Katedra matematickej informatiky
Prírodovedecká fakulta UPJŠ

Logické programovanie

Miroslav Repický a Peter Vojtáš

Košice 1999

Miroslav Repický: Matematický ústav SAV Bratislava,
dislokované pracovisko v Košiciach,
Jesenná 5, 041 54 Košice
E-mail: repicky@kosice.upjs.sk

Peter Vojtáš: Katedra matematickej informatiky,
Prírodovedecká fakulta UPJŠ v Košiciach,
Jesenná 5, 041 54 Košice
E-mail: vojtas@kosice.upjs.sk

Obsah

1.1. Základné pojmy. Programovanie v Prologu pozostáva z

- deklarovania určitých *faktov* o objektoch a ich vlastnostiach,
- definovania určitých *pravidiel* o objektoch a ich vlastnostiach, a
- z kladenia *otázok* o objektoch a ich vlastnostiach.

Prolog sprostredkúva určitý druh konverzácie s počítačom o informáciách uložených v databázach. Je vhodným jazykom pre aplikácie symbolických výpočtov zahŕňajúc relačné databázy, matematickú logiku, symbolické riešenie rovníc, riešenie abstraktných problémov, prirodzené jazyky, analýzu biochemických štruktúr, niektoré oblasti umelej inteligencie.

1.1.1. *Fakty.* Predpokladajme, že chceme v Prologu povedať, že Adam je otcom Fera. Tento fakt má dva objekty „Adam“ a „Fero“, a jednu vlastnosť „byť otcom“. Štandardný spôsob zápisu tohto faktu v Prologu je

```
otec(adam,fero).
```

Všimnime si tento zápis.

- Objekty a vzťahy sú napísané malými písmenami.
- Najskôr je napísaný vzťah a objekty sú uvedené za ním (bez medzery) v zátvorkách a navzájom sú oddelené čiarkami.
- Bodka ukončuje fakt.

1.1.2. *Otázky.* Otázka v Prologu vyzerá presne ako fakt s jediným rozdielom, že je pred ním umiestnený symbol `?-`.

1.1.3. *Premenné.* Otázku „otcom koho je Adam?“ použitím premennej zapíšeme

```
?-otec(adam,X).
```

Uvažujme nasledujúcu databázu faktov:

```
otec(jano,maria).
otec(jano,adam).
otec(adam,fero).
otec(adam,eva).
```

Čo sa deje, keď Prolog dostane otázku `?-otec(adam,X)`. Prolog prechádza databázou faktov zhora nadol a hľadá fakty, ktorých funktor je `otec` a prvý argument je `adam`. Keďže premenná `X` na začiatku nie je inštancializovaná, druhý argument môže byť čokoľvek. Keď sa nájde takýto fakt, do premennej `X` sa dosadí druhý argument tohto faktu. V našom prípade je to tretí fakt a premenná `X` sa inštancializuje objektom `fero`. Prolog si poznačí toto miesto v databáze a čaká na ďalšie pokyny. Ak stlačíme ⟨Return⟩ znamenajúci spokojnosť s odpoveďou, prestane s prehľadávaním databázy. Ak namiesto toho napíšeme znak `;` pre alternatívne riešenie, zopakuje prehľadávanie, ale teraz od miesta, kde zanechal značku. V prípade, že sa mu nepodarí cieľ opätovne splniť odpovie `no`. V našom prípade dialóg bude vyzerať takto:

```
?-otec(adam,X).
X=fero ;
X=eva ;
no
```

Prolog hľadá odpovede na všetky premenné, ktoré sú v otázke. Pokiaľ otázka neobsahuje premenné, jeho odpoveď je len `yes`, alebo `no`.

1.1.4. *Konjunkcia.* Otázku „Kto je (spoločným) otcom Evy a Fera?“ v Prologu zapíšeme

```
?-otec(X,eva),otec(X,fero).
```

Čiarka medzi dvoma faktami znamená logickú spojku „a“. Túto otázku sa Prolog pokúša zodpovedať tak, že najskôr hľadá v databáze fakt s rovnakým funktorom a druhým argumentom ako má prvý cieľ. Ak ho nájde, urobí si pri ňom značku, premennú `X` inštancializuje druhým argumentom tohto faktu a pokúša sa splniť druhý cieľ v otázke s takto inštancializovanou premennou `X`. Ak sa mu podarí splniť aj tento cieľ, urobí si značku v databáze pre tento cieľ a oznámi nájdené riešenie. Ak sa mu nepodarí splniť tento cieľ, pokúsi sa hľadať iné riešenie predchádzajúceho cieľa prehľadávajúc databázu od miesta značky pre tento cieľ a až keď je úspešný, pokračuje v spĺňaní ďalšieho cieľa prehľadávajúc databázu od začiatku.

1.1.5. *Pravidlá.* V Prologu sa používajú pravidlá na vyjadrenie skutočnosti, že jeden fakt nejako závisí na skupine ďalších faktov. V hovorovej reči takýmito pravidlami sú napríklad vety:

Živočích je vtákom, ak má perie.
Dvaja ľudia sú súrodenci, ak majú spoločného rodiča.
Jano má rád každého, kto má rád pivo.
Jano je starým otcom Fera, ak je otcom Ferovej matky.
Jano je starým otcom Fera, ak je otcom Ferovho otca.

Pravidlo v Prologu má hlavu a telo oddelené od seba symbolom `:-`. Rovnako ako fakty, aj pravidlá sú ukončené bodkou. Uvedené pravidlá sa v Prologu dajú vyjadriť takto

```
vtak(X):-zivocich(X),perie(X).
surodenci(X,Y):-rodic(Z,X),rodic(Z,Y),X\=Y.
ma_rad(jano,X):-ma_rad(X,pivo).
stary_otec(jano,fero):-otec(jano,X),matka(X,fero).
stary_otec(jano,fero):-otec(jano,X),otec(X,fero).
```

Cieľ `X\=Y` v druhom pravidle vylučuje možnosť, aby niekto bol súrodencom samého seba. Všimnime si, že premenné sú napísané veľkými písmenami.

Nasledujúca databáza je databáza `otec` z časti 1.1.3 doplnená o jedno pravidlo.

1. `otec(jano,maria).`
2. `otec(jano,adam).`
3. `otec(adam,fero).`
4. `otec(adam,eva).`
5. `stary_otec(X,Y):-otec(X,Z),otec(Z,Y).`

Položme otázku `?-stary_otec(jano,X).` Prolog pri hľadaní odpovede postupuje takto:

1. Najskôr v danej databáze hľadá klauzulu (fakt alebo pravidlo), ktorá začína funktorom `stary_otec`. V tomto prípade nájde piatu klauzulu databázy. Nie je žiadna súvislosť medzi premennou `X` v nájdenom pravidle a premennou `X` v otázke.

2. Na miesto premennej `X` v pravidle sa dosadí `jano` a na miesto premennej `Y` sa dosadí premenná `X`. Zapamätá si substitúciu `X/jano`, `Y/X` pre toto pravidlo a poznačí si dané miesto v databáze pre cieľ `stary_otec(jano,X)`. Nasleduje spĺňanie tela nájdeného pravidla s touto substitúciou, t.j. spĺňanie cieľov `otec(jano,Z),otec(Z,X)`.
3. Prechádzajúc databázou snaží sa nájsť odpoveď pre cieľ `otec(jano,Z)`. Tento cieľ sa splní prvým faktom v databáze pre `Z=maria`. Poznačí si toto miesto pre tento fakt.
4. Spĺňa cieľ `otec(maria,X)`. Táto snaha bude neúspešná, lebo `maria` nie je prvým argumentom žiadneho faktu v databáze. Znamená to návrat k predchádzajúcemu cieľu.
5. Prechádzajúc databázou od značky pre cieľ `otec(jano,Z)`, druhý fakt v databáze poskytne ďalšie splnenie tohto cieľa. Značka pre cieľ `otec(jano,Z)` sa premiestni na tento fakt, `Z` sa inicializuje na `adam`. Nasleduje spĺňanie druhého cieľa pravidla, `otec(adam,X)`.
6. Prechádzajúc databázou od začiatku, cieľ `otec(adam,X)` sa splní tretím faktom, kde sa umiestni značka pre tento cieľ. Premenná `X` sa inštancializuje na `fero` a na termináli sa vypíše odpoveď `X=fero`.

Podobným postupom využijúc umiestnené značky pre jednotlivé ciele v databáze sa nájde aj druhé riešenie `X=eva`, ale požiadavka na ďalšie riešenie už nebude úspešná.

Spravidla otázka v Prologu je konjunkciou cieľov, ktoré sa majú splniť. Pri hľadaní odpovede na danú otázku Prolog využíva klauzuly v databáze. Fakt v databáze znamená okamžité splnenie cieľa, zatiaľ čo pravidlo redukuje úlohu splniť daný cieľ na splnenie podcieľov v tele pravidla. Samozrejme, klauzula sa môže použiť len v prípade, keď sa môže zhodnúť (pomocou vhodnej substitúcie) s daným cieľom. Ak sa cieľ nedá splniť, začne sa backtrack, ktorý zohľadňuje to čo už bolo urobené a pokúša sa znova splniť ciele, ale iným spôsobom. Backtrack sa dá vyvolať tiež napísaním bodkočiarky `;` za úspešnou odpoveďou na otázku.

1.2. Unifikácia. V konvenčných programovacích jazykoch je základným výpočtovým krokom priradenie hodnôt premenným. Prolog využíva oveľa všeobecnejší typ priradenia, ktorý nazývame *unifikácia*. Unifikácia vždy buď *uspeje* (succeeds) alebo *neuspeje* (fail). V prípade úspešnej unifikácie dvoch termov, ak je to nutné, tieto termy sa zmenia (presnejšie za premenné vyskytujúce sa v týchto termoch sa dosadia iné termy) tak, že sa dané termy stanú ekvivalentnými. V unifikácii premenné môžu získať konkrétne hodnoty, ale môžu byť aj unifikované s inými premennými a vo všeobecnosti s ľubovoľnými termami. Unifikácia zložených termov prebieha rekurzívne: termy sa unifikujú, ak ich funktory sa zhodujú a ich argumenty sa unifikujú. V Prologu sa môžu unifikovať nasledujúce typy dát (v časti 2 je prehľad jednotlivých typov dát v Arity/Prologu):

- Premenné sa môžu unifikovať s ľubovoľnými termami. Špeciálne sú tieto dva prípady:
  - Premenné sa vždy môžu unifikovať s premennými. Ak sa neskôr jedna z premenných inštancializuje, rovnako sa inštancializujú aj ostatné s ňou unifikované premenné.
  - Premenné sa môžu unifikovať s atomickými dátami. Túto skutočnosť môžeme považovať za istú formu konvenčného priradenia (assignment).

- Atomické dáta sa unifikujú s atomickými dátami za predpokladu, že sa navzájom zhodujú. Napríklad, celé číslo `3` sa neunifikuje s atómom `'Jozef'`, ale unifikuje sa s číslom `3`. Podobne atóm `'Jozef'` sa neunifikuje s reťazcom `$Jozef$`, ale atóm `jozef` sa unifikuje s atómom `'jozef'`.
- Zložené termy sa unifikujú s inými zloženými termami za predpokladu, že ich funktory sa zhodujú a odpovedajúce argumenty sa unifikujú.

Funktory sa zhodujú, ak majú zhodný názov a rovnaký počet argumentov, t.j. *árnosť funktora*. Teda `otec(X,Y)` a `otec(X,Y,Z)` sú rôzne termy a ich funktory rozlišujeme v poradí zápisom `otec/2` a `otec/3`. Dvojica termov

`kniha(Autor,Nazov)` a `kniha(jan,$Zelený vrch$)`

sa unifikuje, ale nasledujúce dvojice termov sa neunifikujú:

`meno(kapusta,jan)` a `meno(sokol,jozef)`
`otec(jan,eva,ema)` a `otec(jan,eva)`

Prirodzený predikát, ktorý vyvolá proces unifikácie je binárny predikát `=`. Ako negácia tohto predikátu sa používa binárny predikát `\=`. Cieľ `X\=Y` uspeje, ak `X=Y` neuspeje.

Príklad 1.2.1. *Nasledujúce dvojice zložených termov sa unifikujú. Uvedená je aj substitúcia určená unifikáciou.*

1. term	*2. term*	*substitúcia*
`otec(adam,X)`	`otec(Y,zuza)`	`X/zuza,Y/adam`
`a(X)`	`a(a(0))`	`X/a(0)`
`a(b(X,Y),c(Z))`	`a(Z,c(b(X,Y)))`	`Z/b(X,Y)`
`a(b(x),Y)`	`a(X,b(y))`	`X/b(x),Y/b(y)`
`a(b(X),Y)`	`a(b(c(Y,d)),r(a))`	`X/c(r(a),d),Y/r(a)`

1.3. Výpočtový strom. Výpočet prologovského programu možno stotožniť s tzv. *výpočtovým stromom*. Koreň tohto stromu predstavuje otázka a jednotlivé vrcholy stromu pozostávajú z postupností cieľov, v poradí v akom sa majú spĺňať. V každom kroku výpočtu sa spĺňa jeden konkrétny cieľ a tento cieľ môže byť obyčajne splnený viacerými spôsobmi, čo sa vo výpočtovom strome prejaví vetvením. Každému vetveniu odpovedá celkom konkrétna unifikáciou, čo je v podstate nejaká substitúcia premenných. Postupnosť cieľov vo vrchole môže byť prázdna, čo znamená, že daný vrchol je koncovým vrcholom stromu odpovedajúci úspešnej odpovedi na otázku v koreni stromu. Takýto vrchol budeme nazývať *terminálový vrchol výpočtového stromu*. Odpoveď na otázku sa potom dá získať zložením substitúcií pozdĺž vetvy vedúcej k tomuto terminálovému vrcholu.

Samotný priebeh výpočtu si môžeme predstaviť ako prehľadávanie vrcholov výpočtového stromu s cieľom nájsť terminálový vrchol. Strom sa prehľadáva zhora od koreňa stromu a postupuje sa po vetvách smerom nadol, pričom sa najskôr prehľadávajú vetvy vľavo. Ak sa prehľadávanie ocitne v koncovom vrchole stromu, ktorý nie je terminálovým vrcholom, prehľadávanie sa vráti k poslednému vrcholu, od ktorého vedie vetva vpravo, v krajnom prípade až ku koreňu stromu. Tento návrat predstavuje backtrack vo výpočte. Návrat späť sa vyvolá aj napísaním bodkočiarky po dosiahnutí terminálového vrcholu. Teda backtrack vo výpočte sa vo výpočtovom strome redukuje na prehľadávanie vrcholov tohto stromu backtrackom. Ak výpočtový strom obsahuje nekonečnú vetvu, je zrejmé, že reálny výpočet sa nikdy nedostane napravo od tejto vetvy.

Pripomeňme, že prologovský program je konečná množina klauzúl. Predpokladáme, že klauzuly v programe sú očíslované prirodzenými číslami v rastúcom poradí. Kvôli zjednodušeniu definície výpočtového stromu predpokladajme, že v tele žiadneho pravidla sa nevyskytuje disjunkcia. Nie je to žiadne obmedzenie, pretože pravidlo

```
a:-b,(c;d),e.
```

sa dá ekvivalentne nahradiť dvojicou pravidiel

```
a:-b,c,e.
a:-b,d,e.
```

bez akéhokoľvek vplyvu na samotný výpočet. Jednotlivé kroky výpočtu sú vo výpočtovom strome reprezentované jednotlivými hranami. Očíslujme hrany výpočtového stromu prirodzenými číslami, v poradí v akom prebieha výpočet. Popíšme *výpočtový krok programu*.

Predpokladajme, že postupnosť cieľov $A_1, \ldots, A_n$ je vrchol, v ktorom začína *i-tý výpočtový krok*. Priebeh výpočtového kroku môžeme popísať nasledujúcimi troma bodmi:

(1) Očíslovanie premenných v klauzulách programu indexom i.

(2) Označenie klauzuly φ v takto očíslovanom programe pre cieľ A_1, ktorú možno unifikovať s cieľom A_1. Označí sa prvá klauzula v poradí zhora, ktorá ešte nebola pre tento cieľ označená a určí sa substitúcia θ určená unifikáciou faktu A_1 s označenou klauzulou.

(3) (a) Ak nebola nájdená klauzula, ktorá by vyhovovala bodu (2), tak výpočtový krok skončí návratom k predchádzajúcemu vrcholu výpočtového stromu (takto sa postupuje smerom späť do posledného vrcholu, od ktorého smeruje vetva vpravo a ak takýto vrchol v strome nie je, výpočet skončí v koreni stromu bez úspešného splnenia otázky).

(b) Ak v bode (2) bola nájdená klauzula φ unifikovateľná s cieľom A_1 a θ je príslušná unifikácia, tak sú tieto dve možnosti:

(i) Klauzula φ je fakt. Potom výpočtový krok skončí vo vrchole $(A_2, \ldots, A_n)\theta$. V prípade $n = 1$ tento vrchol je terminálový.

(ii) Klauzula φ je pravidlo A_1`:-`$B_1, \ldots, B_m$. Potom výpočtový krok skončí vo vrchole $(B_1, \ldots, B_m, A_2, \ldots, A_n)\theta$.

Príklad 1.3.1. a) *Zostrojme výpočtový strom pre otázku* `?-stryko(X,juraj)` *v nasledujúcom programe (pripomeňme, že každá klauzula prologovského programu musí byť na novom riadku):*

```
1. muz(adam).
2. muz(peter).
3. muz(pavol).
4. zena(maria).
5. zena(eva).
6. rodic(adam,peter).
7. rodic(maria,peter).
8. rodic(adam,pavol).
9. rodic(maria,pavol).
10. rodic(pavol,juraj).
11. rodic(eva,juraj).
12. stryko(X,Y):-rodic(A,Y),rodic(B,A),rodic(B,X),muz(X),X\=A.
```

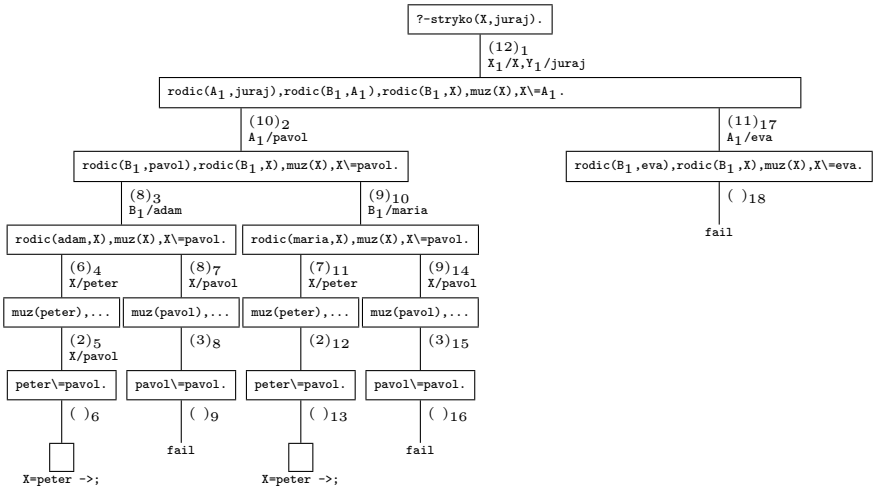
b) *V predchádzajúcej úlohe zameňme definíciu predikátu* `stryko` *pravidlom*

```
12. stryko(X,Y):-muz(X),rodic(A,Y),X\=A,rodic(B,X),rodic(B,A).
```

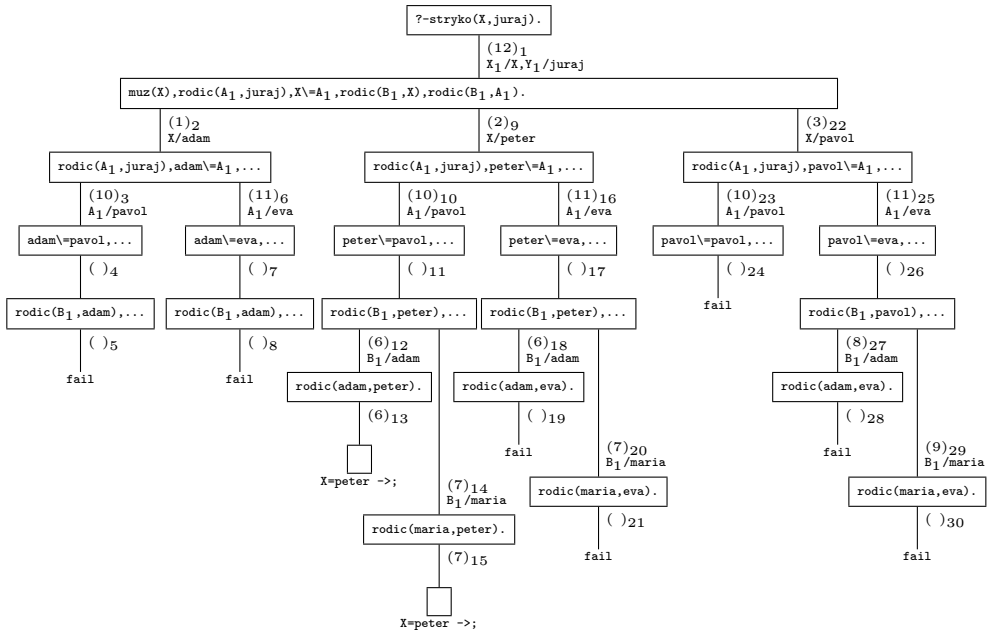
a popíšme výpočtový strom pre rovnakú otázku.

Riešenie. Kvôli stručnosti výber j-tej klauzuly v i-tom výpočtovom kroku označujeme formulou $(j)_i$, pričom číslo i je zároveň indexom, ktorým sa indexujú všetky premenné v klauzulách programu. Formula `X/`t označuje substitúciu premennej `X` za term t, čo znamená, že v každom cieli daného vrcholu výpočtového stromu treba nahradiť premennú `X` termom t. Substitúciu volíme vždy čo najúspornejšie v tom zmysle, že pokiaľ je to možné zamieňame premenné vo vybraných klauzulách a nie premenné vo vrcholoch stromu.

a) Výpočtový strom k uvedenej otázke vyzerá takto:



b) Výpočtový strom v tomto prípade vyzerá takto:



Tento príklad ilustruje dôležitosť poradia jednotlivých cieľov v definíciách predikátov. Použili sme síce ekvivalentnú definíciu predikátu `stryko`, ale výpočtový strom

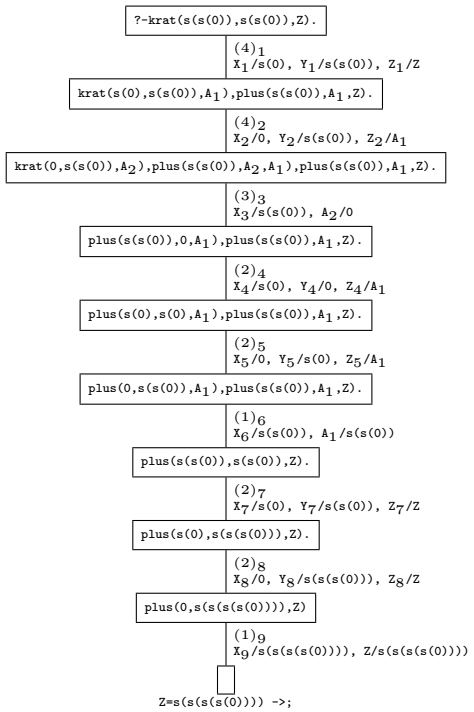
pre odpoveď na rovnakú otázku je teraz zložitejší. Rozdiel bude oveľa výraznejší pri veľkej databáze predikátu `muz`. □

Príklad 1.3.2. *Uvažujme jazyk predikátového počtu pozostávajúci z unárneho symbolu* `s` *a konštanty* `0`*. Na množine všetkých termov tohto jazyka definujme prologovský predikát* `krat(X,Y,Z)` *reprezentujúci násobenie čísiel a zostrojme výpočtový strom pre otázku* `?-krat(s(s(0)),s(s(0)),Z).`*. Pripomeňme, že správna odpoveď na túto otázku by mala byť* `Z=s(s(s(s(0))))`*.*

Riešenie. Budeme potrebovať predikát `plus(X,Y,Z)` reprezentujúci rovnosť $x+y=z$. S jeho použitím výsledný program môže vyzerať takto:

1. `plus(0,X,X).`
2. `plus(s(X),Y,Z):-plus(X,s(Y),Z).`
3. `krat(0,X,0).`
4. `krat(s(X),Y,Z):-krat(X,Y,A),plus(Y,A,Z).`

Všimnime si, že výpočtový strom pre danú otázku sa nevetví.

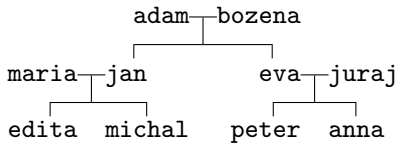


□

V prípade zložitejších výpočtových stromov z priestorových dôvodov sa nám nemusí podariť zakresliť ich stromovú štruktúru. V takom prípade budeme výpočtový strom zapisovať lineárne. Jednotlivé vrcholy stromu budeme zapisovať pod seba v poradí, v akom prebieha výpočet tak, že v tomto zápise za každým vrcholom stromu nasleduje (1) formula v tvare $(n)_i$, ktorá vyjadruje fakt, že v danom výpočtovom kroku sa vyberá n-tá klauzula programu a premenné v nej sa indexujú indexom i, a (2) unifikačná substitúcia. Pre lepšiu prehľadnosť a kontrolovateľnosť tohto zápisu každý vrchol označíme kódom v tvare postupnosti číslic 1, 2, ... , 9, a,

b, c, ... vyjadrujúcim polohu tohto vrcholu v strome. Každá číslica v tejto postupnosti vyjadruje poradie vetvy z predchádzajúceho vrcholu.

Príklad 1.3.3. *Vytvoríme databázu obsahujúcu dva predikáty* `syn(Syn,Rodic)` *a* `dcera(Dcera,Rodic)` *vyjadrené rodokmeňom*



V tejto databáze definujeme predikát `sesternica(X,Y)` *a popíšeme výpočtový strom otázky* `sesternica(anna,X)`.

Riešenie. Nech databáza obsahuje nasledujúcich 12 faktov a 3 pravidlá:

1. `syn(michal,jan).`
2. `syn(michal,maria).`
3. `syn(jan,adam).`
4. `syn(jan,bozena).`
5. `syn(peter,juraj).`
6. `syn(peter,eva).`
7. `dcera(edita,jan).`
8. `dcera(edita,maria).`
9. `dcera(eva,adam).`
10. `dcera(eva,bozena).`
11. `dcera(anna,juraj).`
12. `dcera(anna,eva).`

13. `rodic(X,Y):-syn(Y,X).`
14. `rodic(X,Y):-dcera(Y,X).`

15. `sesternica(A,B):-`
    `dcera(A,R),rodic(S,R),rodic(S,Q),R\=Q,rodic(Q,B).`

Výpočtový strom zapíšeme lineárne podľa popisu pred príkladom. Hodnoty indexov zároveň označujú poradové čísla výpočtových krokov.

$\langle\rangle$ `?-sesternica(anna,X).`
$(15)_1$: $\mathtt{A}_1$`/anna, `$\mathtt{B}_1$`/X`

$\langle 1\rangle$ `dcera(anna,`$\mathtt{R}_1$`),rodic(`$\mathtt{S}_1$`,`$\mathtt{R}_1$`),rodic(`$\mathtt{S}_1$`,`$\mathtt{Q}_1$`),`$\mathtt{R}_1$`\=`$\mathtt{Q}_1$`,rodic(`$\mathtt{Q}_1$`,X).`
Možné vetvenia: $\langle 11\rangle$; $\langle 12\rangle$.
$(11)_2$: $\mathtt{R}_1$`/juraj`

$\langle 11\rangle$ `rodic(`$\mathtt{S}_1$`,juraj),rodic(`$\mathtt{S}_1$`,`$\mathtt{Q}_1$`),juraj\=`$\mathtt{Q}_1$`,rodic(`$\mathtt{Q}_1$`,X).`
Možné vetvenia: $\langle 111\rangle$; $\langle 112\rangle$.
$(13)_3$: $\mathtt{X}_3$`/`$\mathtt{S}_1$`, `$\mathtt{Y}_3$`/juraj`

$\langle 111\rangle$ `syn(juraj,`$\mathtt{S}_1$`),rodic(`$\mathtt{S}_1$`,`$\mathtt{Q}_1$`),juraj\=`$\mathtt{Q}_1$`,rodic(`$\mathtt{Q}_1$`,X).`
$(\ )_4$ fail; návrat do $\langle 11\rangle$.
$\longmapsto$ $(14)_5$: $\mathtt{X}_5$`/`$\mathtt{S}_1$`, `$\mathtt{Y}_5$`/juraj`

$\langle 112\rangle$ `dcera(juraj,`$\mathtt{S}_1$`),rodic(`$\mathtt{S}_1$`,`$\mathtt{Q}_1$`),juraj\=`$\mathtt{Q}_1$`,rodic(`$\mathtt{Q}_1$`,X).`
$(\ )_6$ fail; návrat do $\langle 1\rangle$.
$\longmapsto$ $(12)_7$: $\mathtt{R}_1$`/eva`

$\langle 12\rangle$ `rodic(`$\mathtt{S}_1$`,eva),rodic(`$\mathtt{S}_1$`,`$\mathtt{Q}_1$`),eva\=`$\mathtt{Q}_1$`,rodic(`$\mathtt{Q}_1$`,X).`
Možné vetvenia: $\langle 121\rangle$; $\langle 122\rangle$.
$(13)_8$: $\mathtt{X}_8$`/`$\mathtt{S}_1$`, `$\mathtt{Y}_8$`/eva`

$\langle 121\rangle$ `syn(eva,`$\mathtt{S}_1$`),rodic(`$\mathtt{S}_1$`,`$\mathtt{Q}_1$`),eva\=`$\mathtt{Q}_1$`,rodic(`$\mathtt{Q}_1$`,X).`
$(\ )_9$ fail; návrat do $\langle 12\rangle$

$\longmapsto$ $(14)_{10}$: `X10/S1, Y10/eva`

⟨122⟩ `dcera(eva,S1),rodic(S1,Q1),eva\=Q1,rodic(Q1,X).`
Možné vetvenia: ⟨1221⟩; ⟨1222⟩.
$(9)_{11}$: `S1/adam`

⟨1221⟩ `rodic(adam,Q1), eva\=Q1,rodic(Q1,X).`
Možné vetvenia: ⟨12211⟩; ⟨12212⟩.
$(13)_{12}$: `X12/adam, Y12/Q1`

⟨12211⟩ `syn(Q1,adam), eva\=Q1,rodic(Q1,X).`
$(3)_{13}$: `Q1/jan`

⟨122111⟩ `eva\=jan,rodic(jan,X).`
$(\,)_{14}$

⟨1221111⟩ `rodic(jan,X).`
Možné vetvenia: ⟨12211111⟩; ⟨12211112⟩.
$(13)_{15}$: `X15/jan, Y15/X`

⟨12211111⟩ `syn(X,jan).`
$(1)_{16}$: `X/michal`

⟨122111111⟩ □
`X=michal ->`; návrat do ⟨1221111⟩

$\longmapsto$ $(14)_{17}$: `X17/jan, Y17/X`

⟨12211112⟩ `dcera(X,jan).`
$(7)_{18}$: `X/edita`

⟨122111121⟩ □
`X=edita ->`; návrat do ⟨1221⟩

$\longmapsto$ $(14)_{19}$: `X19/adam, Y19/Q1`

⟨12212⟩ `dcera(Q1,adam), eva\=Q1,rodic(Q1,X).`
$(9)_{20}$: `Q1/eva`

⟨122121⟩ `eva\=eva,rodic(eva,X).`
$(\,)_{21}$ fail; návrat do ⟨122⟩

$\longmapsto$ $(10)_{22}$: `S1/bozena`

⟨1222⟩ `rodic(bozena,Q1),eva\=Q1,rodic(Q1,X).`
Možné vetvenia: ⟨12221⟩; ⟨12222⟩.
$(13)_{23}$: `X23/bozena, Y23/Q1`

⟨12221⟩ `syn(Q1,bozena),eva\=Q1,rodic(Q1,X).`
$(4)_{24}$: `Q1/jan`

⟨122211⟩ `eva\=jan,rodic(jan,X).`
$(\,)_{25}$

⟨1222111⟩ `rodic(jan,X).`
Možné vetvenia: ⟨12221111⟩; ⟨12221112⟩.
$(13)_{26}$: `X26/jan, Y26/X`

⟨12221111⟩ `syn(X,jan).`
$(1)_{27}$: `X/michal`

⟨122211111⟩ □
`X=michal ->;` návrat do ⟨1222111⟩
⟼ $(1)_{27}$: `X/michal`
⟨12221112⟩ `dcera(X,jan).`
$(7)_{28}$: `X/edita`
⟨122211121⟩ □
`X=edita ->;` návrat do ⟨1222⟩
⟼ $(14)_{29}$: $\texttt{X}_{29}$`/bozena,` $\texttt{Y}_{29}$`/`$\texttt{Q}_1$
⟨12222⟩ $\texttt{dcera(Q}_1\texttt{,bozena),eva\=Q}_1\texttt{,rodic(Q}_1\texttt{,X).}$
$(10)_{23}$: $\texttt{Q}_1$`/eva`
⟨122221⟩ $\texttt{eva\=eva,rodic(Q}_1\texttt{,X).}$
$(\,)_{24}$ fail; návrat do ⟨⟩ □

2. Syntax jazyka Arity/Prolog

Jazyk Arity/Prolog pozostáva z celého radu základných typov dát, ktoré je možné kombinovať do zložitejších typov. Všetky tieto typy dát tvoria základ štruktúry programovacieho jazyka Arity/Prolog a spoločne sa nazývajú termy. Prirodzeným spôsobom sa delia na tieto podtypy: *premenné* (variables), *atomické konštantné termy* skrátene *konštanty* (atomics) a *zložené termy* (nonatomics).

Názvy premenných a konštánt v Prologu sú postupnosti znakov. Preto je užitočné vedieť, aké znaky Prolog pozná. Sú dva druhy znakov: viditeľné a neviditeľné. Neviditeľné znaky sa síce na termináli nezobrazia, ale spôsobia nejakú činnosť, napríklad vytvoria medzeru, alebo spôsobia posun na nový riadok. Viditeľné znaky sú rozdelené do štyroch kategórií: veľké písmená, malé písmená, číslice a špeciálne symboly.

```
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
! " # $ % & ' ( ) * + , - . / : ; < = > ? @ \ [ ] ^ _ ` { | } ~
```

Prolog pracuje so znakmi prostredníctvom ich ASCII kódov, t.j. s číslami od 0 do 127. Viditeľné znaky majú kódy od 33 do 126.

2.1. Premenné. Premenná je ľubovoľná postupnosť alfanumerických znakov a symbolov podčiarknutia _, ktorá začína veľkým písmenom alebo symbolom _. Postupnosti znakov

`Bratislava` `X` `_` `_0039` `_a_` `A4`

sú premenné. Premenná `_` sa nazýva *anonymná premenná*. Napríklad v terme `pozri(_,_,C,_,E)` sú dve premenné `C`, `E` explicitne pomenované a tri navzájom rôzne anonymné premenné.

V Arity/Prologu zabudovaný predikát `var(X)` uspeje, ak `X` je neinštancializovaná premenná, a `nonvar(X)` uspeje, ak `X` je inštancializovaná premenná, t.j. premenná `X` je unifikovaná s nejakým termom, ktorý nie je premennou. Premennú možno unifikovať s ľubovoľným termom. Ak sa nejaká premenná unifikuje s inou premennou

a následne sa jedna z premenných inštancializuje, to isté sa stane aj s druhou premennou. Napríklad nasledujúca konjunkcia cieľov uspeje:

```
var(X),X=3,nonvar(X).
```

2.2. Konštanty. Arity/Prolog má nasledujúce typy konštantných atomických termov: (1) *atómy* (atoms), (2) *reťazce* (strings), (3) *celé čísla* (integers), (4) *reálne čísla* (floating-point numbers), (5) *databázové referenčné čísla*.

2.2.1. *Atómy.* Atómy sú textové konštanty, ktoré môžu obsahovať najviac 255 znakov. Môžeme ich považovať tiež za 0-*árne funktory* (alebo 0-árne názvy predikátov). Uvažujeme 3 skupiny atómov:

- Postupnosť alfanumerických znakov a symbolu podčiarknutia `_`, ktorá začína malým písmenom. Napríklad: `otec`, `otec2`, `max_min`.
- Postupnosť znakov uzavretých dvojicou pravých apostrofov, pričom prípadný výskyt apostrofu vo vnútri postupnosti sa musí napísať dvakrát. Napríklad: `'Jang-c''-tiang'`, `'23'`, `'Bratislava'`.
- Postupnosť špeciálnych symbolov. Napríklad: `?-`, `:-`, `>=`, `=\=`, `[]`, ... Takáto postupnosť nesmie obsahovať napríklad symboly `% $ ' ( )`, ktoré sú výnimočné (`%` je znak pre komentár a ostatné majú význam zátvoriek).

Dva atómy sa unifikujú, ak pozostávajú z tých istých symbolov. Teda napríklad `'otec'` sa unifikuje s `otec`. Atómy sú v Arity/Prologu klasifikované zabudovaným predikátom `atom(X)`.

2.2.2. *Reťazce.* Reťazce sú textové konštanty určené na manipuláciu s textom a ich veľkosť nepresahuje 4 kilobajty. Sú to postupnosti ASCII znakov uzavretých dvojicou symbolov `$`, pričom prípadný výskyt symbolu `$` v texte sa musí napísať dvakrát. Príklady reťazcov sú nasledujúce postupnosti:

```
$$
$Dopravná situácia je hrozivá.$
$Cena cestovného lístka je $$5 a bude ešte vyššia.$
```

Reťazce a atómy sa neunifikujú, ani keď majú zhodný text. Reťazce sú v Arity/Prologu klasifikované zabudovaným predikátom `string(X)`.

2.2.3. *Celé čísla.* Celé čísla sú klasifikované predikátom `integer(X)`. Ich rozsah je závislý na implementácii. V Arity/Prologu hodnota celého čísla sa nachádza v intervale

$$\langle -2^{31}, 2^{31} - 1\rangle = \langle -2147483648, 2147483647\rangle.$$

S celými číslami možno pracovať aj prostredníctvom ASCII kódov. Napríklad `0'a` a `97` sú identické atomické termy.

2.2.4. *Reálne čísla.* Reálne čísla sú klasifikované predikátom `float(X)`. Ich rozsah je závislý na implementácii. V Arity/Prologu hodnota reálneho čísla sa nachádza v množine

$$\langle -9.99999999 \cdot 10^{99}, -1.0 \cdot 10^{-99}\rangle \cup \{0.0\} \cup \langle 1.0 \cdot 10^{-99}, 9.99999999 \cdot 10^{99}\rangle$$

s presnosťou na 8 desatinných miest a s exponentom v intervale $\langle -99, 99\rangle$. Napríklad `0.5`, `55.3`, `-83.0E21`, `122e-25` sú legálne čísla. Naproti tomu termy `5.`, `.5` nie sú prípustné.

2.2.5. *Databázové referenčné čísla.* Každému údaju v danej databáze je priradené jediné databázové referenčné číslo. Je tvorené symbolom `~` a ôsmimi hexadecimálnymi ciframi. Napríklad `~0025A015`, `~0035D180` sú databázové referenčné čísla. Sú klasifikované predikátom `ref(X)`.

2.3. Termy. Premenné a konštantné atomické termy sú *atomické termy*. *Zložený term* sa skladá z *funktora* a zoznamu argumentov uvedených v zátvorkách za funktorom. V úlohe funktora môže byť ľubovoľný atóm. Argumentom termu je opäť term. Medzi funktorom a ľavou zátvorkou uvádzajúcou argumenty termu nesmie byť medzera.

Termy sa unifikujú s inými termami, ak sa funktory zhodujú a ak odpovedajúce argumenty sa unifikujú. Napríklad term `sum(2+3)` sa neunifikuje s termom `sum(5)`, ale sa unifikuje s termom `sum(X+Y)`, pričom premenná `X` sa inštancializuje hodnotou `2` a premenná `Y` sa inštancializuje hodnotou `3`. Ak naozaj chceme vypočítať súčet musíme použiť predikát `is`. V otázke `?-X is 2+3.` premenná `X` sa inštancializuje na hodnotu `5` (pozri odsek 2.3.2).

2.3.1. *Funktory.* V Prologu sú termy väčšinou reprezentované prefixnou Poľskou formou, t.j. najskôr sa uvedie funktor a za ním (v zátvorkách) nasledujú argumenty. Samozrejme, že sú výnimky. Napríklad `:-/2` je infixný operátor v pravidle

```
otec(X,Y):-muz(X),rodic(X,Y).
```

a aj operátory `+/2`, `*/2` sú infixné v termoch `3+2`, `3*2`. Arity/Prolog má zabudovaný predikát `op(Priorita,Pozícia,Meno)`, ktorý umožňuje rozhodnúť sa pre ktorúkoľvek formu zápisu. Prvý argument `Priorita` je číslo od 50 do 1200 a určuje prednosť operátora pred inými operátormi v prípade, že sa poradie operátorov neurčí zátvorkami. Čím menšie číslo, tým vyššia priorita operátora. Druhý argument určuje umiestnenie operátora voči operandom. Jeho hodnotou môžu byť nasledujúce atómy (písmeno `f` v nich označuje operátor, písmená `x`, `y` označujú operandy):

- `fx` Unárna prefixná forma.
- `fy` Unárna prefixná forma. Funktor je možné reťaziť. Napríklad `not not X` je to isté ako `not(not X)`.
- `xf` Unárna postfixná forma.
- `yf` Unárna postfixná forma. `X f f` je to isté ako `(X f)f`.
- `xfx` Binárna infixná forma, pri ktorej sa predpokladá, že daný funktor sa nebude reťaziť (tak ako funktor `:-`, pri ktorom nemá zmysel písať `(X:-Y):-Z` a ani `X:-(Y:-Z)`).
- `xfy` Binárna infixná forma s prednosťou vpravo (tak ako operátor `=`, pre ktorý `a=b=c` je to isté ako `a=(b=c)`).
- `yfx` Infixná forma s prednosťou vľavo (tak ako operátor `/`, pre ktorý `a/b/c` je to isté ako `(a/b)/c`).

Hodnotou posledného argumentu `Meno` je názov operátora, ktorým môže byť ľubovoľný atóm.

Konkrétny predikát môže mať viac deklarácií. Napríklad operátory `+` a `-` sú tak unárne predikáty ako aj binárne. Pre obidva sú dovolené zápisy `yfx` a `fx`. Uvedomme si ešte, že symbol `-` môže byť súčasťou záporného čísla a preto na rozdiel od symbolu `+` sa zdanlivo môže používať ako operátor typu `fy`. Rozdiel sa prejaví v tom, že zatiaľ čo zápisy `-(-3)`, `- -3` sú zhodné, pod `+ +3` sa rozumie `'+'('+',3)`.

Niekedy sa môže stať, že argument predikátu obsahuje operátor s prioritou vyššou ako 1000. V takom prípade sa argument musí uzavrieť do zátvoriek. Napríklad predikát `assertz` môže obsahovať takýto argument:

```
assertz((otec(X,Y):-muz(X),rodic(X,Y)))
```

Arity/Prolog má tieto zabudované operátory s prioritou vyššou ako 1000:

```
:- --> ?- ;
```

Prioritu operátora ako aj formu zápisu jeho argumentov je možné získať pomocou predikátu `current_op/3` s rovnakým významom jednotlivých argumentov ako má predikát `op`. Napríklad na otázku „`?-current_op(X,Y,(:-)).`“ dostaneme odpoveď `X=1200`, `Y=xfx`. Deklarácie operátorov v Arity/Prologu zhŕňa nasledujúca tabuľka.

```
200 yfx  ..              670 yfx  and
300  fy  \ & *           675 yfx  or
300 xfy  ^               700 xfx  is @>= @> @=< @< \== \= == =..
400 yfx  mod // / *      700 xfy  =
500 xfy  :               800 xfy  ->
500 yfx  - +             900  fx  case
500  fx  - +             900  fy  nospy not spy \+
550 yfx  << >>          1000 xfy  ,
600 xfx  >= > =< <      1100 xfy  ;
650 xfx  =:= =\=        1150  fy  extrn mode module public visible
655 yfx  /\             1200 xfx  --> :-
660 yfx  \+/            1200  fx  :- ?-
665 yfx  \/
```

Arity/Prolog transformuje všetky termy do prefixnej formy a napríklad termy `2+3` a `2+3^5` sú v poradí zhodné s termami `'+'(2,3)` a `'+'(2,'^'(3,5))`.

2.3.2. *Aritmetické výrazy.* Aritmetické operácie umožňujú numerické výpočty. Použitie aritmetického výrazu v argumente nejakého predikátu nemusí znamenať jeho vyhodnotenie. Len niekoľko predikátov spôsobí vyhodnotenie aritmetického výrazu. Predikáty nasledujúceho zoznamu vyhodnocujú aritmetické výrazy (`A1` a `A2` označujú aritmetické výrazy, `X` a `Y` označujú premenné, `N` je celé číslo):

`X is A1` Vyhodnotí sa aritmetický výraz `A1` a premenná `X` sa unifikuje s touto hodnotou.

`inc(X,Y)` Rovnaké ako cieľ „`Y is X+1`“.

`dec(X,Y)` Rovnaké ako cieľ „`Y is X-1`“.

`A1>A2` Tento cieľ uspeje, ak hodnota výrazu `A1` je väčšia ako hodnota výrazu `A2`.

`A1<A2` Tento cieľ uspeje, ak hodnota výrazu `A1` je menšia ako hodnota výrazu `A2`.

`A1>=A2` Tento cieľ uspeje, ak hodnota výrazu `A1` je väčšia alebo rovná hodnote výrazu `A2`.

`A1=<A2` Tento cieľ uspeje, ak hodnota výrazu `A1` je menšia alebo rovná hodnote výrazu `A2`.

`A1=:=A2` Tento cieľ uspeje, ak hodnoty výrazov `A1` a `A2` sa zhodujú.

`A1=\=A2` Tento cieľ uspeje, ak hodnoty výrazov `A1` a `A2` sa nezhodujú.

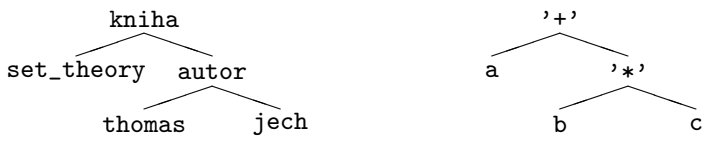
`randomize(N)` Tento predikát prehodnotí generátor náhodných čísiel a hodnota 0-árneho funktora `random` závisí od hodnoty celého čísla `N`.

2.3.3. *Aritmetické operátory v Arity/Prologu.* Nasledujúce operátory sa môžu vyskytovať v aritmetických výrazoch:

`random`	Obsahuje náhodne vygenerované reálne číslo z intervalu $[0, 1]$, ktoré závisí na argumente predikátu `randomize(N)`.
`pi`	Konštanta π s hodnotou na 8 desatinných miest.
`X+Y`	Súčet.
`X-Y`	Rozdiel.
`X*Y`	Násobenie.
`X/Y`	Delenie s výsledkom reálne číslo.
`X//Y`	Celočíselné delenie.
`X^Y`	Umocňovanie.
`-X`	Unárna operácia mínus.
`X/\Y`	Konjunkcia po bitoch (konjunkcia číslic v dvojkovom zápise).
`X\/Y`	Disjunkcia po bitoch.
`\(X)`	Negácia po bitoch.
`X<<Y`	Posun o `Y` bitov doľava (výsledok je $x \cdot 2^y$).
`X>>Y`	Posun o `Y` bitov doprava (výsledok je $[x/2^y]$).
`X mod Y`	Zvyšok pri delení `X` číslom `Y`.
`abs(X)`	Absolútna hodnota.
`acos(X)`	Arkus kosínus.
`asin(X)`	Arkus sínus.
`atan(X)`	Arkus tangens.
`cos(X)`	Kosínus.
`exp(X)`	e^x
`ln(X)`	Prirodzený logaritmus.
`log(X)`	Dekadický logaritmus.
`sin(X)`	Sínus.
`sqrt(X)`	Druhá odmocnina.
`tan(X)`	Tangens.
`round(X,N)`	Zaokrúhlenie reálneho čísla na `N` miest.

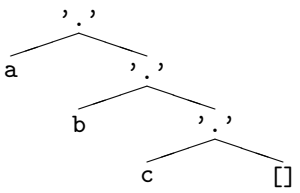
Ak v aritmetickom výraze sú reálne čísla aj celé čísla, výsledok bude reálne číslo. Predikát `float(X)` robí konverziu celého čísla na reálne ešte pred výpočtom. Podobne predikát `integer(X)` z reálneho čísla odsekne desatinnú časť a výsledkom je celé číslo. Keď sa pri výpočte vyskytne chyba (napríklad pretečenie pamäti), výsledkom je atóm `err`. Akékoľvek aritmetické porovnávanie s týmto atómom zlyhá.

2.3.4. *Zoznamy.* Termy je užitočné predstaviť si ako stromy. Každý funktor termu predstavuje vrchol stromu a argumenty tohto funktora predstavujú vetvenie vo vrchole. Napríklad termy `kniha(set_theory,autor(thomas,jech))` a `a+b*c` si môžeme predstaviť ako tieto stromy:



Zoznam je špecifickým typom termu v jazyku Prolog. Zoznamy sú rekurzívne vytvorené z atómu `[]`, ktorý reprezentuje prázdny zoznam, pomocou binárneho

funktora `'.'`. Zoznam pozostávajúci z atómov `a`, `b` a `c` si môžeme predstaviť pomocou tohto stromu



Arity/Prolog má zabudovaných niekoľko spôsobov (viac alebo menej pohodlných, ale každý nejako užitočný), ako zoznam zapísať.

(1) Zápis v tvare zoznamu: `[]` je prázdny zoznam, `[a,b,c]` je zoznam prvkov `a`, `b`, `c`.
(2) Funktorový zápis: Zoznam má hlavu a chvost, ktoré sú spojené prostredníctvom funktora `'.'/2`. Napríklad `[a]` možno napísať ako `'.'(a,[])` a `[a,b,c]` je to isté ako `'.'(a,'.'(b,'.'(c,[])))`.
(3) Zápis v tvare hlava-chvost: Zoznam `[a,b,c,d]` sa reprezentuje výrazom `[a|[b,c,d]]` (ale aj `[a,b|[c,d]]`, `[a,b,c|[d]]`).

Číselné zoznamy je možné zapísať aj postupnosťou ASCII znakov reprezentujúcich ich ASCII kódy. Nasledujúce zápisy sú ekvivalentné:

```
"abc"      [97,98,99]      '.'(97,'.'(98,'.'(99,[])))
```

sú ekvivalentné. Konverziu medzi atómami a zoznamami ich ASCII kódov sprostredkuje predikát `name(Atom,Zoznam)`. Binárny predikát `pred(Atom1,Atom2)` definovaný nasledujúcim programom porovnáva atómy v lexikografickom usporiadaní.

```
pred(X,Y):-name(X,A),name(Y,B),mensi(A,B).
mensi([],[_]).
mensi([X|_],[Y|_]):-X<Y.
mensi([H|X],[H|Y]):-mensi(X,Y).
```

Prvkami zoznamu môžu byť ľubovoľné termy a teda aj premenné a zoznamy. Teda zoznam `[X,set_theory,[thomas,jech]]` je v Prologu legálny. Unifikácia zoznamov prebieha presne podľa schémy unifikácie zložených termov. To znamená, že dva zoznamy sa unifikujú, ak sa unifikujú odpovedajúce členy zoznamov.

Prolog prirodzeným spôsobom, použitím rekurzie, umožňuje spracovávať informácie uložené v zoznamoch. Príkladom takejto rekurzie je táto všeobecná schéma:

```
sprac_zoznamu([],R):-e(R).
sprac_zoznamu([X|T],R):-sprac_zoznamu(T,R1),f(X,R1,R).
```

ktorá vhodným výberom predikátov `e/1` a `f/3` umožňuje urobiť nejakú operáciu na prvkoch zoznamu. Uvedieme štyri príklady použitia tejto schémy.

Príklad 2.3.1. a) *Súčet prvkov zoznamu:*

```
sz1([],0).
sz1([H|T],S):-sz1(T,A),R is A+H.
```

b) *Priemerná hodnota prvkov zoznamu:*

```
priemer(L,P):-sz2(L,X),P is X.
sz2([],0/0).
sz2([H|T],R/N):-sz2(T,S/M),R is S+H,N is M+1.
```

c) *Usporiadanie prvkov zoznamu:*

```
sz3([],[]).
sz3([H|T],R):-sz3(T,S),f3(H,S,R).
f3(X,[],[X]).
f3(X,[Y|T],[X,Y|T]):-X=<Y.
f3(X,[Y|T],[Y|S]):-X>Y,f3(X,T,S).
```

Cieľ `f3(H,R1,R)` *sa splní, ak* `R1`*,* `R` *sú usporiadané zoznamy a* `R` *vznikol z* `R1` *vložením prvku* `H`*.*

d) *Generovanie všetkých permutácií daného zoznamu:*

```
sz4([],[]).
sz4([H|T],R):-sz4(T,R1),f4(H,R1,R).
f4(X,T,[X|T]).
f4(X,[H|T],[H|T1]):-f4(X,T,T1).
```

Cieľ `f4(H,R1,R)` *sa splní, ak* `R` *vznikne z* `R1` *vložením prvku* `H` *na ľubovoľné miesto, pri každom ďalšom spĺňaní na iné miesto.*

Zostrojte výpočtové stromy pre otázky

```
?-sz1([1,3,2],X).
?-priemer([1,3,2],X).
?-sz3([1,3,2],X).
?-sz4([1,2,3],X).
```

2.3.5. *Usporiadanie termov.* Arity/Prolog používa nasledujúce usporiadanie termov od najnižších k najvyšším:

- premenné (navzájom ekvivalentné),
- celé čísla a reálne čísla od $-\infty$ do $+\infty$,
- atómy a reťazce usporiadané lexikograficky,
- databázové referenčné čísla (navzájom ekvivalentné),
- zložené termy usporiadané najskôr podľa arity, potom podľa názvu hlavného funktora, a ďalej rekurzívne podľa argumentov.

Napríklad `A -9 9 zoo A=B a(0,2)` sú usporiadané v rastúcom poradí, `1+2` je pred `2+1`, a reťazec `$zoo$` je v tomto usporiadaní ekvivalentný s atómom `zoo`. Predikáty, ktoré testujú toto usporiadanie:

`T1==T2` Určí, či termy `T1` a `T2` sú ekvivalentné.

`T1\==T2` Určí, či termy `T1` a `T2` nie sú ekvivalentné.

`T1@<T2` Overí, či term `T1` je pred termom `T2`.

`T1@>T2` Overí, či term `T1` je za termom `T2`

`T1@=<T2` Overí, či term `T1` je pred alebo ekvivalentný s termom `T2`

`T1@>=T2` Overí, či term `T1` je za alebo ekvivalentný s termom `T2`

`eq(T1,T2)` Určí či termy `T1` a `T2` sú rovnaké a s totožným umiestnením v pamäti. Napríklad otázka `?-s(a)==s(a).` je úspešná, otázka `?-eq(s(a),s(a)).` nie je úspešná, ale `?-X=s(a),eq(X,X).` je úspešná otázka.

`compare(C,T1,T2)` Porovnáva termy `T1` a `T2` ak `C` je `=`, `<`, `>`. Ak premenná `C` nie je inštancializovaná, potom odpoveďou pre `C` je jeden z týchto symbolov.

`sort(L,X)` Usporiada zoznam `L` a výsledkom je zoznam `X`.

`keysort(L,X)` Usporiada zoznam `L` pozostávajúci z termov tvaru symbol-číslo. Napríklad otázka `?-keysort([c-1,a-4,d-4],X).` je zodpovedaná odpoveďou `X=[a-4,c-1,d-4]`.

2.3.6. *Konverzia termov.* Prístup k jednotlivým položkám termu prípadne konverziu termu na zoznam obsahujúci názov funktora a jeho argumenty prípadne naopak možno dosiahnuť nasledujúcimi príkazmi.

`functor(Term,Meno,Arita)` Z termu získa názov funktora a jeho aritu a naopak z názvu a arity vytvorí term. Napríklad otázka `?-functor(strom(buk,semeno(zalud)),X,Y),functor(Z,kniha,3).` dostane odpoveď `X=strom, Y=2, Z=kniha(A,B,C)`.

`arg(N,Term,Argument)` Výsledkom je hodnota n-tého argumentu termu `T`. Premenné `N` a `Term` musia byť inštancializované. Napríklad otázka `?-arg(2,rodic(adam,fero),X).` dostane odpoveď `X=fero`.

`arg0(N,Term,Argument)` má podobný význam ako `arg(N,Term,Argument)` s jediným rozdielom, že argumenty sa číslujú od 0.

`Term =.. Zoznam` Tento predikát, nazýva sa *univ*, dokáže získať všetky argumenty termu naraz. Napríklad otázka `?-strom(dub,semeno(zalud))=..X.` dostane odpoveď `X=[strom,dub,semeno(zalud)]`.

`argrep(Term,N,Argument,Novyterm)` Vymení argument v terme novým argumentom a výsledkom je nový term. Všetky argumenty tohto predikátu okrem premennej `Novyterm` musia byť inštancializované. Otázka `?-argrep(rodic(adam,jano),1,fero,X).` dostane odpoveď `X=rodic(fero,jano).`

`name(Atom,Zoznam)` Zmení daný atóm na zoznam ASCII kódov jednotlivých znakov daného atómu a naopak zoznam ASCII kódov zmení na atóm. Otázka `?-name(hallo,"hallo").` je úspešná.

`length(Zoznam,N)` Hodnotou premennej `N` je dĺžka daného zoznamu.

2.4. Štandardné predikáty pre vstup a výstup. Program nejakým spôsobom musí získavať informácie. Prijíma ich buď z klávesnice alebo ich prečíta z nejakého súboru. Výstup je spôsob, akým program oznámi vypočítané údaje. Stane sa to buď pomocou monitoru, tlačiarne alebo nejakého výstupného súboru. Predmetom vstupu a výstupu môžu byť prologovské termy, ASCII znaky a reťazce. Pre každý z týchto troch typov objektov má Prolog určené iné vstupné a výstupné predikáty. Všetky tieto predikáty nie sú viacnásobne splniteľné a teda pri backtracku sa nespĺňajú ešte raz.

2.4.1. *Predikáty pre vstup a výstup termov.* Tieto predikáty môžu čítať a písať premenné, atómy, reťazce, klauzuly a vo všeobecnosti ľubovoľné termy v štandardnom formáte, pričom term napísaný z klávesnice musí byť ukončený bodkou a klávesou ⟨Return⟩ (tie sa neprečítajú ako súčasť termu).

`read(X)`, `write(X)`, `writeq(Term)` Tieto predikáty čítajú a píšu termy berúc do úvahy operátorové definície pomocou predikátu `op(X,Y,Z)`. Predikát `writeq` na rozdiel od `write`, ak je to nutné, zapisuje atómy a funktory v apostrofoch v tvare, ktorý je čitateľný pre `read`. Pre porovnanie, otázka `?-write('...'(a,b)).` vypíše `...(a,b)` a `?-writeq('...'(a,b)).` vypíše `'...'(a,b)`.

`display(Term)` Zapisuje termy používajúc prefixnú formu zápisu operátorov bez ohľadu na operátorové definície. Napríklad, otázka `?-write(a+b*c).` vypíše `a + b * c` a `?-display(a+b*c).` vypíše `'+'(a,'*'(b,c))`.

2.4.2. *Predikáty pre vstup a výstup ASCII znakov.*

`get0(X)`, `get(X)` Tieto predikáty prečítajú znak a ak `X` je neinštancializovaná premenná, hodnotou `X` bude tento znak. Ak `X` je inštancializovaná, prečítaný znak porovnajú s hodnotou `X`. Predikát `get` na rozdiel od `get0` si všíma len viditeľné znaky a neviditeľné znaky preskočí.

`put(X)` Napíše znak s ASCII kódom `X`.

`tab(N)` Napíše `N` medzier.

`nl` Napíše koniec riadku.

2.4.3. *Práca s reťazcami.* Z hľadiska užívateľa je ťažkopádne zadávať informácie prostredníctvom termov a spracovávanie zoznamov je príliš pomalé. Tretí spôsob je používanie reťazcov. Reťazce sú uložené v pamäti ako polia znakov a zaberajú menej miesta než keby tieto znaky boli uložené v zoznamoch. Arity/Prolog má mnoho predikátov, ktoré dovoľujú hľadanie, porovnávanie a spájanie slov v reťazcoch, konverziu reťazcov na termy, atómy, čísla, zoznamy znakov a späť.

2.5. Štandardný vstup a výstup do súboru.

`see(X)` Otvorí súbor s názvom `X` pre čítanie, ak už predtým nebol otvorený. Súbor sa uzavrie po použití predikátu `seen` alebo otvorením iného súboru pomocou predikátu `see`.

`seeing(X)` Ak je otvorený nejaký súbor pre čítanie, `X` sa inštancializuje názvom tohto súboru.

`seen` Zatvorí súbor otvorený príkazom `see(X)`.

`tell(X)` Otvorí súbor s názvom `X`. Keď súbor je otvorený pomocou `tell`, všetok výstup sa posiela do tohto súboru kým sa neuzavrie pomocou predikátu `told`, alebo sa neotvorí iný súbor predikátom `tell`

`telling(X)` Zistí názov súboru otvoreného pre výstup.

`told` Zatvorí momentálne otvorený súbor pre výstup. Nasledujúci výstup je na terminál.

3. Základné programovacie triky

Konvenčné programovacie jazyky používajú explicitné cykly a vetvenia a priebeh samotného výpočtu je tu možné znázorniť výpočtovým diagramom. V Prologu priebeh výpočtu ovplyvňuje usporiadanie faktov v programe a usporiadanie podcieľov v každej klauzule. Prolog sa pokúša splniť jeden podcieľ predtým, než začne spĺňať ďalší v poradí. Ak sa mu to nepodarí, vráti sa k predchádzajúcemu podcieľu, ktorý sa snaží splniť iným spôsobom a tak pokračuje ďalej. Tejto stratégii sa hovorí *depth-first search* a proces, ktorý umožňuje návrat späť, aby sa našlo vhodnejšie splnenie podcieľa sa nazýva *backtrack*.

Prolog umožňuje využívať rôzne programové techniky, ktoré ovplyvňujú tento proces (rekurzia, cyklus repeat-fail, rez, ...). Nasledujúci zoznam predikátov umožňuje priamo ovplyvňovať priebeh výpočtu.

`,` Určuje *konjunkciu* cieľov v klauzule.

`;` Určuje *disjunkciu* cieľov v klauzule. Rovnaký význam má predikát `|`.

`!` *Rez* (cut) je cieľ, ktorý je vždy úspešný, ale ak sa program dostane do tohto cieľa pri návrate späť (backtrack), potom predikát, ktorý je v hlave klauzuly s telom obsahujúcim tento rez sa nesplní.

`[!P!]` *Ústrižok* (snip). Ciele `P` obsiahnuté v ústrižku sa preskočia počas trvania backtracku v prípade, že do ústrižku sa dostane program pri návrate späť.

`call(P)` Ak hodnotou `P` je term, ktorý je možné interpretovať ako cieľ, potom `call(P)` uspeje práve vtedy, keď tento cieľ uspeje.

`not(P)` (alebo tiež `\+P`, `not P`) Ak hodnotou `P` je term, ktorý je možné interpretovať ako cieľ, potom `not(P)` uspeje práve vtedy, keď tento cieľ je neúspešný pozri Príklad 3.3.3.

`P and Q` Tento cieľ je úspešný, ak oba ciele `P` a `Q` sú súčasne úspešné.

`P or Q` Tento cieľ je úspešný, ak aspoň jeden z cieľov `P` a `Q` je úspešný. Ak je úspešný cieľ `P`, cieľ `Q` sa neskúša splniť (na rozdiel od predikátu `;`).

`true` Tento cieľ je vždy úspešný.

`fail` Tento cieľ je vždy neúspešný (v prípade potreby vyvolá backtrack).

`repeat` Tento cieľ je vždy úspešný a pri backtracku sa opätovne splní (pozri časť 3.2).

`ifthen(P,Q)` Ak cieľ `P` sa úspešne splní, tak nasleduje spĺňanie cieľa `Q`. Ak cieľ `P` sa nesplní, cieľ `Q` sa nebude spĺňať.

`ifthenelse(P,Q,R)` Ak cieľ `P` sa úspešne splní, tak nasleduje spĺňanie cieľa `Q`. V opačnom prípade nasleduje spĺňanie cieľa `R`.

`case([`P_1`->`Q_1`,`P_2`->`Q_2`,...,`P_n`->`Q_n`|R])` Ak i je prvý index taký, že P_i sa splní, potom sa začne spĺňať cieľ Q_i. Ak také i nie je, tak sa začne spĺňať cieľ `R`.

`case([`P_1`->`Q_1`,`P_2`->`Q_2`,...,`P_n`->`Q_n`])` Tento cieľ má rovnaký význam ako cieľ `case([`P_1`->`Q_1`,`P_2`->`Q_2`,...,`P_n`->`Q_n`|true])`

`abort` Simuluje stlačenie kláves ⟨Ctrl⟩-c.

`break` Simuluje stlačenie kláves ⟨Ctrl⟩-z.

3.1. Rekurzia. Príkladmi rekurzie sú nasledujúce definície predikátov `member/2` a `predok/2`. `member(X,Y)` sa splní, ak `X` je členom zoznamu `Y` a `predok(X,Y)` sa splní, ak človek `X` je priamym predkom človeka `Y`.

```
member(X,[X|_]).
member(X,[_|Y]):-member(X,Y).
predok(X,Y):-rodic(X,Y).
predok(X,Y):-rodic(X,Z),predok(Z,Y).
```

Príklad 3.1.1.

(1) *Definujme predikát* `gen_rozdiel(Predok,Potomok,Rozdiel)` *pre určenie generačného rozdielu predka a jeho priameho potomka.*

(2) *Predpokladajme, že v databáze je uložený popis orientovaného grafu pomocou predikátu* `hrana(X,Y)`. *Definujme predikát* `cesta(X,Y)`, *ktorý zisťuje, či existuje cesta z* `X` *do* `Y`.

Riešenie.

```
gen_rozdiel(X,Y,1):-rodic(X,Y).
gen_rozdiel(X,Y,N):-rodic(X,Z),gen_rozdiel(Z,Y,M),N is M+1.

cesta(X,X).
cesta(X,Y):-hrana(X,Z),cesta(Z,Y).
```

□

Príklad 3.1.2. *Definujme predikát* `sum(X)`*, ktorý opakovane vyzýva o nové číslo z klávesnice až kým neprečíta nečíselný údaj tak, že odpoveď na otázku* `?-sum(X).` *je* `X=`*s, kde s je súčet zadaných čísiel.*

Riešenie.

```
sum(X):-write(=>),read(Y),integer(Y),sum(A),X is A+Y,!.
sum(0).
```

Predikát `sum` rekurzívne volá samého seba, zakaždým s novou premennou. Aby sa po vynorení z rekurzie nespustil backtrack s následným generovaním nesprávnych odpovedí, je potrebné ukončiť rekurziu rezom. V ďalších príkladoch potrebu takého rezu už nebudeme komentovať. □

3.2. Cyklus repeat-fail. Predikát `repeat/0` je v Prologu už definovaný. Je definovaný nasledujúcou rekurziou.

```
repeat.
repeat:-repeat.
```

Tento predikát sa používa v spojení s inými cieľmi, ktoré obyčajne nevyvolajú backtrack, za účelom generovať alternatívne riešenia, kým sa nesplní určitá podmienka. Napríklad nasledujúci zložený cieľ

```
repeat,read(X),write('napíš stop: '),X=stop.
```

preruší beh programu (to spôsobí cieľ `read(X)`) a vyžiada si odpoveď z klávesnice. Po prečítaní odpovede, túto požiadavku opakuje, kým odpoveďou nebude atóm `stop` (ukončený bodkou a klávesou ⟨Return⟩).

Podobný efekt ako predikát `repeat` možno dosiahnuť s ktorýmkoľvek predikátom, ktorý je viacnásobne splniteľný.

3.3. Kombinácia cut-fail. Kombinácia rezu a príkazu `fail` sa v Prologu používa na vylúčenie výnimiek nejakého všeobecného pravidla. Príkaz `fail` vyvolá backtrack a ak tomuto cieľ predchádza rez, definovaný predikát sa nesplní. Napríklad voličom je každý, kto dovŕši vek 18 rokov a je občanom štátu. V tomto pravidle sú výnimky. Napríklad, odsúdenci nemajú právo voliť, hoci spĺňajú tieto dve podmienky.

Predpokladajme, že máme definované predikáty `obcan(Obcan)`, `vek(Meno,Vek)`, `odsudenec(Obcan)`. Predikát `volic/1` určujúci všetkých voličov môžeme definovať týmito klauzulami:

```
volic(X):-odsudenec(X),!,fail.
volic(X):-obcan(X),vek(X,Y),Y>=18.
```

Dôležité je umiestniť klauzuly definujúce výnimky pred všeobecnými pravidlami.

Príklad 3.3.1. *Definujme predikát* `rozklad(X)`*, ktorý pre kladné celé číslo* `X` *vypíše všetky jeho rozklady na súčet nezáporných celých čísiel.*

Riešenie.

```
rozklad(X):-X>0,r(X,X).
r(0,X):-write(0+X=X),!,fail.
r(A,X):-B is X-A,write(A+B=X),nl,C is A-1,r(C,X).
```

□

Príklad 3.3.2. *Definujme predikát* `klasifikuj(X)`*, ktorý oznámi typ termu v argumente.*

Riešenie.

```
klasifikuj(X):-var(X),!,
    write('Premenná.'),nl.
klasifikuj(X):-atomic(X),!,
    write('Konštanta: '),typ_termu(X).
klasifikuj([_|_]):-!,
    write('Zložený term. Zoznam.'),nl.
klasifikuj(X):-write('Zložený term.'),nl.

typ_termu([]):-!,
    atom([]),write('Atóm (prázdny list).'),nl.
typ_termu(X):-atom(X),!,
    write('Atóm.'),nl.
typ_termu(X):-integer(X),!,
    write('Celé číslo.'),nl.
typ_termu(X):-float(X),!,
    write('Reálne číslo.'),nl.
typ_termu(X):-string(X),!,
    write('Reťazec.'),nl.
typ_termu(X):-ref(X),!,
    write('Databázové referenčné číslo.'),nl.
```

□

Príklad 3.3.3. *Definujme predikát* `neg(X)`*, ktorý nahradí negáciu, t.j. predikát* `not(X)`*.*

Riešenie.

```
:-op(900,fy,neg).
neg X:-X,!,fail.
neg X:-true.
```

□

Príklad 3.3.4. *Predpokladajme, že databáza obsahuje predikát* `vek(Meno,Vek)`*.*

(1) *Definujme predikát* `vyber(Meno,Min,Max)`*, ktorý bude generovať mená ľudí vo veku medzi* `Min` *a* `Max`*.*

(2) *Definujme predikát* `vyber2(Min,Max)`*, ktorý vypíše pomocou* `write` *mená ľudí z úlohy* (1)*.*

Riešenie.

```
vyber(X,Y,Z):-Y>Z,!,fail.
vyber(X,Y,Z):-vek(X,V),Y=<V,V=<Z.

vyber2(X,Y):-X>Y,!,fail.
vyber2(X,Y):-vek(M,X),write(M),nl,fail.
vyber2(X,Y):-Z is X+1,vyber2(Z,Y).
```

□

3.4. Použitie ústrižku. Predpokladajme, že predikát `pobrezie(Ocean,Stat)` určuje pobrežia všetkých štátov a oceánov a máme predikáty identifikujúce všetky africké a americké štáty `africky(Stat)` a `americky(Stat)`. Chceme definovať predikát na určenie tých oceánov, ktoré majú pobrežie súčasne s nejakým africkým a nejakým americkým štátom. Možné riešenie dáva táto definícia:

```
riesenie(X):-pobrezie(X,Y),americky(Y),
             pobrezie(X,Z),africky(Z).
```

Ale oveľa efektívnejšia je nasledujúca definícia, v ktorej sa využíva úsudok, že ak oceán `X` nemá pobrežie so žiadnym africkým štátom, tak túto skutočnosť stačí overiť pre jeden americký štát `Y`.

```
riesenie(X):-[!pobrezie(X,Y),americky(Y)!],
             pobrezie(X,Z),africky(Z).
```

3.5. Cyklus for. Definujme predikát `for/3` (v Prologu nie je definovaný) nasledujúcou definíciou:

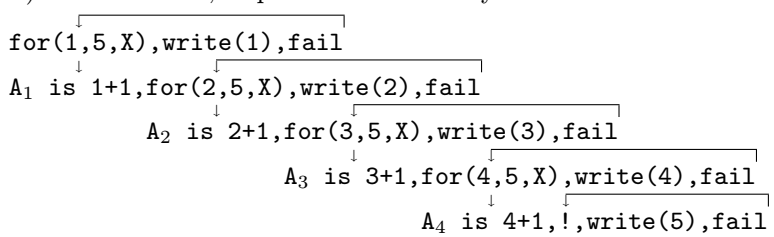
```
for(X,X,X):-!.
for(X,Y,X).
for(X,Y,Z):-A is X+1,for(A,Y,Z).
```

Kedykoľvek databáza obsahuje takto definovaný predikát `for`, ľahko dosiahneme účinok cyklu s riadiacou premennou. Napríklad nasledujúci zložený cieľ je cyklus s riadiacou premennou `X`, ktorým sa na termináli vypíšu čísla od `1` do `10`.

```
for(1,10,X),write(X),fail.
```

Príklad 3.5.1. *Vytvorme zoznam cieľov v poradí v akom sa budú spĺňať pri hľadaní odpovede na otázku* `?-for(1,5,X),write(X),fail.`

Riešenie. Nasledujúci diagram znázorňuje poradie spĺňania cieľov najskôr zľava doprava a potom v smere orientovaných čiar (lomená orientovaná čiara znázorňuje backtrack). Uvedomme si, že predikát `write` nevyvoláva backtrack.



□

Príklad 3.5.2. *Pomocou cyklu* `for` *definujme predikát* `rozklad2(X)` *s rovnakým významom ako má predikát v príklade* 3.3.1

Riešenie. Definícia predikátu `for` musí byť samozrejme prítomná v databáze s touto definíciou:

```
rozklad2(X):-for(0,X,A),B is X-A,write(A+B=X),nl,fail.
```

□

3.6. Cyklus s akumulátorom.

Príklad 3.6.1. *V databáze sú uložené fakty* `a(`n`)` *pre konečnú množinu prirodzených čísiel* n*. Definujme predikát* `sum2(X)`*, ktorý určí súčet všetkých argumentov predikátu* `a(X)`*, t.j. odpoveď na otázku* `?-sum2(X).` *je* `X=`s*, kde* s *je hľadaný súčet.*

Riešenie. Úlohu rozdelíme na dve časti. Najskôr definujeme predikát `generuj`, ktorý vytvorí zoznam všetkých argumentov predikátu `a(X)`. Toto dosiahneme pridaním jedného argumentu, v ktorom sa bude „akumulovať“ výsledný zoznam postupným pridávaním tých argumentov databázy, ktoré ešte nie sú v zozname. Pomocou

tohto zoznamu a pridaním definície predikátu `sz1` z príkladu 2.3.1 ľahko nájdeme súčet.

```
sum2(X):-generuj(Y),sz1(Y,X).
generuj(X):-g([],X).
g(L,X):-a(H),not_member(H,L),g([H|L],X),!.
g(X,X).
not_member(X,[H|T]):-X\=H,not_member(X,T).
not_member(X,[]).
```

Generovaný zoznam je v opačnom poradí než je poradie faktov v databáze. □

Namiesto predikátu `not_member` môžeme vziať tiež negáciu predikátu `member` definovanú v časti 3.1. Arity/Prolog má štandardné predikáty `bagof`, `findall`, `setof`, ktoré umožňujú pohodlne vytvárať zoznamy termov. Napríklad sme mohli jednoducho definovať:

```
generuj(X):-bagof(Y,a(Y),X).
```

3.7. Pridávanie a odoberanie klauzúl z databázy. Predikát `asserta` vloží klauzulu na začiatok databázy a `assertz` na koniec databázy. Naopak predikát `retract` odstráni klauzulu z databázy.

Ak v definícii predikátu `sum(X)` z príkladu 3.1.2 vymeníme cieľ `read(Y)` cieľom `a(Y)` nedostaneme riešenie príkladu 3.6.1, pretože v rekurzii cieľ `a(Y)` sa vždy bude unifikovať s prvým faktom v databáze. Tomu sa dá predísť jednoducho tak, že tieto fakty budeme z databázy po použití odstraňovať.

Príklad 3.7.1. *Zefektívnime predikát* `sum2(X)` *z príkladu* 3.6.1.

Riešenie.

```
sum3(X):-retract(a(Y)),sum3(A),X is A+Y,!.
sum3(0).
```

Predikát `sum3(X)` má možno jednu nevýhodu, že odstráni predikát `a(X)` z databázy. Ak chceme, aby tento predikát ostal v databáze zachovaný, treba ho tam po skončení rekurzie uložiť. Pri vynáraní sa z rekurzie tieto fakty budú zapisované v obrátenom poradí.

```
sum4(X):-retract(a(Y)),sum4(A),X is A+Y,asserta(a(Y)),!.
sum4(0).
```

□

Príklad 3.7.2.

(1) *Zjednodušme definíciu* `generuj(X)` *z príkladu* 3.6.1 *odoberaním faktov z databázy.*

(2) *Definujme predikát* `zoznam(Otazka,Term,Zoznam)`*, ktorý vytvorí zoznam všetkých inštancií daného termu inštancializovaného jednotlivými odpoveďami danej otázky (podobne ako predikát* `bagof`*).*

Riešenie. (1) Nasledujúci predikát generuje zoznam argumentov v rovnakom poradí ako sú fakty v databáze.

```
generuj2([H|T]):-retract(a(H)),generuj2(T),asserta(a(H)),!.
generuj2([]).
```

Pre vytvorenie zoznamu v opačnom poradí treba použiť ďalší argument a cyklus s akumulátorom.

```
generuj3(X):-g3([],X).
g3(T,X):-retract(a(H)),g3([H|T],X),!,asserta(a(H)).
g3(X,X).
```

(2) Najskôr pomocou predikátu `pridaj` sa vytvorí databáza s „novými“ faktami, ktorých argumentami sú inštancie daného termu úspešnými odpoveďami na danú otázku a potom pomocou predikátu `odober` sa tieto fakty odstránia z databázy s dodatočným akumulovaním argumentov do výsledného zoznamu:

```
zoznam(X,Y,Z):-pridaj(X,Y),odober(Z).
pridaj(X,Y):-call(Y),assertz('****'(X)),fail.
pridaj(X,Y).
odober([H|T]):-retract('****'(H)),odober(T),!.
odober([]).
```

□

4. Arity/Prolog interpreter

Štandardnou databázou pre Arity/Prolog interpreter je binárny súbor `api.idb`. Preto prv než začneme pracovať s Arity/Prolog interpreterom, treba sa presvedčiť, že táto databáza je prítomná v pracovnom adresári. Samotný interpreter sa inicializuje príkazom `api.exe`, pomocou ktorého sa ocitneme v hlavnej obrazovke (main window). V nej je prompt označený atómom `?-`. V najvyššom riadku hlavnej obrazovky je menu obsahujúce položky

```
File    Edit    Buffers    Info    Debug    Switch    Help
```

Je možné ich aktivovať v tom istom poradí stláčaním kláves

⟨Alt⟩-f ⟨Alt⟩-e ⟨Alt⟩-b ⟨Alt⟩-i ⟨Alt⟩-d ⟨Alt⟩-s ⟨Alt⟩-h

Význam niektorých kláves:

- ⟨Alt⟩ Prepínač medzi hlavnou obrazovkou a menu, alebo medzi editorom a menu.
- ⟨F8⟩ Prepínač medzi hlavnou obrazovkou a editorom (to isté ako ⟨Alt⟩-s).
- ⟨Esc⟩ Vypínač menu.
- ⟨F1⟩ Zobrazí Arity/Prolog help súbor (to isté ako ⟨Alt⟩-ha).
- ⟨F6⟩ Umožňuje voľbu jedného z deviatich bufferov editora (to isté ako ⟨Alt⟩-bg).
- ⟨F7⟩ Prepínač medzi naposledy použitým bufferom editora a hlavnou obrazovkou (to isté ako ⟨Alt⟩-bl).

Napríklad ⟨Alt⟩-fo umožní zadať meno súboru, ktorý sa má otvoriť.

4.1. Program v Arity/Prologu. Základnými formami výpovede v Prologu sú klauzuly. Klauzula má vo všeobecnosti tvar

⟨hlava klauzuly⟩ :- ⟨telo klauzuly⟩.

Každá klauzula je ukončená bodkou a koncom riadku. Niektoré klauzuly ustanovujú fakty, iné pravidlá. Napríklad klauzuly

```
clovek(jan).
zivocich(X):-clovek(X).
```

predstavujú v Prologu výpovede: „Ján je človek.“ a „Človek je živočích.“ Symbol `:-` tu reprezentuje podmieňovaciu spojku „ak“: `X` je živočích, ak `X` je človek.

Program v Prologu je zoznam faktov a pravidiel. Pri písaní Prologovskej klauzuly treba dodržať tieto pravidlá:

- Bodka a koniec riadku označujú koniec klauzuly. Preto je nutné ich vždy napísať.
- Medzi názvom predikátu a ľavou zátvorkou nesmie byť medzera. Napríklad `clovek(jan)` je správne, ale `clovek (jan)` nie je správne.
- Treba rozlišovať medzi veľkými a malými písmenami.

Je dôležité napísať do programu stručný komentár, podľa ktorého sa možno rýchlo dozvedieť, čo daný program obsahuje. Komentár je možné zapísať pomocou symbolu `%` takto

```
% Príklad programu.
% Program na určenie, či je niečo vták alebo cicavec.
```

alebo pomocou zátvoriek `/*` ... `*/`.

Po inicializovaní interpretera príkazom `api.idb` pomocou kláves ⟨Alt⟩-`fo` a pomocou tabelátora v príslušnom okienku zadajme názov `zivocich` (prípona `ari` sa implicitne doplní) a týmto sa ocitneme v editori interpretera s otvoreným súborom `zivocich.ari`. Ako príklad použitia interpretera do tohto súboru zapíšme nasledujúcu databázu obsahujúcu niekoľko faktov a pravidiel.

```
/* Príklad programu.
   Program na určenie, či je niečo vták alebo cicavec.*/

vtak(X):-teplo_krvny(X),stavovec(X),ma_perie(X).
cicavec(X):-teplo_krvny(X),stavovec(X),ma_srst(X).

teplo_krvny(medved).
teplo_krvny(orol).
teplo_krvny(vrabec).
teplo_krvny(clovek).

stavovec(medved).
stavovec(orol).
stavovec(vrabec).
stavovec(clovek).

ma_perie(orol).
ma_perie(vrabec).

ma_srst(medved).
ma_srst(clovek).
```

Po ukončení zapíšme obsah editora do súboru s názvom `zivocich.ari` stlačením kláves ⟨Alt⟩-`fs`.

Aby sme v interpreteri mohli spustiť program, musíme ho najskôr uložiť do prologovskej databázy. To dosiahneme voľbou `Consult File...` v menu `File` t.j. stlačením kláves ⟨Alt⟩-`fc` alebo voľbou `Reconsult On Exit` v menu `Buffers`, t.j. stlačením kláves ⟨Alt⟩-`br`.

V hlavnej obrazovke teraz môžeme napísať nejakú *otázku* (query):

```
?-vtak(X).
```

Po stlačení klávesy ⟨Return⟩ dostaneme

```
?-vtak(X).
X=orol ->
```

Po napísaní bodkočiarky sa objaví

```
?-vtak(X).
X=orol ->;
X=vrabec ->
```

Po opätovnom napísaní bodkočiarky dostaneme

```
?-vtak(X).
X=orol ->;
X=vrabec ->;
no
```

Odpoveď `no` znamená, že posledný pokus nájsť (ďalšie) riešenie pre splnenie cieľa neuspel. Ak namiesto bodkočiarky po prvej odpovedi stlačíme ⟨Return⟩ dostaneme

```
?-vtak(X).
X=orol ->
yes
```

Odpoveď `yes` znamená, že posledný pokus nájsť vyhovujúce riešenie uspel.

4.2. Predikáty pre načítanie súborov do databázy. Vieme už, že súbor obsahujúci prologovsky program môžeme načítať do databázy prostredníctvom menu `File` voľbou `Consult File...`. Ten istý výsledok dosiahneme prostredníctvom klauzuly

```
?-consult(subor).
```

Predikát `consult` predpokladá, že názov súboru má príponu `ari`. Ak chceme načítať súbor, ktorý nemá príponu `ari` treba celý názov uzavrieť dvojicou pravých apostrofov včítane bodky oddeľujúcej príponu aj v prípade, že názov nemá žiadnu príponu). Napríklad klauzula

```
?-consult('subor.','subor.doc',subor).
```

načíta do databázy v tom istom poradí súbory `subor`, `subor.doc` a `subor.ari`. Rovnako sa používa predikát `reconsult`.

Rozdiel medzi výsledkom použitia predikátu `consult` a predikátu `reconsult` je v tom, že zatiaľ čo `consult` pridá obsah súborov na koniec databázy bez ohľadu na to, či už databáza obsahuje predikáty definované v tomto súbore alebo nie, predikát `reconsult` porovnáva predikáty v súbore s predikátmi uloženými v databáze a v prípade, že nájde v databáze predikát, ktorý je definovaný aj v súbore, tak definíciu tohto predikátu nahradí definíciou obsiahnutou v danom súbore.

K dispozícii je aj skrátená forma zápisu týchto predikátov. Príkaz

```
?-[subor1,-subor2,subor3].
```

konzultuje súbory `subor1.ari` a `subor3.ari` a rekonzultuje súbor `subor2.ari` v poradí uvedenom v zátvorkách.

V prípade, že chceme, aby sa nejaký súbor `subor.ari` pri štarte interpretera automaticky načítal do databázy, dosiahneme to vytvorením súboru `prolog.ini` v danom adresári, ktorý bude obsahovať klauzulu

```
:-[subor].
```

4.3. Debugger. Dovoľuje sledovať beh programu krok za krokom a v prípade potreby zistiť, kde sa výpočtová vetva líši od nami predpokladaného výpočtu. Každý cieľ v Arity/Prologu má štyri porty:

`call` Týmto portom vstúpi program do cieľa pri postupe vpred a začne ho spĺňať.
`exit` Týmto portom program opustí cieľ po jeho úspešnom splnení (succeed).
`fail` Týmto portom program opustí daný cieľ, ak jeho splnenie zlyhalo (fail).
`redo` Týmto portom vstúpi program do cieľa, pri postupe späť (backtrack).

Rôzne parametre pre debugger je možné nastaviť pomocou menu `Debugger`, t.j. pomocou ⟨Alt⟩-`d`. Na druhej strane, v Arity/Prologu sú zabudované predikáty `trace/0` a `notrace/0` na aktiváciu a deaktiváciu debuggera, `spy` na vyznačenie predikátov, ktoré sa majú sledovať, a `leash/1` na vyznačenie portov, na ktorých má debugger pozastaviť výpočet. Na nastavenie rôznych parametrov debuggera môžeme použiť nasledujúce klauzuly, ktoré sa napíšu v hlavnej obrazovke:

`?-trace.` Zaktivuje sa debugger.
`?-notrace.` Deaktivuje sa debugger.
`?-spy(muz/1,otec/2).` Vyznačia sa predikáty `muz/1`, `otec/2`, ktoré sa majú sledovať.
`?-nospy(muz/1,otec/2).` Odstránia sa predikáty `muz/1`, `otec/2` zo zoznamu tých, ktoré sa majú sledovať.
`?-leash(half).` Vyznačia sa porty `call`, `redo`.
`?-leash(full).` Vyznačia sa porty `call`, `redo`, `fail`, `exit`.
`?-leash(off).` Nevyznačia sa žiadne porty a debugger nezastaví výpočet.
`?-leash(tight).` Vyznačia sa porty `call`, `redo`, `fail`.
`?-leash(loose).` Vyznačí sa port `call`.

Nech `p` je n-árny predikátový symbol a nech $A \subseteq \{1, \ldots, n\}$ je ľubovoľná množina. Hovoríme, že *program P je* $(\texttt{p}, A)$*-korektný*, ak pre každú postupnosť konštantných termov $\{t_i : i \in A\}$ je každá vypočítaná odpoveď programu P na otázku `?-p(`r_1`,...,`r_n`)` správna, kde $\{r_i : i \in A\}$ sú navzájom rôzne premenné a $r_i = t_i$ pre $i \notin A$. *Program P je* $(\texttt{p}, A)$*-úplný*, ak každá správna odpoveď na predchádzajúcu otázku je tiež vypočítaná odpoveď programom P. Nech $\mathcal{A} \subseteq \mathcal{P}(\{1, \ldots, n\})$ je ľubovoľný systém množín. Hovoríme, že *program P je* $(\texttt{p}, \mathcal{A})$*-korektný* (prípadne $(\texttt{p}, \mathcal{A})$*-úplný*), ak pre každú množinu $A \in \mathcal{A}$, P je $(\texttt{p}, A)$-korektný (prípadne $(\texttt{p}, A)$-úplný) program. V prípade $\mathcal{A} = \mathcal{P}(\{1, \ldots, n\})$ hovoríme, že P je `p`-korektný (prípadne `p`-úplný) program.

Kvôli stručnosti vo vyjadrovaní často ztotožňujeme pojem programu s pojmom predikátu definovaným týmto programom. Preto často hovoríme „predikát `p` je A-korektný (úplný)“ namiesto „program P je $(\texttt{p}, A)$-korektný (úplný)“.

5.1. Pár skôr použitých príkladov.

1. *Nájdením vhodných substitúcií unifikujte nasledujúce dvojice termov:*

a) `otec(adam,X)`, `otec(Y,zuza)`.
a) `a(X)`, `a(a(0))`.
b) `a(b(X,Y),c(Z))`, `a(Z,c(b(X,Y)))`.
c) `a(b(x),Y)`, `a(X,b(y))`.
d) `a(b(X),Y)`, `a(b(c(Y,d)),r(a))`.

2. *Definujte predikát* `sum(X)`*, ktorý opakovane vyzýva o nové číslo z klávesnice až kým neprečíta nečíselný údaj tak, že odpoveď na otázku* `?-sum(X).` *je* `X=`s*, kde s je súčet zadaných čísiel.*

3. *Definujte predikát* `rozklad(X)`*, ktorý pre kladné celé číslo* `X` *vypíše všetky jeho rozklady na súčet dvoch nezáporných celých čísiel.*

4. *Definujte predikát* `klasifikuj(X)`*, ktorý oznámi typ termu v argumente.*

5. *Predpokladajme, že databáza obsahuje predikát* `vek(Meno,Vek)`.

(1) *Definujte predikát* `vyber(Meno,Min,Max)`*, ktorý bude generovať mená ľudí vo veku medzi* `Min` *a* `Max`.
(2) *Definujte predikát* `vyber2(Min,Max)`*, ktorý vypíše pomocou* `write` *mená ľudí z úlohy* (1).

6. *V databáze sú uložené fakty* `a(`n`)` *pre konečnú množinu prirodzených čísiel n.*

a) *Definujte, predikát* `generuj(X)`*, ktorý vytvorí zoznam všetkých argumentov predikátu* `a/1`.
b) *Definujte predikát* `sum2(X)`*, ktorý určí súčet všetkých argumentov predikátu* `a(X)`.

5.2. Rodinná databáza.

7. a) *Databáza obsahuje predikáty* `rodic(Rodic,Potomok)`, `muz/1` *a* `zena/1`. *Pomocou týchto predikátov definujte nasledujúce (dvojmiestne) rodinné vzťahy:*

a) *otec*	g) *súrodenci*	m) *vnuk*	s) *ujo*
b) *matka*	h) *úplní súrodenci*	n) *vnučka*	t) *teta*
c) *syn*	i) *čiastoční súrodenci*	o) *predok*	u) *strýko*
d) *dcéra*	j) *starý rodič*	p) *potomok*	v) *stryna*
e) *brat*	k) *starý otec*	q) *bratranec*	w) *synovec*
f) *sestra*	l) *stará mama*	r) *sesternica*	x) *neter*

b) *Pridajte definície týchto predikátov* `rodinny_vztah(X,Y)` *do databázy v príklade* 1.3.1 *a pre ktorýkoľvek z nich zostrojte výpočtový strom pre otázku*

```
?-rodinny_vztah(X,Y).
```

8. *Definujte predikát* `gen_rozdiel(Predok,Potomok,Rozdiel)` *pre určenie generačného rozdielu predka a jeho priameho potomka.*

9. *Pre databázu obsahujúcu predikáty* `syn(Syn,Rodič)` *a* `dcera(Dcera,Rodič)` *definujte predikát* `pocet_deti(X,Y)` *vyjadrujúci, že počet detí rodiča* `X` *je* `Y`*:*

a) *pridávaním a odstraňovaním predikátov do databázy,*
b) *bez pridávania a odstraňovania predikátov do databázy.*

Pre databázu v príklade 1.3.3 *zostrojte výpočtový strom pre otázku*

```
?-pocet_deti(adam,X).
```

5.3. Základné funkcie na zoznamoch. Pripomeňme základné kombinatorické pojmy. *Permutácia n-prvkovej množiny* je ľubovoľná usporiadaná n-tica obsahujúca všetky prvky tejto množiny. Počet všetkých permutácií je $n!$. *Kombinácia k-tej triedy z n prvkovej množiny* je jej ľubovoľná k-prvková podmnožina. Počet kombinácií k-tej triedy z n prvkov je $\binom{n}{k}$. *Kombinácia k-tej triedy s opakovaním z n prvkovej množiny* M je ľubovoľná neusporiadaná k-tica prvkov množiny M, v ktorej sa prvky môžu opakovať. Počet kombinácií s opakovaním k-tej triedy z n prvkovej množiny je $\binom{n+k-1}{k}$. *Variácia k-tej triedy z n prvkovej množiny bez opakovania* je každá usporiadaná k-tica navzájom rôznych prvkov tejto množiny. Počet variácií k-tej triedy z n prvkov bez opakovania je $\binom{n}{k}k!$. *Variácia k-tej triedy z n prvkovej množiny s opakovaním* je každá usporiadaná k-tica prvkov tejto množiny. Počet variácií k-tej triedy z n prvkov s opakovaním je n^k.

10. *Definujte predikáty vyjadrujúce nasledujúce operácie so zoznamami a popíšte ich korektnosť.*

a) `first(X,Y)` `X` *je prvý člen zoznamu* `Y`.
b) `last(X,Y)` `X` *je posledný člen zoznamu* `Y`.
c) `append(X,Y,Z)` *Zoznam* `Z` *vznikne pripojením zoznamu* `Y` *na koniec zoznamu* `X`.
d) `prefix(X,Y)` *Zoznam* `X` *je počiatočný úsek zoznamu* `Y`.
e) `suffix(X,Y)` *Zoznam* `X` *je koncový úsek zoznamu* `Y`.
f) `sublist(X,Y)` *Zoznam* `X` *je úsek zoznamu* `Y`.
g) `subseq(X,Y)` *Zoznam* `X` *je podpostupnosť prvkov zoznamu* `Y`.

h) `revert(X,Y)` *Zoznam* `Y` *vznikne zo zoznamu* `X` *obrátením poradia jeho členov.*

i) `halve(X,Y,Z)` *Ak zoznam* `X` *má párny počet členov, tak* `Y` *je jeho prvá polovica a* `Z` *je jeho druhá polovica.*

j) `odd_even(X,Y,Z)` *Zoznam* `Y` *pozostáva z nepárnych členov a zoznam* `Z` *z párnych členov zoznamu* `X`.

k) `member(X,Y)` *Prvok* `X` *je členom zoznamu* `Y`.

l) `not_member(X,Y)` *Prvok* `X` *nie je členom zoznamu* `Y`.

m) `nth_member(N,X,Y)` *Prvok* `X` *je* `N`*-tým členom zoznamu* `Y` *(*`N` *je číslo).*

n) `dlzka(X,Y)`, *Číslo* `Y` *je dĺžka zoznamu* `X` *(rovnako ako zabudovaný predikát* `length(X,Y)`*).*

11. *Definujte* {2}*-korektný a* {2}*-úplný predikát* `perm(X,Y)`, *ktorý v argumente* `Y` *generuje zoznamy, ktoré sú permutáciou daného zoznamu* `X`. *Na otázku*

```
?-perm([1,2,3],X)
```

dostaneme postupne odpovede, po každej bodkočiarke jednu: `X=[1,2,3]`, `X=[1,3,2]`, `X=[2,1,3]`, `X=[2,3,1]`, `X=[3,1,2]`, `X=[3,2,1]`.

Návod. Rekurziou pomocou predikátu `delete(X,Y,Z)`, ktorý generuje zoznamy `Z` zo zoznamu `Y` vymazaním práve jedného prvku `X` (jednotlivé výskyty v zozname považujeme za rôzne prvky). □

12. a) *Definujte* {3}*-korektný a* {3}*-úplný predikát* `comb(K,X,Y)`, *ktorý generuje všetky kombinácie* `K`*-tej triedy zo zoznamu* `X` *v argumente* `Y`. *Predpokladáme, že daný zoznam* `X` *má aspoň* `K` *prvkov.*

b) *Definujte predikát* `comb2(K,N,X)`, *ktorý pre dané prirodzené čísla* `K`=k, `N`=n *dáva rovnaké výsledky ako* `comb(K,[1,2,...,`n`],X)`.

13. a) *Definujte* {3}*-korektný a* {3}*-úplný predikát* `comb_rep(K,X,Y)`, *ktorý v argumente* `Y` *generuje všetky kombinácie s opakovaním* `K`*-tej triedy zo zoznamu* `X`.

b) *Definujte predikát* `comb_rep2(K,N,X)`, *ktorý pre dané prirodzené čísla* `K`=k, `N`=n *dáva rovnaké výsledky ako* `comb_rep(K,[1,2,...,`n`],X)`.

14. a) *Definujte predikát* `subset(X,Y)`, *ktorý v argumente* `Y` *generuje zoznamy, ktoré vzniknú mazaním ľubovoľného počtu prvkov daného zoznamu* `X`. *Teda ak zoznam* `X` *má* n *prvkov, tak pre* `Y` *je* 2^n *odpovedí. Otázka* `?-subset([1,2,3,4],[X,Y])` *generuje kombinácie druhej triedy zo štyroch prvkov.*

b) *Definujte predikát* `subset2(N,X)`, *ktorý pre dané prirodzené číslo* `N`=n *dáva rovnaké výsledky ako* `subset([1,2,...,`n`],X)`.

15. a) *Definujte predikát* `varia(K,X,Y)`, *ktorý v argumente* `Y` *postupne generuje všetky variácie* `K`*-tej triedy bez opakovania zo zoznamu* `X`. *Predpokladáme, že daný zoznam* `X` *má aspoň* `K` *prvkov.*

b) *Definujte predikát* `varia2(K,N,X)`, *ktorý pre dané prirodzené čísla* `K`=k, `N`=n *dáva rovnaké výsledky ako* `varia(K,[1,2,...,`n`],X)`.

16. a) *Definujte predikát* `varia_rep(K,X,Y)`, *ktorý v argumente* `Y` *postupne generuje všetky variácie* `K`*-tej triedy s opakovaním zo zoznamu* `X`.

b) *Definujte predikát* `varia2_rep(K,N,X)`, *ktorý pre dané prirodzené čísla* `K`=k, `N`=n *dáva rovnaké výsledky ako* `varia_rep(K,[1,2,...,`n`],X)`.

5.4. Aritmetické operácie a usporiadanie čísiel.

17. *Definujte nasledujúce predikáty tak, aby spĺňali tieto podmienky:*

a) `sucet(X,Y)`*, ak* `Y` *je súčtom prvkov zoznamu* `X`*.*
b) `priemer(X,Y)`*, ak* `Y` *je aritmetický priemer prvkov zoznamu* `X`*.*
c) `usporiadaj(X,Y)`*, ak* `Y` *je zoznam, ktorý vznikne zo zoznamu* `X` *usporiadaním jeho prvkov v neklesajúcom poradí.*

Návod. Najjednoduchšie riešenie úlohy d), hoci najmenej efektívne, je v zmysle hesla „generuj a testuj“. Teda použitím generátora všetkých permutácií (pozri úlohu 11) stačí otestovať každý získaný zoznam a v prípade úspešného testu skončiť výpočet. □

5.5. Syntaktická aritmetika. Jazyk Peanovej aritmetiky pozostáva z jedného konštantného symbolu `0` a z jedného unárneho symbolu `s`. Prirodzené čísla v tomto jazyku predstavujú v poradí termy `0`, `s(0)`, `s(s(0))`, ... Kvôli stručnosti ich budeme nazývať `s`-termy.

18. *Definujte predikát* `unary_num(X)`*, ktorý generuje všetky* `s`*-termy.*

19. a) *Definujte predikát* `plus(X,Y,Z)`*, ktorý pre dané dva* `s`*-termy* `X`*,* `Y` *nájde* `s`*-term v argumente* `Z` *predstavujúci ich súčet. Zostrojte výpočtový strom pre otázku*

```
?-plus(s(s(0)),s(s(0)),X).
```

b) *Definujte predikát* `plus2(X,Y,Z)`*, ktorý oproti príkladu a) je naviac* $\{1,2\}$*-korektný a* $\{1,2\}$*-úplný. Zostrojte výpočtový strom pre otázku*

```
?-plus2(s(s(0)),s(0),X),plus2(s(s(0)),Y,X).
```

c) *Definujte* $\{1,2,3\}$*-korektný a*$\{1,2,3\}$*-úplný predikát* `plus3(X,Y,Z)`*. Zostrojte časť výpočtového stromu pre otázku* `?-plus3(X,Y,Z)` *po tretiu vypočítanú odpoveď zľava.*

20. *Definujte predikát* `minus(X,Y,Z)`*, ktorý reprezentuje odčítanie* `s`*-termov.*

21. *Definujte predikát* `krat(X,Y,Z)` *pre násobenie* `s`*-termov. Zostrojte výpočtový strom pre otázku* `?-krat(s(s(0)),s(s(0)),X)`*.*

22. *Definujte predikát* `faktorial(X,Y)` *pre výpočet faktoriálu* $x!$ *na množine všetkých* `s`*-termov a zostrojte výpočtový strom pre otázku* `?-faktorial(s(s(0)),X).`

23. *Definujte* $\{1,2\}$*-korektný predikát* `exponent(X,Y)` *pre výpočet mocnín dvojky na množine všetkých* `s`*-termov. Zostrojte časť výpočtového stromu pre otázku*

```
?-exponent(X,Y).
```

po tretiu vypočítanú odpoveď zľava.

24. a) *Definujte predikát* `pair2(X,Y)`*, ktorý generuje všetky usporiadané dvojice* `s`*-termov.*

b) *Definujte predikát* `pair(X,Y)`*, ktorý generuje všetky usporiadané dvojice* `s`*-termov v maximo-lexikografickom usporiadaní (pripomeňme, že* $(n,m) < (n',m')$*, ak* $\max\{n,m\} < \max\{n',m'\}$*, alebo* $\max\{n,m\} = \max\{n',m'\}$ *a* $n < n'$*, alebo* $\max\{n,m\} = \max\{n',m'\}$*,* $n = n'$ *a* $m < m'$*).*

25. *Upravte definície predikátov v príkladoch* 20, 21 *tak, aby boli* $\{1,2,3\}$*-korektné a úplné.*

26. *Definujte predikát* `triple(X,Y,Z)`*, ktorý generuje všetky usporiadané trojice* `s`*-termov v nejakom usporiadaní.*

Reprezentujme kladné racionálne čísla v tvare podielu dvoch `s`-termov, t.j. s použitím dodatočnej binárnej operácie ‚`/`' a v prípade záporných racionálnych čísiel pripustíme znamienko ‚`-`' v čitateli. Napríklad termy `0`, `s(s(0))/s(s(s(s(0))))`, `(-s(s(0)))/s(s(s(0)))` sú `s`-racionálne čísla, ale term `-s(s(0))/s(s(s(0)))` nie.

27. a) *Definujte predikát* `racio(X)`*, ktorý generuje všetky* `s`*-racionálne čísla v intervale* $[0,1)$ *v nejakom usporiadaní (treba využiť fakt, že racionálne čísla sa dajú zoradiť do jednej nekonečnej postupnosti).*

b) *Definujte predikát* `racio2(X)`*, ktorý generuje všetky* `s`*-racionálne čísla v intervale* $(-1,1)$*.*

28. *Definujte predikáty* `plus_rac(X,Y,Z)`*,* `krat_rac(X,Y,Z)` *pre sčítanie a násobenie* `s`*-racionálnych čísiel.*

5.6. Syntax výrokového počtu.

29. *Pomocou operátorových definícií (predikát* `op(X,Y,Z)`*) definujte infixné operátory* `'&'`*,* `'v'`*,* `'=>'`*,* `'<=>'` *a jeden unárny prefixný operátor* `'non'` *s obvyklými preferenciami. Definujte predikát* `vyhodnot(X,Y)`*, ktorý určuje pravdivostné hodnoty* `Y` *výrokov* `X` *vytvorených pomocou operátorov* `'&'`*,* `'v'`*,* `'=>'`*,* `'<=>'`*,* `'non'` *a konštantných pravdivostných hodnôt* `0` *a* `1`*.*

30. *Upravte definíciu predikátu* `vyhodnot(X,Y)` *v predchádzajúcej úlohe tak, že v prípade výskytu prologovských premenných vo výroku* `X` *generuje všetky možnosti pre ich pravdivostné hodnoty. Napríklad otázka* `?-vyhodnot(X & Y => non Z,V)` *má osem odpovedí a jedna z nich je* `X=1, Y=0, Z=0, V=1`*.*

5.7. Dátum.

31. *Definujte predikát* `datum(Den,Mesiac,Rok)` *s celočíselnými argumentami, ktorý testuje, či daná trojica prirodzených čísiel je skutočným dátumom.*

Návod. Treba zobrať do úvahy, že každý mesiac nemá rovnaký počet dní a každý rok deliteľný štyrmi je o deň dlhší. □

32. *Definujte predikáty* `zajtra(Dnes,Zajtra)` *a* `vcera(Dnes,Vcera)`*, ktoré pre term tvaru* `datum(Den,Mesiac,Rok)` *overia, či argumenty tohto termu sú skutočným dátumom a zostroja nový term* `datum(D,M,R)`*, reprezentujúci zajtrajší prípadne včerajší dátum tak, že napríklad nasledujúce dve otázky sú úspešné:*

```
?-zajtra(datum(1,3,1999),datum(2,3,1999)).
?-vcera(datum(1,3,1999),datum(28,2,1999)).
```

33. *Použijúc predchádzajúce definície definujte nasledujúce predikáty popísané uvedenými podmienkami:*

a) `pozajtra(X,Y)` *nájde pozajtrajší dátum* `Y` *pre daný dátum* `X`*.*
b) `predvcerom(X,Y)` *nájde predvčerajší dátum* `Y` *pre daný dátum* `X`*.*

c) `potom(N,X,Y)` *nájde dátum* `Y` *nasledujúci* `N` *dní po dátume* `X`.
d) `predtym(N,X,Y)` *nájde dátum* `Y` *predchádzajúci* `N` *dní pred dátumom* `X`.
d) `doba(X,Y,N)` *určí počet dní* |`N`|*, ktoré uplynú medzi dátumami* `X` *a* `Y`*, pričom* `N` *je kladné práve vtedy, keď* `X` *predchádza dátumu* `Y`.
e) `aky_je_den(X,Y)` *pre daný dátum* `X` `Y` *je príslušný deň v týždni (*`pondelok`, ... , `nedeľa`*).*

5.8. Rôzne úlohy.

34 (Rozklady čísla). *Definujte predikát, ktorý nájde všetky rozklady daného kladného celého čísla na súčet kladných celých čísiel*

a) *v tvare zoznamu jednotlivých sčítancov,*

b) *v tvare súčtu jednotlivých sčítancov.*

35 (Hanojské veže). *Dané sú tri tyče* `a`, `b`, `c`. *Na tyči* `a` *je navlečených* `N` *rôzne veľkých kotúčov tak, že žiaden kotúč nie je položený na menšom kotúči. Úlohou je presunúť kotúče z tyče* `a` *na tyč* `b` *použitím tyče* `c` *tak, že v žiadnom kroku nie je položený väčší kotúč na menšom kotúči.*

a) *Napíšte program v Prologu, ktorý pre daný počet* `N` *kotúčov vytvorí zoznam jednotlivých krokov tvaru* ‚`X na Y`‘ *s významom presuň kotúč z tyče* `X` *na tyč* `Y`, *kde* `X`, `Y` *sú ľubovoľné z písmen* `a`, `b`, `c`.

b) *Napíšte program, ktorý zapíše jednotlivé kroky návodu pre presun daného počtu kotúčov do súboru s názvom* ‚`postup.txt`‘, *každý krok na nový riadok.*

Návod. Predpokladajme, že poznáme návod na premiestnenie `N` diskov z tyče `a` na tyč `b` pomocou tyče `c`. Vzájomnými výmenami písmen `a`, `b`, `c` v tomto návode zostrojíme návod na premiestnenie $\mathtt{N}+1$ diskov: (1) Vytvoríme návod na premiestnenie `N` diskov z tyče `a` na tyč `c` pomocou tyče `b`. (2) Do návodu pridáme term ‚`a na b`‘ a pripojíme návod na premiestnenie `N` diskov z tyče `c` na tyč `b` pomocou tyče `a`. □

36 (Rozmiestnenie dám na šachovnici). a) *Napíšte program v Prologu, ktorým sa nájdu všetky rozmiestnenia ôsmich dám na šachovnici rozmeru* 8×8 *tak, aby sa jednotlivé dámy neohrozovali, t.j. aby žiadne dve dámy neboli umiestnené v tom istom stĺpci, v tom istom rade alebo na tej istej diagonále.*

b) *Napíšte program v Prologu, ktorým sa nájdu všetky rozmiestnenia* `N` *dám na šachovnici rozmeru* $\mathtt{N} \times \mathtt{N}$ *tak, aby sa jednotlivé dámy neohrozovali.*

Návod. Dámy ukladajte na šachovnicu do jednotlivých stĺpcov zľava doprava tak, aby pridaná dáma neohrozovala predchádzajúce, až kým nie sú obsadené všetky stĺpce. Postupne sa vyskúšajú všetky polohy pre dámu v stĺpci počnúc prvým radom a končiac posledným. □

37 (Latinské štvorce). *Latinský štvorec rádu* n *je matica čísiel typu* $n \times n$ *taká, že v každom riadku a v každom stĺpci matice sú všetky čísla od* 1 *do* n.

a) *Napíšte program, ktorý pre dané číslo* n *nájdite všetky latinské štvorce rádu* n.

b) *Napíšte program, ktorý pre dané číslo* n *nájdite všetky latinské štvorce rádu* n *také, že aj obe hlavné diagonály obsahujú všetky čísla.*

38 (Magické štvorce). *Magický štvorec radu n je taká tabuľka tvaru $n \times n$ obsahujúca čísla od 1 do n^2, že súčet čísiel v každom riadku, v každom stĺpci a na oboch diagonálach je rovnaký. Definujte predikát, ktorý pre dané prirodzené číslo n nájde všetky magické štvorce rádu n.*

39 (Prelievanie nádob). *Daných je niekoľko nádob, ich objem a obsah vody v litroch v nich. Ktorúkoľvek nádobu je dovolené buď vyprázdniť, preliať celý jej objem bezo zvyšku do inej nádoby, alebo doplniť inú nádobu doplna s prípadným zvyškom vody v nej. Definujte predikát, ktorý pre dané číslo n nájde postup prelievanie vody medzi nádobami tak, aby v niektorej z nádob ostalo presne n litrov vody.*

40 (Hra so zápalkami). *Hra s dvoma kôpkami zápaliek má dvoch hráčov. Obaja hráči striedavo odoberajú ľubovoľný nenulový počet zápaliek z jednej z kôpok. Prehrá ten hráč, ktorý zoberie poslednú zápalku.*

a) *Definujte predikát pre interaktívnu hru druhého hráča.*

b) *Definujte predikát pre interaktívnu hru druhého hráča v hre s ľubovoľným počtom kôpok.*

41. *Definujte predikát, ktorý v danom atóme vymení veľké a malé písmená. Ostatné znaky ostanú bezo zmeny.*

Návod. Je možné využiť fakt, že ASCII kódy veľkých písmen sú 65–90 a ASCII kódy malých písmen sú 97–122. Rozdiel medzi malým a odpovedajúcim veľkým písmenom je presne 32. □

42 (Jazdec na šachovnici). *Definujte predikát, ktorý nájde všetky cesty koňom po šachovnici začínajúce z rohového políčka $(1, 1)$,*

a) *tak, aby sa kôň ocitol na každom políčku šachovnice práve raz,*

b) *tak, aby sa kôň ocitol na každom políčku šachovnice najviac raz a všetky políčka dostupné z posledného políčka boli už použité. Zároveň sa určí dĺžka tejto cesty.*

43 (Algebrogramy). *Algebrogram je aritmetická rovnosť, v ktorej namiesto číslic sú písmená. Rôzne písmená treba nahradiť rôznymi číslicami od 0 do 9, tak aby získaná rovnosť bola pravdivá pričom prvé písmená nemožno nahradiť nulou. Napríklad algebrogram* `potom` + `pocit` = `ostane` *má tieto tri riešenia* $81213 + 81492 = 162705$ $51216 + 71732 = 102948$, $51217 + 51632 = 102849$. *Napíšte program v Prologu na riešenie algebrogramov s dvoma sčítancami tak, že troma argumentami definovaného predikátu budú*

a) *zoznamy odpovedajúcich veľkých písmen k písmenám v slovách algebrogramu, t.j. zoznamy premenných,*

b) *atómy zhodné so slovami v algebrograme.*

Teda vyššie uvedený príklad budú riešiť obe nasledujúce otázky:

```
?-algebrogram([P,O,T,O,M],[P,O,C,I,T],[O,S,T,A,N,E]).
?-algebrogram2(potom,pocit,ostane).
```

c) *Vyriešte tieto algebrogramy:* `send` + `more` = `money`, `skola` + `zaklad` = `zivota`, `trva` + `tento` = `pocit`, `slnko` + `teplo` = `svieti`, `zima` + `jar` = `leto`, `tento` + `dojem` = `ostane`, `auto` + `moto` = `revue`.

Návod. b) Vbudovaným predikátom `name(X,L)` prekonvertujte atómy v argumentoch predikátu `algebrogram2(X,Y,Z)` na zoznamy ASCII kódov a získané zoznamy spracujte podobne ako zoznamy v úlohe a) priradením navzájom rôznych premenných týmto kódom. □

44. *Je 5 domov, každý inej farby, v každom dome žije obyvateľ inej národnosti, každý obyvateľ pije iný nápoj, fajčí iné cigarety a chová iné zvieratá ako ostatní.*

1. *Angličan žije v červenom dome.*
2. *Švéd chová psov.*
3. *Dán pije čaj.*
4. *Zelený dom je naľavo od bieleho.*
5. *Obyvateľ zeleného domu pije kávu.*
6. *Ten čo fajčí Pall Mall chová vtáky.*
7. *Obyvateľ žltého domu fajčí Dunhill.*
8. *Ten čo žije v strednom dome pije mlieko.*
9. *Nór žije v prvom dome.*
10. *Ten čo fajčí Blend žije vedľa toho, čo chová mačky.*
11. *Ten čo chová kone žije vedľa toho, čo fajčí Dunhill.*
12. *Ten čo fajčí Blue Master pije pivo.*
13. *Nemec fajčí Prince.*
14. *Nór žije vedľa modrého domu.*
15. *Ten čo fajčí Blend ma suseda, ktorý pije vodu.*

Otázka: Kto chová ryby?

Návod. Permutujte zoznamy farieb, národností, nápojov, cigariet a zvierat, a testujte, či spĺňajú uvedených 15 podmienok. Keďže počet všetkých možností pre takéto permutácie je $(5!)^5 = 120^5$, výpočet treba urýchliť zmenou poradia generovania permutácii a spĺňania uvedených podmienok. Napríklad hneď po vytvorení permutácie všetkých farieb treba uviesť všetky podmienky, v ktorých je určená farba. Okrem toho netreba zabudnúť ešte na jeden atribút – čísloavanie domov: prvý dom môže byť vľavo ale rovnako aj vpravo (pozri podmienku 9). □

5.9. Zápis čísiel.

45 (Prevod čísiel). a) *Definujte predikát na prevod zadanej celočíselnej hodnoty na slovné vyjadrenie tejto hodnoty.*

b) *Možnosť prevodu číselnej hodnoty na slovné vyjadrenie je obmedzená maximálnym číslom* 2 147 483 647. *Definujte predikát prevádzajúci dané čísla na slovné vyjadrenie, v ktorom sa číslo zadá atómom obsahujúcim príslušné číslice.*

46 (Rímsky zápis čísiel). a) *Definujte predikát na prevod arabských čísiel na rímske. Výsledný rímsky zápis čísla bude* (1) *zoznam písmen (atómov dĺžky* 1*), alebo* (2) *jeden atóm (t.j. číslo* 34 *prevedie buď na* `['X','X','X','I','V']`*, alebo na* `'XXXIV'`*). V prípade* (1) *definujte predikát na vypísanie výsledku na obrazovku v čitateľnom tvare.*

b) *Definujte predikát na prevod rímskeho zápisu čísiel na arabský.*

47. *Reprezentujme prirodzené čísla pozičným zápisom v tvare zoznamu číslic p-adickej sústavy* 0, 1, ... , $p-1$. *Pomocou známych algoritmov pre pre operácie s číslami reprezentovanými v pozičnej p-adickej sústave definujte predikáty pre*

a) *sčítanie,*
b) *násobenie,*
c) *odčítanie,*
d) *čiastočné delenie a*
e) *výpočet celej časti druhej odmocniny prirodzeného čísla.*

Návod. Keďže algoritmy pre sčítanie, násobenie a odčítanie postupujú zprava doľava počnúc najnižším radom poziˇcného zápisu, bude výhodnejšie reprezentovať čísla zoznamami cifier (čísiel) v obrátenom poradí.

e) Základom algoritmu pre výpočet odmocniny je rovnosť $(pa+i)^2 = p^2a^2 + (2pa+i)i$. Nech $x_1x_2\ldots x_{2k-1}x_{2k}$ je zápis čísla x v p-adickom zápise (pripusťme možnosť $x_1 = 0$ v prípade nepárneho počtu cifier). Ak pre $x^{(j)} = x_1x_2\ldots x_{2j-1}x_{2j}$ sa v predchádzajúcich krokoch našlo najväčšie j ciferné číslo a také, $y_j = x^{(j)} - a^2 \geq 0$, tak $(j+1)$-vá cifra i je najväčšie číslo také, že rozdiel $p^2y_j + px_{2j+1} + x_{2j+2} - (2pa+i)i$ je nezáporný. Tento rozdiel je zároveň číslo $y_{j+1} = x^{(j+1)} - (pa+i)^2$. □

48. *Definujte predikát na prevod čísiel z p-adického zápisu do q-adického zápisu pre ľubovoľné čísla $p, q \geq 2$.*

6. Riešenia cvičení

Príklad 7. a), 8.

```
otec(A,B):-rodic(A,B),muz(A).        % A je otcom B.
matka(A,B):-rodic(A,B),zena(A).
syn(A,B):-rodic(B,A),muz(A).         % A je synom B.
dcera(A,B):-rodic(B,A),zena(A).
brat(A,B):-surodenci(A,B),muz(A).    % A je bratom B.
sestra(A,B):-surodenci(A,B),zena(A). % A je sestrou B.
surodenci(A,B):-rodic(X,A),rodic(X,B),A\=B.
uplni_surodenci(A,B):-rodic(X,A),rodic(X,B),A\=B,muz(X),
     rodic(Y,A),rodic(Y,B),zena(Y).
polo_surodenci(A,B):-rodic(X,A),rodic(X,B),rodic(Y,A),
     rodic(Z,B),X\=Y,Y\=Z,X\=Z.      % Dohromady majú 3 rodičov.
stary_rodic(A,B):-rodic(A,X),rodic(X,B).
stary_otec(A,B):-stary_rodic(A,B),muz(A).
stary_mama(A,B):-stary_rodic(A,B),zena(A).
vnuk(A,B):-stary_rodic(B,A),muz(A).
vnucka(A,B):stary_rodic(B,A),zena(A).
predok(A,B):-rodic(A,B).
predok(A,B):-rodic(A,X),predok(X,B).
potomok(A,B):-predok(B,A).
bratranec(A,B):-cousin(A,B),muz(A).
sesternica(A,B):-cousin(A,B),zena(A).
cousin(A,B):-rodic(X,A),rodic(Y,B),X\=Y,rodic(Z,X),rodic(Z,Y).
ujo(A,B):-matka(X,B),surodenci(A,X),muz(A).
teta(A,B):-matka(X,B),surodenci(A,X),zena(A).
stryko(A,B):-otec(X,B),surodenci(A,X),muz(A).
stryna(A,B):-otec(X,B),surodenci(A,X),zena(A).
synovec(A,B):-muz(A),rodic(A,X),surodenci(X,B).
neter(A,B):-zena(A),rodic(A,X),surodenci(X,B).
gen_rozdiel(X,Y,1):-rodic(X,Y).
gen_rozdiel(X,Y,N):-rodic(X,Z),gen_rozdiel(Z,Y,M),N is M+1.
```

Príklad 9. a). Cieľ `ma_deti(X)` v nasledujúcej definícii predikátu `pocet_deti/2` je možné vynechať. Zabezpečuje iba korektnosť predikátu `pocet_deti/2`. Otázka `?-pocet_deti(X,Y)` generuje všetky dvojice rodič-počet detí (tieto odpovede sa opakujú toľkokrát, koľko detí má rodič).

```
pocet_deti(X,Y):-ma_deti(X),pocet_synov(X,A),pocet_dcer(X,B),Y is A+B.
ma_deti(X):-syn(_,X).
ma_deti(X):-dcera(_,X).
pocet_synov(X,Y):-retract(syn(A,X)),pocet_synov(X,Z),
     Y is Z+1,asserta(syn(A,X)),!.
pocet_synov(X,0).
pocet_dcer(X,Y):-retract(dcera(A,X)),pocet_dcer(X,Z),
     Y is Z+1,asserta(dcera(A,X)),!.
pocet_dcer(X,0).
```

Príklad 9. b).

```
pocet_deti2(X,Y):-ma_deti(X),zoznam_deti(X,A),dlzka_zoznamu(A,Y).
ma_deti(X):-syn(_,X).
ma_deti(X):-dcera(_,X).
zoznam_deti(X,Y):-z(X,[],Y).
z(X,L,Y):-syn(H,X),not_member(H,L),z(X,[H|L],Y),!.
z(X,L,Y):-dcera(H,X),not_member(H,L),z(X,[H|L],Y),!.
z(X,Y,Y).
not_member(X,[H|T]):-X\=H,not_member(X,T).
not_member(X,[]).
dlzka_zoznamu([],0).
dlzka_zoznamu([H|T],X):-dlzka_zoznamu(T,Y),X is Y+1.
```

Príklad 10.

```
first(F,[F|_]).
last(L,[L]).
last(L,[_|T]):-last(L,T).
append([],L,L).
append([H|T],S,[H|L]):-append(T,S,L).
prefix(P,L):-append(P,_,L).
suffix(S,L):-append(_,S,L).
sublist([],L).
sublist([H|S],L):-append(T,_,L),append(_,[H|S],T).
subseq([],_).
subseq([H|S],[H|T]):-subseq(S,T).
subseq([H|S],[_|T]):-subseq([H|S],T).
revert(L,R):-rev(L,[],R).
rev([],R,R).
rev([H|T],A,R):-rev(T,[H|A],R).
halve(L,A,B):-hv(L,L,A,B).
hv([],B,[],B).
hv([_,_|R],[H|S],[H|T],B):-hv(R,S,T,B).
odd_even([],[],[]).
odd_even([H|T],[H|O],E):-odd_even(T,E,O).
member(H,[H|_]).
member(X,[_|T]):-member(X,T).
not_member(X,[]).
not_member(X,[H|T]):-X\=H,not_member(X,T).
nth_member(1,X,[X|_]).
nth_member(N,X,[_|T]):-N>1,M is N-1,nth_member(M,X,T).
dlzka([],0).
dlzka([H|T],X):-dlzka(T,A),X is A+1.
```

Predikáty `append/3`, `revert/2` sú $\{1,2\}$-korektné, predikáty `first/2`, `last/2`, `prefix/2`, `suffix/2`, `sublist/2`, `subseq/2`, `member/2`, `not_member/2` sú $\{1\}$-korektné, predikáty `halve/3`, `odd_even/3` sú $\{\{1\},\{2,3\}\}$-korektné a predikáty `nth_member/3`, `dlzka/2` sú $\{2\}$-korektné.

Príklad 11.

```
perm([],[]).
perm(L,[H|T]):-delete(H,L,R),perm(R,T).
delete(H,[H|S],S).
delete(X,[H|S],[H|T]):-delete(X,S,T).
```

Príklad 12. a), b).

```
comb(0,_,[]).
comb(K,[H|S],[H|T]):-K>0,I is K-1,comb(I,S,T).
comb(K,[_|S],T):-K>0,comb(K,S,T).
comb2(K,N,X):-n_tica(N,Y),comb(K,Y,X).
n_tica(N,X):-tica(N,1,X).
tica(N,K,[K|T]):-K<N,M is K+1,tica(N,M,T).
tica(N,N,[N]).
```

Príklad 13. a), b).

```
comb_rep(0,_,[]).
comb_rep(K,[H|S],[H|T]):-K>0,I is K-1,comb_rep(I,[H|S],T).
comb_rep(K,[_|S],T):-K>0,comb_rep(K,S,T).
comb_rep2(K,N,X):-n_tica(N,Y),comb_rep(K,Y,X).
n_tica(N,X):-tica(N,1,X).
tica(N,K,[K|T]):-K<N,M is K+1,tica(N,M,T).
tica(N,N,[N]).
```

Príklad 14. a), b).

```
subset(_,[]).
subset([H|S],[H|T]):-subset(S,T).
subset([_|S],[H|T]):-subset(S,[H|T]).
subset2(K,N,X):-n_tica(N,Y),subset(K,Y,X).
n_tica(N,X):-tica(N,1,X).
tica(N,K,[K|T]):-K<N,M is K+1,tica(N,M,T).
tica(N,N,[N]).
```

Predikát `subset/2` je definovaný rovnako ako `subseq/2` (pozri príklad 10) a stačilo definovať:
`subset(X,Y):-subseq(Y,X).`

Príklad 15. a), b).

```
varia(0,_,[]).
varia(K,L,[H|T]):-K>0,I is K-1,delete(H,L,R),varia(I,R,T).
delete(H,[H|T],T).
delete(X,[H|T],[H|T1]):-delete(X,T,T1).
varia2(K,N,X):-n_tica(N,Y),varia(K,Y,X).
n_tica(N,X):-tica(N,1,X).
tica(N,K,[K|T]):-K<N,M is K+1,tica(N,M,T).
tica(N,N,[N]).
```

Príklad 16. a), b).

```
varia_rep(0,_,[]).
varia_rep(K,S,[H|T]):-K>0,I is K-1,member(H,S),varia_rep(I,S,T).
member(H,[H|_]).
member(X,[_|T]):-member(X,T).
varia_rep2(K,N,X):-n_tica(N,Y),varia(K,Y,X).
n_tica(N,X):-tica(N,1,X).
tica(N,K,[K|T]):-K<N,M is K+1,tica(N,M,T).
tica(N,N,[N]).
```

Príklad 17. Nasledujúce definície sú prevzaté z riešenia úlohy 2.3.1.

```
sucet([],0).
sucet([H|T],S):-sucet(T,A),S is H+A.
priemer(L,P):-sz2(L,X),P is X.
sz2([],0/0).
sz2([H|T],R/N):-sz2(T,S/M),R is S+H,N is M+1.
usporiadaj([],[]).
usporiadaj([H|T],R):-usporiadaj(T,S),f3(H,S,R).
f3(X,[],[X]).
f3(X,[Y|T],[X,Y|T]):-X=<Y.
f3(X,[Y|T],[Y|S]):-X>Y,f3(X,T,S).
```

Príklady 18, 19.

```
unary_num(0).
unary_num(s(X)):-unary_num(X).
plus(0,X,X).
plus(s(X),Y,s(Z)):-plus(X,Y,Z).
plus2(0,X,X).
plus2(s(X),Y,Z):-plus(X,s(Y),Z).
plus3(X,Y,Z):-unary_num(Z),plus2(X,Y,Z).
```

Príklad 20.

```
minus(X,0,X).
minus(X,s(Y),Z):-minus(X,Y,s(Z)).
```

Príklad 21.

```
krat(0,X,0).
krat(s(X),Y,Z):-krat(X,Y,A),plus2(A,Y,Z).
plus2(0,X,X).
plus2(s(X),Y,Z):-plus2(X,s(Y),Z).
```

Príklad 22.

```
faktorial(0,s(0)).
faktorial(s(X),Y):-faktorial(X,A),krat(s(X),A,Y).
krat(0,X,0).
krat(s(X),Y,Z):-krat(X,Y,A),plus2(A,Y,Z).
plus2(0,X,X).
plus2(s(X),Y,Z):-plus2(X,s(Y),Z).
```

Príklad 23.

```
exp(0,s(0)).
exp(s(X),Y):-exp(X,A),plus2(A,A,Y).
plus2(0,X,X).
plus2(s(X),Y,Z):-plus2(X,s(Y),Z).
```

Príklad 24. a).

```
pair(X,Y):-unary_num(Z),plus2(X,Y,Z).
unary_num(0).
unary_num(s(X)):-unary_num(X).
plus2(0,X,X).
plus2(s(X),Y,Z):-plus2(X,s(Y),Z).
```

Príklad 24. b).

```
pair2(X,Y):-unary_num(Z),dvojice(X,Y,Z).
unary_num(0).
unary_num(s(X)):-unary_num(X).
dvojice(X,Y,Z):-Y=Z,mensi(X,Y).
dvojice(X,Y,Z):-X=Z,mensi_rovny(Y,X).
mensi(0,s(X)).
mensi(s(X),s(Y)):-mensi(X,Y).
mensi_rovny(0,X).
mensi_rovny(s(X),s(Y)):-mensi_rovny(X,Y).
```

Príklad 25.

```
minus2(X,Y,Z):-unary_num(X),minus(X,Y,Z).
unary_num(0).
unary_num(s(X)):-unary_num(X).
minus(X,0,X).
minus(X,s(Y),Z):-minus(X,Y,s(Z)).
krat2(X,Y,Z):-pair(X,Y),krat(X,Y,Z).
pair(X,Y):-unary_num(Z),plus2(X,Y,Z).
plus2(0,X,X).
plus2(s(X),Y,Z):-plus2(X,s(Y),Z).
krat(0,X,0).
krat(s(X),Y,Z):-krat(X,Y,A),plus2(A,Y,Z).
```

Príklad 26.

```
triple(X,Y,Z):-unary_num(A),plus2(X,B,A),plus2(Y,Z,B).
unary_num(0).
unary_num(s(X)):-unary_num(X).
plus2(0,X,X).
plus2(s(X),Y,Z):-plus2(X,s(Y),Z).
```

Príklad 27. a).

```
racio(0).
racio(s(X)/s(s(Y))):-unary_num(Y),mensi_rovny(X,Y).
mensi_rovny(0,Y).
mensi_rovny(s(X),s(Y)):-mensi_rovny(X,Y).
```

Príklad 27. b).

```
racio2(0).
racio2(X/s(s(Y))):-unary_num(Y),mensi_rovny(Z,Y),plus_minus(X,s(Z)).
mensi_rovny(0,Y).
mensi_rovny(s(X),s(Y)):-mensi_rovny(X,Y).
plus_minus(X,X).
plus_minus(-X,X).
```

Príklad 28.

```
plus_rac(0,X,X).
plus_rac(X/Y,0,X/Y).
plus_rac(A/B,C/D,Z):-
     krat_c(B,D,Y),krat_c(A,D,S),krat_c(C,B,T),plus_c(S,T,X),X\=0,Z=X/Y.
plus_rac(A/B,C/D,Z):-krat_c(A,D,S),krat_c(C,B,T),plus_c(S,T,X),X=0,Z=0.
plus_c(0,X,X).
plus_c(X,0,X):-X\=0.
plus_c(s(X),-s(Y),Z):-plus_c(X,-Y,Z).
plus_c(-s(X),s(Y),Z):-plus_c(-X,Y,Z).
plus_c(-X,-Y,-Z):-plus_c(X,Y,Z).
plus_c(s(X),s(Y),s(s(Z))):-plus_c(X,Y,Z).
krat_c(0,X,0).
krat_c(X,0,0):-X\=0.
krat_c(s(X),-Y,-Z):-krat_c(s(X),Y,Z).
krat_c(-X,-Y,Z):-krat_c(X,Y,Z).
krat_c(-X,s(Y),-Z):-krat_c(X,s(Y),Z).
krat_c(s(X),s(Y),Z):-krat_c(X,s(Y),A),plus_c(A,s(Y),Z).
krat_rac(X,0,0).
krat_rac(0,X,0):-X\=0.
krat_rac(A/B,C/D,X/Y):-krat_c(A,C,X),krat_c(B,D,Y).
```

Príklady 29, 30. Všetky logické spojky budeme definovať pomocou konjunkcie a využijeme skutočnosť, že v nasledujúcom cieľ `X&Y=a&b&c` má jedinú správnu odpoveď `X=a&b`, `Y=c`.

```
:-op(400,yfx,&).
:-op(500,yfx,v).
:-op(300,fy,non).
:-op(600,yfx,=>).
:-op(600,yfx,<=>).
konjunkcia(0,0,0).
konjunkcia(0,1,0).
konjunkcia(1,0,0).
konjunkcia(1,1,1).
negacia(0,1).
negacia(1,0).
konstanta(0).
konstanta(1).
vyhodnot(X&Y,Z):-konstanta(X),konstanta(Y),konjunkcia(X,Y,Z).
vyhodnot(X&Y,Z):-nonvar(X),konstanta(Y),vyhodnot(X,A),konjunkcia(A,Y,Z).
vyhodnot(X&Y,Z):-konstanta(X),nonvar(Y),vyhodnot(Y,B),konjunkcia(X,B,Z).
vyhodnot(X&Y,Z):-nonvar(X),nonvar(Y),vyhodnot(X,A),vyhodnot(Y,B),
     konjunkcia(A,B,Z).
vyhodnot(non X,Z):-konstanta(X),negacia(X,Z).
vyhodnot(non X,Z):-nonvar(X),vyhodnot(X,A),negacia(A,Z).
vyhodnot(X v Y,Z):-vyhodnot(non(non X & non Y),Z).
vyhodnot(X=>Y,Z):-vyhodnot(non X v Y,Z).
vyhodnot(X<=>Y,Z):-vyhodnot((X=>Y)&(Y=>X),Z).
```

Príklady 31, 32, 33.

```
datum(29,2,R):-Y is R mod 4,Y=0.
datum(D,M,R):-1=<M,M=<12,mesiace(X),member([M,D1],X),1=<D,D=<D1.
mesiace([[1,31],[2,28],[3,31],[4,30],[5,31],[6,30],[7,31],[8,31],
     [9,30],[10,31],[11,30],[12,31]]).
member(H,[H|_]).
member(X,[_|T]):-member(X,T).
zajtra(datum(D,M,R),datum(Z,M,R)):-datum(D,M,R),Z is D+1,datum(Z,M,R),!.
     % Dnes nie je posledný deň mesiaca.
zajtra(datum(D,M,R),datum(1,Z,R)):-datum(D,M,R),Z is M+1,datum(1,Z,R),!.
     % Dnes je posledný deň mesiaca, ale nie posledný deň roka.
zajtra(datum(D,M,R),datum(1,1,Z)):-datum(D,M,R),Z is R+1,datum(1,1,Z).
     % Dnes je posledný deň roka.
vcera(datum(D,M,R),datum(V,M,R)):-datum(D,M,R),V is D-1,V>0,!.
     % Dnes nie je prvý deň mesiaca.
vcera(datum(D,M,R),datum(W,V,R)):-datum(D,M,R),V is M-1,mesiace(X),
     member([V,P],X),(W=P+1;W=P),datum(W,V,R),!.
     % Dnes je prvý deň mesiaca, ale nie prvý deň roka.
vcera(datum(D,M,R),datum(31,12,V)):-datum(D,M,R),V is R-1,datum(31,12,V).
     % Dnes je prvý deň roka.
pozajtra(X,Y):-zajtra(X,Z),zajtra(Z,Y).
predvcerom(X,Y):-vcera(X,Z),vcera(Z,Y).
potom(0,X,X).
potom(N,X,Y):-N>0,zajtra(X,Z),M is N-1,potom(M,Z,Y).
predtym(0,X,X).
predtym(N,X,Y):-N>0,vcera(X,Z),M is N-1,predtym(M,Z,Y).
doba(X,X,0).
doba(X,Y,N):-prv(X,Y),zajtra(X,Z),doba(Z,Y,M),N is M+1.
doba(X,Y,N):-prv(Y,X),zajtra(Y,Z),doba(Z,X,M),N is -M-1.
prv(datum(_,_,R1),datum(_,_,R2)):-R1<R2.
prv(datum(_,M1,R),datum(_,M2,R)):-M1<M2.
prv(datum(D1,M,R),datum(D2,M,R)):-D1<D2.
dni_v_tyzdni(['nedeľa'/0,pondelok/1,utorok/2,streda/3,'štvrtok'/4,piatok/5,
     sobota/6]).
aky_je_den(X,Y):-doba(X,datum(1,1,1999),A),B is (abs(A)+5)mod 7,dni_v_tyzdni(C),
     member(Y/B,C).         % 1.1.1999 je piatok
```

Príklad 34. a), b).

```
rozklady(0,[]).
rozklady(X,[X]):-X\=0.
rozklady(X,[H|Y]):-zoznam(X,[],L),member(H,L),A is X-H,A>0,rozklady(A,Y).
zoznam(0,X,X).
zoznam(N,T,X):-N>0,K is N-1,zoznam(K,[N|T],X).
member(H,[H|_]).
member(X,[_|T]):-member(X,T).
rozklady2(X,X).
rozklady2(X,H+Y):-zoznam(X,[],L),member(H,L),A is X-H,A>0,rozklady2(A,Y).
```

Príklady 35. a), b).

```
:-op(500,xfx,na).
hanoi(N,R):-presun(N,a,b,c,R),!.
presun(0,_,_,_,[]).
presun(N,A,B,C,X):-N>0,M is N-1,presun(M,A,C,B,Y),presun(M,C,B,A,Z),
     append(Y,[A na B|Z],X).
append([],L,L).
append([H|T],S,[H|L]):-append(T,S,L).
hanoi2(N):-tell('postup.txt'),presun(N,a,b,c),told.
presun(0,_,_,_).
presun(N,A,B,C):-N>0,M is N-1,presun(M,A,C,B),write(A na B),nl,presun(M,C,B,A).
```

Príklad 36. a). Poloha dám je jednoznačne určená permutáciou množiny $\{1,2,3,4,5,6,7,8\}$, keďže v každom stĺpci a v každom riadku musí byť práve jedna dáma. Predikát `ukladaj(0,[],X)` hľadá správne permutácie `X` a predikát `diagram(X)` vypíše odpovedajúci diagram.

```
damy8x8(X):-ukladaj(0,[],X),diagram(X).
ukladaj(8,X,X).
ukladaj(K,L,X):-K<8,I is K+1,stlpec(H),testuj(H,1,L),ukladaj(I,[H|L],X).
stlpec(1).
stlpec(2).
stlpec(3).
stlpec(4).
stlpec(5).
stlpec(6).
stlpec(7).
stlpec(8).
testuj(_,_,[]).
testuj(H,K,[X|T]):-H\=X,A is abs(H-X),A\=K,I is K+1,testuj(H,I,T).
diagram(X):-nl,riadok(X,1,X,X).
riadok([],_,_,_).
riadok([D|T],_,[],X):-write('|'),nl,riadok(T,1,X,X).
riadok([D|T],I,[H|L],X):-D\=I,write('|_'),J is I+1,riadok([D|T],J,L,X).
riadok([D|T],D,[H|L],X):-write('|D'),J is D+1,riadok([D|T],J,L,X).
```

Príklad 36. b). Predikát `damy(N,X)` je jednoduchým zovšeobecnením predikátu `damy8x8(X)`.

```
damy(N,X):-poukladaj(N,0,[],X),diagram(X).
poukladaj(N,N,X,X).
poukladaj(N,K,L,X):-K<N,I is K+1,stlpec(N,H),testuj(H,1,L),poukladaj(N,I,[H|L],X).
stlpec(N,H):-zoznam(N,[],X),member(H,X).
zoznam(0,X,X).
zoznam(N,T,X):-N>0,K is N-1,zoznam(K,[N|T],X).
member(H,[H|_]).
member(X,[_|T]):-member(X,T).
testuj(_,_,[]).
testuj(H,K,[X|T]):-H\=X,A is abs(H-X),A\=K,I is K+1,testuj(H,I,T).
diagram(X):-riadok(X,1,X,X).
riadok([],_,_,_).
riadok([D|T],_,[],X):-write('|'),nl,riadok(T,1,X,X).
riadok([D|T],I,[H|L],X):-D\=I,write('|_'),J is I+1,riadok([D|T],J,L,X).
riadok([D|T],D,[H|L],X):-write('|D'),J is D+1,riadok([D|T],J,L,X).
```

Príklad 37. a).

```
latin(N,X):-zoznam(N,[],Y),stvorec(Y,Y,[],X),vypis_s(N,X).
zoznam(0,X,X).
zoznam(N,T,X):-N>0,K is N-1,zoznam(K,[N|T],X).
stvorec(Y,[],X,X).
stvorec(Y,[A|B],L,X):-perm(Y,H),test(H,L),stvorec(Y,B,[H|L],X).
perm([],[]).
perm(L,[H|T]):-delete(H,L,R),perm(R,T).
delete(H,[H|S],S).
delete(X,[H|S],[H|T]):-delete(X,S,T).
test(_,[]).
test(H,[X|T]):-porovnaj(H,X),test(H,T).
porovnaj([],[]).
porovnaj([A|X],[B|Y]):-A\=B,porovnaj(X,Y).
vypis_s(N,[]).
vypis_s(N,[H|T]):-vypis_r(N,H),vypis_s(N,T).
vypis_r(N,[]):-nl.
vypis_r(N,[H|T]):-A is integer(log(10*N))-integer(log(H)),
     tab(A),write(H),vypis_r(N,T).
```

Príklad 37. b).

```
latin_d(N,X):-zoznam(N,[],Y),stvorec_d(Y,N,0,[],X),vypis_s(N,X).
zoznam(0,X,X).
zoznam(N,T,X):-N>0,K is N-1,zoznam(K,[N|T],X).
stvorec_d(Y,N,N,X,X).
stvorec_d(Y,N,K,L,X):-K<N,I is K+1,perm(Y,H),test(H,L),test_d(N,I,H,L),
     stvorec_d(Y,N,I,[H|L],X).
perm([],[]).
perm(L,[H|T]):-delete(H,L,R),perm(R,T).
delete(H,[H|S],S).
delete(X,[H|S],[H|T]):-delete(X,S,T).
test(_,[]).
test(H,[X|T]):-porovnaj(H,X),test(H,T).
test_d(N,K,H,X):-M is N-K+1,nth_member(K,L,H),nth_member(M,R,H),
     lava_d(K,L,X),prava_d(M,R,X).
nth_member(1,X,[X|_]).
nth_member(N,X,[_|T]):-N>1,M is N-1,nth_member(M,X,T).
lava_d(_,_,[]).
lava_d(K,L,[H|T]):-I is K-1,nth_member(I,A,H),L\=A,lava_d(I,L,T).
prava_d(_,_,[]).
prava_d(M,R,[H|T]):-I is M+1,nth_member(I,A,H),R\=A,prava_d(I,R,T).
vypis_s(N,[]).
vypis_s(N,[H|T]):-vypis_r(N,H),vypis_s(N,T).
vypis_r(N,[]):-nl.
vypis_r(N,[H|T]):-A is integer(log(10*N))-integer(log(H)),
     tab(A),write(H),vypis_r(N,T).
```

Príklady 38. a), b).

```
magic(N,S):-M is N*N,zoznam(M,[],X),zoznam(N,[],Z),K is integer((N^2+1)*N/2),
     m(K,Z,Z,S,[[]],X),vypis_s(N,S).
zoznam(0,X,X).
zoznam(N,T,X):-N>0,K is N-1,zoznam(K,[N|T],X).
m(K,Z,[A|T],S,[H|W],[V|X]):-delete(B,[V|X],Y),m(K,Z,T,S,[[B|H]|W],Y).
m(K,Z,[],S,[H|W],[V|X]):-sucet(H,K),m(K,Z,Z,S,[[],H|W],[V|X]).
m(K,Z,[],[H|W],[H|W],[]):-sucet(H,K),v_sucty(Z,[H|W],K),diag1(Z,[H|W],K),
     diag2(Z,[],[H|W],K).
delete(H,[H|S],S).
delete(X,[H|S],[H|T]):-delete(X,S,T).
sucet([],0).
sucet([H|T],S):-sucet(T,A),S is H+A.
v_sucty([],_,_).
v_sucty([_|Z],S,K):-v_sucet(Z,S,K),v_sucty(Z,S,K).
v_sucet(_,[],0).
v_sucet(Z,[H|S],K):-o_member(Z,H,L),v_sucet(Z,S,M),K is L+M.
o_member([],[B|Y],B).
o_member([A|X],[B|Y],Z):-o_member(X,Y,Z).
diag1(_,[],0).
diag1([_|Z],[H|S],K):-o_member(Z,H,L),diag1(Z,S,M),K is L+M.
diag2([],Z,[],0).
diag2([X|Y],Z,[H|S],K):-o_member(Z,H,L),diag2(Y,[X|Z],S,M),K is L+M.
vypis_s(N,[]).
vypis_s(N,[H|T]):-vypis_r(N,H),vypis_s(N,T).
vypis_r(N,[]):-nl.
vypis_r(N,[H|T]):-A is integer(log(10*N))-integer(log(H)),
     tab(A),write(H),vypis_r(N,T).
```

Príklad 39. Počiatočné podmienky úlohy určuje prvá premenná predikátu `prelievanie(X,A,R)`. Jej hodnotou je zoznam termov tvaru objem/obsah a hodnotou `A` je hľadané množstvo vody. Návod `N` je zoznam inštrukcií v jednom z tvarov

```
[I/(V/A),J/(W/B)]->[I/(V/E),J/(W/F)] a
[I/(V/A)]->[I/(V/0)]
```

znamenajúcich prelievanie z `I`-tej nádoby do `J`-tej nádoby, prípadne vyliatie `I`-tej nádoby.

```
prelievanie(Z,A,N):-ocisluj(Z,X,0),p(X,A,[X],[],R),revert(R,N).
ocisluj([],[],_).
ocisluj([H|X],[K/H|Y],L):-K is L+1,ocisluj(X,Y,K).
p(X,A,[X|K],R,R):-member(_/(_/A),X).
p(X,A,K,T,R):-not_member(_/(_/A),X),ukon(X,Y,U),not_member(Y,K),
     p(Y,A,[Y|K],[U|T],R).
member(H,[H|_]).
member(X,[_|T]):-member(X,T).
not_member(X,[]).
not_member(X,[H|T]):-X\=H,not_member(X,T).
ukon(X,Y,[I/(V/A),J/(W/B)] -> [I/(V/E),J/(W/F)]):-member(I/(V/A),X),
     member(J/(W/B),X),I\=J,A\=0,B\=W,min(A,W-B,C),E is A-C,F is B+C,
     prelej(I,J,E,F,X,Y).
ukon(X,Y,[I/(V/A)] -> [I/(V/0)]):-member(I/(V/A),X),A\=0,prelej(I,I,0,0,X,Y).
prelej(_,_,_,_,[],[]).
prelej(I,J,E,F,[K/(U/V)|X],[K/(U/V)|Y]):-K\=I,K\=J,prelej(I,J,E,F,X,Y).
prelej(I,J,E,F,[I/(V/A)|X],[I/(V/E)|Y]):-I\=J,prelej(I,J,E,F,X,Y).
prelej(I,J,E,F,[J/(W/B)|X],[J/(W/F)|Y]):-I\=J,prelej(I,J,E,F,X,Y).
prelej(I,I,0,0,[I/(V/A)|X],[I/(V/0)|Y]):-prelej(I,I,0,0,X,Y).
min(A,B,A):-A<B.
min(A,B,B):-A>=B.
revert(L,R):-rev(L,[],R).
rev([],R,R).
rev([H|T],A,R):-rev(T,[H|A],R).
```

Príklad 40. a).

```
zapalky:-repeat,
     write($Zadaj zoznam dvoch čísiel pre počty zápaliek v kôpkach: $),
     read([A,B]),A>1,B>1,integer(A),integer(B),[!moja_hra(A,B)!],nl,
     write($Nová hra? (ano/nie): $),read(X),X=nie.
moja_hra(0,0):-prehra.
moja_hra(A,0):-A>1,C is A-1,tah(1,C,1,0).
moja_hra(0,B):-B>1,C is B-1,tah(2,C,0,1).
moja_hra(A,B):-1 is A+B,nl,write($Blahoželám k výhre! $),nl.
moja_hra(1,B):-B>0,tah(2,B,1,0).
moja_hra(A,1):-A>1,tah(1,A,0,1).
moja_hra(A,A):-A>1,I is integer(1+random),E is A-2+I,F is A+1-I,tah(I,1,E,F).
moja_hra(A,B):-A\=B,A>1,B>1,C is abs(A-B),I is (3+(B-A)//C)//2,
     E is A-(A-B)*(2-I),F is B-(B-A)*(I-1),tah(I,C,E,F).
tah(I,C,E,F):-nl,write($Moja voľba je: $),write(I/C),
     write($. Počty zápaliek v kôpkach sú: $),write(1/E+2/F),tvoja_hra(E,F).
tvoja_hra(E,F):-E+F>1,repeat,nl,
     write($Tvoja voľba (číslo kôpky/počet zápaliek.): $),
     read(J/A),(J=1;J=2),1=<A,A=<E*(2-J)+F*(J-1),X is E-A*(2-J),Y is F-A*(J-1),
     write($Počty zápaliek v kôpkach sú: $),write(1/X+2/Y),moja_hra(X,Y).
tvoja_hra(E,F):-1 is E+F,prehra.
prehra:-nl,write($Prehral si! $),nl.
```

Príklad 40. b). Nie je ťažké overiť, že hráč na ťahu je vo vyhrávajúcej pozícií a) ak najviac jedna kôpka má viac ako jednu zápalku a počet všetkých zápaliek je párny, alebo b) ak aspoň v dvoch kôpkach je viac ako po jednej zápalke a v dvojkovom zápise počtov zápaliek je v niektorej pozícií nepárny počet cifier 1. Na vyjadrenie toho použijeme binárne aritmetické operácie \/ a /\, ktoré znamenajú zjednotenie a prienik výskytov jednotiek v dvojkovom zápise.

```
zapalky2:-repeat,
     write($Zadaj zoznam kladnych čísiel pre počty zápaliek v kôpkach: $),
     read(X),ocisluj(X,Y,0),[!moja_hra(Y)!],nl,write($Nová hra? (ano/nie): $),
     read(Z),Z=nie.
ocisluj([],[],_).
ocisluj([H|X],[K/H|Y],L):-H>0,K is L+1,ocisluj(X,Y,K).
moja_hra(X):-prazdne(X),nl,write($Prehral si! $),nl.
moja_hra(X):-aspon_dva(X),parita(X,0),member(I/A,X),A\=0,tah(I,1,X,Z).
moja_hra(X):-aspon_dva(X),parita(X,P),P\=0,rad(P,M),member(I/A,X),
     A is A\/M,B is A-((A\/P)-P),tah(I,B,X,Z).
moja_hra(X):-jeden(X,I/A),parne(X),tah(I,A,X,Z).
moja_hra(X):-jeden(X,I/A),neparne(X),B is A-1,tah(I,B,X,Z).
moja_hra(X):-ziadny(X),member(I/1,X),tah(I,1,X,Z).
prazdne([]).
prazdne([I/0|X]):-prazdne(X).
aspon_dva(X):-member(I/A,X),A>1,member(J/B,X),B>1,I\=J.
jeden([J/A|X],I/B):-A=<1,jeden(X,I/B).
jeden([I/A|X],I/A):-A>1,ziadny(X).
ziadny([]).
ziadny([I/A|X]):-A=<1,ziadny(X).
parita([],0).
parita([I/A|X],Y):-parita(X,B),Y is (A\/B)-(A/\B).
rad(1,1).
rad(P,M):-P>1,Q is P//2,rad(Q,N),M is N*2.
member(H,[H|_]).
member(X,[_|T]):-member(X,T).
parne([]).
parne([I/0|X]):-parne(X).
parne([I/A|X]):-A\=0,neparne(X).
neparne([I/0|X]):-neparne(X).
neparne([I/A|X]):-A\=0,parne(X).
tah(I,A,X,Z):-nl,write($Moja voľba je: $),write(I/A),write($.$),t(I,A,X,Z),nl,
     write($Počty zápaliek v kôpkach sú: $),write(Z),tvoja_hra(Z).
t(I,A,[],[]).
t(I,A,[J/B|X],[J/B|Z]):-I\=J,t(I,A,X,Z).
t(I,A,[I/B|X],[I/C|Z]):-C is B-A,t(I,A,X,Z).
tvoja_hra(X):-member(J/C,X),C>0,repeat,nl,
     write($Tvoja voľba (’číslo kôpky/počet zápaliek.’ alebo ’koniec.’): $),
     read(U),(U=koniec;U=I/A,member(I/B,X),1=<A,A=<B,t(I,A,X,Z),
     write($Počty zápaliek v kôpkach sú: $),write(Z),moja_hra(Z)).
tvoja_hra(X):-prazdne(X),nl,write($Blahoželám k výhre! $),nl.
```

Príklad 41.

```
inverzia(X,Y):-name(X,Z),vymena(Z,W),name(Y,W).
vymena([],[]).
vymena([A|X],[A|Y]):-A<65,vymena(X,Y).
vymena([A|X],[A|Y]):-90<A,A<97,vymena(X,Y).
vymena([A|X],[A|Y]):-122<A,vymena(X,Y).
vymena([A|X],[B|Y]):-65=<A,A=<90,B is A+32,vymena(X,Y).
vymena([A|X],[B|Y]):-97=<A,A=<122,B is A-32,vymena(X,Y).
```

Príklad 42. a), b).

```
jazdec(N,Y):-M is N*N-1,j(N,M,[[1,1]],Y).
j(N,0,Y,Y).
j(N,M,[A|X],Y):-M>0,skok(N,A,B),not_member(B,X),K is M-1,j(N,K,[B,A|X],Y).
skok(N,[X,Y],[S,T]):-Z=[1,2,-1,-2],member(A,Z),member(B,Z),3 is abs(A)+abs(B),
     S is X+A,T is Y+B,1=<S,S=<N,1=<T,T=<N.
not_member(X,[H|T]):-X\=H,not_member(X,T).
not_member(X,[]).
member(H,[H|_]).
member(X,[_|T]):-member(X,T).
jazdec2(N,M,Y):-j2(N,1,M,[[1,1]],Y).
j2(N,K,M,[A|X],Y):-smie(N,X,A,B),L is K+1,j2(N,L,M,[B,A|X],Y).
j2(N,M,M,[A|X],[A|X]):-not smie(N,X,A,_).
smie(N,X,A,B):-skok(N,A,B),not_member(B,X).
```

Príklady 43. a), b).

```
algebrogram(X,Y,Z):-revert(X,A),revert(Y,B),revert(Z,C),
     ag(A,B,C,0,[0,1,2,3,4,5,6,7,8,9]),pivot(X),pivot(Y),pivot(Z).
revert(L,R):-rev(L,[],R).
rev([],R,R).
rev([H|T],A,R):-rev(T,[H|A],R).
ag([A|X],[B|Y],[C|Z],S,L):-volba(A,L,L1),volba(B,L1,L2),volba(C,L2,L3),
     C is (A+B+S) mod 10,S1 is integer((A+B+S)/10),ag(X,Y,Z,S1,L3).
ag([],[B|Y],[C|Z],S,L):-ag([0],[B|Y],[C|Z],S,L).
ag([A|Y],[],[C|Z],S,L):-ag([A|Y],[0],[C|Z],S,L).
ag([],[],[C],S,L):-volba(C,L,_),nonvar(C),C=S.
ag([],[],[],0,_).
volba(X,[],[]):-nonvar(X).
volba(H,[H|T],T).
volba(X,[H|T],[H|L]):-volba(X,T,L).
pivot([H|_]):-H\=0.
algebrogram2(X,Y,Z):-write(X+Y=Z),nl,revert2(X,A),revert2(Y,B),revert2(Z,C),
     ag2(A,B,C,0,[],[0,1,2,3,4,5,6,7,8,9]),pivot2(A),pivot2(B),pivot2(C),
     vypis(A),write(' + '),vypis(B),write(' = '),vypis(C),nl,fail.
revert2(X,Y):-name(X,L),rev2(L,[],Y).
rev2([],R,R).
rev2([H|T],L,R):-rev2(T,[H/_|L],R).
ag2([A/I|X],[B/J|Y],[C/K|Z],S,L,T):-volba(A/I,L,L1,T,T1),volba(B/J,L1,L2,T1,T2),
     volba(C/K,L2,L3,T2,T3),K is (I+J+S) mod 10,S1 is integer((I+J+S)/10),
     ag2(X,Y,Z,S1,L3,T3).
ag2([],[B|Y],[C|Z],S,L,T):-ag2([_/0],[B|Y],[C|Z],S,L,T).
ag2([A|Y],[],[C|Z],S,L,T):-ag2([A|Y],[0],[C|Z],S,L,T).
ag2([],[],[C/K],S,L,T):-volba(C/K,L,_,T,_),nonvar(K),K=S.
ag2([],[],[],0,_,_).
volba(X/Y,L,L,T,T):-member(X/Y,L).
volba(X/Y,L,[X/Y|L],T,T1):-not_member(X/Y,L),delete(Y,T,T1).
member(H,[H|_]).
member(X,[_|T]):-member(X,T).
not_member(X,[]).
not_member(X,[H|T]):-X\=H,not_member(X,T).
delete(H,[H|S],S).
delete(X,[H|S],[H|T]):-delete(X,S,T).
pivot2(X):-last(A/B,X),B\=0.
last(L,[L]).
last(L,[_|T]):-last(L,T).
vypis([]).
vypis([X/Y|T]):-vypis(T),write(Y).
```

Príklad 44.

```
farby([biela,'červená','modrá','zelená','žltá']).
narodnosti(['Angličan','Dán','Nemec','Nór','Švéd']).
napoje(['čaj','káva',mlieko,pivo,voda]).
cigarety(['Pall Mall','Dunhill','Blend','Blue Master','Prince']).
zvierata([pes,'vták','mačka','kôň',ryba]).
perm([],[]).
perm(L,[H|T]):-delete(H,L,R),perm(R,T).
delete(H,[H|S],S).
delete(X,[H|S],[H|T]):-delete(X,S,T).
nth_member(1,X,[X|_]).
nth_member(N,X,[_|T]):-N>1,K is N-1,nth_member(K,X,T).
kth_member(1,X,[X|_]).
kth_member(K,X,[H|T]):-X\=H,kth_member(N,X,T),K is N+1.
obsadenie([A,B,C,D,E]):-
  nth_member(3,'mlieko',C),                        % 8.
  (nth_member(1,'Nór',B);nth_member(5,'Nór',B)),  % 9.
  narodnosti(B0),perm(B0,B),
  kth_member(I1,'Angličan',B),nth_member(I1,'červená',A),
  kth_member(I2,'Švéd',B),nth_member(I2,pes,E),
  kth_member(I3,'Dán',B),nth_member(I3,'čaj',C),
  kth_member(I13,'Nemec',B),nth_member(I13,'Prince',D),
  farby(A0),perm(A0,A),
  kth_member(I5,'zelená',A),nth_member(I5,'káva',C),
  kth_member(I7,'žltá',A),nth_member(I7,'Dunhill',D),
  napoje(C0),perm(C0,C),
  cigarety(D0),perm(D0,D),
  kth_member(I6,'Pall Mall',D),nth_member(I6,'vták',E),
  kth_member(I12,'Blue Master',D),nth_member(I12,pivo,C),
  zvierata(E0),perm(E0,E),
  kth_member(I4,'zelená',A),kth_member(J4,biela,A),I4<J4,
  kth_member(I10,'Blend',D),kth_member(J10,'mačka',E),1 is abs(I10-J10),
  kth_member(I11,'kôň',E),kth_member(J11,'Dunhill',D),1 is abs(I11-J11),
  kth_member(I14,'Nór',B),kth_member(J14,'modrá',A),1 is abs(I14-J14),
  kth_member(I15,'Blend',D),kth_member(J15,voda,C),1 is abs(I15-J15).
kto_chova_ryby(X):-obsadenie(Y),nth_member(5,Z,Y),nth_member(2,N,Y),
  kth_member(K,ryba,Z),nth_member(K,X,N).
```

Príklad 45. a). `preklad(N,X)` prevedie číslo na zoznam, ktorého prvky sú zoznamy elementárnych jazykových slabík a slov vyjadrujúce príslušné číselné skupiny; `preklad(N,X)` prevedie číslo na zoznam číselných skupín v slovnom vyjdrení; `vypis(N)` napíše slovné vyjadrenie čísla X. Číslo `N` nesmie byť väčšie ako 2 147 483 647.

```
vypis(N):-prepis(N,Y),tlac(Y).
tlac([A|B]):-write(A),tab(1),tlac(B).
prepis(N,X):-preklad(N,Y),uprava(Y,X).
uprava([],[]).
uprava([H|T],[A|X]):-zlep(H,A),uprava(T,X).
zlep([X],X).
zlep([H|T],X):-T\=[],zlep(T,G),name(H,E),name(G,F),append(E,F,A),name(X,A).
append([],L,L).
append([H|T],S,[H|L]):-append(T,S,L).
preklad(0,[[nula]]).
preklad(N,X):-N>0,A is N mod 1000, B is N//1000,do_tisic(A,Y),nad_tisic(1,B,Z),
     append(Z,Y,X).
do_tisic(0,[]).
do_tisic(N,[X]):-N>0,A is N mod 100,B is N//100,do_sto(A,Y),nad_sto(B,Z),
     append(Z,Y,X).
do_sto(0,[]).
```

```
do_sto(N,X):-N>0,A is N mod 10,B is N//10,dve_cifry(B,A,X).
dve_cifry(0,A,[X]):-do_10(A,X).
dve_cifry(1,A,[X]):-do_20(A,X).
dve_cifry(B,0,[X,Y]):-B>1,do_10(B,X),volba(B,['dsať',desiat],Y).
dve_cifry(B,A,[X,Y,Z]):-B>1,A>0,do_10(B,X),volba(B,['dsať',desiat],Y),do_10(A,Z).
do_10(X,Y):-member(X/Y,[0/nula,1/jeden,2/dva,3/tri,4/'štyri',5/'päť',6/'šesť',
     7/sedem,8/osem,9/'deväť']).
member(H,[H|_]).
member(X,[_|T]):-member(X,T).
do_20(X,Y):-member(X/Y,[0/'desať',1/'jedenásť',2/'dvanásť',3/'trinásť',4/'štrnásť'
     ,5/'pätnásť',6/'šestnásť',7/'sedemnásť',8/'osemnásť',9/'deväťnásť']).
volba(X,[Y,_],Y):-X<5.
volba(X,[_,Y],Y):-X>=5.
nad_sto(0,[]).
nad_sto(1,[sto]).
nad_sto(2,[dve,sto]).
nad_sto(N,[X,sto]):-N>2,do_10(N,X).
nad_tisic(K,0,[]).
nad_tisic(K,N,X):-N>0,A is N mod 1000,B is N//1000,M is K+1,nad_tisic(M,B,Z),
     tri_cifry(K,A,Y),append(Z,Y,X).
tri_cifry(K,0,[]).
tri_cifry(K,N,X):-N>0,I is K//2,J is K mod 2,rad(I,J,N,X).
rad(0,1,1,[['tisíc']]).
rad(0,1,2,[[dve,'tisíc']]).
rad(0,1,N,[X]):-N>2,do_tisic(N,[Y]),append(Y,['tisíc'],X).
rad(I,J,1,[[X,Y]]):-I>0,predpona(I,X),member(J/Y,[0/'lión',1/liarda]).
rad(I,J,2,[[A],[X,Y]]):-I>0,member(J/A,[0/dva,1/dve]),predpona(I,X),
     member(J/Y,[0/'lióny',1/liardy]).
rad(I,0,N,[A,[X,Y]]):-I>0,N>2,do_tisic(N,[A]),
     predpona(I,X),volba(N,['lióny','liónov'],Y).
rad(I,1,N,[A,[X,Y]]):-I>0,N>2,do_tisic(N,[A]),
     predpona(I,X),volba(N,[liardy,'liárd'],Y).
predpona(I,Y):-member(I/Y,[1/mi,2/bi,3/tri,4/kvadri,5/penti,6/sexti,7/septi,
     8/okti,9/nona,10/deka]).
predpona(I,I):-I>10.
```

Príklad 45. b). `X` v `preklad2(X,Y)`, `prepis2(X,Y)`, `vypis2(X)`, je atóm pozostávajúci z číslic.

```
vypis2(X):-prepis2(X,Y),tlac(Y).
tlac([A|B]):-write(A),tab(1),tlac(B).
prepis2(X,Y):-preklad2(X,Z),uprava(Z,Y).
uprava([],[]).
uprava([H|T],[A|X]):-zlep(H,A),uprava(T,X).
zlep([X],X).
zlep([H|T],X):-T\=[],zlep(T,G),name(H,E),name(G,F),append(E,F,A),name(X,A).
preklad2(X,Y):-name(X,A),obrat(A,[],B),prevod(B,Y).
obrat([],B,B).
obrat([H|T],S,B):-name(A,[H]),obrat(T,[A|S],B).
prevod([A],[[X]]):-do_10(A,X).
prevod([A,B],[X]):-do_sto2(B,A,X).
prevod([A,B,C|T],X):-do_tisic2(C,B,A,Y),nad_tisic2(1,T,Z),append(Z,Y,X).
do_tisic2(0,0,0,[]).
do_tisic2(C,B,A,[X]):-C+B+A\=0+0+0,do_sto2(B,A,Y),nad_sto(C,Z),append(Z,Y,X).
do_sto2(0,0,[]).
do_sto2(0,A,[X]):-A\=0,do_10(A,X).
do_sto2(1,A,[X]):-do_20(A,X).
do_sto2(B,0,[X,Y]):-B>1,do_10(B,X),volba(B,['dsať',desiat],Y).
do_sto2(B,A,[X,Y,Z]):-B>1,A>0,do_10(B,X),volba(B,['dsať',desiat],Y),do_10(A,Z).
do_10(X,Y):-member(X/Y,[0/nula,1/jeden,2/dva,3/tri,4/'štyri',5/'päť',6/'šesť',
```

```
     7/sedem,8/osem,9/'deväť']).
member(H,[H|_]).
member(X,[_|T]):-member(X,T).
do_20(X,Y):-member(X/Y,[0/'desať',1/'jedenásť',2/'dvanásť',3/'trinásť',4/'štrnásť'
     ,5/'pätnásť',6/'šestnásť',7/'sedemnásť',8/'osemnásť',9/'devätnásť']).
volba(X,[Y,_],Y):-X<5.
volba(X,[_,Y],Y):-X>=5.
nad_sto(0,[]).
nad_sto(1,[sto]).
nad_sto(2,[dve,sto]).
nad_sto(N,[X,sto]):-N>2,do_10(N,X).
append([],L,L).
append([H|T],S,[H|L]):-append(T,S,L).
nad_tisic2(K,[],[]).
nad_tisic2(K,[A],X):-nad_tisic2(K,[A,0,0],X).
nad_tisic2(K,[A,B],X):-nad_tisic2(K,[A,B,0],X).
nad_tisic2(K,[0,0,0|T],X):-M is K+1,nad_tisic2(M,T,X).
nad_tisic2(K,[A,B,C|T],X):-A+B+C\=0+0+0,I is K//2,J is K mod 2,M is K+1,
     nad_tisic2(M,T,Z),rad2(I,J,[C,B,A],Y),append(Z,Y,X).
rad2(0,1,[0,0,1],[['tisíc']]).
rad2(0,1,[0,0,2],[[dve,'tisíc']]).
rad2(0,1,[C,B,A],[X]):-[C,B,A]\=[0,0,1],[C,B,A]\=[0,0,2],do_tisic2(C,B,A,[Y]),
     append(Y,['tisíc'],X).
rad2(I,J,[0,0,1],[[X,Y]]):-I>0,predpona(I,X),member(J/Y,[0/'lión',1/liarda]).
rad2(I,J,[0,0,2],[[A],[X,Y]]):-I>0,member(J/A,[0/dva,1/dve]),predpona(I,X),
     member(J/Y,[0/'lióny',1/liardy]).
rad2(I,0,[C,B,A],[S,[X,Y]]):-I>0,[C,B,A]\=[0,0,1],[C,B,A]\=[0,0,2],
     do_tisic2(C,B,A,[S]),predpona(I,X),N is A+10*(B+10*C),
     volba(N,['lióny','liónov'],Y).
rad2(I,1,[C,B,A],[S,[X,Y]]):-I>0,[C,B,A]\=[0,0,1],[C,B,A]\=[0,0,2],
     do_tisic2(C,B,A,[S]),predpona(I,X),N is A+10*(B+10*C),
     volba(N,[liardy,'liárd'],Y).
predpona(I,Y):-member(I/Y,[1/mi,2/bi,3/tri,4/kvadri,5/penti,6/sexti,7/septi,
     8/okti,9/nona,10/deka]).
predpona(I,I):-I>10.
```

Príklad 46. a) (1), (2). V rímskom zápise pozície dekadických cifier rádov 0, 1, alebo 2, možno vyjadriť vhodnou (vždy rovnakou) variáciou trojice písmen `IVX`, `XLC`, alebo `CDM`.

```
riman(0,[]).
riman(N,X):-N>0,cislice(N,A,B,C,D),ra(D,'I','V','X',[],U),
     ra(C,'X','L','C',U,V),ra(B,'C','D','M',V,W),rb(A,'M',W,X).
cislice(N,A,B,C,D):-A is N//1000,B is (N mod 1000)//100,C is (N mod 100)//10,
     D is N mod 10.
rim_zapis(N):-riman(N,X),tlac2(X).
tlac2([A|B]):-write(A),tlac2(B).
ra(0,I,V,X,T,T).
ra(1,I,V,X,T,[I|T]).
ra(2,I,V,X,T,[I,I|T]).
ra(3,I,V,X,T,[I,I,I|T]).
ra(4,I,V,X,T,[I,V|T]).
ra(5,I,V,X,T,[V|T]).
ra(6,I,V,X,T,[V,I|T]).
ra(7,I,V,X,T,[V,I,I|T]).
ra(8,I,V,X,T,[V,I,I,I|T]).
ra(9,I,V,X,T,[I,X|T]).
rb(0,M,T,T).
rb(N,M,S,T):-N>0,K is N-1,rb(K,M,[M|S],T).
riman2(0,[]).
```

```
riman2(N,X):-N>0,cislice(N,A,B,C,D),ra(D,‘I,‘V,‘X,[],U),ra(C,‘X,‘L,‘C,U,V),
     ra(B,‘C,‘D,‘M,V,W),rb(A,‘M,W,T),name(X,T).
```

Príklad 46. b). Pri vyhodnotení rímskeho zápisu sa jednotlivé písmená nahradia ich hodnotami. Tie hodnoty, ktoré predchádzajú väčším číselným hodnotám sa odčítavajú a ostatné sa pripočítavajú.

```
arabsky(X,Y):-name(X,Z),arab(Z,V),Y is V.
arab([],0).
arab(X,A+B):-X\=[],cifra(X,Z,A),Z\=[],arab(Z,B).
arab(X,Y):-X\=[],cifra(X,[],Y).
cifra([H],[],I):-rc(H,I).
cifra([H,G|X],X,-I+J):-rc(H,I),rc(G,J),I<J.
cifra([H,G|X],[G|X],I):-rc(H,I),rc(G,J),J=<I.
rc(‘I,1).
rc(‘V,5).
rc(‘X,10).
rc(‘L,50).
rc(‘C,100).
rc(‘D,500).
rc(‘M,1000).
```

Príklad 47, 48.

```
% a)
scitanie(P,X,Y,Z):-s(P,X,Y,0,Z).
s(P,[A|X],[B|Y],I,[C|Z]):-C is (A+B+I) mod P,J is (A+B+I)//P,s(P,X,Y,J,Z).
s(P,[],[],0,[]).
s(P,[],[],I,[I]):-I>0.
s(P,[],[B|Y],I,Z):-s(P,[0],[B|Y],I,Z).
s(P,[A|X],[],I,Z):-s(P,[A|X],[0],I,Z).
% b)
nasobenie(P,[0],X,[0]).
nasobenie(P,X,[0],[0]).
nasobenie(P,X,Y,Z):-X\=[0],Y\=[0],nasob(P,X,Y,[0],Z).
nasob(P,X,[],Z,Z).
nasob(P,X,[A|Y],T,Z):-n(P,X,A,0,B),scitanie(P,T,B,C),nasob(P,[0|X],Y,C,Z).
n(P,[A|X],B,I,[C|Z]):-C is (A*B+I) mod P,J is (A*B+I)//P,n(P,X,B,J,Z).
n(P,[],B,0,[]).
n(P,[],B,I,[I]):-I>0.
% c)
odcitanie(P,X,X,[0]).
odcitanie(P,X,Y,Z):-X\=Y,o(P,X,Y,Z). % X-Y=Z
o(P,[A|X],[B|X],[C]):-C is A-B,C>0.
o(P,[A|X],[B|X],[]):-0 is A-B.
o(P,[A|X],[B|Y],[C|Z]):-X\=Y,C is A-B,C>=0,o(P,X,Y,Z).
o(P,[A|X],[B|Y],[C|Z]):-C is P+A-B,C<P,scitanie(P,Y,[1],V),o(P,X,V,Z).
o(P,[],[],[]).
o(P,[A|X],[],[A|X]).
% d)
delenie(P,X,Y,Q,R):-revert(X,A),d0(P,A,[],Y,Q,R).
d0(P,[],R,Y,[0],R).
d0(P,[A|X],T,Y,Q,R):-vacsi(Y,[A|T]),d0(P,X,[A|T],Y,Q,R).
d0(P,[A|X],T,Y,Q,R):-nie_vacsi(Y,[A|T]),d1(P,[A|X],T,Y,[],Q,R).
vacsi([_|_],[]).
vacsi([A|X],[B|X]):-A>B.
vacsi([A|X],[B|Y]):-A=<B,vacsi(X,Y).
vacsi([A|X],[B|Y]):-A>B,vacsi(X,Y).
nie_vacsi(X,X).
nie_vacsi(X,Y):-vacsi(Y,X).
d1(P,[A|X],[0],Y,S,Q,R):-krok_delenia(P,[A],Y,I,W),d1(P,X,W,Y,[I|S],Q,R).
d1(P,[A|X],T,Y,S,Q,R):-T\=[0],krok_delenia(P,[A|T],Y,I,W),d1(P,X,W,Y,[I|S],Q,R).
```

```
d1(P,[],R,Y,Q,Q,R).
krok_delenia(P,U,Y,I,W):-kd(P,U,Y,I,W,0).
kd(P,W,Y,I,W,I):-vacsi(Y,W).
kd(P,U,Y,I,W,J):-odcitanie(P,U,Y,V),K is J+1,kd(P,V,Y,I,W,K).
revert(L,R):-rev(L,[],R).
rev([],R,R).
rev([H|T],A,R):-rev(T,[H|A],R).
% e)
odmocnina(P,X,Y):-revert(X,A),od(P,A,Y).
od(P,[A|X],Y):-parne(X),odm(P,[A],X,[],Y).
od(P,[A,B|X],Y):-parne(X),odm(P,[B,A],X,[],Y).
parne([]).
parne([A,B|X]):-parne(X).
odm(P,A,[],R,[I|R]):-J is P-1,krok_odm(P,A,R,J,I,Z).
odm(P,A,[B,C|X],R,Y):-J is P-1,krok_odm(P,A,R,J,I,Z),odm(P,[C,B|Z],X,[I|R],Y).
krok_odm(P,A,R,J,J,[]):-'(2pr+j)j'(P,R,J,A).
krok_odm(P,A,R,J,J,Z):-'(2pr+j)j'(P,R,J,B),A\=B,odcitanie(P,A,B,Z).
krok_odm(P,A,R,J,I,Z):-'(2pr+j)j'(P,R,J,B),vacsi(B,A),K is J-1,
     krok_odm(P,A,R,K,I,Z).
'(2pr+j)j'(P,[],J,X):-nasobenie(P,[J],[J],X).
'(2pr+j)j'(P,R,J,X):-R\=[],nasobenie(P,[0,2],R,A),scitanie(P,A,[J],B),
     nasobenie(P,B,[J],X).
% f)
pq_prevod(P,Q,[0],[0]):-P>1,Q>1.
pq_prevod(P,Q,X,Y):-P>1,Q>1,X\=[0],p_zapis(P,Q,A),pq(P,Q,X,A,[],Y).
pq(P,Q,[0],A,Y,Y).
pq(P,Q,X,A,S,Y):-X\=[0],delenie(P,X,A,E,F),hodnota(P,F,B),p_zapis(Q,B,T),
     append(S,T,U),pq(P,Q,E,A,U,Y).
p_zapis(P,N,[N]):-N<P.
p_zapis(P,N,[A|X]):-N>=P,A is N mod P,M is N//P,p_zapis(P,M,X).
hodnota(P,[A|X],Y):-hodnota(P,X,B),Y is P*B+A.
hodnota(P,[],0).
append([],L,L).
append([H|T],S,[H|L]):-append(T,S,L).
```

Literatúra

1. R. Barták *Learning Prolog via examples, Interactive Prolog guide*, 1997, 42 strán.
2. W. F. Clocksin, C. S. Mellish, *Programming in Prolog,* Springer-Verlag, New York, 1987.
3. J. Csontó, *Aplikácie jazyka Prolog v UI*, Elektrotechnická fakulta TU v Košiciach, Košice, 1992.
4. P. Töpfer, *Sbírka úloh z programování*, Matematicko-fyzikální fakulta UK, Praha, 1990.
5. Using the Arity/Prolog compiler and interpreter, Arity Corporation, 1987.
6. The Arity/Prolog language reference manual, Arity Corporation, 1988.