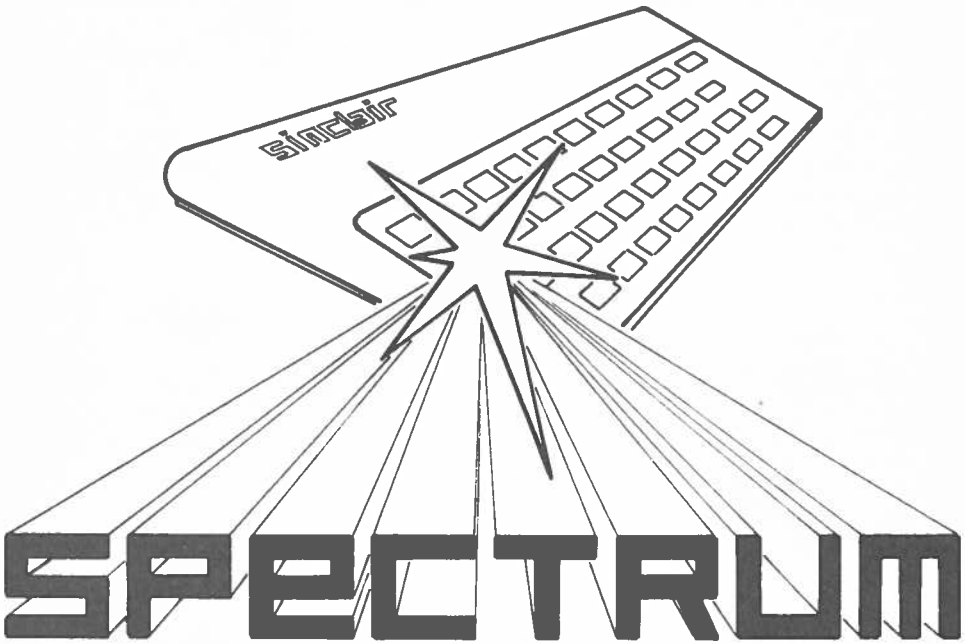
ZO SVAZARM KAROLINKA

SBORNÍK č. 10

MICROPROLOG



MICRO PROLOG

pro

ZX SPECTRUM

P r o g r a m o v a c í j a z ý k

P R O L O G

verze: micro-PROLOG : verze FELCVUT

Komentovaný výpis jazyka micro-PROLOG pro ZX Spectrum

Typ logického programování pro úkoly a pro práce s nečíselnými údaji, relační databáze, matematickou logiku, řešení abstraktních problémů, pro práci s přirozeným jazykem a v oblastech umělé inteligence.

Prolog - slouží jako základ logického programování oficiálně přijatý v Japonsku pro perspektivní vývoj.
Je určený pro:
- paralelní počítačovou architekturu /dataflow machine/
- uplatnění umělé inteligence /Artificial Inteligence AI/
- pro znalost bází /Knowledge Base/
- pro systémy se znalostí bází /Knowledge-Based Systems/
V ČSSR: implementace jazyka PROLOG v ČVUT na FEL. Jednodušší verze jazyka PROLOG je označována jako:

micro-PROLOG

Vhodný pro rozsáhlé aplikace a pro seznámení se strukturou jazyka.

2. Implementace na ZX Spectru + výuková varianta SIMPLE.

Micro-PROLOG je verzí rodiny jazyků PROLOG, kde každá verze je určitou podmnožinou verze základní. Hlavní rozdíly mezi jednotlivými verzemi spočívají v rozdílnosti syntaxe programů a přípustných forem dotazů.
Připomínka: PROLOG má odlišnou koncepci od všech dosavadních jazyků.

3. Nahrání překladače micro-PROLOGu do počítače.

Překladač v jazyku micro-PROLOG je k dispozici pro počítače Sinclair Spectrum s pamětí 48 K. K uložení do počítače použijte výraz:

LOAD "PROLOG"

nebo

LOAD "" - pokud je kazeta nastavena přímo před záznam s překladačem, nyní proveďte následující činnost:

- umístěte kazetu s programem micro-PROLOG do mgf
- ujistěte se, že je kazeta na počátku nahrávky
- nastavte hlasitost přibližně na polovinu
- nastavte tónovou clonu přibližně do poloviny
- stiskněte klávesu ENTER
- stiskněte tlačítko PLAY na mgf

Jestliže jsou všechny ovládače na magnetofonu správně nastaveny, zobrazí se na krajích obrazovky televizoru vodorovné pruhy. Zaváděcí program - červené a modré silnější pruhy, přenos dat - žluté a modré tenké pruhy.
Micro-PROLOG se nahrává po několika částech. V každé části se opakuje čtení zaváděcího záznamu a vlastních dat.

Pokud probíhá nahrávání správně, trvá nahrání micro-prologu asi 1 minutu. Po ukončení nahrávání pruhy zmizí a zobrazí se následující text:

```
Spectrum micro-PROLOG T1.Ø
    1983 LPA Ltd
XXXXXXX
 aend .L
```

Dále je možné zavést výukový program SIMPLE front-end.

4. Systém SIMPLE

Micro-PROLOG je velmi flexibilní jazyk. Při seznamování s ním je výhodné sestavit několik programů různého typu. Program SIMPLE ukazuje způsob použití jazyka. SIMPLE je zaznamenán na kazetě bezprostředně za micro-PROLOGEM, a musí se nejprve nahrát do paměti počítače.
Na levé straně obrazovky máte pod úvodním textem znak "aend". Vedle tohoto znaku je blikající kurzor L. Kurzor indikuje pozici, na které bude znázorněn následující znak. Na počítači Spectrum zároveň indikuje typ znaku, který bude zobrazen při použití odpovídající /přeřaďovací/ klávesy.
Všimněte si, že po stisknutí CAPS LOCK / CAPS SHIFT současně s klávesou 2 se mění kurzor na C, což naznačuje že všechna následující písmena budou velká. Po dalším stisknutí naopak,

Kombinaci znaků aend.L označujeme jako příznak aktivity /prompt/ a ten informuje uživatele o tom, že micro-PROLOG očekává vstup z klávesnice. Napište:

```
LOAD SIMPLE
```

LOAD a SIMPLE se musí napsat normálně jako z klávesnice psacího stroje. Během psaní držte stisknutou klávesu CAPS SHIFT,

aby byl text napsán velkými písmeny.

Jestliže napíšete chybu a chcete ji opravit, použijte klávesu DELETE /CAPS SHIFT a současně Ø/. Tak bude zrušen poslední znak vlevo od kurzoru. Po stisknutí klávesy ENTER spusťte Váš magnetofon. Po několika sekundách se zobrazí na kraji obrazovky vodorovné pruhy. Je tedy nahrán program SIMPLE, který tvoří celou řadu krátkých datových bloků s dobou nahrání 1 minuty. Po nahrání každého bloku prověřuje micro-PROLOG správnost dat a vypíše:

SIMPLE Ø1 BLOCK OK

Toto sdělení udává název souboru a číslo přečteného bloku. Když z jakéhokoli důvodu nepřečte počítač blok správně, vypíše se na display místo BLOCK OK tento text:

NEW /nezapomeňte zastavit mgf!/

Tímto způsobem se v paměti zruší všechny dosud nahrané záznamy. Převiňte pásku a nahrajte SIMPLE znovu. Po ukončení nahrávání se zobrazí příznak aktivity "aend". Vypněte pak mgf, SIMPLE je nahráno na počítač a tento je připraven přijmout další příkaz.

5. Používání klávesnice v micro-PROLOGU.

Micro-PROLOG nepoužívá systému klíčových slov. Flexibilita jazyka umožňuje uživateli definovat v případě potřeby vlastní klíčová slova. Po nahrání SIMPLE je definována množina klíčových slov. S těmito se seznámíte v dalším oddíle. Při chybě vypíše ERROR, zrušit text můžete klávesou DELETE. Pokud napíšete nesprávný text, můžete se kdykoliv vrátit k výpisu příznaku aktivity "aend" stiskem: BREAK/SIMBOL SHIFT společně s BREAK. Kontrola syntaxe takto zadaného příkazu bude provedena. Výpis po stisku BREAK říká, co počítač prováděl v okamžiku přerušení.

6. Velká a malá písmena.

Ve většině případů je výhodné používat malá písmena.

Oddělovače a aritmetické operátory.

Dalším druhem znaků, které budeme potřebovat, jsou oddělovače a aritmetické operátory. K jejich zadání je třeba stisknout červenou klávesu SYMBOL SHIFT a klávesu s požadovaným /červeným/ znakem.

Po stisknutí klávesy s černě vypsanými klíčovými slovy BASICU se vypíše otazník, který vyznačuje, že jste vypsali něco, čemu micro-PROLOG nerozumí.

Několik operátorů je natištěno pod klávesami. Jeden z nich je důležitý pro micro-PROLOG / je nazýván " bar " / a ještě několik dalších lze používat. Zapisují se v rozšířeném modulu, t.j. při současném stlačení kláves CAPS SHIFT a SYMBOL SHIFT. Zjistíte, že blikající kurzor L / očekává se písmeno, číslo nebo operátor / se změní na E, což označuje přechod do rozšířeného /extendet/ modu.

7. Rušení znaků

Užitečná je klávesa DELETE. Poslední znak zobrazený vlevo od kurzoru se zruší klávesou Ø, přičemž se musí držet klávesa CAPS SHIFT. Klávesy pro posuv kurzoru vlevo a vpravo /klávesy 5 a 8/ dovolují pohybovat kurzorem na řádku. Kurzor lze přesunout na požadované místo, opravit, či změnit, respektive zrušit znaky a potom předat text počítači stisknutím klávesy ENTER.

8. Použití programu SIMPLE pro popis jevů.

Většina programovacích jazyků se skládá ze sekvence příkazů pro počítač. PROLOG je odlišný, protože pracuje s popisem běžných jevů a vztahů.
Pro porozumění:
Máte obrázek /obdélník/ ve kterém jsou čtyři jednoduché geometrické obrazce: malý čtverec, šestiúhelník, trojúhelník a obdélník/. Tuto informaci lze předat počítači prostřednictvím SIMPLE micro-PROLOGu formou, které bude rozumět:

čtverec nad trojúhelník

Jev se rozdělil na tři části: máme dva objekty - čtverec a trojúhelník a relaci, která je spojuje: "nad".

Když připravíme převod našich popisů do takové formy dovolí nám micro-PROLOG pokračovat v používání běžných českých slov v ASCII kódu, bez gramatiky.
Sdělme tedy počítači popis obrázku:
Počítač zobrazí nejprve vždy jako první příznak aktivity /"AEND"/. To indikuje, že můžeme micro- PROLOGu zadat informaci. Pokud se tento znak nezobrazí, stiskněte BREAK /SYMBOL SHIFT a BREAK/. To je jediná cesta jak zastavit počítač v jeho činnosti a zahájit novou činnost.

Poznámka: Klíčová slova, která se naučíme v této části, jsou podmnožinou klíčových slov programu SIMPLE front-end. Abychom je mohli používat, musíme nahrát SIMPLE. Pokud chceme nahrát micro-PROLOG bez SIMPLE, musíte se řídit popisem vlastního jazyka micro-PROLOG Primer.

9. add - příkaz pro vložení výrazu.

Příkaz používáme tehdy, když chceme počítač naučit něco nového. Zadejte svůj první příkaz:

```
add /čtverec nad trojůhelník/
```

Za klíčovým slovem následuje levá závorka, potom text a nakonec pravá závorka. Pamatujte si, že mezi slovy můžete napsat jednu nebo více mezer, aby byl text zřetelnější. Potom stiskněte ENTER. Počítač zadanou informaci zadrží v paměti. K získání přehledného výpisu všech dosud zadaných informací napište:

```
list all  / a stiskněte ENTER /
```

Počítač odpoví:

```
čtverec nad trojůhelník
aend
```

Teď předáme počítači další informaci o našem obrázku:

Poznámka: Při zadávání nových informací se můžeme dopustit chyb. V tomto případě se obraťte k odstavci o opravách, rušení řádků a informací v počítači.

Při zadávání výrazů příkazem add nemusíte zadat pravou uzavírací závorku a můžete stisknout ENTER. Zjistíte, že místo příznaku aend se vypíše 1. na začátku nového řádku. Neztraťte odvahu, protože micro-PROLOG je velmi tolerantní, dovoluje zadat výraz na více řádcích. Výpis"1" označuje, že počítač očekává jednu uzavírající závorku. Počítač se nevrátí k výpisu aend dokud nenapíšete odpovídající počet závorek.

```
aend. add /šestiůhelník nad obdélník/
aend. add /šestiůhelník vpravo-od čtverce/
aend. add /obdélník vpravo-od trojůhelníku/
aend.
```

V dalších výrazech jsme použili nový relační operátor vpravo - od. Každý výraz tohoto typu se skládá ze tří slov - dvou objektů a relačního operátoru. Slova jsou navzájem oddělena jednou nebo více mezerami. Kdybychom napsali "vpravo od" zjistil by počítač výskyt čtyř slov a ohlásil chybu. Proto při nutném spojení více slov musíme použít pomlčky.

Nyní, co jsme zadali počítači:

aend. list all /nezapomeňte vždy zadat ENTER/
čtverec nad trojúhelník
šestiúhelník nad obdélník
šestiúhelník vpravo - od čtverce
obdélník vpravo - od trojúhelníku
aend.

Rovněž lze vypsat výrazy zadané s určitým relačním operátorem:

aend. list vpravo - od
šestiúhelník vpravo - od čtverce
obdélník vpravo - od trojúhelníku
aend.

Zadávání objektů počítači při zápisu informací tímto způsobem je označováno jako výstavba databáze. Databázi tvoří soubor dat, tedy množin informací.

10. Používání databáze - dotazy.

Informace uložené v databázi micro-PROLOGu budou užitečné až při jejich využití prostřednictvím dotazů.

11. is - Potvrzení předpokladu.

Prvním typem dotazu je otázka is /= je ?/, kterou můžeme zodpovědět jednoduše YES nebo NO /ano, ne/. Je čtverec nad trojúhelníkem? Je to trojúhelník vpravo od šestiúhelníku? Tyto otázky klademe prostřednictvím klíčového slova SIMPLE a tím je právě is. Otázka "Je čtverec nad trojúhelníkem?" Bude realizována v micro-PROLOGu takto:

is /čtverec nad trojúhelník/ počítač

odpoví:

Evalution error 2. No definition for
relation tryng : pod /čtverec trojúhelník/

Počítač nám tímto sdělil, že nerozumí slovu pod. Abychom zjistili relační operátory uložené v paměti počítače, můžeme napsat:

aend. list diet

Pro naší databázi počítač odpoví:

nad fict
vpravo - od dict
aend

Tato slova představují slovník relačních operátorů, kterým

počítač rozumí. Pouze těchto slov můžete použít ke konstrukci dotazů.
Ještě několik příkladů dotazů:

```
aend. is /šestiúhelník nad obdélník/
YES
aend. is /šestiúhelník vpravo - od čtverce/
YES
aend. is /šestiúhelník nad šestiúhelník/
NO
aend. is /druhé patro nad první patro/
NO
aend.
```

Poznámka: Počítač odpoví pouze na ty dotazy, které má k dispozici v paměti,
I když je odpověď pravdivá / 2.patro je vždy nad prvním patrem/ není pro počítač logická. Odpověď NO.
Jestli jste nedostal očekávanou odpověď, zkontrolujte stejnost slov dotazu a databáze. Micro-PROLOG rozlišuje malá a velká písmena, takže šestiúhelník a Šestiúhelník jsou dvě odlišná slova.
Chceme-li klást obšírnější dotazy jako např:

```
aend.is /šestiúhelník vpravo-od čtverce and
        1. šestiúhelník nad obdélník/
YES
```

Odpoví počítač YES jen v případě, že jsou všechny části dotazu pravdivé, jinak je odpověď NO.

Musíme si uvědomit, že v češtině často vynecháváme slova místo úplného vyjádření podmínky. Zkusme dotaz:

```
aend.is /šestiúhelník vpravo - od čtverce and
1. nad obdélník
NO
```

Tento dotaz je NO. Úplný dotaz se vždy tvoří ze základních dotazů, z nichž každý musí být úplně jiný.

Dotazu is může být rovněž použito pro jiný typ dotazů jako např: "je něco nad trojúhelníkem?". V takových dotazech slovo "něco" nebo cokoliv nahrazují objekty, které neznáme. Jestliže je šestiúhelník vpravo od čtverce, "něco" v původním dotazu ukazuje na čtverec.

V micro-PROLOGu se používá písmena x,y, z /rovněž X,Y,Z x1,y1 atd./ která nahražují neznámé údaje.
Zadejme takové dotazy v SIMPLE micro-PROLOGU :

```
aend. is /šestiúhelník vpravo - od x /
YES
aend. is /x nad trojúhelník/
YES
```

Znak x v těchto otázkách nahražuje slovo "něco" nebo "cokoliv" používané v češtině.

Další způsob užití této techniky užití neznámých proměnných nahrazujících část našich dotazů.

12. which - výběr informací

Na většinu dotazů se nelze ptát ani jednoduchými slovy jako YES nebo NO. Např: Jaká je nejbližší cesta ke stanici, kolik vážíte? atd. Na takové dotazy se odpovídá podrobnějšími informacemi. V micro-PROLOGu se požaduje často více odpovědí /informací jako odpověď/ na kladenou otázku. Proto další slovo which.

Předpokládejme, že chceme nalézt odpověď na otázku: "který obrazec je nad obdélníkem?" Tážeme se takto:

```
aend.which /x : x nad obdélník/        /který/
```

odpověď:

```
šestiúhekník
NO /more/ answers /žádná další odpověď/
```

Probereme si dotaz podrobněji:

```
which /x        : x nad obdélníkem/
    vzor odpovědi    ukazatel výběru
```

Ukazatel výběru se podobá probíranému typu dotazu. Znak "x" nahrazuje "cokoliv", co může počítač nalézt nad obdélníkem. Vzor odpovědi říká počítači co má vytisknout: V tomto případě to bude hodnota x

Podívejme se nyní na několik dalších příkladů dotazů:

```
aend.which /x:šestiúhelník vpravo-od x/
čtverec
NO /more/ answers
aend,which /x : x nad čtverec/
NO /more/ anwers
```

Na našem obrázku není nic nad čtvercem a proto není vytištěne žádná odpověď.

```
aend.which /x : x nady/
čtverec
šestiúhelník
NO /more/ answers
```

Často je nalezeno více odpovědí než jedna. V tomto případě představuje y další neznámou. Česky můžeme tento dotaz vyjádřit takto: "Jaký obrazec je umístěn nad libovolným obrazcem?".

Jako dotazu is můžeme použít množinu podmínek, jejichž splnění požadujeme:

```
aend.which /x : x nad obdélník and x /
1. vpravo od čtverce
šestiúhelník
NO /more/ answers
```

Již víme, že první část dotazu je určena k definování údajů, které chceme vytisknout. Těch může být více, nikoliv pouze jeden.

```
aend.which /x y : x nad y /
čtverec trojúhelník
šestiúhelník trojúhelník
NO / more/ answers
```

V tomto dotazu se ptáme na dvojici neznámých objektů, z nichž je jeden nad druhým. Do části pro tisk může být rovněž zařazen další text jehož výpis požadujeme:

```
aend.which /OBRAZEC x : y vpravo - od x /
OBRAZEC čtverec
OBRAZEC trojúhelník
NO /more/ answers
```

Nejčastější chybou připoužívání dotazu which je opomenutí pravidla, že každý which musí obsahovat vzor odpovědi a ukazatel výběru. Uživatelé často píší:

```
aend.which /x nad y/
```

místo správného

```
aend.which /x : x nad y/
```

První z těchto dotazů vyvolá chybu, protože chybí uvedení vzorů odpovědí. Na druhý výraz bude napsána odpověď:

```
čtverec
šestiúhelník
NO /more/ answers
```

13. Pravidla pro rozšíření schopností databáze.

Prozatím jsme se naučili jak vkládat výrazy, vypisovat je a tvořit dotazy. Jsme tedy už připraveni na využití dalších rozšíření programů v micro-PROLOGu

Vypište si obsah databáze:

```
aend.list all
čtverec nad trojúhelník
šestiúhelník nad obdélník
šestiúhelník vlevo - od čtverce
obdélník vpravo - od čtverce
aend
```

Nyní nás bude zajímat nový dotaz - jaký je obrazec pod čtvercem? Již jsme dostali správnou odpověď od počítače na ekvivalentní dotaz:

```
aend.which /x : čtverec nad x/
trojúhelník
NO /more/ answers
```

Tento dotaz můžeme vyjádřit uvedeným způsobem, protože víme, že "pod" a "nad" jsou vzájemným opakem. Počítač vůbec nic neví o relačním operátoru pod, ale můžeme mu ho předat prostřednictvím dalších výrazů:

```
aend.add /trojúhelník pod čtverec/
aend.add /obdélník pod šestiúhelník/
```

To však pouze duplikuje informace již obsažené v databázi. Místo toho můžeme definovat vztah mezi "pod" a "nad" jako obecné pravidlo stejným způsobem. Je zřejmé, že když je x nad y, potom musí být y pod x.

```
aend.add /x pod y if y nad x/
```

Přidali jsme nové pravidlo stejným způsobem jako výraz t.j. vztah mezi objekty. Opět jsme použili proměnné /x,y/ jako náhradu neznámých hodnot. Pravidla mohou být vypsána pomocí příkazu list

```
aend.list pod
X pod Y if
        Y nad X
```

Výraz "pod" můžeme používat stejným způsobem jako "nad".

```
aend.is /čtverec pod trojúhelník/
NO
aend.which /x : x pod šestiúhelník/
obdélník
NO /more/ answers
```

Přidejme další pravidlo:

```
aend.add /x vlevo - od y if y vpravo - od x/
```

Vyzkoušejte činnost databáze zadáváním dotazů s novými pravidly.

14. Opravy, změny a rušení výrazů.

Při návrhu je často chyba špatně provedeného zápisu nebo je třeba používat jiná slova pro popisy relací. Jedna možnost opravy je použití příkazu DELETE. Nejprve se vypíší příkazy z nichž chceme některý zrušit:

```
aend.list nad
čtverec nad trojúhelník
šestiúhelník nad obdélník
```

Předpoklad: chceme vyměnit druhý výraz ve kterém je nesprávně napsáno slovo obdélníkk. Napíše se:

```
aend. delete nad 2
aend
```

Navíc je k dispozici radikálnější prostředek ke zrušení informací dané množiny výrazů. Je to příkaz kill

```
aend kill nad
```

zruší všechny výrazy se slovem nad a vypíše:

```
Definition for nad deleted /výrazy obsahující nad zrušeny/
```

Úplné zrušení všech výrazů v paměti provádí příkaz

```
kill all
```

Počítač se pro kontrolu ptá, zda chceme zrušit opravdu všechno.

```
aend.kill all
Entire program? /yes/no/yes     /celý program ano/ne?/
Entire program deleted          /celý program zrušen/
aend
```

Obvykle není účelné abychom rušili větu proto, abychom ji napsali znovu ve správném znění. Můžeme použít příkazu EDIT, který nám umožní zobrazit příkaz a opravit nesprávné slovo.

```
aend, edit nad 1
1 /čtverec nad trojúhelník/
```

Údaj 1 indikuje pozici výrazu v seznamu výrazů obsahujících "nad". Pořadí výrazů můžeme změnit právě změnou vypsaného čísla.

Opravu věty můžeme provádět pohybem kurzoru /kurzory 5,8/ a klávesou DELETE. Rovněž lze přídat znaky pomocí nastavení kurzoru a zápisu nového textu. Po skončení opravy stlač ENTER.

15. Ukládání a nahrávání programů

Výsledek lze uložit kdykoliv během vývoje programu na mgf. Před uložením se program překontroluje. Spojí se konektory v zdířkách MIC. Konektor EAR nesmí být připojen.
Pro uložení naší databáze zapište:

```
save obrazce   / a stiskněte ENTER /
```

Počítač vypíše dotaz:

start tape for recording /spusťte pásku pro záznam/
Hit ENTER when /stiskněte ENTER až budete
ready připraveni/

Po stisknutí klávesy ENTER bude program uložen na pásku. Během nahrávání se na lemu zobrazí stejné pruhy jako při nahrávání programů micro-PROLOGu a SIMPLE a zároveň jsou vypisována čísla bloků.Po ukončení záznamu vypíše počítač příznak aktivity aend.

Uložený soubor má obrazce, soubor můžete nazvat libovolným textem obsahujícím čísla a písmena.
Rovněž lze použít příkaz SAVE překladač micro-PRCLOG. Tento se liší od uvedeného příkazu SAVE /součást SIMPLE/ tím, že po zadání příkazu probíhá okamžitě nahrávání bez kontrolního dotazu.

"Hit ENTER when ready"

Nahrávání:

load obrazce /stiskněte ENTER/

/stejně jako u SIMPLE fornt-end, ale load se píše malými písmeny:

load obrazec /a ENTER/

Č Á S T II.

PODROBNÝ POPIS PROGRAMOVÁNÍ V SIMPLE MICRO-PROLOG.

Typy relačních vztahů

Postup na příkladu rodinných vztahů:

/Henry st. otcem Henry/

/Henry otcem Mary/

Řídíme se vzorem:

jméno objektu - jméno relace - jméno objektu 2

Tyto relace zadávají vždy vztah mezi dvěma fakty, vyjádřením relace je slovo "otec". Jedná se tedy o binární reakci , která je vyjádřena formou infix.

Nevyjadřuje se pouze binárními relacemi. Např: osobám můžeme přiřadit pohlaví "muž", "žena" jedná se o unární relace. Výrazy unárních relací vyjadřujeme formou postfix /Henry - st muž/, /henry muž/, /Mary žena/

Třetí příklad představuje výraz, ve kterém např. "někdo" dává "někomu" - "něco" nebo "někdo" má určitou "výšku","váhu". Opět se jedná o binární relaci, ale zápis má poněkud odlišnou syntaxi, provádí se formou prefix

/dává/ Henry Mary kniha//

/osoba/výška barva- oči váha//

/SUM /2 3 5 // t.j. suma 2 + 3 + 5

Zapamatujte si, že prefix forma je obecná a binární relace i unární relace lze rovněž vyjádřit tímto způsobem, i když v řadě případů je jejich užití člověku srozumitelnější. Např. ekvivalentní jsou zápisy:

otcem /Henry - st Henry/

muž /Henry/

Jednotlivé složky se označují jako argumenty. "Henry" je první argument, "Mary" druhý a "kniha" třetí argument, zatímco slovo "dává" označujeme v logice jako predikát. Vytvořme pro další výklad databázi rodokmenu rodiny:

aend. add /Henry-st otcem Henry/

aend. add /Henry otcem Mary/

aend. add /Elizabeth matkou Henry/

aend. add /Katherine matkou Mery/

aend. add /Henry otcem Elizabeth 2/

aend. add /ANN matkou Elizabeth 2/

aend. add /Henry otcem Edward/

```
aend. add /Jane matkou Edward/
aend. add /Henry-st muž/
aend. add /Henry muž/
aend. add /Elizabeth žena/
aend. add /Katherine žena/
aend. add /Mary žena/
aend. add /Elizabeth 2 žena/
aend. add /Mary žena/
aend. add /Ann žena/
aend. add /Zena/Jane//
aend. add /Muž/Edward//
```

Slovník našeho programu se skládá z jmen relačních operátorů a ze jmen prvků: slovník obsahuje "věci" na které se můžeme následně dotazovat. Náš slovník obsahuje:

Henry st	jména prvků	otcem	jména relací
Henry		matkou	
Mary		muž	
Elizabeth		žena	
Katherine		Zena	
Elizabeth 2		Muž	
Ann			
Edvard			
Jane			

accept - příkaz pro vkládání formou prefix

Příkaz accept urychluje zadávání textů pomocí formy prefix. Když napíšete :

```
accept zena
```

bude na následujícím řádku opakovaně vypisován příkaz aktivity ve tvaru relace:

```
zena
```

Nyní můžete zadávat argumenty pro relaci "žena" ve tvaru seznamu: Režim vstupu accept ukončíte zápisem slova end. Naši databázi bychom mohli rozšířit pomocí příkazu accept takto:

```
aend. acept muz
muz. /Henry-st/
muz. /Henry/
muz. /Edvard/
muz. end
```

```
aend. accept zena
zena. /Elizabeth/
zena. /Katherine/
zena. /Mary/
zena. /Elizabeth2/
zena. /Ann/
zena. /Jane/
zena. end
aend.
```

Shrnutí dosud probraných příkazů

Všechny následující příkazy pracují s programem uloženým v daném okamžiku v paměti. V obecném zápisu používáme špičatých závorek pro určení syntaktické formy. Např:

výraz udává, že obsahem může být libovolný výraz.

add

Vzor 1: add / ⟨výraz⟩ /

Přidání věty argumentů v závorce na konec seznamu výrazů pro danou relaci

Vzor 2: add n / ⟨výraz⟩ /

Přidání věty argumentů v závorce na n-tou pozici v seznamu výrazů pro příslušnou relaci. Pokud je v databázi méně výrazů s touto relací než n, přidá se věta na konec.

delete

Vzor 1: delete / ⟨výraz⟩ /

Zruší zadaný ⟨výraz⟩ v programu.

Vzor 2: delete ⟨jméno relace⟩ n

Zruší n-tý výraz v seznamu uvedené relace.

list

Vzor 1: list ⟨jméno relace⟩

Výpis všech výrazů programu obsahujících uvedenou relaci.

Vzor 2: list all

Výpis všech výrazů v programu.

save

Vzor 1: save jméno programu

Uložení programu na kazetu. Jméno programu musí být odlišné od jmen relací obsažených v databázi a od názvů příkazů micro-PROLOGU. Pokud vypíšete program znovu přečtený z kazety, pak zjistíte, že výrazy nejsou v databázi uloženy tak, jak jste je původně znali, ale jsou převedeny do standartního tvaru micro-PROLOGU.

kill

Vzor 1: kill jméno relace

Zruší všechny výrazy obsahující dané jméno relace.

Vzor 2: kill all

Zruší celý program v paměti. Použijte pouze až po uložení programu na kazetu.

edit

Vzor edit jméno relace n

Umožní opravu n-tého řádku v seznamu s obsahem jména relace.

NEW

Resort micro-PROLOGU. Působí stejně jako kill all, ale navíc zruší i všechny další pomocné programy, jako např. SIMPLE. Tečka za NEW musí být uvedena, protože každý příkaz micro-PROLOGU musí mít argument.

Dotazy:

Pro ilustraci dotazů se použije již užitá databáze RODINA

Potvrzení dotazů

Nejjednodušším případem otázky je dotaz, kterým chceme potvrdit nějaký fakt. Dotazy vysvětlíme na dotazech v češtině. Pod českým dotazem je uveden ekvivalent dotazu v micro-PROLOGU. Př:

```
Je Henry otcem Elizabeth 2 ?
aend.is /Henry otcem Elizabeth 2/
YES
```

Dotaz prakticky ověřuje přítomnost druhého faktu v databázi. Výraz v závorce představuje masku, která je srovnávána s výrazem v databázi. Výraz v dotazu "smečuje" výraz v databázi, t.j. byl nalezen stejný krok /ztotožnil se výraz v dotazu s výrazem v databázi/.

Jako další příklad poslouží dotaz:

```
Je známa matka Mary ?
aend.is /x matkou Mary/
YES
```

V tomto dotazu se pokoušíme nalézt výrok v databázi, který obsahuje informaci o tom, kde je matkou Mary. Malé písmeno "x" nahražuje jméno nám neznámé matky. Micro-PROLOG hledá mezi výroky s relací "matkou" výrok:

x matkou Mary
nalezne výrok Katherine matkou Mary
a vrátí odpověď YES

Nedozvíme se však jméno matky: K tomu je třeba dotaz WHICH.

Proměnné v dotazech

Písmena x, y, z, X, Y, Z, atd. jsou proměnnými micro-PROLOGU. Proměnné nemohou být použity jako jména objektů či relačních operátorů. Nemůžeme tedy objekt označit jako x. Pokud bychom přesto potřebovali objekt takto pojmenovat, musíme použít k zápisu uvozovek, tedy "x", což už nebude chápáno jako proměnná. Potom "x" může označovat objekt či relaci: Písmena x, y, z, a X, Y, Z, jsou tzv. prefix písmena /počáteční písmena/ za kterými mohou následovat číslice. Proměnné označené různými indexy jsou různé. Tak x a y jsou odlišné od x2 a y2 či xl a yl. Ale pozor!, xl a xØl nejsou odlišné protože představují stejné celé číslo.

Vyhledávání dat

Pro vyhledávání jmen neznámého objektu použijeme dotaz which:

```
Kdo je otcem Edwarda ?
aend.which /x : x otcem Edward/
Henry
No /more/ ansvers
Kdo je dcerou Henryho?
 aend. which /x/ : Henry otcem x aend x zena/
Mary
Elizabeth2
No/more/ answers
```

Všimněte si, že aend lze v micro-PROLOGU nahradit znakem " ".

Vyjmenujte všechny matky s dítětem:

```
aend.which /<: x matkou Y/
Elizabeth
Katherine
Jane
Anne
No /more/ answers
```

Protože ve vzoru odpovědi nebylo jméno dítěte, nezískali jsme zároveň jeho jméno. Pokud je toho třeba, napíšeme:

Vyjmenujte všechny matky a jména jejich dětí:

```
aend.which /x y : x matkou y/
Elizabeth Henry
Katherine Mary
Jane Edward
Anne Elizabeth2
No /more/ answers
```

Vyjmenujte všechny otce, kteří mají syny / a jejich jména/:

```
aend.which /x otcem syna y : x otcem Y/
1.aend y muz/
Henry st otcem syna Henry
Henry otcem syna Edward
No /more/ answers
```

Shrnutí dotazů /druhů dotazů/

is

- is /⟨ podmínka⟩ [and ... podmínka⟩] /

kde každá podmínka je jednoduchý výraz, ve kterém mohou být jeden nebo více objektů nahrazeny názvem proměnné. Jestliže je stejná proměnná součástí více podmínek, musí tato proměnná představovat stejný objekt. Odpověď na tento jednoduchý dotaz zní YES nebo NO.

which

- which /⟨ vzor odpovědi⟩ : ⟨podmínka⟩ [and...⟨podmínka⟩]

tento dotaz odpovídá na podmínku v otázce nebo konjunkci.

one

- one /⟨vzor odpovědi⟩ : ⟨podmínka⟩ [and ...⟨podmínka⟩]/

Dotaz one je podobný dotazu which s tím rozdílem, že po wicht dostaneme všechny správné odpovědi, zatímco po one bude vypsána pouze první odpověď, výpočet bude přerušen a vypíše se příznak aktivity ve tvaru "more /y/n", po zadání "y" bude vypsána další odpověď po zadání "n" bude výpočet ukončen.

Vyjmenujte děti Henryho :

```
aend.one /x: Henry otcem x/
Mary
more? /y/x/ y
Elizabeth2
more? /y/n/ n
aend.
```

Protože jsme výpočet ukončili dříve než micro-PROLOG prohledal všechny varianty, nevypsal se text " No /more/ answers a systém pokračuje výpisem obvyklého příznaku aktivity "aend".

Vyhledávání zadaných jmen relací

Při každém zadání nového výrazu je nová relace v programu zařazena do slovníku relací. Proto můžete kdykoliv vypsat relace v programu dotazem:

all /x : x dict/

nebo identicky

list dict

Když zrušíte všechny výroky s určitým relačním operátorem příkazem kill, bude tento operátor rovněž ve slovníku relací. Avšak když budete rušit věty postupně příkazem DELETE, zůstane slovník relací nezměněn. Tak tedy není zajištěno, že zobrazená relace je použita v některém výroku v programu. Pro kontrolu, zda je relace R použita v definovaném výroku, se použije dotaz

is /r defined/

Odpověď YES nám říká, že existuje nejméně jedno použití zadané relace, odpověď No říká, že relace není v programu nikde použita. Dotaz na definovanou relaci lze použít ke kontrole. Naopak příkaz dict vypíše všechny relace slovníku. K výpisu relací definovaných v programu se použije v programu např:

all/x : x dict aend x defined/

Předdefinované relace a moduly

Microprolog obsahuje řadu předdefinovaných relací, se kterými se seznámíme při podrobnějším studiu jazyka. Micro-PROLOG nedovoluje měnit definice těchto relací. Jestliže se však pokusíte vložit relaci, odpovídající některé z předdefinované, obdržíte odpověď:

"Cannot add sentence for R", kde R je jméno relace.

Stejnou odpověď obdržíte při pokusu o přidání relace definované programem SIMPLE front-end./ Tak tomu bude např. pokud se pokusíte vložit relaci odpovídající některé z předdefinovaných/, Tak tomu bude např. když se pokusíte definovat vlastní relaci "is". I když SIMPLE je program sestavený v micro-PROLOGu, jsou jeho definice chráněny stejným způsobem, protože zahrnuje tři speciální formy programů nazývané moduly. Modulem se nazývá souhrn relačních definic, které explicitně zahrnují názvy určitých relací. Pouze tyto exportované relace mohou používat ostatní programy a jejich definice jsou chráněny před změnami. Tvorba a používání modulů jsou obsahem zvláštního oddílu o programování v micro-PROLOGu a nejsou popsány v tomto textu. Jména exportovaných relací programu SIMPLE lze zjistit dotazem:

all / x : x reserved /

V odpovědi obdržíte seznam jmen, které nesmíte použít ve svých relacích. Tento dotaz využijte k tomu, abyste se vyhnuli definování takových relací jaké jsou již v programu.

Aritmetické relace

Micro-PROLOG není určen ani vybaven pro aplikace vyžadující množství číselných manipulací. Může však provádět aritmetické úkony prostřednictvím zabudovaných aritmetických relací: SUM. TIMES, LESS a INT a lze používat aritmetické výrazy v podmíněných dotazech. Problematiku nebudeme řešit detailně, pouze budeme demonstrovat využití aritmetických relací na příkladech, které se používají a provádí dotazy stejným způsobem jako u databázových relací. Aritmetické výrazy jsou vyhodnocovány relacemi SUM a TIMES. Také každá je implementována ve strojovém kódu, takže k využití hardwardových operací lze použít každou relaci jako by tato byla definována implicitně v databázi.

To je prostředek, který umožňuje provádět dotazy na aritmetické operace stejně jako při dotazech na relace definované v reálné databázi faktů.

Sčítání a odčítání pomocí relace SUM

Relace SUM má tři argumenty ve tvaru:

SUM /x,y,z/ a vyjadřuje z = x + y

Implicitní databáze vyjadřuje relace výroků jako SUM /2,3,5/ nebo SUM /3 10.6 7.6/. S touto implicitní bází můžeme provádět dotazy na sčítání a odčítání.

Použití relace SUM

```
Kontrola : aend.is /SUM/ /20 30 50//
           YES
sečítání : aend.which /x : SUM /5.6 - 2.34 x//
           3.26
           No /more/ answers
odčítání : aend.which /x : SUM /x 34 157//
           123
           No /more/ answers
nebo       aend.which /x : SUM /34 x 157//
           123
           No /more/ answers
```

Omezení použití SUM

Podmíněné dotazy prostřednictvím relace SUM mohou mít pouze jeden neznámý argument. Toto omezení se nevyskytovalo u reálné

databáze. Vzniká proto, že micro-PROLOG simuluje databázi a vzhledem k efektivnosti nahrazuje pouze omezený rozsah vzorů v dotazu. To znamená, že takový dotaz jako:

which /x y : SUM / x y 1Ø //

nebude zodpovězen. Vypíše se sdělení chyby "Too many variables" /příliš mnoho proměnných/. Takové sdělení bude vypsáno vždy při použití zabudovaných relací micro-PROLOGu s větším počtem neznámých argumentů.

Číselné konstanty

V relacích můžeme používat celá čísla /integer/ i racionální čísla /floating point/. Všechny aritmetické relace pracují buď s celými nebo racionálními čísly. Jestliže použijete v jedné relaci oba typy, micro-PROLOG automaticky převede celá čísla na racionální.

Kladné celé číslo

Je posloupnost dekadických číslic bez předřazeného znaménka "-". Viz následující příklady:

234 7Ø56 89ØØ1 jsou celá čísla a

záporné

-34 -56ØØ4 -11ØØØ jsou celá záporná čísla.

Kladné racionální číslo je posloupnost dekadických čísel, opět bez předřazeného znaménka "+" s desetinou tečkou. Za číslem může následovat celočíselný exponent za písmenem E. Viz:

23.45 2.345E1 Ø. Ø234560E+2 2345.OE-2

jsou různá vyjádření stejného racionálního čísla. Za desetincu tečkou "." musí nutně následovat minimálně jedna další číslice i když je to jen Ø. Exponent má základ 1Ø a je jím zadané číslo vynásobeno.

Záporné racionální číslo

Má stejný tvar jako kladné číslo, pouze mu předchází znaménko "-". Tedy:

-34.678 -Ø.Ø783E-34 -1ØØ.Ø5 jsou záporná rac. čísla.

Poznámka: Při výpisu je exponent vytlačen u čísel v intervalu -1Ø až 1Ø. Číslo 3.26 bude vypsáno jako 3.26 ale číslo 32.6 bude jako 3.26E1. Celé číslo musí ležet v intervalu -32767 až +32767. Racionální čísla mohou mít až 8 číslic bez počátečních nul. Exponent může být v intervalu -127 až 127. Pokud při výpočtu vznikne nezobrazitelné malé číslo, vypíše se: "Aritmetic underflow", při příliš velkém čísle: "Aritmetic overflow",

Pokud výsledná hodnota celého čísla přesáhne přípustný interval, bude výsledek převáděn na racionální číslo.

Převody a testování číselných typů

Relace INT se používá dvěma způsoby. Může být použita jako standartní relace k testování, zda je zkoumané číslo celé, nebo přesněji, zda je číslo celé /typu integer/ nebo zda racionální číslo nemá desetinnou čárku. Relace INT může být rovněž použita jako binární relace k nalezení celé části binárního čísla.

Použití INT pro testování

```
aend.is /45 INT/
YES
aend.is /4.67 INT/
NO
aend.is /3.567E3 INT/
YES
```

Použití INT pro konverze

```
which /x : 3.5 INT x/
3
No /more/ answers
which /x : -3.56498E3 INT x/
-3564
No /more/ answers
```

Omezení používání INT

Při použití v základním tvaru relace je zadán pouze jeden argument. Relace může být použito pouze k testování a ne k vyhledávání celého čísla. Při použití binární relace musí být zadán první argument a druhý musí být neznámý, t.j. představuje ho proměnná. Při vyčíslení podmínky je přiřazena proměnné celočíselná část prvního argumentu. Při relaci s dvěma argumenty nelze číslo testovat, pouze nalézt celou část. K porovnání, zda celá část čísla je stejná jako jiné číslo, musíme použít INT a potom další primitivní relace micro-PROLOGu EQ k testování, zda je celá část čísla identická se zadanou hodnotou.

```
aend.is /ž.78 INT x aemd x EQ 6/
YES
```

Použití testu EQ po podmínce INT je důležité a bude o něm pojednáno později.

Násobení a dělení pomocí TIMES

Relace TIMES má tento význam:

TIMES /x y z/ vyčíslí se jako z = x . y

Použití relace TIMES

kontrcla výsledků:

```
aend.is /TIMES /3 4 12//
YES
```

kontrola dělení:

```
aend.is /TIMES /3 y 12/ aend y INT
YES
aend.is /TIMES /3 y 11/ aend y INT/
NO
```

násobení:

```
aend.which /x : TIMES /5 4.3 x//
2.15E1
No /more/ answers
```

dělení:

```
aend.which /x : TIMES /x 24 126//
5.25
No /more/ answers
aend.which /X : TIMES /24 y 126/ aend INT y/
5
No /more/ answers
aend.is /TIMES/ x 3 1Ø / aend TIMES /x 3 1Ø //
No
```

V posledním dotazu může odpověd činit No čtenáři potíže. Vždyť srovnáváme dvě stejné relace a výsledek není shodný. Příčina spočívá v zobrazení čísel a tím vznikajících nepřesnostech. Výsledek 1Ø : 3 bude 3.33333333. Toto číslo vynásobené třemi v druhé relaci však už nebude 1Ø a proto záporná odpověd No.

Omezení použití TIMES

Omezení při používání relace TIMES jsou shodná s omezeními pro relaci SUM. Pouze jeden argument musí být neznámý, ale může to být kterýkoliv ze tří argumentů. Tak je zabezpečeno vyčíslení dělení nebo násobení.

Testování pořadí pomocí relace LESS

Primitivní relace LESS může být použita pouze pro kontrolu. Podmínka LESS /x y/ je splněna, když x je menší než y ve smyslu obvyklého uspořádání čísel.

```
aend.is /3 LESS 4/
YES
aend.is /4 LESS 3/
No
aend.is /TIMES /3 x 10 / aend /TIMES /3 x y / aend
I.SUM /y z 10/ aend z LESS 0.1E-5/
YES
```

LESS můžeme rovněž použít pro srovnání dvou jmen. Uspořádání odpovídá pořadí ve slovníku. Podmínka LESS /x y/ je splněna pro slova x a y jen tehdy, když je ve slovníku x před y.

```
aend.is /FRED LESS FREDY/
YES
aend.is /Albert LESS Harold/
YES
aend.is /SAM LESS BILL/
No
```

Abecední uspořádání odpovídá uspořádání ASCII kódu. V tomto uspořádání jsou nejprve znaky velké abecedy a potom znaky malé.

Rozhodněte o výsledku výroků:

```
SAM LESS Sam
SAM1 LESS Samath2
Sam41 LESS Sam1
```

Vyčíslování dotazů

V tomto okamžiku je vhodné sdělit jak micro-PROLOG vyhodnocuje dotazy. Při kladení dotazů se nemusí znát způsob jeho vyhodnocení. V některých případech se zjistí, že uspořádání podmínek v konjunktivních dotazech má vliv na čas, který micro-PROLOG vyžaduje pro získání odpovědi. Prodloužení neovlivní odpověď, kterou získáme, pokud nedojde k chybě. Volba uspořádání a usnadnění vyčíslení je součástí pragmatiky použití micro-PROLOGu.

U některých konjunktivních dotazů, např:

```
which /x : TIMES/ 37 51 y/ aend SUM /y 73 x//
```

musíme mít představu o pořadí vyčíslování jednotlivých částí podmínky. Řeší micro-PROLOG nejprve podmínku SUM nebo TIMES? Kdyby byla první podmínka SUM byla by odpověď "Too many variabless" jako chyba, protože jsou zadány dva neznámé argumenty x a y. Pokud bude první řešená podmínka TIMES, nebude s řešením potíž, protože do podmínky SUM se proměnná y dostane už instalovaná, t.j. vyhodnocená. Naštěstí právě tak tomu je.

Micro-PROLOG vyhodnocuje konjuktivní dotazy při řešení podmínek zleva do prava, přičemž jsou postupně obsazovány /instalovány/ neznámé proměnné, které se ve výrazu vyskytují. Uspořádáním podmínek zleva do prava řídíme postup vyhodnocování.

Uspořádání podmínek je řídící částí dotazu. Konjunkce podmínek je logickým prvkem. Při sestavování dotazu se musíme předem soustředit na správný logický popis toho, na co se ptáme nebo co požadujeme. Další pozornost pak věnujeme uspořádání podmínek pro efektivní a bezchybné vyčíslení.

Vyhodnocení dotazu is s jednou podmínkou

Nejjednodušší formou dotazu je is:

is /C/ kde je C jednoduchý výrok bez proměnných

Micro-PROLOG vyhodnocuje takový dotaz prohledáváním výroků v databázích, které obsahují stejnou relaci jako je C. Neprohledává se tedy celá databáze. Micro-PROLOG ukládá výroky o každé relaci v pořadí, jaké vypisuje po příkladu list. Micro-PROLOG prochází tento seznam postupně a srovnává podmínku C s každým výrokem. Pokud se podmínka C ztotožní s výrokem v databázi, proces se přeruší a vypíše se odpověď YES. Pokud se dosáhne konce seznamu bez ztotožnění, je odpověď NO.

Příklad 1:

is /Henry muz/

Tato věta v naší databázi je ve výrocích "muz" uložena takto:

Henry -st muz
Henry muz
Edward muz

protože v tomto pořadí jsou výroky vypsány po příkaz:
"list muz?.

Micro-PROLOG nejprve porovnává podmínky v dotazu:

Henry muz s výrokem
Henry -st muz

který tvoří hlavu seznamu. Výroky se neztotožnily, protože "Henry" a Henry -st" jsou odlišná jména. Protože nedošlo k ztotožnění micro-PROLOG přechází k dalšímu výroku. Nyní dojde k ztotožnění, takže micro-PROLOG ukončí vyhledávání a vypíše odpověď "YES". Položíme-li dotaz:

is /Edward3 muz/

micro-PROLOG porovnává výrok postupně se všemi výroky a nedojde k ztotožnění. Tím vyjde odpověď NO.

Dotaz is se vzorem výroku

Zadáme-li dotaz is způsobem:

is /C/

kde C je jednoduchý vzor výroku, t.j. jednoduchý výrok s minimálně jednou proměnnou, bude odpověď realizována prakticky stejně. Jediný rozdíl je v tom, že při hledání totožnosti může micro-PROLOG každé proměnné ve výroku C přiřadit hodnotu označení nějakého prvku.

Příklad 2:

is /x otcem Elizabeth2/

Věty po relaci "otcem" jsou uloženy v pořadí:

Henry -st otcem Henry
Henry otcem Mary
Henry otcem Elizabeth 2
Henry otcem Edward

Micro-PROLOG porovnává vzor výroku: x otcem Elizabeth2 s každým výrokem v uvedeném pořadí. Ke ztotožnění dojde u třetí věty při obsažení proměnné x hodnotou "Henry". V tomto okamžiku micro-PROLOG přeruší prohledávání a vypíše odpověď "YES"

Příklad 3:

is /x otcem x/

Tento dotaz zkoumá obsah databáze tak, že se pokouší nalézt výrok, ve kterém je někdo svým vlastním otcem. Micro-PROLOG odpoví tedy "NO". Pokouší se totiž ztotožnit vzor výroku:

x je otcem x s každým výrokem seznamu. Částečně dojde k ztotožnění v první větě:

Henry -st otcem Henry při dosazení za x hodnoty "Henry-st". Vzor výroku pak dostane tvar:

Henry -st otcem Henry -st ale nedojde k ztotožnění, protože při ztotožnění musí micro-PROLOG obsadit obě x ve vzoru výroku jménem "Henry-st". Dojde k nesrovnalosti při porovnávání jména dítěte. Stejně tak ztroskotá srovnávání se všemi ostatními výroky v databázi. Odpověď bude tedy "NO".

Nyní uvažujeme dotaz:

is /x otcem y/

V tomto případě nastane stejný případ, protože micro-PROLOG může přiřadit proměnným x a y různé hodnoty. Okamžitě dojde ke ztotožnění pro případ x = Henry -st a y = Henry

Při vyhodnocování dotazu může micro-PROLOG přiřadit různým proměnným různé hodnoty, ale může jim přiřadit i stejné hodnoty.

Tak například: bude-li databáze obsahovat jednoduchý výrok "má-rád":

Tom má-rád Tom

potom oba dotazy

is /x má-rád y/

is /s má-rád y/

budou zodpovězeny stejně. V druhém dotazu se ptáme na to, zda databáze zná nějaká fakta o relaci x má-rád y. Ano, zná, i když x a y je stejná osoba Tom. Tuto konvenci možnosti přiřazení stejného neznámého objektu různým proměnným převzal micro-PROLOG ze symbolické logiky. Trvám-li na tom, že různým proměnným musí být přiřazeny různé hodnoty, musíme ještě přidat zvláštní podmínku, která tento požadavek vyjádří. K vyjádření takové podmínky ovšem potřebujeme hlubší znalost systému.

Vyhodnocení dotazu which s jednou podmínkou

Jednoduchá podmínka v dotazu which má tvar:

which /P:C/ kde P je vzor odpovědi a C je vzor jednoduchého výroku. Micro-PROLOG porovnává vzor výroku C s každým výrokem databáze obsahujícím danou relaci. K ztotožnění podmínky C s výrokem v databázi dojde během přiřazování hodnot proměnným v C. Při každém ztotožnění C s výrokem v databázi je vypsán vzor odpovědi P, ve kterém jsou proměnné nahrazeny hodnotami tohoto ztotožnění.

Příklad 4:

which /x : Henry otcem y/

Výroky databáze jsou porovnávány se vzorem dotazu v pořadí daného seznamem. Ke ztotožnění nedojde s prvním výrokem

Henry-st otcem Henry

protože otcové Henry-st a Henry se neztotožnili. Ke ztotožnění dochází až u druhého výroku

Henry otcem Mary

pro x = Mary. Protože došlo ke ztotožnění výroku se vzorem výroku, micro-PROLOG nalezl odpověď na dotaz. Proto vypíše vzor odpovědi a nahradí x hodnotou "Mary". Dostali jsme první odpověď:

Mary

Vyhodnocení pokračuje dále pokusem o ztotožnění vzoru výroku "Henry otcem x" pro další výroky:

Henry otcem Elizabeth2

Henry otcem Edward

Opět došlo ke ztotožnění hned s následujícím výrokem pro
x = Elizabeth2
Dostáváme tedy druhou odpověď:

Elizabeth2

a nakonec i u posledního výroku dochází ke ztotožnění pro
x = Edward- dostáváme poslední odpověď:

Edward
No /more/ answers

Vyhodnocení složeného výrazu which

Metodu vyhodnocování znázorníme na dvou příkladech:
Příklad 5:

which /x : Henry otcem x aend x muz/

Tento dotaz zužuje dotaz z příkladu 4 na pouhé nalezení synů, Henryho. Micro-Prolog musí nalézt všechna jména, pro která budou nalezeny výroky v databázi splňující podmínky:

Henry otcem x a x = muz

Micro-PROLOG bude hledat hodnoty proměnných x bez ohledu na druhou podmínku dotazu. Začne prohledávat všechna x, která splní podmínku:

Henry otcem x

my již víme, že existují tři věty tohoto typu, z nichž první je:

Henry otcem Mary

Micro-Prolog ztotožní podmínku dotazu s tímto výrokem a nalezne přípustnou odpověď x = Mary, ktrá je východiskem pro další zkoušení složeného dotazu. V tomto okamžiku micro-PROLOG přeruší hledání řešení dle první podmínky za účelem prověření souladu druhé podmínky dotazu "x muz". Hledá, zda lze dosáhnout úspěšně ztotožnění pro túto podmínku, jestliže x má hodnotu "Mary". To je podobný příklad jako při vyhledávání na dotaz:

"Mary muz"

Pokusí se uspokojit tuto podmínku prohledáním seznamu výroků s relací muz. Protože nenalezne tento výrok, nemůže přijmout splnění druhé podmínky pro hodnotu x = Mary. Vrátí se tedy k přerušenému vyhledávání dle první podmínky pro ostatní řešení vztahu:

Henry otcem x

Nalezne další řešení, které se ztotožní s výrokem

Henry otcem Elizabeth2

Proměnné x přiřadí hodnotu Elizabeth2. Micro-PROLOG opět

přeruší srovnávání dle relace "otcem" a začne s kontrolou zda může dojít k ztotožnění pro podmínku "x muz", když x = Elizabeth 2. Takže kontroluje, zda je splněna podmínka:
Elizabeth2 muz
Což se opět nepodaří. Micro-PROLOG se znovu vrací ke svému přerušenému vyhledávání pro všechna x, která splňují první podmínku

Henry otcem x

Další přípustná hodnota proměnné x, pro kterou dojde ke ztotožnění bude ve výroku:

Henry otcem Edward

kde se pokusí přiřadit x = Edward. Znovu dojde k přerušení, micro-PROLOG se pokusí potvrdit druhou podmínku "x muz" s výrokem:

Edward muz

Tentokrát se to podařilo, věta Edward muz je v databázi. Micro-PROLG nakonec vytvoří podle vzoru odpověď na složený dotaz a bezprostředně ji napíše. Protože micro-PROLOG chce nalézt všechna přípustná řešení, znovu se vrátí k přerušenému vyhledávání pro podmínku "Henry otcem x". Takový výrok již není, protože micro-PROLOG již nalezl ztotožnění pro všechny věty databáze dle tohoto vzoru. Je tedy vypsána odpověď

"No /more/"answers".

Metodu vyhodnocení dotazu

which /x : Henry otcem x aend x muz/

lze vyjádřit takto: Pro všechna x, která odpovídají podmínce Henry otcem x, jestliže x je muz, vypiš x.

Efektivní sestavování dotazů

Jak efektivnost ovlivňuje uspořádání podmínek v dotazu efektivnost vyhledávání odpovědí. Např.:

which /x : Henry otcem x aend x muz/

a which /x : x muz aend Henry otcem x /

Jsou ekvivalentní dotazy a povedou k získání naprosto stejných odpovědí. Avšak při vyhodnocování prvního výrazu bude micro-PROLOG nejprve prověřovat podmínku "Henry otcem x", aby nalezl hodnotu x, kterou potom srovná s podmínkou "x muz". Při vyhodnocování druhého dotazu nejprve zkontrolujeme podmínku "x muz", aby nalezl přípustné hodnoty x, které použije následně při kontrole podmínky "Henry otcem x". Takže uvedené dotazy nejsou z hlediska průběhu vyhodnocování ekvivalentní.

Dotazy můžeme vyjádřit takto:
Pro všechna x, které odpovídají podmínce Henry otcem x, jestliže x odpovídá x muž, vypiš x
Pro všechna x, která odpovídají podmínce x muž, jestliže x odpovídá Henry otcem x, vypiš x
V mnohem větší databázi než je naše RODINA, kde bude mnohem méně dětí Henryho než mužů, bude první dotaz zodpovězen daleko rychleji neždruhý. Pro každé dítě Henryho bude prohledávána množina výroků pro relaci "muž". Za obecné pravidlo pak můžeme považovat zásadu, že při dvou a více podmínkách dosadíme na první místo takovou podmínku, která má nejmenší počet řešení.

P R A V I D L A

Často potřebujeme klást složené výrazy víckrát, přičemž je nutné neustálé opakování konjunkce podmínek. Bylo by výhodné nějak podmínku v dotaze zkrátit. Tedy bylo by užitečné konstruovat závěry z faktů uložených v databázi.
Př. fakt, že Henry-st je otcem Henryho obsahuje rovněž význam, že Henry-st je rodičem Henryho.
Abychom mohli konstruovat takové závěry a zkracovat dotazy musí se použít
pravidla:
Logický popis pravidla zní:
x je dědečkem po meči když
x je otcem z a
z je otcem y, pro libovolné z
Procedurální popis vyjadřuje použitý způsob řešení podmínky dotazu pro relaci: "dedeckem-po-meci" Lze to vyjádřit:
Pro řešení: x dedeckem-po-meci y
Řeš podmínku : x otcem z aend z otcem y
Pro různé specifické požadavky můžeme vypracovat příslušné procedurální popisy. Např. pro nalezení vnuků se sestaví popis:
K nalezení y pro dané x, kde x je dědečkem po meči y
najdi takové z, pro které platí x je otcem z a potom
najdi y, kde z je otcem y
Pro různé specifické požadavky můžeme vypracovat příslušné procedurální popisy. Např. pro nalezení vnuků je popis:
K nalezení y pro dané x, kde x je dědečkem po meči y
najdi takové z, pro které platí x je otcem z a potom najdi
y, kde z je otcem y

Používání několika pravidel

Někdy musíme položit několik otázek typu which, abychom kompletně "pokryli" relaci. Např. budeme-li chtít seznam rodičů a dětí, musí se použít dva dotazy, protože taková informace není v databázi obsažena:

```
which /x y : x otcem y/
which /x y : x matkou y/
```

Může se zavést nová relace "rodičem", která bude transportovat vztahem "otcem" a "matkou". Lze zavést pomocí dvou nových pravidel:

```
x rodicem y if x otcem y
x rodicem y if x matkou y
```

Tím se rozšířila databáze tak, že se nalezne buť relace "otcem" či "matkou" odděleně nebo se tato relace vyjádří souhrnně "rodičem"

Na dotaz `which /x : x rodičem Elizabeh2/`

je odpověď

```
Henry
Mary
```

Odpovědi se získají v tomto pořadí, protože pravidlo pro relaci "otcem" je v seznamu dříve než pro relaci "matkou".

Proměnné v pravidlech

```
Jestliže se vypíší pravidla pro relaci "rodičem" dostaneme:
aend.list rodicem
X rodicem Y if
  X otcem Y
X rodicem Y if
  X matkou Y
aend.
```

Pravidla jsou vypisována v takovém pořadí, jaké jsme si zadali. Všimněte si, že micro-PROLOG změnil malé písmena na velká. Proto možné, že jméno proměnné není podstatné, ale podstatné jsou pozice proměnných definovaných v pravidlech.

Podmíněné výroky

Pravidla, která jsme dosud užívali jsou podmíněné výroky tvaru:

```
jednoduchý      if podmínka and  podmínka
výrok
```

kde každá podmínka je jednoduchým výrokem.

Podmíněný výrok se nazývá inplikace. Definice pravidla /consequent-konsekvence/ je jednoduchý výrok vlevo od "if",

podmínka výroku /antecedent/ je jednoduchá podmínka nebo konjukce podmínky vpravo od "if".

Každý výraz obsahující proměnné je pravidlo. Dosud jsme tedy používali jednoduché příkazy bez proměnných a podmíněné výroky s proměnnými. Formule označujeme jako fakta. Mohou být rovněž podmíněné bez proměnných, t. j.:

Honza má-rad David if David má-rad Honza

nebo s proměnnými:

Honza má-rád x /t.j.: má rád každého/

V následujícím textu budeme často pracovat s pravidly a jednoduchými výrazy. Od této chvíle budeme používat pojem fakta /jednoduché výroky bez proměnných/ a podmíněná pravidla /podmíněné výroky s proměnnými.

Množinou všech faktů v programu micro-PROLOGu je databáze. Podmíněná pravidla nám dovolují zkracovat dotazy pomocí definování nových relací, používající relace zahrnuté v databázi. Klademe-li otázky pomocí nových relací, použije micro-PROLOG pravidel při dotazech v databázi.

Pravidla používající relace definované jinými pravidly

Relace nově definované v pravidlech mohou být užívány dalšími pravidly. Může se tak vybudovat hierarchie relací, kde základ tvoří relace databáze. Lze např. definovat vztah "prarodičem" pomocí relace "rodičem".

Česky bychom řekli:

Nějaké x je prarodičem nějakého y
jestliže x je rodičem z a
z je rodičem y pro nějakého z

Podmíněný výrok lze přidat do našeho programu vyjádřením pravidla:

x prarodičem y if
x rodičem zz z and
z rodičem y

Význam tohoto pravidla lze popsat takto:

Pro zodpovězení podmínky ve tvaru x prarodičem y
odpověz na konjunkci podmínek x rodičem z a z rodičem y

Nechť řešitel vypracuje popis nalezení vnuků a prarodičů.

Kontrola:

Pro kontrolu, že x je prarodičem y pro dané x a y najdi z,
pro které x je rodičem z, přičemž se potvrdí, že z je rodičem y

Pravidlo "prarodičem" používá relace "rodičem", která je sama definovaným pravidlem. Není to problém. Micro-PROLOG použije pravidlo o prarodiči nezávisle na tom, jestli relace o rodiči je

zadána explicitně jako fakt nebo pomocí pravidla. Zjistí o jaký se jedná případ a zachová se dle toho, redukujíc podmínku "prarodičem" na konjunkci podmínek rodičem.

K našemu programu

Program o rodině se z jednoduchého počátku rozrostl. Ukončíme jeho vývoj za současného stavu, aby bylo vidět dosažené změny.

```
aend.list all
Henry-st otcem Henry
Henry otcem Mary
Henry otcem Elizabeth2
Henry otcem Edward
Elizabeth matkou Henry
Katherine matkou Mary
Jane matkou Edward
Anne matkou Elizabeth2
Henry-st muz
Edward muz
Elizabeth zena
Katherine zena
Mary zena
Elizabeth2 zena
Anne zena
Jane zena
x dedeckem-po-meci y if
    x otcem z and
    z otcem y
x rodičem y if
    x otcem y
x rodičem y if
    x matkou y
x prarodičem y if
    x rodičem z and
    z rodičem y
aend.
```

Program je značně dlouhý a nevejde se na obrazovku. Micro-PROLOG může zastavit výpis pomocí klávesy STP. Výpis pak bude pokračovat po stisku libovolné klávesy.

Více vzorů odpovědí

Dosud se požadoval výpis hodnot, zadanými proměnými ve vzoru odpovědi. S každou odpovědí lze také vypsat libovolný text. Zadá se prostě text do vzoru odpovědi v dotazu. Viz následující příklad:

Jaká jsou jména matek a jejich dětí:

```
which /x y : z matkou y/
Elizabeth Henry
Katherine Mary
Jane Edward
Anne Elizabeth2
No /more/ answers
```

Dvojice jmen není příliš informativní. Odpovědi lze také získat v tomto tvaru:

```
Elizabeth je matkou Henry
Katherine je matkou Mary    atd.
```

V odpovědi přerušuje vložený text "je matkou" výpis proměnných. Každá z odpovědí má vzor:

```
x je matkou y
```

Pro tento výpis se použije tento vzor místo jednoduchého "x y"

```
aend.which /x je matkou y : x matkou y/
Elizabeth je matkou Henry
Katherine je matkou Mary
Jane je matkou Edward
Anne je matkou Elizabeth2
No /more/ answers
```

Přidal se prostě text do vzoru odpovědi pro požadovaný výstup. Text se podobá vlastnímu dotazu tvaru "x je matkou y". Lze přidat libovolný text do seznamu proměnných ve vzoru odpovědi. Nemá to vliv na vyhodnocení relace. Jedinou podmínkou je: Oddělit proměnné od textu minimálně jednou mezerou. V jiném případě je text považován za proměnnou a hodnoty se nevypíší.

Vyhodnocování dotazů s pravidly

V předchozím jsme poznali vyhodnocování dotazů which. Ostatní druhy dotazů jsou většinou vyhodnoceny stejně. Jediný rozdíl je u dotazu "one" v tom, že vyhodnocování můžeme kdykoliv zrušit po nalezení nějaké odpovědi zadáním n a u dotazu "is", kde vyhodnocování končí, vždy po nalezení prvního řešení.

Při popisu způsobu vyhodnocování dotazů obsahujících relace obsahující pravidla, popíšeme obecnou metodu, kterou micro-PROLOG používá k nalezení řešení konjukce dotazů. Tato

metoda je nezávislá na tom, zda konjunkce relací jsou definovány prostřednictvím faktů, obecných relací nebo jejich kombinací.

Konjunkce dotazů which má tvar:

aend.which/P:C aend C´../

kde C a C´atd. jsou jednoduché výrazy. Podmíněné dotazy C,C´... obsahují proměnné, z nichž některé jsou součástí vzoru odpovědi. Micro-PROLOG musí nyní projít všechny cesty /větve/, aby mohl přiřadit hodnoty proměnným v konjunkci podmínek a nalézt odpovídající výroky v databázi nebo je navíc sestavit prostřednictvím pravidel. Každá taková množina přípustných hodnot je řešením konjunkce podmínek v dotazu. Pro každé nalezené řešení micro-PROLOGU se zobrazí výsledek dle vzoru odpovědi P.

Micro-PROLOG začíná své hledání všech řešení odpovídající konjunkci podmínek tak, že pochází všechny cesty, aby mohl splnit první podmínku C. Pokud nalezené řešení splňuje podmínku C, přeruší se další prohledávání dle C. Pokud C obsahuje proměnné, bude výsledkem přiřazení hodnot těmto proměnným.

Micro-PROLOG dále pokračuje hledáním všech řešení, která splňují zbývající část konjunkce podmínek s nalezenými hodnotami přiřazenými k proměnným. Znamená to tedy, že po vyhodnocování pokračuje s instalovanými proměnými odpovídajícími řešení C ve zbytku dotazu.

Po nalezení všech řešení splňujících podmínky ve zbylé části dalších cest dle C, opět se předají hodnoty proměnným pro zbytek dotazu.

Návrat k řešení podmínky C se realizuje až po nalezení všech řešení odpovídajících zbytku dotazu za podmínkou C. Tento proces pokračuje tak dlouho, dokud není žádné další řešení podmínky C v databázi k dispozici.

Cesty jsou prohledávány systematicky po "nejlevější" cestě do "hloubky". Tento proces se nazývá depth-first search. Mechanizmus navracení po vyčerpání všech možnosti na dané cestě k jakési "vyšší křižovatce" se nazývá bactracking /zpětné prohledávání/.

B A C K T R A C K I N G

Je způsob vyhledávání všech řešení konjunkce podmínek. Pokud micro-PROLOG nalezne řešení první podmínky C a předá je do následující podmínky C´aend C´´....pokračuje vpřed. Pokud se vrací k hledání dalšího řešení dleC, jedná se o postup zpět-bactracing.

Vyhodnocení konjunktivního dotazu which je dopředným a

zpětným posouváním přes podmínky k dotazu. Předpoklad: zadány 3 podmínky:

C aend C'aend C''

Micro-PROLOG vyhledá první řešení podmínky C a předá ho do:

C'aend C''

Dále hledá řešení podmínek C'aend C'', které odpovídají řešení nalezenému dle C. A opět se pokračuje "levou větví", t.j. hledáním řešení dle podmínky C'.

Micro-PROLOG se pokouší řešit C's hodnotami proměnnými nalezenými dle C. Pokud se to podaří postoupí k C''. Pokouší se C''řešit pomocí proměnných, obsažených řešením C aend C'. Po nalezení všech řešení dle C''s hodnotami proměnných dle C aend C'' se vrací k vyhledání dalšího řešení dle C'. Posouvá se vzad a vpřed mezi C'a C''tak dlouho, až nalezne všechna řešení vztahu

C'aend C''

pro první řešení dle C. Potom se vrátí k hledání dalšího řešení dle c. Proces "předávání" řešení zbytku dotazu představuje tok informací zleva do prava v dotazu. První podmínka, ve které se objeví proměnná, je generátor hodnot této proměnné. Tyto hodnoty jsou předávány dalším podmínkám dotazu, ve kterých se uvedené proměnné vyskytnou.

P R A V I D L A

Zpětné vyhledávání /bactracking/ všech řešení složeného dotazu se aplikuje bez ohledu na to, zda je relace v dotazu definována pomocí faktů, pravidel nebo jeho kombinací. Jediný rozdíl je při převzetí podmínky C na počátku vyhodnocování této podmínky.

Předpokládejme, že podmínka C odkazuje na pravidlo definované relací R. PROLOG vyhledává řešení podmínky vzhledem k relaci v databázi. Prohledává seznam výroků o R a za účelem ztotožnění s podmínkou v dotazu. Prohledávání v takovém pořadí, v jakém jsme vkládali výroky do programu /pořadí odpovídá výpisu v seznamu po příkazu list/. Obtíž spočívá v tom, že nyní se musí ztotožnit podmínka z dotazu s příslušným pravidlem, které může obsahovat proměnné. Pokud dojde ke ztotožnění, není ještě nalezeno vlastní řešení. Musí se přerušit vyhledávání výroků R, aby se nalezlo řešení dotazu dané podmínky v pravidlu. Každé řešení tohoto pomocného dotazu je řešením podmínky C.

Po každém nalezení řešení pomocného dotazu přeruší PROLOG

vyhledávání, aby předal řešení zbývajícím podmínkám ve výchozím dotazu. Nyní rozumíme návratem zpět k vyhledání dalšího řešení C návrat k prohledání dalšího řešení pomocného dotazu. Po nalezení všech řešení pomocného dotazu se vrací vyhledání k programovým výrokům pro relaci R. Každé pravidlo, které se ztotožní s podmínkou C, přejde k pomocnému dotazu. Kombinace všech pomocných dotazů dávají všechna řešení C.

Shrnutí metody vyhodnocování

Vyhodnocování můžeme shrnout takto:
Nalezení řešení konjunktivního dotazu:
pro každé přípustné řešení první podmínky
nalézt odpovídající řešení dle zbývající části dotazu
Nalezení řešení jednoduché podmínky:
Pro každý ztotožněný výrok, pokud je jím podmíněné pravidlo, nalézt řešení podmínek v pravidle, v ostatních případech je ztotožněný výrok řešením.
Ztotožněný výrok se vyhledává shora dolů dle seznamu výroků pro relace podmínek.

Příklad vyhodnocování

Vyjdeme z dotazu:

```
which /y : Henry-st dedeckem Y/
```

použije se pravidlo:

```
x dedeckem y if
x otcem z and
z rodicem y                    .../5/
```

které připojíme /respektive v předchozím textu přidáno/ do programu. PROLOG musí najít všechny hodnoty proměnné y, které představují řešení podmínek v dotazu:

```
Henry-st otcem y          .... /F/
```

V programu je pouze jediný výrok o této relaci a tím je pravidlo /5/. Připomeňme, že PROLOG zapomíná konkrétně zadané proměnné použité v pravidle. Pamatuje si pouze jejich pozice. Když začíná ztotožňovat podmínku s důsledky pravidla, předává proměnným v pravidle jména. Vždy přiřazuje odlišná jména od jmen použitých v dotazu. Předpokládejme, že proměnné x v pravidle / / je přiřazeno jméno x1, proměnné y jméno y1 a proměnné z jméno z1. Prolog musí ztotožnit podmíněný dotaz /F/ ve tvaru:

```
x1 dedeckem y1 if
x1 otcem z1 and
z1 rodicem y1
```

Vyhledávání ztotožnění je nyní trochu komplikovanější. Aby bylo nalezeno ztotožnění, musí být proměnným v dotazu i proměnným v pravidle přiřazeny hodnoty. Ale v tomto případě získáme v pravidle pouze proměnné. Hodnoty x = Henry-st a yi = y představují skutečné ztotožnění. Uvědomme si, že yl není přiřazen konkrétní údaj, ale jméno proměnné v dotazu. Nahradíme-li xl,yl hodnotami, dostaneme pro předka konjunkci podmínek ve tvaru:

Henry-st otcem zl and zl rodicem y

Úkol nalézt řešení podmínky:

Henry-st dedeckem y

se změnil nalézt všechny řešení nové konjunkce podmínek:

Henry-st otcem zl and zl rodicem y

Tento úkol je řešen obvykle. PROLOG začne hledat řešení podmínky "Henry-st otcem zl". Nalezne první řešení: zl = Henry se ztotožní s faktem

Henry-st otcem Henry

PROLOG okamžitě přeruší vyhledávání dle relace "otcem" a hledá všechna řešení následující podmínky:

zl rodicem y

která je shodná se zl = Henry. Musí se nalézt všechna řešení podmínky:

Henry rodicem y ...../H/

Nyní jsme došli k další relaci definované pravidlem. Tentokrát jsou dvě pravidla, v nichž jsou nová jména proměnných:

x2 rodicem y2 if x2 otcem y2

x3 rodicem y3 if x3 matkou y3

Dotaz Henry rodicem y se ztotožní sprvním pravidlem, když x2 = Henry, y2 = y a s druhým pravidlem se ztotožní, když y3 = Henry a y3 = y. PROLOG prověřuje tato pravidla jednou a to v uvedeném pořadí. Po úspěšném ztotožnění s prvním pravidlem, pak PROLOG nahradí podmínku /H/ podmínkou:

Henry otcem y ...../J/

Řešení /H/ jsou tři:

y = Mary

y = Elizabeth2

y = Edward

Tato tři řešení /H/ spolu s řešením zl = Henry z první podmínky /G/ jsou vlastním řešením /G/. Nakonec jsou hodnoty y z řešení /G/ řešením jednoduché podmínky /F/ výchozího dotazu. Výsledkem jsou tři odpovědi na /E/:

Mary
Elizabeth2
Edward

budou vypsána na displeji jako řešení nalezená pro podmínku /J/. Po nalezení všech řešení se PROLOG vrátí zpět k hledání dalších cest k řešení /H/, Použije se druhého pravidla "rodicem". Tím vznikne pomocný dotaz

Henry matkou y

který nemá žádné řešení.

Připomeňme si, že podmínka /H/ byla vytvořena tehdy, když PROLOG nalezl první řešení první podmínky vztahem

Henry-st otcem zl and zl rodicem y

Nalezením dalších řešení tohoto složeného dotazu a tím rovněž řešení původního dotazu lze provést návratem k úkolu vyhledání všech dalších řešení první podmínky:

Henry-st otcem zl

Pokračuje tedy prohledávání databáze pro relaci "otcem". Tuto podmínku již nesplňuje žádný výrok. PROLOG se musí vrátit k výchozímu dotazu:

Henry-st dedeckem y

aby zjistil, zda neexistují další výroky v databázi o relaci "dedeckem". Protože žádný takový neexistuje, vyhledávání řešení končí!

Poznámka: K pochopení činnosti PROLOGu je výhodné použít modul SIMTRACE, který umožňuje pomocí dalších příkazů sledovat způsob vyhledávání. Cvěřte si některé dotazy definované pomocí pravidel, abyste pochopili způsob vyhodnocování. Při sledování činnosti programu je použité pravidlo pro vyhledávání totožnosti s dotazem identifikováno pomocí pozice v seznamu výroků společně s výsledkem použitého pravidla. Dojde-li ke ztotožnění, jsou zobrazeny nové podmínky společně s předchozími podmínkami pravidla ve tvaru:

nový dotaz : předchozí podmínky pravidla

Když jste nyní požádáni o nový dotaz a o sdělení co má být sledováno, zjistíte, že podmínka je identifikována seznamem čísel a ne pouze jedním číslem. Např. identifikátor "/1 3/" říká, že je zkoumána první hodnota dotazu, používající pravidlo třetí podmínky výchozího dotazu. Zbytek dotazu je vlastně historickým pohledem zpět k původnímu dotazu.

Rekurzivní relace

Až dosud byly relace definované pravidly vlastně postradatelné. Dotazy používající takové relace mohou být nahrazeny dalšími rozšířenými dotazy, které používají pouze relace databáze. To je umožněno tím, že pravidla definující nové relace se dají rozvinout do dříve definovaných pravidel. Ale rovněž existují relace, které nemohou být definovány tak jednoduše. Jsou to relace, které mohou být popsány pouze rekurzivně, t.j. definicemi, které se vyvolávají právě definovanou relací. Pro takové relace je použití pravidel podstatné.

Předpokládejme, že naše databáze RODINA obsahuje mnoho generací a že požadujeme nalézt v databázi všechny předky Edwarda. Kdybychom věděli, že databáze obsahuje přesně čtyři generace předků Edwarda, mohli bychom je nalézt dotazem:

```
which /x1 x2 x3 x4 : x1 rodicem x2 and
x2 rodicem x3 and
x3 rodicem x4 and
x4 rodicem Edward
```

Jestliže nevíme, kolik předků databáze obsahuje, nemůžeme všechny předky pomocí jednoduchého dotazu zjistit. Je to proto, že nevíme, kolikrát musíme zapsat relaci "rodicem", abychom se vrátili zpět k prvnímu zaznamenanému předkovi. K nalezení předků pomocí jednoduchého dotazu musíme definovat relaci "předkem". Kdybychom chtěli někomu vysvětlit, kdo jsou jeho předkové, asi bychom řekli:

Vaši předci jsou Vaši rodiče a všichni předci Vašich rodičů.

Toto je rekurzivní definice /t.j. volající sama sebe/, protože k vysvětlení jevu používáme termínu, který sama sebe vysvětluje. Jestliže budeme "procházet" touto definicí, zjistíme, že naši předkové jsou:

naši rodiče
naši prarodiče /kteří jsou rodiči- předci našich rodičů/
naši pra-prarodiče
naši pra-pra-prarodiče
atd. /až do vyčerpání všech údajů/

Tuto rekurzivní definici můžeme vyjádřit dvojicí pravidel v micro-PROLOGu:

```
x předkem y if x rodičem y
x předkem y if z rodičem y
and x předkem z
```

Leckdy lze pravidla vyjádřit jednodušeji:

x je předkem y, když x je rodičem y

x je předkem y, když z je rodičem y a zároveň x je předkem z, pro libovolné z

Obecné vyjádření obou pravidel bude:

K řešení podmínky ve tvaru: x je předkem y
řeš podmínku: x je rodičem y
K řešení podmínky ve tvaru: x je předkem y
řeš konjunkci podmínek: x je rodičem y a x je předkem z

Všimněte si, že definice zahrnuje rekurzivní pravidlo i nerekurzivní pravidlo. Všechny definice musí obsahovat minimálně jedno nerekurzivní pravidlo nebo fakt, jinak by došlo k zacyklení při vyhodnocování.

Zadejme úkol nalézt všechny předky pomocí dotazu:

which /x : x předkem Edward/

PROLOG začne použitím prvního pravidla, tak, že podmínku v dotazu nahradí podmínkou:

x rodičem Edward

Po nalezení všech řešení této podmínky, tedy po nalezení a výpisu rodičů Edwarda se vrátí PROLOG ke druhému pravidlu, aby nalezl další předky Edwarda. Podmínka se převede do tvaru:

z rodičem Edward and x předkem z

Protože pravidlo definující rodiče jako otec je první, podmínka

" z rodicem Edward"

bude splněna pro z = Henry, ktrý je otcem Edwarda. Přiřadíme-li tuto hodnotu proměnné z, bude PROLOG hledat všechna řešení druhé podmínky, která zní:

x předkem Hénry

Po zodpovězení této podmínky a nalezení vších předků Henryho, se PROLOG vrátí ke druhé cestě a to k nalezení rodičů Edwarda. Najde jeho matku Jane. Potom vyhledá a vypíše její předky.

Poznámka: Domníváme se, že pro lepší pochopení vyhodnocování relace "předkem" a dalších dotazů obsahujícím "předkem" použijete na počítači úplného režimu sledování /all trace/. Musíte samozřejmě do databáze přidat uvedená dvě pravidla.

Všimněte si důsledků při změně zadání pořadí pravidel pro relaci "předkem". Jestliže se zadá rekurzivní pravidlo před nerekurzivním, budou vzdálenější předkové nalezeni dříve než rodiče-předci. Ale rovněž si všimněte, že v uspořádání s rekurzivním pravidlem na druhém místě dojde k zacyklení v případě požadavku na odpověď typu:

which /x y : x predkem y/

Zkuste sami zkoumat tento příkaz, nebo použijte opět all-trace. Potom vyzkoušejte vyhodnocení s rekurzivním pravidlem na prvním místě. Zjistíte, že PROLOG nikdy nenajde odpověď, protože bude opakovaně vyvolávat rekurzivní pravidlo "predkem" /pokud použijete all-trace, můžete přerušit cyklus stiskem klávesy o/.
Závěr: Vkládejte rekurzivní pravidlo při definici relace až za nerekurzivní, obvzláště , když je definice použita k nalezení všech typů relací.

Oddělené definice inverzních relací

Naše dvě pravidla zcela logicky definují nejen relaci "předkem", ale i inverzní relaci "potomkem". K nalezení potomků můžeme položit dotaz:

which /y : Henry predkem y/

Vyhledání všech řešení této odvozené podmínky je značně neefektivní pro zjištění potomků Henryho. K nalezení řešení "z rodicem y/ bude pro každou relaci rodič/dítě v databázi ověřeno, zda nalezený rodič je potomkem Henryho. Jediná cesta vedoucí k odstranění této nevýhody spočívá v oddělené definici relace " potomkem ", která sice bude ekvivalentní k definici "předkem", ale po vyhodnocení bude probíhat jiným způsobem. Přitom při vyhledávání potomků bude generován stejný postup směru vyhledávání předků. Problém s použitím definice "předkem" pro nalezení potomků spočívá v toku hodnot proměnnými v pravidle. Pravidlo:

x predkem y if z rodicem y and x predkem z

dává účinný postup pro zadané y. Pro první podmínku:

z rodicem y

se známým y, dostaneme menší množinu přípustných hodnot z předaných podmínce "x predkem z". Pro získání podobného toku hodnot v případě, že x je dáno a y máme nalézt, bychom měli použít dané x k nalezení dítěte z rodiče x a potom nalézt všechny potomky z.
Pro optimální postup musíme odděleně definovat relaci "potomkem" pomocí pravidla:

y potomkem x if

x rodičem y
y potomkem x if

x rodicem z and
y potomkem z

Tento postup ustavuje alternativní definici relace, která platí mezi dvěma osobami x a y, kdy x je předkem y, respektive y je potomkem x. Z čistě pragmatických důvodů bychom měli používat tato pravidla pro nalezení potomků a z pravidla o předcích pro nalezení předků. Pro kontrolu toho, zda jsou obě osoby ve vztahu předek/potomek lze použít libovolnou skupinu pravidel
Dotazy:

```
is /Henry predkem Edward/
is /Edward potomkem Henry/
```

jsou logicky zcela ekvivalentní a PROLOG bude provádět srovnání /srovnatelné činnosti/ při jejich vyhodnocování. K zodpovězení prvního dotazu bude pokračovat rodokmenem počínaje od Edwarda a u druhého dotazu začne od Henryho. Jestliže rodiny popsané v databázi budou mít v průměru více než dvě děti, dotaz s použitím relace "předkem" bude efektivnější, proč?
Rozdíl mezi pravidly "předkem" a "potomkem" se projeví při obecném vyjádření pravidel pro jednotlivé definice použité pro vyhledání potomka. Za tímto účelem lze popsat u programu "předkem":

> Pro nalezení takového y, pro které platí, že x je předkem y pro zadané x, najdi y takové, pro které je x rodičem y

nebo:

> Pro nalezení najdi y a z, pro něž platí, že z je rodičem y a zkontroluj zda x je předkem z pro nalezené z.

Pravidla "potomkem" zapíšeme takto:

> Pro nalezení takového y, pro které platí, že y je potomkem x pro zadané x, najdi y takové, pro které x je rodičem y.

nebo:

> Najdi takové z, pro které x je rodičem z pro dané x a potom najdi takové y, pro které y je potomkem z pro nalezené z

Nechá se na čtenáři, aby sestavil podobné logické schéma vyhledávání předků pro obě pravidla i pro kontrolu vztahu se zadanými hodnotami. K lepšímu pochopení rozdílů mezi použitím definic "předkem" a "potomkem" zadejte několik dotazů s použitím all-trace.
Příklad:
V předchozím textu jsme použili relace LESS. Ta může být rovněž použita pro definici pravidel dalších relací. Např. pro definici relace:

"lesseg" /t,j. menší nebo rovno/ musíme použít dvě pravidla:

x lesseg x

toto pravidlo jednoduše říká, že každá hodnota je menší nebo rovna sama sobě. Dalším pravidlem je:

x lesseg y if x LESS y

Toto pravidlo říká, že jestliže jsou dvě čísla /nebo slova/ ve vztahu menší, potom jsou rovněž ve vztahu, t.j. menší nebo rovna.

Obdobným způsobem řešte následující zadání:

1. Definujte relaci "greather-then" /větší než/
2. Definujte relaci "greateg" /větší nebo rovno/
3. Definujte relaci "dělitelný"

Nezapomeňte, že v důsledku omezených možností používání matematických relací mohou být uvedená čtyři pravidla používána pouze k porovnávání hodnot. Nezapomeňte v důsledku omezených možností používání dodat pomocné definice.

Rekurzivní výzvy aritmetických relací

V předchozích příkladech jsme definovali nové aritmetické relace pomocí výroků s aritmetickými relacemi. Jestliže použijeme rekurzivní definice, můžeme definovat každou aritmetickou funkci jako relaci micro-PROLOGU.

Podívejme se na relaci faktoriál. Faktoriál kladného čísla je násobek všech čísel mezi 1 až N:

1 * 2 * 3 * ..... * N

Toto vzájemné násobení můžeme napsat jako:

/1 * 2 * ... n-1// * N pro N větší než 1

vidíme, že faktoriál čísla N, které je větší než 1, je faktoriál čísla /N-1/ násobený N. Tak dostáváme rekurzivní charakteristiku, kterou potřebujeme pro sestavení programu.

Jak bylo poznamenáno, musí mít rekurzivní definice minimálně jeden nerekurzivní výrok. V případě faktoriálu bude nerekurzivním pravidlem fakt, který definuje faktoriál pro N = 1.

Použijeme "x faktoriál y" v tom smyslu, že y faktoriál kladného čísla x. Následující dva výroky podávají kompletní definici relace:

```
1 factoriál 1
x factoriál y if
1 LESS x aend
SUM /x1 1 x/ aend
x1 factoriál y1 aend
TIMES /x y1 y/
```

Po zadání uvedených pravidel je můžeme použít pro výpočet faktoriálu pomocí dotazu, např:

```
which /x : 6 factorial x/
```

Výpočet factoriálu lze slovně vyjádřit:
Nalezni takové y, pro něž platí, že y je factoriál x
Pro zadané x
jestliže x = 1, y = 1 nebo
jestliže 1 je menší než x potom odečti 1 od x a přiřaď výsledek xl
najdi yl takové, že vl je factoriálem xl, vynásob yl hodnotou x ulož do y
Upozorňujeme znovu čtenáře na to, že v důsledku omezení při používání LESS může být relace použita pouze se zadaným prvním argumentem. Můžeme tedy factoriál použít pouze pro výpočet factoriálu nebo pro text, zda dvojice čísel splňuje relaci "factorial"

```
aend.is /3 factorial 6/
YES
```

Jestliže se pokusíme použít definici pro nalezení factoriálu záporného nebo racionálního čísla, přeruší PROLOG okamžitě výpočet. Pro záporné číslo nepoužije žádné pravidlo. Pro kladné necelé číslo bude případně rekurzivní pravidlo redukováno na úlohu vyhledání factoriálu čísla menšího než 1 a opět nebude aplikováno žádné pravidlo. Toto přerušení výpočtu je zde zcela na místě, protože relace "factorial" je určena pro celá kladná čísla. Mohlibychom nahradit relaci "1 factorial" pravidlem:

```
x factorial 1 if x lesseg 1
```

Nyní je pravidlo definováno pouze pro všechna čísla větší nebo rovna 1.Ověřte novou definici dotazem na factorial čísla 4.5. Definice "lesseg" byla již uvedena v předchozím textu.

Rekurzivní definice posloupnosti celých čísel

Definice relace "factorial" nemůže být použita k výpočtu inverzní funkce k funkci factoriálu vzhledem s omezením platným pro používání funkce less. Předpokládejme ale, že naše relace by mohla být výhodně použita ke zjištění čísel, která se podílejí na výpočtu factoriálu, při řešení podmínky, ve které je zadáno y a x máme nalézt pravidlo:

```
x factorial y if
1 LESS x aend
SUM /xl 1 x/ aend
xl factorial yl aend
TIMES /x yl y/
```

pokusí se použít podmínku /0 LESS x/ pro vygenerování kandidáta možné hodnoty větší než Ø. Factoriál bude vypočítáván pro každého kandidáta a kontrolován vzhledem k zadané hodnotě y. Je-li relace LESS definována tímto způsobem, bude generovat různé hodnoty x v pořadí 2,3,4... Potom použití pravidla bude iterativně vyhledat prostřednictvím sekvence hodnot číslo, jehož factorial je y. Víme, že hodnota x nemůže být nikdy větší než y.

Tak pro definici relace factoriálu pro inverzní použití nahradit podmínku

1 LESS x

přísnější podmínkou

x between /2y/

kde "x between /y z/" bude splněno pro y = x a zároveň $x < z$. Podívejme se, jestli můžeme definovat tuto relaci k vygenerování všech x v zadaném rozsahu /y z//

Nejprve zkusme sestavit nerekurzivní pravidlo. Jaké číslo je bezpečně v intervalu $y \leq x < z$? Je to y za předpokladu, že y je menší než z vzniká tak nerekurzivní pravidlo:

y between /y z/ if y LESS z

Tento postup pokrývá takové x, které je na levém okraji intervalu. Podívejme se nyní na rekurzivní pravidlo pro

" x between /y z/"

abychom pokryli všechny hodnoty x uvnitř intervalu. Jaké podmínky vygenerují takovou hodnotu x, která je odlišená od y a leží v intervalu mezi y a z ? Jsou to podmínky že y+1 je menší než z a zároveň x je mezi y+1 a z. Tak dostáváme rekurzivní pravidlo:

x between /y z/ if
SUM /y 1 y1/ aend
y1 LESS z aend
x between /y1 z/

Nyní se pravidlo použije v PROLOGu k nalezení všech čísel v daném rozsahu.

Při zodpovězení dotazu:

all /x : x between /1 3/ .../A/

použije PROLOG nejprve rekurzivní pravidlo, které přiřadí x hodnotu 1 a zredukuje podmínku "x between /1 3/" na

1 LESS x

která má být vyřešena. Získáme tak první odpověď na dotaz. Zpětné prohlížení vyústí v použití rekurzivního pravidla, které přetransformuje /A/ do tvaru:

SUM /1 1 y1/ aend y1 LESS 3 aend x between /y1 3/

Po vyřešení prvních dvou podmínek dostaneme jednoduchou podmínku

x between /2 3/

která dá zbývající odpovědi na výchozí dotaz. Znovu bude aplikováno nerekurzivní pravidlo k nalezení první hodnoty x = 2, která splňuje danou podmínku a zredukuje rekurzivní pravidlo na tvar:

SUM /2 1 y1/ aend y1 LEES 3 x between /3 3/

Tentokrát nemůže být druhá podmínka splněna pro hodnotu y1=3. Tato poslední aplikace pravidla přeruší hledání dalších řešení. Odpovědi na dotaz tedy budou:

2

3

No /more/ answers

Vyhledání čísel v daném rozsahu můžeme pro program "between" rozepsat takto:

Nalezni takové x, pro které platí, že x leží mezi /y z/ pro zadání y z,
jestliže y1 LESS z potom nalezni takové x, které leží v intervalu /y1 z/

Inverzní definice faktoriálu

Vraťme se nyní k definici relace pro výpočet faktoriálu. Můžeme definovat inverzní relaci "fact-of" pomocí jednoduchého pravidla:

y fact-of if x between /1 y/ aend x factorial y

Zvláštní podmínka "x between /1 z/" je logicky renudantní, ale působí jako generátor kandidátů hodnot x pro zadané y. Každý kandidát hodnoty je testován podmínkou:

"x factorial y"

která používá nám již známou definici factorialu ke kontrole zda kandidát představuje faktoriál y. Definici nemůžeme použít k nalezení hodnoty y při zadaném x, protože vyhodnocení relace "x between /1 y/" s neosazenou proměnnou vyvolá chybu:

"Too many variables"

při kontrole podmínky

"1 LESS y"

Avšak stejně jako u definice "factorial" můžeme pravidlo použít ke kontrole, zda zadaná dvojice čísel je v relaci odpovídající funkci faktoriálu

Inverzní definici faktoriálu rozepíšeme takto:

K nalezení takového x, pro které platí, že
y fact-of x pro dané y
najdi x v intervalu 1 až y,
pro které je splněna podmínka x factorial y

Dělitelnost celých čísel

Kladné číslo x má vlastního dělitele, jestliže existuje takové celé číslo v intervalu mezi 2 až x, které je dělitelem x.

x má dělitele if y between /2 x/ aend TIMES /y z x/
aend z INT

Tato definice může být použita jako program v micro-PROLOGu k testování zda má číslo vlastního dělitele. Ve spojení s podmínkou "between"ji můžeme použít k vyhledání všech dělitelných čísel v zadaném rozsahu.
Následující dotaz

all /x : x between/2 1Ø/ aend x ma-delitele/

nám dá všechna dělitelná čísla v rozsahu 2 až 1Ø.

L I T E R A T U R A

1. Briggs J. : Introduction micro-PROLOG for the Sinclair ZX Spectrum. Sinclair Research Limited, London 1983
2. Šebelík : PROLOG-jazyk nové generace počítačů. Výběr informací z organizační a výpočetní techniky 2/85 NOTO Kancelářské stroje k.ú.o,Praha 1985 /235-240/
3. Brůha J. : Programovací jazyk PROLOG. Informační systémy 4. ALFA, Bratislava, 1984
4. Mikuleski : Problémy programovania v predikátovom počte, Informační systémy 2, ALFA, Bratislava 1978
. Staudte R. : Ein PROLOG-interpretes fürSCP. EDV Aspektr 3/1985 Berlin 1985.
. Clocksin W. : Programming in PROLOG Springer Verlag 1981

Ovládání programu PROLOG-80 na počítači ZX SPECTRUM

1. Označení kláves

CS	- CAPS SHIFT
SS	- SYMBOL SHIFT

2. Režim kladení otázek

[user]	- přechod do režimu vstupu klauzulí z klávesnice
SS I	- přechod do režimu kladení otázek
[jméno]	- přečtení programu ze souboru "jméno"
listing	- výpis programu
listing /X/	- výpis všech klauzulí s predikátem X /atom/
listing /X,Y/	- jako listing /X/ s výstupem do souboru y
listing /'',Y/	- výpis celého programu do souboru Y
listing /X,buffer/	- jako listing /X,Y/ s výstupem do aktuálního bufferu
listing /'', buffer/	- jako listing /'', Y/ s výstupem do aktuálního bufferu
eogram	- zrušení programu v paměti
edit	- přechod do režimu editování všech klauzulí programu
edit /X/	- přechod do režimu editování všech klauzulí s predikátem X
edit-buffer	- přechod do režimu editování klauzulí uložených v aktuálním bufferu
new-buffer	- otevření nového bufferu a přechod do režimu editování tohoto nového aktuálního bufferu
old-buffer	- zrušení aktuálního bufferu, a přechod do režimu editování předchozího bufferu, který se stává aktuálním. Obsah předchozího bufferu je ztracen
[buffer]	- přečtení klauzulí z aktuálního bufferu

3. Režim editování

Po přechodu do režmu editování pracuje editor jako celostránkový /full screen editor/. Pro pohyb kurzoru po obrazovce a provádění změn lze použít následující klávesy:

SS I - návrat do režimu kladení otázek
SS 3 - zrušení řádku
SS CS - přepínání rychlého nebo pomalého pohybu kurzoru prostřednictvím šipek
SS Q - nastavení kurzoru na první znak bufferu
SS W - nastavení kurzoru na poslední znak bufferu
SS E - tabelátor pro rychlý posuv po ... znacích směrem doprava
SS 4 - vložení řádku za řádek s nastaveným kurzorem
ENTER - vložení řádku před řádek s nastaveným kurzorem
SS 9 - vynechání znaku na pozici kurzoru
SS 1 - přepínání režimu vkládání nebo přepisu znaků
CS SPACE break - návrat do výchozího stavu intřpretu jazyku PROLOG

Stejný postup platí i při práci s externími buffery /predikáty edit-buffer, new-buffer a old-bufer/. Klauzule z těchto bufferů budou zařazeny do programu až po příkazech consult / buffer/ respektive [buffer]. Součastně může být rozpracováno více bufferů, které se zpracovávají obdobně jako zásobníková paměť, 'stack.'Používání bufferu je vhodné při ověřování různých verzí programu a také z toho důvodu, že v režimu vstupu příkazu user dojde při chybě k nenávratnému ztracení zapsané klauzule, zatímco při použití bufferu zůstávají všechny řádky /i chybné/ zachovány.

4. Poznámky

Zrušení klauzulí můžeme provést pomocí predikátu retract respektive rectractall nebo přechodem do režimu editování výzvou edit a následným zrušením řádku s vyřazenými klauzulemi.

Zrušení definice operátoru se provádí zadáním cíle ve tvaru op /1, xfx, jméno/. Všimněte si, že priorita operátoru je při rušení vždy 1, proto se žádný operátor nesmí deklarovat s prioritou 1 !

Počet vzájemně odlišných atomů a jmen externích proměnných je omezen na 1Ø24. Toto omezení se může projevit pouze v případech použití velké databáze s mnoha fakty. Atom může mít maximálně 253 znaků. Funktor nemůže mít větší

než 15. Celá čísla jsou omezena na interval Ø - 16383. Nelze používat ani racionální ani záporná čísla. Priorita operátoru musí být v intervalu 2 - 255. Term čtený z aktuálního vstupu predikátem 'read' nemůže obsahovat více než 30 různě pojmenovaných proměnných včetně anonymních proměnných. Klauzule v databázi nemůže obsahovat více než 31 různých proměnných použitých vícekráte. Všechny proměnné, které se v klauzuli vyskytují pouze jednou, jsou převedeny predikátem 'assert' na anonymní proměnné, ale jejich jména zůstávají zachována.

Systém 'v vypisuje' termy pouze do 'hloubky vnoření' 20. Místo termu vnořených hlouběji se vypisuje pouze značka jako upozornění na neúplný výpis.

S t a n d a r t n ě z a v e d e n é p r e d i k á t y
v j a z k u P R O L O G

!	- rez; ruší možnost uplatnění mechanizmu navracení až po nadřazený cíl
abort	- okamžité přerušení běhu programu
arg /N, T, A/	- N-tý argument v termu je T je A
assert /C/	- vložení klauzule C do databáze
asserta /C/	- vložení klauzule C do databáze před ostatní klauzule se stejným predikátem
assetz /C/	- vložení klauzule C do databáze za ostatní klauzule se stejným predikátem
atom /X/	- X je atom
atomic /X/	- X je atom nebo celé číslo
break	- návrat do výchozího stavu interpretu jazyka PROLOG
call /G/	- /cíl vyjádřený termem G/
clause /H,B/	- v databázi je klauzule s hlavou H a tělem B
consult /F/	- čti klauzule a cíle ze souboru F /atom/
core	- výpis rozdělení paměti
debussing	- výpis seznamu aktivních bodů spy
deny/H,B/	- v databázi zruší klauzuli s hlavou H a tělem B
display /X/	- výpis x na aktuální výstupní kanál ve tvaru prexix
fail	- /cíl, který nikdy neuspěje; okamžitě se uplatní mechanismus navracení/.
fork	- zahájení dalšího procesu PROLOGU
functor/T,F,N/	- je term z funktorem F a arností N

set /X/	- přečtení následujícího zobrazitelného znaku
set 0 /X/	- přečtení následujícího znaku X /kód znaku/ z aktuálního vstupního souboru
integer /X/	- X je celé číslo
length /L,M/	- délka seznamu L je M
listing	- výpis programu
listing /X/	- výpis všech klauzulí s predikátem X /atom/
listing /X,Y/	- jako listing X s výstupem do souboru Y
name /X,Y/	- Y je seznam znaků /celá čísla/ jména X
next /X,Y/	- změň standartní vstup a výstup na X a Y
nl	- výpis pokračuje na novém řádku
nodebung	- zrušení všech bodů spy v predikátu X /atom nebo seznam/
not /X/	- /cíl uspěje tehdy a jen tehdy, když cíl X neuspěje/
notrace	- zrušení ladícího výpisu chodu programu
op /X,Y,Z/	- deklarace atomu Z jako operátoru Y s prioritou X
pipe /X/	- založ "proud" se jménem X /atom/
portraycl /X/	- vypiš klauzuli X na aktuální výstupní zaříz.
portraygoals /X/	- vypiš cíl na aktuální výstupní zařízení
print /X/	- vypiš X na výstupní zařízení
put /X/	- výstup znaku X /celočíselný výraz/ na aktuální výstupní zařízení
read /X/	- přečtení termu X /ukončeno/ z aktuálního vstupního zařízení
reconsult /X/	- přečtení klauzulí ze souboru X, které nahradí odpovídající klauzule v programu
repeat	- /cíl uspěje v nekonečném cyklu/
retract /X/	- zrušení klauzule X v databázi
see /X/	- přepnutí aktuálního vstupu na výstupní soubor X /atom/
seeins /X/	- X je aktuální vstupní soubor /atom/
seen	- uzavření aktuálního vstupního souboru a přepnutí na standartní vstup
shell /X/	- výzva shell příkazem X /seznam znaků/
skip /X/	- nastavení kontrolního bodu spy pro klauzule X /atom nebo seznam/
tab /X/	- Vypiš X mezer na aktuální výstupní zařízení
tellins /X/	- nastavení aktuálního výstupu na soubor X
told	- uzavření aktuálního výstupního souboru a převedení na standartní výstup

trimcore	- uvolnění nepotřebného prostoru operačního systému
true	- /cíl, který vždy uspěje/
var /X/	- X je proměnná /neinstalována/
write /X/	- výpis termu X na aktuální výstupní zařízení /respektující definici operátoru/
X	- /cíl předaný hodnotou proměnné X/
X,Y	- X and Y /konjunkce cílu/
X;Y	- X or Y /disjunkce cílu/
X<Y	- celočíselný výraz X je menší než výraz Y
X = Y	- X je rovno Y
X =..Y	- Y je seznam složený z funktoru v X a všech argumentů v X
X=:=Y	- celočíselné výrazy X a Y jsou si rovny
X=<Y	- celočíselný výraz X je menší nebo roven výrazu Y
X==Y	- X je identické s Y
X =\= Y	- celočíselné výrazy X a Y si nejsou rovny
X>Y	- celočíselný výraz X je větší než celočíselný výraz Y
X>= Y	- celočíselný výraz X je větší nebo roven celočíselnému výrazu Y
X is Y	- výsledek celočíselného výrazu Y je přiřazen X
X\= Y	- X není rovno Y
X\==Y	- X není identické s Y
X + Y	- součet X a Y
X - Y	- rozdíl X a Y
X * Y	- násobek X a Y
X / Y	- rozdíl x a Y
XmodY	- zbytek podílu X a y
[X X]	- ekvivalentní "reconsult /X/"
[X]	- ekvivalentní s "consult /X/"
[user]	- vstup klauzulí do databáze

Přehled znaků použitelných v termech:

A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z,a,b,c,d,
e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x, y,z,0,1,2,3,4,5,6,7,
8,9,+,-,*,/,\,^,(,),=,',:,.,?,&,#

Přehled zobrazitelných znaků:

abeceda a čísla - stejná

znaky: !,",#,%,&,;,),=,-,~,^,\,/,},],{,[,_,`, ,+,*,;,(,*,?,/