

## 5.9 Expertní systémy

Za expertní systémy se považují programy, které umožňují uživateli získat takové informace o vybrané speciální oblasti, které vyplývají z její podrobné znalosti. Často se používají pro diagnostické i terapeutické účely, vytváření návrhů technických zařízení, ekonomické a právní úkony apod. Expertní systémy se v poslední době staly velice oblíbenou technikou při vytváření konzultačních programů. Literatura o expertních systémech je velmi bohatá (viz např. [30]), informaci o využití Prologu pro tvorbu expertních systémů lze najít např. v publikacích [9], [22], [25], [32], [43], [57] a řady dalších.

Effektivnost funkce expertního systému i způsob zacházení s ním závisí nejen na množství a kvalitě poznatků o oblasti, které se program týká, ale i na volbě programových prostředků. V tomto článku ukážeme, že Prolog je vhodným prostředkem pro tvorbu expertních systémů. Byl již mnohokrát úspěšně využit jak při realizaci inferenčního stroje, báze znalostí, vysvětlovacího mechanizmu, tak i při vytváření uživatelského prostředí.

### 5.9.1 Složky expertního systému

Expertní systém, podobně jako systém databázový, se obvykle skládá z báze znalostí, deduktivního (inferenčního) mechanizmu, vysvětlovacího mechanismu a uživatelského prostředí. Expertní systémy se liší kvalitou jednotlivých složek. V některých systémech např. není přítomen vysvětlovací modul a uživatelské prostředí je chudé. Každý expertní systém však musí obsahovat bázi *znalostí* a *inferenční mechanizmus*. V bázi znalostí jsou obsaženy znalosti týkající se oblasti aplikace; inferenční mechanismus tyto znalosti interpretuje a zpracovává, vysvětlovací modul by měl uživateli předvést důvody, na základě nichž expertní systém dospěl ke svým závěrům. Modifikovatelný expertní systém by měl být vybaven pružným uživatelským prostředím, které umožňuje snadno vytvářet, testovat a modifikovat báze znalostí.

V následujících odstavcích popíšeme jednotlivé složky podrobněji.

### 5.9.2 Jednoduchý příklad

[21], ukážeme, jaké základní požadavky klade uživatel na práci expertního systému a jaké programovací prostředky k tomu účelu nabízí Prolog. Oblastí, pro kterou budeme vytvářet expertní systém, bude zoologie. Požadujeme, aby systém dokázal určit zvíře na základě údajů o jeho vlastnostech podle těchto pravidel:

- |      |   |
|------|---|
| P1:  | Má – li zvíře srst, je savec.                                       |
| P2:  | Dává – li zvíře mléko, je savec.                                    |
| P3:  | Má – li zvíře peří, je pták.  |
| P4:  | Pokud zvíře letá a klade vejce, je to pták.                         |
| P5:  | Savec, který žere maso, je šelma.                                   |
| P6:  | Savec, který má špičaté zuby, drápy, a oči namířené dopředu,        |
| P7:  | je šelma.   |
| P8:  | Savec, který má kopyta, je kopytnatec.                              |
| P9:  | Savec, který přežívá, je kopytnatec.                                |
| P10: | Šelma, která je žlutohnědá a má tmavé skvrny,                       |
| P11: | je gepard.  |
| P12: | Šelma, která je žlutohnědá a má černé pruhy,                        |
| P13: | je tygr.  |
| P14: | Kopytnatec, který má dlouhé nohy, žlutohnědou barvu a tmavé skvrny, |
| P15: | je žírafa.  |
| P16: | Kopytnatec, který má bílou barvu a černé pruhy,                     |
| P17: | je tučnák.  |
- Na jednoduchém příkladě, který se často vyskytuje v učebnicích Prologu [21], ukážeme, jaké základní požadavky klade uživatel na práci expertního systému a jaké programovací prostředky k tomu účelu nabízí Prolog.
- Oblastí, pro kterou budeme vytvářet expertní systém, bude zoologie. Požadujeme, aby systém dokázal určit zvíře na základě údajů o jeho vlastnostech podle těchto pravidel:

P18: *je kočka.*  
*Šelma,*  
*je pes.*

která žije u lidí  
 a (štěká nebo chodí na vodítku),

Povšimněme si, že každě z použitých pravidel má tvar

"*Jestliže Podmínka1 a ... a PodmínkaN, pak Závěr.*"

Tato struktura doslova zeadlí způsob vytváření klausul jazyka Prolog. Odlišnost je jen ve formě zápisu, neboť pravidla v Prologu mají tvar

Závěr :- Podmínka1, ..., PodmínkaN.

ktorý zdůrazňuje člověk orientovaný přístup k řešení úloh. Stačilo by tedy vhodně zvolit predikáty jazyka a přímo přepsat uvedená pravidla ve tvaru klausul.

*L.program:*

Nechť každě vlastnosti uvedené v pravidlech odpovídá unární predikát, který vypovídá o tom, zda uvažovaný objekt má či nemá danou vlastnost. Konstanty jazyka nechť jsou vlastní jména uvažovaných zvěřat. Proměnné jazyka budou libovolná slova začínající velkým písmenem, např. X, Y, Z.

Soustava pravidel P1 až P18 tak dostane v Prologu tvar klausul

C1 až C18:

C1: savec(X) :- má\_srst(X).

C2: savec(X) :- dává\_mléko(X).

C3: savec(X) :- žije\_u\_lidi(X).

C4: savec(X) :- žere\_maso(X).

C5: selma(X) :- savec(X), žere\_maso(X).

C6: selma(X) :- žije\_u\_lidi(X), štěká(X).

C18a: pes(X) :- selma(X), žije\_u\_lidi(X), štěká(X).

C18b: pes(X) :- selma(X), žije\_u\_lidi(X), chodí\_na\_vodítku(X).

Stojí za zmínu, že pravidlo P18, které obsahuje disjunkci vlastností, jsme museli rozdělit do dvou klausul C18a a C18b, neboť tělo klausule se vždy interpretuje pomocí konjunkce.

(Samozřejmě, že s využitím operátoru pro disjunkci ; a závorek jsme mohli pravidlo P18 zapasat do jediné klausule

C18: pes(X) :-

    selma(X),  
     žije\_u\_lidi(X),  
     štěká(X); chodí\_na\_vodítku(X).

Pro další výklad zústaneme u zápisu pomocí předchozích klausul C1 až C18b.) Konzultační systém v Prologu tvorený klausulami C1 až C18b umožňuje např. zjistit, zda zvíře jménem cola je tučňák. Otázku budeme formulovat takto:

?- tučňák(cola).

Ovšem díky tomu, že klausule C1 až C18b obsahují pouze obecná pravidla, není žádná z uvedených klausul aplikovatelná na nás dotaz (není možno ji unifikovat s naší člověkovou klausulou). Odpověď tedy bude znít no.

Chceme-li se o zvratu cola něco zajímavého dovédat, musíme systému dodat výchozí informaci, např.

má\_srst(cola).

přítulný(cola).

žije\_u\_lidi(cola).

Tyto údaje stačí, aby nám systém zodpověděl otázku

?- selma(cola).

Odpověď bude yes, jak vyplyvá z klausul C5 a C1. Ovšem některé poskytnuté informace jsou k odvození tohoto poznatku zbytečné, protože je systém neumí využít, např. fakt: přítulný(cola). Naopak na otázku

?- pes(cola).

dostaneme odpověď no. Nemůžeme totiž použít ani jedno z pravidel C18, protože systému některé informace chybějí.

Vidíme, že navržený "přímý překlad" odvozovacích pravidel P1 až P18 by jako návrh expertního systému neobstál, neboť:

- Všechny znamená údaje o objektu, o kterém chceme získat další informace, se musí před vznesením dotazu přesně specifikovat. Přitom mnoho vložených údajů může být zbytečných. Vzniká i nebezpečí, že použijeme formulaci mírně odlišnou od toho, co systém očekává, a dodané informace nebude z neporu-  
zumění použito.
- Mohou být vzneseny pouze konkrétní dotazy typu "Je colo pták?" nebo "Je rek pes?", ale nemáme možnost formulovat otázku "Jaké zvíře je colo?".
- Odpovědi systém kladně na nějakou otázku, nevyvstávají uživateli dívody, které k odpovědi vedou – uživatel tedy musí projevit stejnou důvěru k danému systému.

Z těchto důvodů výplývá, že dokonalejší program by měl s uživatelem vést dialog, a tak mu pomáhat vybrat relevantní informace. Navíc by na požadání měl vysvětlit důvody, které ho vedly k přijetí závěrů. Uvedeme jiný program pro rozpoznávání zvěří podle pravidel P1 až P18, který bude reprezentovat tyto požadavky. Program je převzat z [21].

Obecná jména i jména druhů zvěří, např. zvíře, kopytnatec, šelma, žirafa přijmeme jako konstanty jazyka, v němž budeme říšti. Vlastnosti, vyskytující se v pravidlech P1 až P18, označme konstantami c1 až c21 (přesná formulace odpovídající jednotlivým konstantám bude zřejmá z celkového popisu programu).

Každé pravidlo P1 až P18 může být popsáno pomocí predikátu pravidlo o čtyřech argumentech s významem:

**pravidlo(Císlo-pravidlo,  
Vstupní\_argumentent,  
Závěr,**

**Seznam\_vlastností\_které\_musí\_argument\_mít\_abo\_mohou\_být\_odvozen\_závěr)**

Například pravidlo P10 pak bude mít tvar

**pravidlo(10,šelma,tygr,[c12,c14]).**

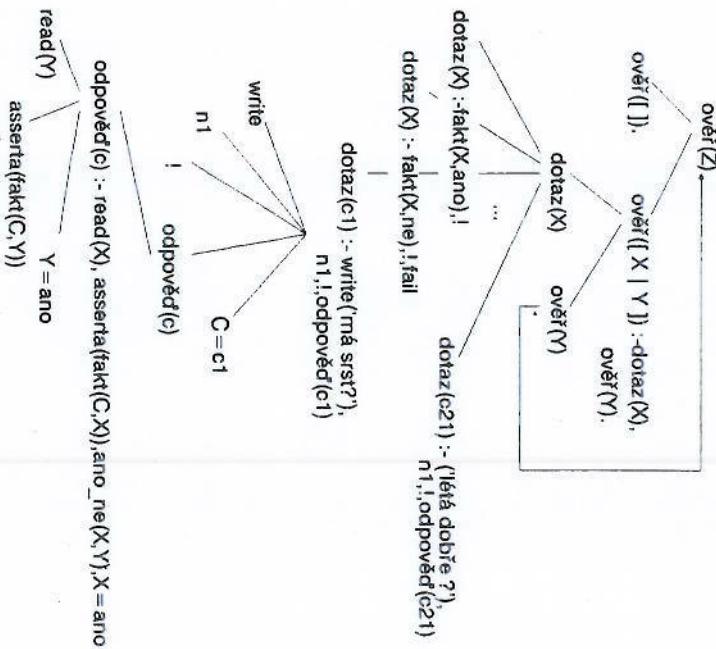
Postupné ověřování platnosti seznamu vlastností odpovídajících zvolenému pravidlu provede predikát ověř (obr.5.8), jehož jediným argumentem bude právě tento seznam.

Uskutečnění dotazu, zda uvažovaný objekt má vlastnost specifikovanou danou konstantou, obstarává predikát dotaz. Tento predikát nejen položí uživateli vhodnou otázku, ale zajistí, aby uživatel nemusel na stejnou otázkou opakováně odpovídat, pokud se stejná vlastnost vyskytuje ve více použitých pravidlech. Takové vlastnosti jsou např. "mít dlouhé nohy" nebo "mít dlouhý krk" z pravidel P13 a P16. Definice predikátu dotaz využívá možnosti jazyka Prolog upravovat během výpočtu množinu klauzul, které tvoří program. K tomu slouží vestavěné predikáty assert, retract a retractall (odst. 4.4.8) s jediným argumentem, kterým je klauzule. Tyto predikáty umožňují nový přístup k řešení našeho problému.

Informace, které uživatel poskytl systému, budeme ukládat pomocí predikátu fakt(X,Y) přímo mezi klauzule programu. Na začátku dialogu o novém objektu neobsahuje program klauzule s predikátem fakt v hlavě klauzule. Postupně je však získává pomocí asserta v procedurě odpověď (obr. 5.8).

Zde má zkoumaný objekt vlastnost X, zjištěje predikát dotaz třemi možnými způsoby:

- Zjistí, zda databáze obsahuje fakt(X,ano). Pokud ano, pak úspěšně končí.



Obr. 5.8

b) Zjistí, zda databáze obsahuje fakt(X,ne). Pokud ano, končí dotaz po platnosti X neúspěšněm.

c) Neobsahuje-li databáze ani kladou, ani zápornou informaci o vlastnosti X, položí program uživateli otázku na vlastnost X pomocí zápisu na obrazovce (write(...)).

Pokud získaná odpověď byla ano, je zodpovězený dotaz přidán do databáze procedurou odpověď a procedura dotaz úspěšně skončí.

Nyní máme připraveny stavební kameny pro návrh požadovaného programu. Máme-li podat co nejpresnější informaci o objektu X v souladu se stanovenými pravidly, mohou nastat opět tři případy:

- Existuje pravidlo aplikovatelné na X, s nimž se odvodí blížší určení.
- Takové pravidlo v našem výpočtu není, ale informace o X už byla odvozena pomocí jiného pravidla.

c) Na X se žádne pravidlo neshodí a během celé konzultace o něm z pravidel nic nevyplývalo.

Pro rozlišení situací a a b použijeme opět výhody predikátu asserta, který po úspěšné aplikaci libovolného pravidla přidá do databáze predikát rozpoznej. Celý proces rozpoznání zvítězí probíhá tedy podle algoritmu pro predikát rozpoznej popsaného výpočtovým stromem na obr. 5.9, kde všechno a, b, c odpovídají třem uvažovaným případům. Pokus o aplikaci nějakého pravidla (případ a) na objekt X probíhá takto (obr. 5.9):

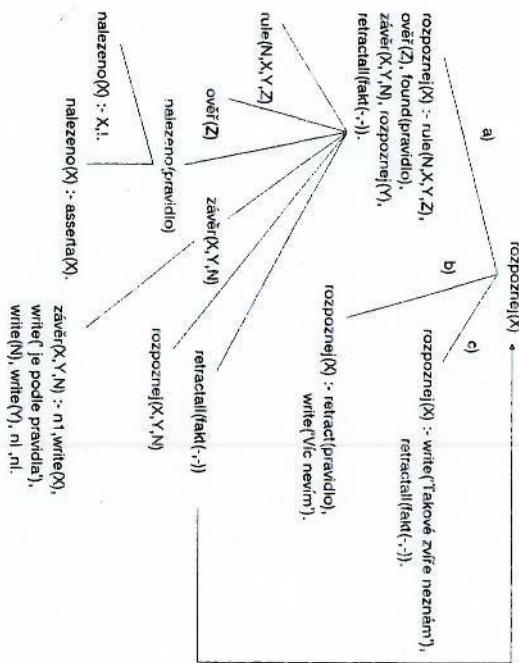
- V prologovské databázi, která obsahuje výčet pravidel pravidlo(„„, „“), se vybere pravidlo, jehož vstupní (tj. druhý) argument je týž jako X.
- Zkontroluje se, zda X splňuje všechny vlastnosti dané výčtem z uvažovaného pravidla pomocí predikátu ověř. Pokud procedura ověř skončí neúspěchem, hledá se nové pravidlo (popř. se řízení vrací k proceduře rozpoznej).
- Po úspěchu procedury ověř zajistí predikát nalezeno přítomnost predikátu pravidlo v databázi.
- Vytiskne se závěr použitého pravidla, tj. hlášení, že X je Y podle pravidla N (viz predikát závěr).

- O nově odvozeném závěru se opět hledají údaje pomocí procedury rozpoznej, az do úplného vyčerpání všech pravidel. Vytiskne se zpráva o ukončení a vymaže se pravidlo z databáze.
- Na závěr se z databáze odstraní všechny odvozené údaje tvaru fakt(„“). Po konzultaci tedy program obsahuje stejně katalogu jako před ní.

Nyní uvedeme úplné znění programu, které by, až na procedury pravidlo a døaz, bylo možno vyuèist z kluauzí obsažených ve výpočtových stromech na obr. 5.8 a 5.9:

% Báze znalostí (pravidla)

```
pravidlo(1,zvíře,savec,[c1]).  
pravidlo(2,zvíře,savec,[c2]).  
pravidlo(3,zvíře,pták,[c3]).  
pravidlo(4,zvíře,pták,[c4,c5]).  
pravidlo(5,savec,šelma,[c6]).  
pravidlo(6,savec,šelma,[c7,c8,c9]).  
pravidlo(7,savec,kopytnatec,[c10]).  
pravidlo(8,savet,sudokopystík,[c11]).  
pravidlo(9,šelma,tygr,[c12,c14]).  
pravidlo(10,kopytnatec,žirafa,[c15,c16,c12,c13]).  
pravidlo(11,kopytnatec,žirafa,[c15,c16,c12,c13]).
```



Obr. 5.9

```
pravidlo(12,kopytnatec,zebra,[c17,c14]).  
pravidlo(13,pták,pštros,[c18,c15,c16,c19]).  
pravidlo(14,pták,tučňák,[c18,c20,c19]).  
pravidlo(15,pták,albatros,[c21]).  
pravidlo(16,pták,čáp,[c23,c15,c16,c17]).  
pravidlo(17,šelma,kočka,[c22,c25]).  
pravidlo(18,šelma,pes,[c22,c24]).  
pravidlo(20,šelma,pes,[c22,c26]).
```

% Proces rozpoznavání, tj. nalezení jména zvítězite

```
rozpoznej(X) :-  
    pravidlo(N,X,Y,Z),  
    ověř(Z),  
    nalezeno(pravidlo),  
    závěr(X,Y,N),
```

```

rozpoznej(Y),
retractall(fakt(_, _)). 

write(' Víc nevím. '), nl.

rozpoznej() :- 
write(' Takové zvíře neznám. '),
retractall(fakt(_, _)), nl.

ověř([]).

ověř([H | T]) :- dotaz(H), ověř(T).

nalezeno(X) :- call(X), !.
nalezeno(X) :- asserta(X).

dotaz(X) :- fakt(X,ano), !.
dotaz(X) :- fakt(X,ne), !, fail.

dotaz(c1) :- write('Má srst?'), nl, !, odpověď(c1).
dotaz(c2) :- write('Dává mléko?'), nl, !, odpověď(c2).

dotaz(c3) :- write('Má peří?'), nl, !, odpověď(c3).
dotaz(c4) :- write('Létá?'), nl, !, odpověď(c4).

dotaz(c5) :- write('Klade vejce?'), nl, !, odpověď(c5).

dotaz(c6) :- write('Žíví se masem?'), nl, !, odpověď(c6).

dotaz(c7) :- write('Má špičaté zuby?'), nl, !, odpověď(c7).

dotaz(c8) :- write('Má drápy?'), nl, !, odpověď(c8).

dotaz(c9) :- write('Míří jeho oči dopředu?'), nl, !, odpověď(c9).

dotaz(c10) :- write('Má kopyta?'), nl, !, odpověď(c10).

dotaz(c11) :- write('Prežívkuje?'), nl, !, odpověď(c11).

dotaz(c12) :- write('Má žlutohnědou barvu?'), nl, !, odpověď(c12).

dotaz(c13) :- write('Ma tmavé skvrny?'), nl, !, odpověď(c13).

dotaz(c14) :- write('Má černé pruhý?'), nl, !, odpověď(c14).

dotaz(c15) :- write('Má dlouhý krk?'), nl, !, odpověď(c15).

dotaz(c16) :- write('Má bílou barvu?'), nl, !, odpověď(c16).

dotaz(c17) :- write('Má bílou barvu?'), nl, !, odpověď(c17).

dotaz(c18) :- write('Je pravda, že neumí létat?'), nl, !, odpověď(c18).

dotaz(c19) :- write('Je to černobilíš?'), nl, !, odpověď(c19).

dotaz(c20) :- write('Umí plavat?'), nl, !, odpověď(c20).

dotaz(c21) :- write('Žije v domácnosti u lidí?'), nl, !, odpověď(c21).

dotaz(c22) :- write('Je to šelma-pes podle pravidla 19. '),
            write(' Víc nevím. ').

dotaz(c23) :- write('Zjde u vodě?'), nl, !, odpověď(c23).

dotaz(c24) :- write('Štěká?'), nl, !, odpověď(c24).

dotaz(c25) :- write('Mňouká?'), nl, !, odpověď(c25).

```

závěr(X,Y,N) :-  
nl, write('Je to '), write(X),  
write(' -'), write(Y),  
write(' podle pravidla '), write(N), nl.

Pro úspěšné určení zvířete navrženým programem začíná dialog dotazem  
?- rozpoznej(zvíře).

Počítac klade otázky, uživatel na ně odpovídá:

|                                     |                                     |
|-------------------------------------|-------------------------------------|
| Má srst?                            | Má žlutohnědou barvu?               |
| ano                                 | ano                                 |
| Je to savec-šelma podle pravidla 1. | Je to žlutohnědou barvu?            |
| Žíví se masem?                      | ne                                  |
|                                     | Je to savec-šelma podle pravidla 5. |
|                                     | Má žlutohnědou barvu?               |
|                                     | ano                                 |
|                                     | Žije v domácnosti u lidí?           |
|                                     | ne                                  |
|                                     | Mňouká?                             |
|                                     | ano                                 |
|                                     | Je to šelma-pes podle pravidla 19.  |
|                                     | Víc nevím.                          |

Povšimněte si blíže struktury programu, který jsme uvedli. Procedury rozpoznej, ověř, závěr i odpověď jsou viceméně nezávislé na oblasti, která nás zajímá. Daly by se použít i v případě, že bychom místo zvěřat chtěli rozpoznávat květiny, značky aut apod. Speciálnost systému je dána pra-

vidly a dotazy, tj. procedurami pravidlo a dotaz, které systém přizpůsobují dané konkrétní oblasti. Tento přístup je běžný při vytváření expertních systémů různých stupňů složitosti. Obykle se expertní systém navrhuje ve dvou vrstvách. První tvorí problémově nezávislou (prázdnou) kostru či skořápkou (shell), která definuje jak odvozovací schopnosti systému bez ohledu na aplikační oblast, tak i způsob komunikace s uživatelem. Druhou tvoří báze znalostí, která definuje vztahy specifické pro danou oblast aplikace.

Nás program pro rozpoznávání zvířat má jen hrubé rysy dobrého expertního systému. Chybí mu mj. možnost uvádět jako podmínky pravidel složitější kombinace vlastností než pouhé konjunkce. Nedává uživateli možnost odpovídat na dotazy hodnotou "nevím" nebo "vím to jen s jistotou dávkou určitosti". Stejně i pravidla jsou v našem systému příliš kategorická – odvozují důsledky s naprostou jistotou! To však není běžné ve složitých prostředích, především např. v lékařské diagnostice, pro která jsou expertní systémy zamýšleny. Problém, jak zacházet s neurčitými informacemi, je jedním z nejaktuálnějších v oblasti expertních systémů. Budeme se mu věnovat v následujícím odstavci, kde také ukážeme, jak učinit jazyk pro vyjadřování znalostí (pravidel) pružnějším.

Zádný z naznačených nedostatků našeho programu na rozpoznávání zvířat však není způsoben programovacím jazykem Prolog. Naopak, jeho základní vlastnosti umožňují dalším zjednodušením struktury úlohy vytvořit systém, který se zmíněným nedostatkům vyhne.

Zvrubnou a snadno srozumitelnou informaci o obdobném systému je možno nalézt např. v [9], [21] a [43]. Praktické zkušenosti z návrhu expertních systémů v Prologu popisují [31], [32] a [44].

### 5.9.3 Pravidla a jejich překlad do logického programu

V tomto a v následujících odstavcích si ukážeme jeden z možných přístupů k tvorbě problémově nezávislého (prázdného) expertního systému. Nejprve učiníme rozhodnutí o struktuře báze znalostí. Ta bude obsahovat pravidla, ale i dotazy a specifikaci vrcholových hypotéz. Budeme předpokládat, že pravidla, která tvoří rozhodovací strom, jsou vytvořena textovým editorem ve tvaru

```
veličina1(Hodnota1),...,veličinaN(HodnotaN)
-> závěr((Text závěru 1)Váha závěru 1,...,
závěrM(Text závěru M)Váha závěru M.
```

S každou vstupní (dotazovací) veličinou je spojen dotaz tvaru

```
dotaz(veličina,Seznam_možných_hodnot) = 'Text dotazu'.
```

Například s dvouhodnotovou veličinou ohezita bude spojen dotaz

dotaz(ohezita,[ano,ne]) = 'Je pacient obézní?'.  
kdežto s dvouhodnotovou veličinou barva očí bude spojen dotaz

dotaz obarva\_ocí,[hnědá,černá,modrá,šedá,zelená]) =

'Jakou barvu očí má pacient?'.

Definujme predikát **překlad\_P**, který v cyklu repeat-fail převede dotazy, specifikaci číslové hypotézy a pravidla do prologovských klausul:

```
překlad_p(end_of_file).
překlad_p(dotaz(Jmeno,Legální_hodnoty) = Dotaz);-
Jméno_terminu = ..([Jméno,X,Váha], assertz(
(Jméno_terminu:-dotaz(Dotaz,Legální_hodnoty,X,Váha))), !,fail.
překlad_p(Goal = Obj);-
GoalTerm = ..[Goal,X,Váha],
ObjTerm = ..[Obj,X,Váha],
assertz((GoalTerm:-ObjTerm)), !,fail.
překlad_p((Předpoklady -> Závěry));-
překlad_p(p(Předpoklady,Předp,Váha),
překlad_závěr(Závěry,Předp,Váha),
! ,fail.

překlad_předp((A1,A2),(P_A1,P_A2),Váha);-
překlad_1_předp(A1,P_A1,Out),
překlad_zbytek_předp(A2,P_A2,Out,Váha).
překlad_předp(A,P_A,Váha);-
překlad_1_předp(A,P_A,Váha).

překlad_zbytek_předp((A1,A2),(P_A1,Temp2,is
In*Temp1,P_A2),In,Out);!,
překlad_1_předp(A1,P_A1,Temp1),
překlad_zbytek_předp(A2,P_A2,Temp2,Out).
překlad_zbytek_předp(A,(P_A,Out is In*Temp),In,Out);-
překlad_1_předp(A,P_A,Temp).

překlad_1_předp((A1,A2),(P_A1,P_A2),Out);!,
překlad_předp(A1,P_A1,Out),
překlad_předp(A2,P_A2,Out),
překlad_1_předp(Term,inference(Atom,Val,Out),Out);-
Term = ..[Atom,Val].
```

```

překlad_závěr(Term/Váha,P_A,Out):-!
překlad_závěr(C,P_A,Out);-
překlad_1_závěr(C,P_A,Out).

```

```

překlad_1_závěr(Term/Váha,P_A,Out);!,
Term =..[H,Val],
RealTerm =..[H,Val,Out1],
assertz((RealTerm:-P_A,Out1 is Out#Váha)).

```

```

překlad_1_závěr(Term,P_A,Out);-
Term =..[H,Val],
RealTerm =..[H,Val,Out],
assertz(RealTerm->P_A)).

```

První klausule predikátu `překlad_p` testuje konec souboru, druhá překládá definice dotazů a ukládá je do prologovské databáze. Term `dotaz(Jméno,Legální_hodnoty)`

je převeden na klausuli tvaru

```
Jméno(X,Váha) :- dotaz(Dotaz,Legální_hodnoty,X,Váha).
```

kde místo "proměnné" `Jméno` bude stát ta hodnota, kterou byla vázána v předchozí klausuli. Analogicky, vložková hypotéza (`cíl`) báze znalostí, tj. sítě pravidel, je převedena na klausuli tvaru

```
cíl(X,Váha) :- Cíl(X,Váha).
```

Nakonec je definován vlastní překlad pravidel, který se provádí postupně, a vzniklé prologovské klausule jsou opět uloženy do databáze. Při překladu je třeba brát v úvahu nejen počet předpokladů, ale i počet záverů. Vypočítá – li z týchž předpokladů `n` záverů, je třeba vytvořit `n` klausuli Prologu. Takoto vytvořená báze znalostí je potom zpracovávána inferenčním mechanismem.

#### 5.9.4 Inferenční stroj

Popišeme zjednodušený inferenční stroj, který je modifikací známého expertního systému M1 firmy Tecknowledge, Inc., jak byl popsán v [43].

```

go:-                                         % Vyjištění databáze, obrazovky a
reinitializace,cls,gol.                      % spuštění inferenčního stroje
gol:-                                         % Odvození
                                              % Vlastní fail cyklus
                                              % odvození(cíl,Hodnota,Váha),   % inferenčního stroje
                                              % write(vrcholový cíl = Hodnota),nl,
                                              % write(váha = Váha),nl,nl,
                                              % fail.

```

```

go!:-                                         % Zpracování dotazu
write('Nová konzultace, zadaj: go.'),nl,
write('Výstup z interpretu Prologu, zadaj: halt.').

```

```

dotaz(Dotaz,Legální_hodnoty,X,Váha):-          % Zpracování dotazu
write(Dotaz),
tab(1),
read(Y),
správná_odpověď(Y,Legální_hodnoty),!,          % odpověď(Y,Legální_hodnoty,X,Váha):
všechny_odpovědi(Y,X,Váha).

```

```

dotaz(Dotaz,Legální_hodnoty,X,Váha):-          % odpověď(Y,Legální_hodnoty,X,Váha):
write('Chybna odpověď. Legální hodnoty jsou z množiny '),
write(Legální_hodnoty),
nl,
dotaz(Dotaz,Legální_hodnoty,X,Váha),
správná_odpověď((A,B),Legální_hodnoty);-,      % Kontrola správnosti
správná_odpověď(A,Legální_hodnoty),               % odpovědi
správná_odpověď(B,Legální_hodnoty),
správná_odpověď(Y,Legální_hodnoty);-,           % odpovědi(Y,Legální_hodnoty)
správná_odpověď(Y,Legální_hodnoty);-            % odpovědi(Y,Legální_hodnoty)
(member(Y,Legální_hodnoty);Y = unknown),!.

```

```

všechny_odpovědi((A,B),X,Váha);-,              % odpovědi(A,B,X,Váha):
(všechny_odpovědi(A,X,Váha);všechny_odpovědi(B,X,Váha));-
všechny_odpovědi(X/Váha,X,Váha);-.
všechny_odpovědi(X,X,1.0).

```

```

member(H,[H|_]).
member(H,[_|T]);-member(H,T).
odvození(Atom,Hodnota,Váha);-          % Odvození váhy
recorded(Atom,_),                         % výroku
!,                                     %
recorded(Atom,zjištěno(Hodnota,Váha),).    %
odvození(Atom,Hodnota,Váha);-          % Odvození
                                              % pomocná odvozen((Atom,V,VáhaTemp),
                                              % zjištěny_fakt(Atom,V,VáhaTemp),
                                              % fail.
recorded(Atom,Hodnota,Váha);-          %
recorded(Atom,_),                         %

```

!,  
recorded(Atom,zjištěno(Hodnota,Váha),\_).  
odvození(Atom,unkown,1,0),-  
recordz(Atom,zjištěno(unknow,1,0),\_).

zjištěny fakt(Atom,Hodnota,VáhaNew):-  
zjištěno(Atom,Hodnota,VáhaOld)!,  
VáhaMerge is VáhaOld + VáhaNew \* (1.0 - VáhaOld),  
retract(zjištěno(Atom,Hodnota,VáhaOld)),  
assert(zjištěno(Atom,Hodnota,VáhaMerge)).

zjištěný fakt(Atom,Hodnota,VáhaNew):-  
assert(zjištěno(Atom,Hodnota,VáhaNew)).

reinicializace :-

recorded(rule\_consulted,X,\_),  
eraseall(X),  
fail.

reinicializace.

V definici predikátu odvození a reinicializace jsme použili vestavěných predikátů Arity Prologu record a recorded. Umožňují ukládat a vyhledávat termy podle klíčů, avšak nejsou běžné v jiných implementacích. Predikát eraseall(K) vymaže z prologovské databáze všechny termy uložené pod klíčem K. (Význam těchto predikátů viz odst. 7.1.3.)

Nakonec se ještě zmíníme o strategiích odvozování. Inferenční mechanismy deklarativních expertních systémů nejčastěji využívají *strategii zpětného řešení* pravidel. Ta je pro logické programování typická. Zpětné (backward) řešení je vedeno člověkem hypotézami, naproti tomu *přímé* (forward) řešení je vedeno daty. Systémy s přímým řešením pravidel lze sice také snadno kódovat v Prologu, ale většinou, pokud jsou realizovány jako interprety pravidel, bývají neefektivní. Yamamoto a Tanaka [62] navrhli alternativní metodu překladu pravidel do prologovských klauzul. Vykonání programu v Prologu se pak obejde bez interpretu pravidel a je podstatně rychlejší. Metodu ve stručnosti popiseme.

Předpokládáme jako dosud, že pravidla jsou zapisována ve tvaru

$p_1, \dots, p_n \rightarrow z$

kde  $p_1, \dots, p_n$  jsou předpoklady, z je závěr pravidla. Nezájímá nás tedy vnitřní struktura výroků  $p_i$  a z. Každé takové pravidlo přeložíme na klauzuli Prologu tvaru

$p_1(G) \leftarrow \text{deriv}(z, G), \text{goal}(p_2), \dots, \text{goal}(p_n), z(G).$

přičemž v případě negativních výroků se místo podcelku  $\text{goal}(p_i)$

generuje podcelk not  $\text{goal}(p_i)$ . Pravidla bez levé strany, tj. akceptovaná fakta, se přeloží na nepodmíněné příkazy tvaru

fakt(p).

Kromě toho jsou generovány tři druhy klauzul: *cílová deriváční* a *terminální*. Cílová klauzule má tvar

goal(G) :- fakt(F), deriv(F,G), P = ..[F,G], call(P).

Tato klauzule vyhledá v databázi vhodný fakt F a vyvolá pravidlo, které má tento fakt jako předpoklad. Spojovací klauzule definují reflexivní a transzitivní relaci odvoditelnosti mezi uzly sítě produktivních pravidel. Terminální klauzule definují podmínky ukončení výpočtu pro jednotlivé výroky. Všechny mají tvar atom(atom), tedy např. z(z), p1(p1), ..., pn(pn).

Prologovský program pracuje takto: V okamžiku kdy má být učiněn pokus o splnění podcelku p1, je činěn pokus splnit postupně podcelky p2, ..., pn. Přitom proměnná G je konkretizována buď finálním závěrem, nebo některým z mezikláuzul. Na začátku při volání

?- goal(G).

je ovšem vlná.

*Příklad:* Uvažujme systém dvou produkčních pravidel a fakt.

p1 -> p2  
p2,p3 -> p4  
p1  
p3

Přeložme jej do prologovského programu:

% Překlad pravidel  
p1(G) :- deriv(p2,G), p2(G).  
p2(G) :- deriv(p4,G), goal(p3), p4(G).

% Relace odvoditelnosti  
deriv( \_, G) :- var(G) !.  
deriv(p1, p2).  
deriv(p1, p4).  
deriv(p2, p4).  
deriv(X, X).

% Překlad faktů  
fakt(p1).  
fakt(p3).

a konečně

```
% Cílová klauzule
goal(G) :- fakt(F), deriv(F,G), P = ..[F,G], call(P).
```

## 5.10 Produkční systémy

V tomto odstavci se budeme zabývat produkčními systémy. S jistou dávkou tolerance bychom je mohli označit za zvláštní druh expertních systémů. Jsou to systémy, které jsou založené na pravidlech tvaru

Jestliže <situace>, potom <akce>

Předpokládáme, že <situace> je seznam podmínek, které musejí být splněny, aby pravidlo mohlo být aplikováno. Dále předpokládáme, že <akce> je seznam procedur, které jsou vykonány, jakmile je pravidlo spuštěno. Produkční systém pracuje tak, že na základě aktuálního obsahu databáze vybere nejprve jedno aplikovatelné pravidlo. To je spuštěno, což znamená, že jsou postupně provedeny všechny akce obsažené v jeho akční části. provedením akcí může být změněn aktuální obsah databáze. Po úspěšné aplikaci pravidla je vybráno další aplikovatelné pravidlo (pokud existuje) a celý postup se opakuje. Říkáme, že produkční systém pracuje v cyklu "rozpoznej-vykonej". V daném kroku výpočtu se však může stát, že je více pravidel, která mohou být aplikována. Řídící mechanismus produkčního systému pak musí vybrat jediné, jež bude spuštěno. To, jak se výběr provede, bude záležet na typu úlohy. Protože v prologovské databázi jsou pravidla usporádána do postoupnosti, je nejjednodušší strategii výběru výběr prvního aplikovatelného pravidla. V Prologu ji lze realizovat programem:

```
CF1 + CF2*(1.0 - CF2).

(viz procedura zjištěný fakt).

Je zřejmé, že volba kombinační funkce je záležitostí teoretických úval o podstatě faktorů věření (vah) a nesouvise s vlastnostmi jazyka Prolog. Bude proto třeba se podrobnejší zabývat možnostmi jak zobecnit rezoluční princip dokazování, aby bylo možné jej využít i na logické programy s neurčitostmi. Teoretické úvahy na toto téma jsou obsaženy v čl. 3.
```

Dosud jsme hovořili pouze o bázi znalostí a odvozovacím mechanismu, tj. o jádru expertního systému. Expertní systém může být ještě doplněno další složky. Mezi důležité patří uživatelské prostředí a prostředek pro tvorbu bází znalostí. I zde je Prolog vhodným programovacím prostředkem.

Zbyvá tedy definovat procedury rozpoznej a vykonej. K tomu nejprve předepišeme tvar pravidel. Jednou z možností je zapisovat pravidla jako ternární predikát

```
pravidlo(Jméno, Podmínky, Akce)
```