což zaručuje, že program neobsadí tolik vnitřní paměti a provoz bude probíhat rychleji.
Mnoho odborníků vyslovuje názor, že problémy programování, tj. pronikavé snížení nákladů a zkrácení lhůt, vyřeší teprve rozsáhlé použití umělé inteligence, zvláště expertních systémů. Ale tyto postupy jsou teprve ve stadiu laboratorních výzkumů a potřebné práce jsou rozvrženy na mnoho let. Např. známý MIT (Massachusettský technologický institut — USA) předpokládá, že vytvoření automatického učedníka, který by pomáhal při programování (Knowledge-Based Software Assistant), bude trvat nejméně 15 let.

Podle zahraničních časopisů a firemní literatury

Dr. Dušan Novák, CSc.

Programovací jazyk PROLOG

Ruku v ruce s rozvojem umělé inteligence se uplatňuje programovací jazyk Prolog a podobně jako jazyk LISP je vzhledem ke své struktuře, odlišné od klasických programovacích jazyků jako je Pascal, C a další, právě vhodný pro úlohy z oblasti umělé inteligence, kde konvenční jazyky nevyhovují. Jako příklad je možno uvést:

— symbolické matematické systémy
— překlady přirozených jazyků
— expertní systémy
— logické závěry

Prolog vznikl v roce 1970 a zpočátku byl implementován pouze na velkých výpočetních systémech, až teprve v poslední době se stal užitečným prostředkem i pro systémy malé (CP/M, C-64).
Program napsaný v Prologu se svou podobá způsobu, jakým se vyjadřuje algoritmus v konvenčních jazycích. V Prologu se programátor „ptá", které vztahy mezi objekty se vyskytují v jeho problému a které z nich jsou pravdivé. Prolog je tedy popisný jazyk a neudává bezpodmínečně veškeré kroky, jichž je k řešení daného problému třeba. Výpočty se provádějí jednak pomocí deklarativní sémantiky Prologu, jednak pomocí nových fakt, které lze vydedukovat z daných objektů a jen zčásti pomocí řídících informací ze strany programátora.

Termy

Prolog zná jedinou strukturu dat: *term*, z něhož se konstruují veškerá data i programy. Program v Prologu se skládá z jednotlivých vět, přičemž každá věta obsahuje *fakt* o určitém objektu nebo předpis, jak je možno odvodit nebo v jakém vztahu je řešení problému k určitým objektům. Příkladem takového vztahu může být věta: „Karel vlastní počítač." Vztah „vlastní" spojuje tedy dva objekty: „Karel" a „počítač". Také pořadí, ve kterém se objekty uvádějí, je důležité, neboť není např. pravda, že počítač vlastní Karla. Na otázku: „Vlastní Karel počítač?" očekáváme odpověď ANO nebo NE.
Fakt je výrok o vlastnostech objektů nebo o vztahu mezi nimi. Předpis (pravidlo) zní např: „Dva lidé jsou sestrami, jestliže jsou oba ženského pohlaví a mají stejné rodiče." Předpis vypovídá, co znamená výraz „sestra" a kromě toho uvádí, jak je možno zjistit, zda dvě osoby jsou sestry.
Program v Prologu se skládá z těchto částí:
1. deklarování faktů o objektech a jejich vztazích,
2. definování pravidel o objektech a jejich vztazích,
3. kladení otázek o objektech a jejich vztazích.

Prolog je dialogový jazyk, tzn. vloží se údaj a Prolog ukáže reakci; pak se celý cyklus opakuje.

Konstanty

Konstanty slouží k pojmenování určitých objektů nebo vztahů. Jsou dva druhy konstant: atomy a celá čísla. U atomů rozeznáváme dvě skupiny: jedna skupina obsahuje abecední a číselné znaky, druhá zvláštní symboly. První skupina musí začínat malým písmenem a může obsahovat i podtrhávací znaménko, druhá se může skládat pouze ze symbolů. Atomy s jednoduchými uvozovkami mohou obsahovat všechny znaky:

otto, =, —>, 'Halo!' a12,

Čísla jsou známa z normálních jazyků a tvoří základ pro aritmetické operace. Často se omezuje rozsah jejich hodnoty na 16 bitů, nicméně Prolog je aplikován na oblasti, kde vystačí jednoduchá aritmetika s jednoduchými čísly.

Fakty

Abychom vyjádřili vztah „Karel vlastní počítač", musíme jej formalizovat takto:

vlastní (karel, počítač)

— protože jména objektů a vztahů jsou konstantní, musí začínat malými písmeny;
— vztah se uvádí na prvním místě a pak se uvádějí objekty. Oba objekty se od sebe oddělují čárkami a jsou v závorkách;
— znak '.' ukončuje fakt.

Další příklad: rychlý (počítač), tzn. „Počítač je rychlý". Vztah nazýváme *výrokem* a slova v závorce *argumenty*. Vrátíme-li se k předchozím příkladům, pak vztah „vlastní" je vztah se dvěma argumenty, vztah „rychlý" s jedním argumentem. Více faktů tvoří *databázi*.

Formulace otázek

Obsahuje-li databáze více faktů, je možno se na tato fakta dotazovat a otázky se formulují pomocí atomu: ?—
Např. ?— vlastní (karel, počítač).
Jakmile se tato otázka položí, hledá Prolog fakty, které se hodí k faktu obsaženém v otázce. Nalezne-li odpovídající fakt, odpoví na otázku ANO, v opačném případě NE. Položíme-li např otázku: ?— vlastní (jiří, počítač), bude odpověď znít NE.
Kdybychom se chtěli dovědět, kdo vlastní počítač, mohli bychom klást otázky „Vlastní počítač Jiří?", „Vlastní počítač Karel?" atd. Prolog zná ovšem jednodušší způsob, je možno se zeptat: „Kdo vlastní počítač". Pro tento účel existují v Prologu proměnné.

Proměnné

Proměnné jsou termy, které jsou na první pohled podobné konstantám, liší se pouze tím, že začínají velkými písmeny nebo podtrhávacím znaménkem. Byla by možná otázka: „Vlastní X počítač?" V okamžiku, kdy klademe otázku, je význam X neznámý.
Proměnné v Prologu mohou nabývat dva stavy:
— mohou být volné; v tom případě je význam proměnné neznámý,
— mohou být vázané; tzn. proměnná znamená určitý objekt.

Způsob, jakým Prolog postupuje, je patrný z tohoto příkladu:
V paměti jsou uloženy tyto fakty:
— vlastní (karel, počítač).
— vlastní (jiří, počítač).

Jestliže se zeptáme: ?— vlastní (X, počítač), pak je zpočátku X volnou proměnnou a Prolog vyhledává v databázi fakt, který odpovídá dané otázce. Protože X je dosud volné, hodí se otázka na každý argument každého faktu. Prolog tedy hledá fakt, který obsahuje výrok „vlastní" a dále druhý argument

„počítač". První argument je cosi dosud libovolného. Vyhledávání faktů v databázi probíhá v pořadí, v jakém byly uloženy. V našem případě bude prvním nalezeným faktem: vlastní (karel, počítač).

V tom okamžiku je X vázáno na první argument „karel" a Prolog vytiskne:

X = karel

a čeká na další vstupy. Výsledek hledání je zaznamenán do databáze. Stiskneme-li nyní RETURN, další hledání se zastaví.

Vložíme-li jako další vstupní údaj středník, hledání pokračuje. Jestliže se tedy vytiskne ' ; ', změní se stav X na „volné" a Prolog se zastaví u druhého faktu a váže X na „jiří". Vložíme-li znovu středník, pak se opět uvolní X a hledání pokračuje. Protože v databázi žádný další fakt není, vytiskne Prolog „ne".

Slučování otázek

Více otázek je možno slučovat pomocí konjunkce. Předpokládejme, že máme tato fakta:

```
chce (karel, pivo).
chce (karel, víno).
chce (susi, víno).
chce (karel, susi).
```

Chceme-li se dovědět, zda Karel a Susi se vzájemně chtějí, můžeme se zeptat:

?— chce (karel, susi), chce (susi, karel).

Čárka mezi oběma otázkami je konjunkce a používá se ke spojení jednotlivých otázek, na které se musí odpovědět ANO, jestliže odpověď na celou otázku má znít ANO. V našem případě bude odpověď NE, protože Karel chce Susi, ale nikoliv naopak.

Je možné se také zeptat, co Karel a Susi oba chtějí. Pak bude opět třeba aplikovat proměnné.

Formulace v Prologu:

?— chce (karel, X), chce (susi, X).

Tzn. pokyn: Hledej, co chce Karel a podívej se, zda to Susi také chce.

Prolog začíná první dílčí otázkou. Jakmile je nalezen fakt odpovídající otázce, tj.: chce (karel, pivo), je X vázáno na druhý argument (pivo) a následuje záznam v databázi. X si podrží svou hodnotu i pro druhou část otázky, která zní:

?— chce (susi, pivo).

Protože takový fakt neexistuje, odpověď bude znít NE, X se uvolní a hledání bude pokračovat od posledního záznamu v databázi.

Další hodící se fakt je:

?— chce (karel, víno).

X nyní nabývá významu „víno" a následuje záznam v databázi. Druhá část otázky zní:

?— chce (susi, víno).

Hledání začíná opět od začátku databáze, protože předchozí záporný výsledek (Karel a Susi chtějí pivo) zrušil záznamy v databázi. Tentokrát je výsledek úspěšný, takže obě dílčí otázky jsou zodpovězeny ANO. Prolog vytiskne:

X = víno

Chceme-li se dovědět, zda Karel a Susi chtějí kromě vína ještě něco společně, vložíme ' ; ' a Prolog pokračuje podle stejného schématu.

Z uvedeného příkladu vyplývá, že konjunkci dvou nebo více dílčích otázek zpracovává Prolog zleva doprava. Jestliže odpověď na dílčí otázku zní ANO, provede se záznam do databáze, případně volná proměnná se váže a zůstane vázána i v dalších dílčích otázkách. Potom provede Prolog pokus zodpovědět

další otázku (vpravo) a přitom začíná hledání od začátku databáze. Při kladné odpovědi dílčí otázky se provede záznam do databáze pro případ, že by bylo dílčí otázky později ještě třeba.

Při záporné odpovědi dílčí otázky se Prolog vrací k levému sousedovi a začíná hledání u existujících záznamů, aby zjistil, zda ještě některé další fakty se hodí. K tomu se musí uvolnit proměnné vázané v dílčí otázce, aby se hodily na další libovolné argumenty. Jestliže je dílčí otázka stojící zcela vlevo zodpovězena záporně, je celá konjunkce záporná.

Tento princip — pokus zodpovědět otázky a opakovaný návrat při záporných odpovědích — se nazývá Backtracking (sledování zpětné stopy) a je jednou ze základních charakteristik Prologu.

Odpovědi pomocí pravidel

Jestliže chceme vyjádřit, že Karel vlastní všechny počítače, můžeme zavést tuto formulaci:

```
vlastní (karel, apple).
vlastní (karel, mc68000).
atd.
```

Mnohem racionálnější je teze: „Karel vlastní každý objekt, kterým je počítač." V pseudoprologu by to znělo: „X je vlastněn Karlem, proto X je počítač." Ve správném Prologu se fakt vyjádří takto:

```
vlastní (Karel, X) : — počítač (X).
```

Symbol ' : — ' je opět atom, který určuje funkci. Řídící část pravidla: vlastní (karel, X) popisuje, co pravidlo definuje a podružná část pravidla: počítač (X) popisuje konjunkci dílčích otázek, které musí být zodpovězeny kladně, aby řídící část pravidla byla pravdivá. Aby bylo pravidlo použitelné, je nutno připojit několik faktů:

```
počítač (apple).
počítač (mc68000).
počítač (c64).
```

Položíme-li otázku:

```
?— vlastní (karel, X).
```

obdržíme tento sled odpovědí:

```
X = apple
X = mc68000
X = c64
no
```

přičemž za každou odpovědí je nutno vložit symbol ' ; '.

Jako další uveďme příklad, kde se jedná o příbuzenské vztahy. Nejprve fakty:

```
muž (otto).
muž (karel).
žena (susi).
žena (karin).
rodiče (karel, karin, otto).
rodiče (susi, karin, otto).
```

rodiče (x, y, z) znamená, že rodiče dítěte x jsou y a z. A nyní pravidlo: sestra (X, Y) (= X je sestra Y). To platí, jestliže X je ženského rodu a jestliže X a Y mají stejnou matku M a stejného otce O.

Vyjádřeno v Prologu:

```
sestra (X, Y) : — žena (X),
                  rodiče (X, M, O),
                  rodiče (Y, M, O).
```

První možná otázka:

?— sestra (susi, karel).

X bude vázáno na „susi" a Y na „karel", pak bude zkoumána první otázka:

žena (susi).

Toto tvrzení je pravdivé, proto se přechází ke zkoumání další dílčí otázky:

rodiče (susi, M, O).

M a O jsou volné, proto se hodí na libovolné argumenty. Hodící se fakt je:

?— rodiče (susi, karin, otto).

M znamená Karin a O znamená Otto. Protože odpověď na tuto druhou dílčí otázku je kladná, pokračuje se třetí a poslední otázkou:

?— rodiče (karel, karin, otto).

Protože fakt se hodí i na toto tvrzení, odpoví Prolog na dotaz, zda Susi je sestrou Karla ANO.

Položíme-li poněkud složitější otázku: „Čí sestra je Susi", vyjádřeno v Prologu:

?— sestra (susi, X).

V pravidle je X vázáno na Susi, Y je volné, v otázce je X volné.

První dílčí otázka:

žena (susi).

je zodpovězena kladně. Pokračuje se druhou otázkou:

rodiče (susi, M, O).

M znamená Karin a O Otto. Protože i druhá otázka byla zodpovězena kladně, pokračuje se třetím krokem:

rodiče (Y, karin, otto).

Y je zde ještě volné, protože v otázce bylo Y také volné.

Tato třetí dílčí otázka se hodí na tvrzení: rodiče (karel, karin, otto), z čehož vyplývá, že Y je vázáno na „karel". Protože i třetí dílčí otázka byla zodpovězena kladně, je konjunkce pravdivá. X v otázce je rovnocenné Y v pravidle, takže X bude vázáno na „karel" a Prolog odpoví:

X = karel.

Aritmetika

Pro aritmetické operace jsou k dispozici operátory: +, —, *, / a mod pro porovnání hodnot: =, \=, >, <, >= a =<.

Jako příklad vezmeme hustotu obyvatelstva v některých zemích. Nejprve uvedeme několik faktů, které vypovídají, kolik miliónů obyvatel má ta která země:

obyv (usa, 203).
obyv (indie, 548).
obyv (čína, 800).

Dále uvedeme rozlohu těchto zemí v miliónech čtverečních kilometrů:

plocha (usa, 6).
plocha (indie, 2).
plocha (čína, 4).

Pro výpočet hustoty obyvatelstva vydělíme počet obyvatel počtem čtverečních kilometrů podle následujícího pravidla:

```
hustota (X, Y) : — obyv (X, B),
                   plocha (X, F),
                   Y is B/F.
```

Čteme: Hustota obyvatelstva země X je Y, jestliže počet obyvatel země X je B a plocha země X je F a jestliže Y se vypočte zlomkem B/F. Operátor „is" přiřazuje volné proměnné vlevo (Y), výsledek matematického výrazu vpravo (B/F). Odpověď počítače na otázku:

?— hustota (čína, X)

zní: X = 200, tzn. 200 obyvatel na čtvereční kilometr. Na otázku:

?— hustota (Francie, X)

odpoví počítač „no", protože v databázi není k této otázce žádná informace.

Struktury

Struktury jsou dalším druhem termů. Struktura je objekt, který se skládá z více dalších objektů, v nejširším slova smyslu je to srovnatelné s Record v Pascalu. Můžeme např. zapsat fakt:

vlastní (karel, mc68000); podrobnější informace by zněla:

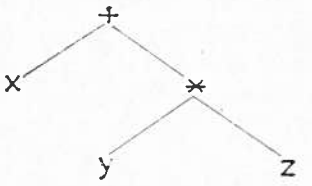
vlastní (karel, počítač (mc68000, motorola)); nebo ještě podrobněji:

vlastní (karel, počítač (mc 68000, firma (motorola, usa))).

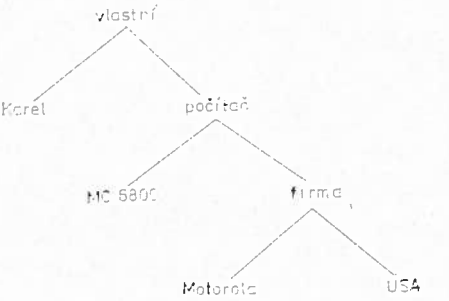
Abychom s dověděli, které Karlovy počítače pocházejí z USA, zeptáme se:

?— vlastní (karel, počítač (X, firma Y (usa))).

V odpovědi udává X typ počítače a Y jméno firmy. Pokud není možno vystačit s označováním variant nebo je nepohodlné si stále vymýšlet nová označení, je možné použít anonymní variantu '_', která se může vyskytnout i vícekráte v jednom termu. Přitom se chová tak, jako by při každém výskytu měla jiné jméno. Jak ukazuje příklad, je Prolog vhodný pro relační databanky, kde je třeba vyhledávat podle určitých kritérií. Také matematické výrazy je možno pojmout jako struktury, např. vzorec x + y*z je možno vyjádřit jako strukturu takto: + (x, * (y, z)), viz obr. 1. Jako strukturu je možno rovněž vyjádřit Karlův počítač, viz obr. 2.



Obr. 1



Obr. 2

Rovnosti

V Prologu je možno psát X = Y. Interpretace závisí vždy na povaze X a Y:

— Je-li X volné a Y je libovolný objekt, pak je výrok pravdivý a X bude vázáno na objekt označovaný Y.
— Rovněž, jestliže X a Y jsou volné, je výrok pravdivý a X a Y jsou ekvivalenty. Jestliže je tedy vázána jedna z těchto dvou proměnných, bude vázána i druhá, a to na týž objekt.
— Čísla typu integer a atomy jsou si sobě rovny.
— Dvě struktury se rovnají, jestliže obsahují shodný výrok, stejný počet argumentů a jestliže jsou všechny argumenty stejné.

Soubory

Soubory (Lists) jsou dalším známým objektem v nenumerickém programování. Soubor je utříděný sled prvků. Přitom tyto prvky mohou být atomy, struktury nebo jakékoliv jiné termy, ale také opět soubory. Proto se hodí pro řešení takových problémů, u nichž není předem určena velikost datové struktury nebo typ elementů. Soubor může být buď prázdný, tzn. neobsahuje žádné prvky, nebo se skládá z „hlavičky" (první prvek) a „zbytku". Konec každého souboru se vyznačuje prázdným souborem. Prázdný soubor se označuje [].

Soubory je možno vytvářet pomocí operátoru '.', přičemž první argument je hlavička a druhý argument zbytek:

```
.(a,[])
```

Analogicky je možno vytvořit soubor s hlavičkou 'a' a se zbytkovým souborem, jehož hlavičku představuje prvek 'b', atd.

```
.(a,.(b,.(c,[]))).
```

Používání operátoru '.' není právě příjemné a proto existuje ještě další možnost zápisu, a to: prvky jsou uzavřeny v hranatých závorkách a jsou od sebe odděleny čárkami.

```
[a, b, c].
```

Takto vytvořený soubor je zcela identický se souborem vytvořeným podle předchozího zápisu shora.

V zápisu:

```
[a, [halo, otto], b, c].
```

jsou dva soubory v sobě, tj. druhý prvek souboru je opět soubor. Takovéto prolínání souborů lze opakovat libovolněkrát.

Kromě operátoru pro vytvoření souboru je přirozeně také zapotřebí další operátor, kterým by bylo možno provádět výběry ze souboru. K tomu účelu užívá Prolog tento výraz: X/Y.

Tento výraz váže X na hlavičku a Y na zbytek souboru. Jestliže zapíšeme tyto fakty:

```
soubor ([1, 2, 3]).
soubor ([a, [halo, otto], b, c]).
```

je možno položit tuto otázku:

```
?— soubor ([X/Y]).
```

a získáme tuto odpověď:

```
X = 1, Y = [2, 3].
X = 1, Y = [[halo, otto], b, c].
```

Řetězce znaků se rovněž zobrazují jako soubory.

V souborech se dále užívá velmi často pravidlo 'member', kterým se zjišťuje, zda určitý prvek je obsažen v určitém souboru. Pravidlo je definováno rekurzívně, nejprve však nerekurzívní část: prvek je v souboru, jestliže tvoří hlavičku souboru, vyjádřeno v Prologu:

```
member (X, [X|_]).
```

Znak '_' je opět anonymní proměnnou. Podmínka je pravdivá, jestliže X je hlavička souboru a zbytek je něco, co dále nebude zajímavé. Jestliže podmínka není pravdivá, je nutno postupovat rekurzívně: prvek je obsažen v souboru, jestliže je obsažen ve zbytku souboru. Zbytek souboru je většinou opět souborem a proto je vyjádřen rekurzívní funkcí:

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

Podmínka je pravdivá, jestliže je X obsaženo ve zbytku souboru, jehož hlavička obsahuje něco, co není zajímavé.

Příklady:

```
?— member (a, [halo, a, otto]).
yes
?— member (a, [b, c]).
no
```

Úvodní informace, která v rámci tohoto příspěvku byla možná, ukazuje výkonnost programovacího jazyka Prolog. Ve srovnání s jinými jazyky umožňuje relativně jednoduchým způsobem, prakticky pomocí několika málo faktů, vyjádřit i složité vztahy, tzn. jeho aplikace je velmi efektivní.