů a pravidel, jak s těmito fakty zacházet a v následném kladení dotazů, kdy řídicí systém PROLOGu vyhledává odpovídající fakta a odvozuje z nich odpovědi na dotazy, aniž by uživatel byl nucen jeho činnost nějak organizovat.

2.1 Datové typy

V PROLOGu rozlišujeme jediný datový typ - logický term, který zahrnuje konstanty, proměnné a struktury.

Konstanty

Konstanty jsou tvořeny dvěma typy - celými čísly a atomy. Atomem nazýváme

- alfanumerickou posloupnost znaků začínající malým písmenem
- symboly jako např. =, /, +
- posloupnost znaků uzavřených v apostrofech (')

Proměnné

Názvy proměnných v PROLOGu začínají vždy velkým písmenem, aby se odlišily od konstant, nebo znakem podtržítko (_), u interpretu PROLOGu pro počítače JSEP také znakem #. Nikde v programu je nedeklarujeme, přičemž s jednou proměnnou mohou být v různých chvílích svázány různé "datové typy".

Speciální význam má v PROLOGu nespecifikovaná proměnná, označená symbolem podtržítko. Je to obdoba např. FILLER v Cobolu a použijeme ji tehdy, nezajímá-li nás její obsah.

Struktury

Struktury jsou tvořeny z libovolných termů, tedy i dalších struktur, spojených funktorem, kterému říkáme predikát.

```
clovek(jan,novak)
```

například je struktura s predikátem "člověk", který spojuje dva termy typu atom "křestní jméno" a "příjmení". V klasickém programu bychom tuto strukturu mohli chápat jako větu s dvěmi položkami či proceduru s dvěma parametry.

Stuktura je nejobecnější logický term, konečně i program v PROLOGu můžeme chápat jako logický term typu struktura.

termy	-	konstanty	1, ahoj, 'petr pije'
	-	proměnné	A, Ahoj, _n
	-	struktury	prvek(radon)
			zlocinec(jan,novak,vrah)
			bankovky(mince(1,2,5),10,20,100)

Zvláštním případem struktury je tzv. seznam. Seznam je množina objektů uzavřených v hranatých uvozovkách ([ ]), oddělených čárkami (,).

	-	seznamy	[] prázdný seznam
			[petr,pavel] seznam má 2 prvky
			[1,[2,3]] 2 prvky, druhý je opět seznam

2.2 Fakta a pravidla

Program i data jsou v PROLOGu uloženy ve formě klauzulí, které můžeme rozdělit na fakta a pravidla.

klauzule	-	fakta
	-	pravidla

Programování v PROLOGu si přiblížíme na příkladu postupného vytváření elementárního informačního systému evidence literatury.

Pro začátek se usneseme, že budeme evidovat informace o názvech knih a jménech jejich autorů.

Fakta

Skutečnost, že Jan Novák je autorem románů Oběšenec a Neznámý hrob a napsal je v letech 1970 a 1989 můžeme zapsat jako fakta do databáze například takto:

```
roman(jan,novak,obesenec,1970).
roman(jan,novak,'Neznamy hrob',1989).
```

Predikát, který spojuje dílčí fakta jsme pojmenovali "román", v kulatých závorkách jsou uvedeny jednotlivé "položky", oddělené čárkami a celá struktura je podle konvencí PROLOGu ukončena tečkou. Konstanty píšeme malým písmenem, abychom je odlišili od proměnných, nebo jako řetězce v uvozovkách.

Je velmi zajímavé, že již v dané fázi tvorby databáze můžeme PROLOGu klást dotazy. Je-li PROLOG připraven k dialogu, což indikují znaky

```
?-
```

můžeme položit otázku, zda Jan Novák je autorem Oběšence z roku 1970

```
?-roman(jan,novak,obesenec,1970).
```

Reakce PROLOGu je taková, že prohledává postupně databázi, a najde-li fakt "román", který je zcela srovnatelný s vzorem v otázce, odpověď PROLOGu bude

```
YES
```

Na otázku, zda Jan Novák napsal Oběšence v roce 1990 odpoví PROLOG záporně (NO), neboť naše databáze žádný takový fakt neobsahuje.

Vést dialog podobným způsobem, ač je to možné, není patrně příliš atraktivní. Položme tedy otázku jinak:

```
?-roman(Jmeno,Prijmeni,Nazev,Rok).

Jmeno = jan;
Prijmeni = novak;
Nazev = obesenec;
Rok = 1970
```

Místo abychom nechali porovnat databázi a konstantami v predikátu, použili jsme jako parametrů otázky proměnné. PROLOG vyhledal první predikát román s čtyřmi parametry a svázal (unifikoval) jeho termy se zvolenými proměnnými. Odpověď PROLOGu je zachycena výše.

Bylo nalezeno první řešení otázky a PROLOG očekává pokyn, jak má postupovat dále. Chceme-li, aby se pokusil o další řešení, tedy vyhledal další libovolný román, odpovíme znakem středník (;). Odpověď bude "NO", protože databáze žádný jiný fakt neobsahuje. Postačuje-li nám však nalezené řešení, odpovíme tečkou (.), pokynem pro ukončení práce. PROLOG odpoví "YES", neboť řešení bylo nalezeno.

Potřebujeme-li zjistit fakta splňující nějakou podmínku, například vypsat jména knih autora Nováka z let 1960 až 80, můžeme otázku modifikovat takto:

```
?-roman(_,Jmeno,Nazev,Rok),Rok>1960,Rok<1980,Jmeno=novak.

Jmeno = novak;
Nazev = obesenec;
Rok = 1970  )

NO
```

Standartní predikáty

Standartními predikáty jazyka PROLOG rozumíme předdefinova-

na pravidla ve smyslu klasických příkazů jiných vyšších programovacích jazyků. Uveďme si na ukázku několik z nich.

read( )	-	predikát čte term ukončený tečkou z aktuálního vstupu a ztotožní jej s proměnnou uvedenou jako parametr
write( )	-	term uvedený jako parametr se zapíše do aktuálního výstupu
tab( )	-	výstup určeného počtu mezer (tabulátor)
nl	-	přechod na nový řádek při výstupu (new line)
fail	-	predikát představuje cíl, který nikdy není splnitelný
repeat	-	predikát představuje cíl, který je splnitelný vždy

Je snad na místě předeslat, že PROLOG nemá příkazy typu IF THEN či příkazy cyklu typu FOR nebo WHILE.

Pravidla

Konstrukci pravidel si přiblížíme na příkladu pravidla pro výpis jmen autorů a názvů jejich děl, nazvaného výpis.

```
vypis:-write('----- vypis romanu ----- '),nl,nl,
       roman(_,J,N,_),
       write(J),write(','),tab(2),write(N),nl,
       fail.
```

Pravidlo spustíme příkazem .

```
?-vypis.

------ vypis romanu -------

novak:  obesenec
novak:  Neznamy hrob

NO
```

Znaky ":-" odělují vždy tzv. hlavu pravidla a případnými parametry v kulatých závorkách od vlastního pravidla, tzv. těla. Pravidlo je tvořeno vždy jednou až několika klauzulemi, což mohou být další pravidla, predikáty nebo standartní predikáty PROLOGu.

Jednotlivé klauzule pravidla jsou od sebe odděleny čárkami, které zde představují vazbu mezi klauzulemi "a současně" - konjunkci. Obdobně bychom v pravidle mohli použít oddělovače "nebo", středníku ve smyslu disjunkce.

Vysvětlíme si, jak bude PROLOG postupovat při řešení našeho pravidla. Nejprve vytiskne řetězec s nadpisem sestavy, příkazem NL přejde na nový řádek a opětovným NL vynechá mezi hlavičkou sestavy a dalším tiskem prázdný řádek.

Příkaz 'roman(_,J,N,_)' způsobí prohledávání databáze. Je-li v ní nalezen predikát srovnatelný s tímto cílem, jako je v našem případě záznam o Oběšenci Jana Nováka, dojde k unifikování, svázání jeho parametrů s dosud volnými proměnnými J a N; obě zbylé položky se svážou s nedostupnou proměnnou '_'. Další příkazy vytisknou obsahy proměnných J a N s dvojtečkou po příjmení autora, dvěmi mezerami a přejde se v tisku na další řádek.

Až potud je pravidlo jasné. Kdybychom však v tomto okamžiku ukončili popis pravidla, PROLOG by splnil všechny požadované cíle nalezením prvního výskytu románu v databázi a ukončil by práci s konstatováním YES, pravidlo uspělo.

Náš úmysl však je, aby PROLOG vypsal všechny romány, nikoli vždy pouze první z nich. Proto jako poslední cíl pravidla stanovíme cíl 'FAIL', který nikdy nemůže uspět. A tady se

poprvé setkáváme se zázračnou schopností PROLOGu, zvanou backtracing - navracení. Pokud některý z cílů pravidla neuspěje, PROLOG zastaví postup po pravidle směrem doprava, uvolní vázané proměnné a začne se vracet zpět. Tento návrat trvá tak dlouho, dokud se mu nepodaří nalézt cíl, který je znovu splnitelný, nebo se navrátí k hlavě pravidla. Nalezne-li splnitelný cíl, zastaví navracení a pokračuje opět doprava. Tato činnost probíhá do tá doby, dokud neopustí pravidlo zprava se stavem YES (pravidlo uspělo), nebo zleva s neúspěchem splnění pravidla, stavem NO.

Graficky bychom tuto ideu činnosti mohli znázornit takto

```
Pravidlo :-   Cil 1 ,  Cil 2 ,  Cil 3 .


start -------->  ano  --> ano  --> ano   ---> konec YES


start -------->  ano  -->  ne
                  <-------
                  ano  -->  ne
                  <-------
                   ne
konec NO <----------
```

Cíle, která nemají alternativu, jako např. většina standartních predikátů, nelze splnit znovu při návratu po pravidle směrem doleva. Znovusplnitelné jsou pouze ty cíle, které se v databázi vyskytují několikrát, například skupiny faktů se stejným typem predikátů (román), ale také vícenásobně definovaná pravidla a standartní predikát REPEAT.

znovu splnitelné cíle - fakta obsažená vícekrát v bázi
- pravidla obsažená vícekrát v bázi
- predikát repeat

Pro prohledávání bází skupin stejných klauzulí platí pravidlo:

- plníme-li cíl "ZLEVA", začíná se báze prohledávat od začátku

- plníme-li cíl "ZPRAVA", pokračuje se v prohledávání báze od místa, kde se naposledy skončilo.

Aplikujeme-li tato pravidla, pochopíme, proč po vypsání 'Oběšence' naše pravidlo nalezlo další knihu morbidního autora Nováka a po vyčerpání všech románů skončilo s hlášením NO.

2.3 Příklady

Některé možnosti PROLOGu si ukážeme na souhrnném příkladu, umožňujícím srovnání s řešením obdobných úloh v jiných programovacích jazycích.

Pokusme se o vytvoření jednoduchého informačního systému evidence literatury v knihovně, včetně sledování výpůjček knih čtenáři. K tomu budeme potřebovat tyto dvě struktury:

čtenář

- číslo legitimace
- jméno čtenáře
- příjmení čtenáře
- půjčené knihy (seznam)

kniha

- číslo knihy
- jméno autora
- příjmení autora
- název knihy

- rok vydání
- cena knihy
- číslo čtenáře, který si vypůjčil knihu

V PROLOGu bychom data podle těchto struktur mohli zaznamenat například takto:

```
ctenar(1,josef,doubek,[]).

kniha(1,autor(jan,novak),'Obesenec',1970,17,0).
```

Všimněme si, jak jsou tyto vytvořené "věty" zvláštní z pohledu konvenčního hromadného zpracování dat. Predikát čtenář má čtyři parametry, z nichž jeden je seznam a v našem případě neobsahuje žádnou vypůjčenou knihu. Predikát kniha má šest parametrů, druhý z nich je další struktura ("autor"), a hodnota 0 v položce "číslo čtenáře" znamená, že kniha není vypůjčena. Každá položka je proměnné délky.

Na datech uchovávaných v databázi je milé, že stejně jako program jsou přístupné editorem, přičemž není takový problém vytvořit konverzní programy pro přenos dat mezi PROLOGem a jinými programovácími jazyky.

Vstup a výstup

Pro vkládání nových vět z terminálu můžeme zkonstruovat pravidlo "vstup_knihy".

```
vstup_knihy:-nl,write('--- zapis nove knihy ---'),nl,nl,
             write('cislo knihy      :'),read(C),
             not kniha(C,_,_,_,_,_),
             write('jmeno autora     :'),read(J),
             write('prijmeni autora  :'),read(P),
             write('nazev knihy      :'),read(N),
             write('rok vydani       :'),read(R),
               R>1000,R<2000,
```

```
        write('cena knihy      :'),read(H),
          H>0,
        assertz(kniha(C,autor(J,P),N,R,H,0)),
        nl,write('kniha zapsana ...'),nl.


?-vstup_knihy.

   --- zapis nove knihy ---

   cislo knihy      :2.
   jmeno autora     :jan.
   prijmeni autora  :novak.
   nazev knihy      :'Neznamy hrob'.
   rok vydani       :1989.
   cena knihy       :32.

   kniha zapsana ...

   YES
```

Výsledkem je věta v databázi ve tvaru

```
kniha(2,autor(jan,novak),'Neznamy hrob',1989,32,0).
```

V pravidle jsou použity některé nové standartní predikáty PROLOGu. Je to především predikát "read", který sváže term čtený z klávesnice a zakončený tečkou s proměnnou, uvedenou v příkazu. Cíl

```
not kniha(C,_,_,_,_,_)
```

prohledá databázi knih a uspěje, pokud v ní NENAJDE knihu příslušného čísla. Standartní predikát "not" způsobí totiž převrácení výsledku cíle. Pokud by v databázi již existovala kniha identického čísla, provádění pravidla by bylo zastaveno

a vstup údajů by ihned skončil s hlášením PROLOGu "NO".

Kontrolu logické správnosti vkládaných dat zde uskutečňují testy

```
R>1000,R<2000    rok vydání musí ležet v intervalu 1000-2000
H>0              cena knihy nesmí být nulová
```

Pokud všechny vstupní údaje byly korektní, provede pravidlo "assertz" zápis struktury uvedené v závorce do databáze na konec skupiny stejnojmenných predikátů. Ekvivalentní pravidlo "asserta" provádí zápis na začátek skupiny.

Vyhledání knih, které splňují určité podmínky můžeme realizovat pravidlem "vypis_knih".

```
vypis_knih:-nl,write('--- vyhledani knih ---'),nl,nl,
           write('jmeno autora     :'),read(J),
           write('prijeni autora   :'),read(P),
           write('nazev knihy      :'),read(N),
           nl,
           kniha(C,autor(J,P),N,R,_,V),
           write(C),tab(2),
           write(P),write(','),
           write(J),write(';'),nl,
           tab(3),write(N),
           tab(3),write(R),
           ((V=0),nl,nl)
           ;
            (V\=0,
             ctenar(V,_,Pc,_),
             tab(5),write('... pujceno - '),
             write(Pc),nl,nl),
           ),
           fail.

vypis_knih:-nl,write('neni (vice) knih)'),nl.
```

Předpokládejme, že čtenář Doubek má půjčeného Oběšence, tedy v záznamu o této knize bude číslo čtenářského průkazu "1" a v seznamu knih čtenáře seznam [1].

```
?-vypis_knih.

--- vyhledani knih ---

jmeno autora      :_.
prijmeni autora   :novak.
nazev knihy       :_.

1  novak,jan:
   Obesenec   1970      ... pujceno - doubek

2  novak,jan:
   Neznamy hrob   1989

neni (vice) knih

YES
```

Toto pravidlo je už zajímavější. Je v něm využito alternativy, disjunkce. Pravidlo začíná zjištěním podmínek pro výběr. Chceme-li kupříkladu, aby se nám vypsali autoři s libovolným křestním jménem, odpověď na dotaz o jménu autora bude nepojmenovaná proměnná "_", která je unifikovatelná s libovolným termem.

Po nastavení výběrových podmínek následuje dotaz na databázi, zda se v ní vyskytuje kniha, vyhovující podmínkám. Pokud je nalezena, dochází k větvení na alternativní postup dalšího řešení.

Obě varianty jsou uzavřeny v závorkách a odděleny středníkem, který symbolizuje vztah "nebo".

```
((V=0,,,,);(V\=0,,,,))
```

v druhé větvi je podmínka pochopitelně nadbytečná, neboť vyplývá z neplatnosti první, nicméně je uvedena pro zvýšení čitelnosti programu.

Není-li kniha vypůjčena, provede se toliko vynechání řádku při tisku. Pokud však je, vyhledá se v databázi čtenář podle čísla legitimace unifikovaného s C a vytiskne se jeho jméno.

Obě větve se opět sejdou na cíli "fail", pravidlo neuspělo. Tím je vynuceno navracení po pravidle až k cíli "kniha". Protože se pokoušíme o znovusplnění cíle zprava, pokračuje se v prohledávání databáze knih od místa, kde naposledy skončilo.

Přicházíme znovu k vyhledání jména čtenáře, a sice zleva. Tím je dáno, že databázi čtenářů prohledává PROLOG vždy znovu od začátku.

Je možná vhodné upozornit, že znovusplnění cíle

```
ctenar(V,_,Pc,_)
```

není možné jenom proto, že v evidenci nesmí být dva čtenáři se stejným číslem čtenářského průkazu.

První pravidlo vypis_knih je konstruhováno tak, aby jako celek nikdy nemohlo uspět. Pokud však přece jenom chceme, aby nekončilo neúspěchem, nebo potřebujeme po jeho skončení provést další činnost, jako v našem případě tisk textu "není více odpovědí", lze definovat alternativní pravidlo (pravidla) pod stejným jménem.

Skončí-li pravidlo z takové skupiny neúspěchem, přejde se automaticky k zpracování dalšího. Pravidlo jako celek neuspěje teprve tehdy, pokud neuspěje žádné z alternativních pravidel.

Práce se soubory

Koncepce souborů je v PROLOGu velmi jednoduchá. Jediným, který používá, je sekvenční soubor ASCII znaků. Vstup probíhá

vždy z aktuálního souboru, výstup do aktuálního výstupního souboru, přičemž pro jejich změnu má PROLOG vestavěny standartní predikáty. Při inicializaci systému je vstup i výstup nasměrován do uživatelského souboru "user", reprezentovaného klávesnicí a obrazovkou terminálu.

Ukažme si pravidlo pro uložení dat do externího souboru, pojmenované "zápis_dat".

```
zapis_dat:-nl,write('--- ulozeni dat ---'),nl,nl,
          write('jmeno souboru :'),read(S),
          nl,nl,
          write('zapsat data ? '),read(Q),
          nl,
          (Q=ano;Q=a;Q=yes;Q=y),
          tell(S),
           listing(ctenar),
           listing(kniha),
          told,
          write('data ulozena ..'),nl.

?-zapis_dat.
--- ulozeni dat ---

jmeno souboru :knihy.

zapsat data ?ano.
data ulozena ..

YES
```

I na tomto pravidle si můžeme demonstrovat disjunkci cílů. Odpověď na otázku "zapsat data ?" může v našem případě nabývat řekněme čtyř správných hodnot - "ano,a,yes,y". Má-li se zápis na médium provést jen tehdy, leží-li odpověď v množině povolených hodnot, můžeme test rozepsat na čtyři alternativy uzavřené v závorkách. Výsledek testu uspěje, bude-li splněna jedna z nich, v opačném případě pravidlo

končí.

Predikát "tell" přepíná výstup do souboru, specifikovaného parametrem predikátu. Predikát "listing" provádí výpis všech teras jména uvedeného v závorce, či v případě, že závorku s maskou neobsahuje, veškerý obsah databáze, do aktuálního výstupního souboru. Konečně predikát "told" uzavře výstupní zařízení a vrátí výstup do standartního souboru "user".

```
cteni_dat,-nl,write('--- cteni dat ---'),nl,nl,
          write('jmeno souboru : '),read(S),
          nl,nl,
           reconsult(S),
          write('data ulozena ...'),nl.
```

Pravidlo "čtení dat" je postaveno na využití standartního predikátu "reconsult", který odstraní z databáze všechny predikáty, které mají stejná jména jako nově zařazované predikáty ze souboru S a nahradí je obsahem tohoto souboru. Ostatní predikáty, které se nevyskytují v souboru S zůstanou beze změny. Ve většině verzí PROLOGu má tento predikát i alternativní zápis v podobě jména souboru v hranatých závorkách se znakem x před jménem souboru ([xS]).

Predikát "reconsult" lze nahradit spojením dvou pravidel, pravidel "retractall" a "consult". Pokud bychom je chtěli použít v našem případě, museli bychom predikát "retracta.." užít dvakrát, k odstranění predikátů "kniha" i "čtenář".

```
retractall(ctenar(_,_,_,_,_))   odstranění predikátů ctenář

consult(S)                      vložení obsahu souboru S do
                                databáze bez ohledu na to,
                                zda predikáty stejného jména
                                v něm již existují.
```

Práce se seznamy

Seznam je uspořádaná posloupnost prvků, která může mít libovolnou délku, přičemž prvkem seznamu může být jakýkoliv term. Používá se k seskupení informací, které spolu nějak souvisí a jejich počet se může případ od případu lišit.

Jedinou, přímo definovanou operací na seznamech, je rozdělení seznamu na "hlavu" a "tělo".

[a,b,c]	-	[a]	hlava seznamu
		[b,c]	tělo seznamu

Operace je definována svislou čarou v následujícím tvaru

```
[X|Y]
```

a způsobí, že s proměnnou X je svázána hlava seznamu a s proměnnou Y tělo. Ukážeme si ji na příkladu pravidla pro výpis knih, které má vypůjčené některý čtenář.

```
vypis_ctenare(C):-nl,
                  ctenar(C,J,P,K),
                  write(C),tab(2),
                  write(P),write(','),
                  write(J),
                  nl,nl,
                   ma_knihy(K).

ma_knihy([]):-nl.
ma_knihy([H|T]):-tab(3),write(H),
                 write('-> '),
                 ((kniha(H,_,N,_,_,_),
                  write(N))
                 ;write('neexistuje')),
                 nl,
                   ma_knihy(T).
```

Pravidlo "vypis_ctenare" je poměrně jednoduché. Vyhledá v databázi čtenáře, jehož číslo je svázáno s proměnnou C, a vytiskne jeho příjmení a jméno. Posledním cílem pravidla je provedení cíle "má_knihy", do kterého se předá seznam čísel knih unifikovaný s proměnnou K.

Pravidlo "má_knihy" má dvě varianty. První uspěje v případě, že čtenář nemá žádné knihy, tedy hlava pravidla je srovnatelná s prázdným seznamem, a provede toliko přechod v tisku na nový řádek.

Druhé využívá již v hlavě mechanismus krájení seznamu na hlavu a tělo. S proměnnou H je svázán první člen seznamu, s proměnnou T zbytek seznamu.

Vytiskne se číslo knihy H, znaky "->" a podle toho, zda taková kniha existuje buď její název, nebo text "neexistuje". Obě větve se pak znovu sejdou na cíli "nl".

Rekurzí je posléze voláno znovu pravidlo "má_knihy" pro zbytek seznamu.

Předpokládejme, že čtenář Doubek má půjčená knihy 1 a 3, která neexistuje. Činnost pravidla by byla následující:

```
?-vypis_ctenare(1).

1  doubek,josef

ma_knihy([1,3]) ->

    ma_knihy([]):-nl.                                   ... ne
    ma_knihy([1|[3]]):-

            1-> Obesenec                                ... ano

        ma_knihy([]):-nl.                               ... ne
        ma_knihy([3|[]]):-

            3-> neexistuje                              ... ano

        ma_knihy([]):-'                                 ... ano
```

```
YES
```

Na stejném principu můžeme zkonstruhovat i pravidla "vráceno" a "půjčeno", které budeme potřebovat na konstrukci pravidla pro automatizaci obsluhy čtenáře při půjčení a vrácení knih.

```
vypujcky:-nl,write('--- vypujcky knih ---'),nl,nl,
          write('cislo ctenare : '),read(C),
           vypis_ctenare(C),nl,
          write('nove knihy  [] : '),read(S),
           retract(ctenar(C,J,P,K)),
           assertz(ctenar(C,J,P,S)),
             vraceno(K),
             pujceno(S,C),
           vypis_ctenare(C),
          nl.

vraceno([]).
vraceno([X|Y]):-
                ((retract(kniha(X,A,B,C,D,_)),
                  assertz(kniha(X,A,B,C,D,0)))
                 ;
                 (write(X),write('kniha neexistuje'),nl)),
                vraceno(Y).

pujceno([],Cl).
pujceno([X|Y],Cl):-
                ((retract(kniha(X,A,B,C,D,_)),
                  assertz(kniha(X,A,B,C,D,Cl)))
                 ;
                 (write(X),write('kniha neexistuje'),nl)),
                pujceno(Y,Cl).
```

Na pravidle je rovněž vidět, jak můžeme v PROLOGu měnit obsah některého faktu. Predikát "retract" odstraní z databáze knihu, jejíž číslo je srovnatelné s X, přičemž současně sváže

původní parametry s proměnnými A..D. Predikát "assertz" zapíše fakt znovu do databáze s modifikovaným parametrem "číslo čtenáře".

```
?-vypujcky.

--- vypujcky knih ---

cislo ctenare : 1.

1 doubek,josef

   1-> Obesenec

nova knihy [] : [2].

1 doubek,josef

   2-> Neznamy hrob

YES
```

Aritmetika

Aritmetickou operaci realizujeme obecně v PROLOGu strukturou

```
X is Y
```

kde Y je struktura, reprezentující platný aritmetický příkaz. Tento výraz je vyhodnocen a výsledek je svázán (nebo porovnán, je-li X již vázáno), s proměnnou X. Výraz Y může být tvořen za pomocí operátorů

```
+    sčítání
```

-	odečítání
*	násobení
/	celočíselné dělení
mod	zbytek po celočíselném dělení

U PROLOGu 80 pro osmibitové počítače, který je v tomto příspěvku popisován musí být hodnota celého výrazu i kteréhokoliv mezivýsledku celá číslo.

Př. X is (10 / A) + B

řez

Vytvořené predikáty můžeme zastřešit "jídelníčkem", menu za kterého uživatel vybírá požadované funkce.

```
menu:-repeat,
      nl,write('--- Agenda literatury ---'),nl,
      nl,write('1 ... zapis nove knihy'),nl,
      nl,write('2 ... vypis knih'),nl,
      nl,write('3 ... vypujcka knihy'),nl,
      nl,write('4 ... data na vnejsi medium'),nl,
      nl,write('5 ... obnova dat z vnejsiho media'),nl,
      nl,write('k ... konec'),nl,nl,
      nl,write('Vase volba = '),read(Q),nl,
      volba(Q),
      Q=k.

volba(1):-!,vstup_knihy.
volba(2):-!,vypis_knih.
volba(3):-!,vypujcka.
volba(4):-!,zapis_dat.
volba(5):-!,cteni_dat.
volba(k).
volba(X):-!,nl,write('chybna volba'),nl.
```

Predikát "repeat" uspěje vždy a způsobí, že pravidlo se stále vrací k výpisu nabídky a volbě uživatele. Jednotlivé funkce se volají srovnáním hodnoty vázané s Q s pravidlem pro příslušnou činnost volba. Leží-li Q mimo rozsah vyjmenovaných voleb, uspěje poslední z nich "volba(X)" vždy, neboť s volnou proměnnou lze svázat cokoli.

Predikát "!" je tzv. řez. Oznamuje PROLOGu, že cíl, v němž byl použit, je nadále nesplnitelný při pokusu o znovuplnění zpráva. Je zde použit proto, aby při návratu po pravidle "menu" nedocházelo k opětnému pokusu o splnění cíle "volba", kde poslední z nich by uspěl a způsobil nechtěné generování textu "chybná volba".

Operátorový zápis struktury

V případě, že struktura má jeden až dva argumenty, je možné použít i čitelnější zápis struktury, tzv. operátorový zápis. Funktor predikátu musíme deklarovat jako operátor s číslem precedence (čím je toto číslo větší, tím slabší je vazba operátoru s operandy), a pozicí, udávající polohu operátoru vzhledem k operandům.

Pozice je vyjádřena takto

fx - prefixový operátor
xf - postfixový
xfx - infixní
xfy - vpravo asociativní
yfx - vlevo asociativní

V standartní PROLOGu existuje například tato definice operátoru

```
op(255,fx,?-)
```

definující otázku v PROLOGu, kterou bychom jinak museli psát

```
?-(menu).
```

ve formě struktury. Obdobné sčítání a odčítání je definováno jako op(31,yfx,+) a op(31,yfx,-).

Operátoru můžeme využít i ke konstrukci např. rozhodovacího bloku IF THEN ELSE, která v PROLOGu není [2].

```
op(108,fx,if)    op(106,xfx,then)    op(104,xfx,else)

if P then C1 else C2 :- P,!,C1;C2.
```

Nyní lze v programu psát například

```
if A<B then V is B-A else V is A-B
```

3 ZÁVĚR

Programovací jazyk PROLOG má za sebou asi patnáct let historie. K jeho popularitě výrazně přispělo oznámení japonských odborníků, že byl vybrán, respektive logické programování, jako předběžný jazyk procesoru počítačů tzv. páté generace.

V současnosti existuje již několik implementací PROLOGu prakticky na všech kategoriích počítačů. Pro osmibitové počítače PROLOG 80, který byl zvolen za standart tohoto textu, Arity PROLOG pro šestnáctibitové počítače řady IBM PC, Turbo PROLOG fy Borland, který ale porušuje nepsaný standart, tzv. Edinburgskou verzi a má blíže k jakési syntéze PROLOGu a Pascalu, konečně zdařilý interpret KS k.o. pro počítače JSEP.

PROLOG je neprocedurální jazyk, v němž se programátor nesoustředí ani tolik na vlastní konstrukci programu, jako spíše na stanovení cílů, které mají být splněny. Vlastní generování výpočtu provádí odvozovací mechanismus jazyka.

Vhodný je především k symbolické manipulaci s daty, symbolické derivaci, projektování expertních systémů, méně

pro aritmetické výpočty.

Osobně se domnívám, že ačkoli PROLOG se nestane jakýmsi universálním jazykem budoucnosti, ale bezesporu sehrává důležitou roli ve vývoji programovacích jazyků. Kdo ví, co bude v tomto prudce se rozvíjejícím oboru za dvacet let, snad budou nové počítače vybaveny komunikačním jazykem s prvky umělé inteligence, vyrostlé z kmenů logického i procedurálního programování, možná tomu bude jinak. V každém případě myslím, že získat základní znalosti PROLOGu by mělo patřit k průpravě každého profesionálního programátora.

literatura

[1] Clockin,W.F.-Mellish,C.S.: Programming in PROLOG. 1.ed. Berlin Heidelberg, Springer Verlag 1981. Český překlad: Programováná v Prologu. Slušovice, JZD Závody aplikované kybernetiky 1986. 352 s.

[2] Csonto,J.: Prolog v příkladech. Sborník č. 19. ZO svazarmu Karolinka. Karolinka, 1988. 208 s.

[3] Koch,M.: Aplikace jazyka PROLOG a jeho postavení v ASŘ podniku. IN Aplikácia prvkov umelej inteligencie v podmienkach priemyselného podniku. Žilina, DT CSVTS 1989, s.100-125.

[4] Možnosti uplatnění poznatků umělé inteligence v automatizovaných informačních systémech řízení. Sborník přednášek. Brno, DT CSVTS 1989. 166 s.

[5] Štěpánek,P.-Štěpánková,O.: Logické programování a Prolog. IN Ročenka výpočetní techniky 1. Informatika. Praha, SNTL 1989. s.71-162.