# Úvod

*venované* Richardovi Nemcovi

Brožúrka, ktorú práve držíte v rukách, obsahuje pedagogické materiály k prednáške 'Logické programovanie' vzniknuvšie v školskom roku 1992/1993. Tento materiál má pedagogické poslanie a ťažko v ňom hľadať výsledky skutočne 'vedeckej' činnosti.

V roku 1992/1993 sme pre účely prednášky vypracovali sadu transparentných fólií, ktoré sme počas prednášok premietali na meotare. Fólie obsahujú základne myšlienky, definície a tvrdenia, o ktorých sa hovorí na prednáške. Neobsahujú veľa vysvetľujúcich komentárov, nakoľko sa predpokladá, že študent ich počuje na prednáške. Rovnako tak neobsahujú dôkazy jednotlivých tvrdení, ktoré sú prezentované na prednáške.

Súbor fólií, ktoré takto počas semestra vznikol, sa stal užitočným materiálom pri príprave študentov na skúšku (resp. zápočet). Zároveň aspoň čiastočne simuloval nedostupnú literatúru z tejto oblasti a chýbajúce skriptá. Najmä tieto dva dôvody nás viedli k vydaniu tejto technickej správy, ktorá je akousi bodkou za prednáškou 1992/1993 a zároveň istou prípravou na rok 1993/1994. Tento materiál zďaleka nedosahuje úroveň skrípt, preto nemôže byť použitý ako samostatný študijný materiál (bez absolvovania prednášok je len veľmi ťažko zrozumiteľný). Študenti ho však efektívne používali ako 'poznámkový blok' s predpísanými hlavnými bodmi, do ktorých si značili detaily, vysvetlenia a dôkazy, ktoré sa dozvedeli na prednáške. Myslíme, že takéto využitie tohoto materiálu je najadekvátnejšie.

Predpokladaná životnosť tohoto materiálu je do roku 1994, kedy po ďalšom semestri prednášok plánujeme vydať ďalší 'upgrade'. Dúfame, že limitou týchto materiálov po niekoľkých rokoch bude skriptum predmetu 'Logické programovanie', ak sa dovtedy ešte tento predmet bude prednášať.

Mimoriadne atraktívna časť tohoto materiálu (pre študentov) je časť, ktorá obsahuje zadania všetkých písomných testov (tzv. besiedok), zápočtových písomiek, skúškových písomiek. Taktiež tu môžete nájsť aj zopár prémiových úloh, ktoré boli vypísané v tomto školskom roku. Zverejnenie týchto materiálov študentom nasledujúcich ročníkov chápte ako 'fair-play' krok. Prakticky to znamená, že túto prácu bude treba vykonať znova, od čoho si sľubujeme ďalšie skvalitnenie prednášky a cvičení. Zároveň sa predpokladá, že podobné typy príkladov by v budúcnosti nemali študentov priveľmi zaskočiť.

Záverom by som chcel poďakovať *Richardovi Nemcovi*, s ktorým už niekoľko rokov cvičíme študentov k tejto prednáške. *Rišo* výraznou mierou prispel k príprave besiedok, písomiek a prémiových príkladov. Len jeho trvalá zaneprázdnenosť zapríčinila, že tento materiál nie je ešte kvalitnejší.

Rovnako tiež ďakujem študentom 3. ročníka 1992/1993 (najmä Monike Obrancovej), ktorí pomohli vychytať a opraviť najhrubšie chyby a prieklepy v tomto materiály.

V neposlednej rade sa chcem poďakovať Ivici Chačaturianovej, ktorá celý text prezrela, opravila a doplnila s dôslednosťou, ktorá mi dych vyrážala.

*autor*

# LOGICKÉ PROGRAMOVANIE

sylabus prednášky,
rozsah: 2+2 (3.ročník),
verzia: 1993/1994

- **Úvod do deklaratívneho a logického programovania,**

  Paradigmy: klasické programovanie vs. deklaratívne programovanie,

  funkcionálne vs. logické programovanie,

- **Rekurzívne programy a dátové štruktúry,**

  čísla: prirodzené čísla, celé a racionálne čísla,

  množiny: množinové operácie,

  databázy: a databázové operácie,

  zoznamy: práca so zoznamami,

  stromy: binárne stromy, binárne vyhľadávacie stromy, všeobecné stromy a lesy,

  operátory,

- **Logické programy, interpretácie a modely logických programov,**

  jazyk prvého rádu, termy, formuly, Herbrandove univerzum, Herbrandova báza, prefixová normálna forma, Skolemova normálna forma, klauzálna forma, rezolučný princíp, strom dôkazu, interpretácia prvorádového jazyka, model, Herbrandovské interpretácie a modely, najmenší Herbrandov model,

- **Monotónne a spojité zobrazenia na úplných zväzoch, pevné body,**

  operátor $T_P$, úplné zväzy,spojitosť, monotónnosť, pevný bod, Tarskeho veta,

  konštrukcia ordinálnych čísel, stromy ordinálnych čísel, iterácia zobrazenia na ordinálne číslo,

  Kleeneho veta,

- **Odpoveďová substitúcia, unifikácia, algoritmy unifikácie,**

  Substitúcia, základné algoritmy unifikácie, problém occur-checku,

  abstraktný interpreter logických programov,

  korektná odpoveďová substitúcia,

- **SLD–rezolúcia, bezospornosť,**

  SLD–strom, SLD–odvodenie, SLD–bezospornosť,

- **Úplnosť SLD rezolúcie, procedurálna sémantika,**

  úplnosť SLD–rezolúcie, nezávislosť výpočtového pravidla,

- **Vypočítateľnosť,**

  vzťah logických programov k čiastočne rekurzívnym funkciám,

- **Vzťah logických dôkazov a typov,**

  prirodzená dedukcia, kalkulus sekventov, Curry – Howardov izomorfizmus, transformácie dôkazov,

- **Náročnejšie techniky programovania v Prologu:**

  1. **Rekurzia a iterácia,**

     technika akumulátorov, technika zásobníka, odstraňovanie stromovej rekurzie, použitie logickej premennej pri odstraňovaní rekurzie,

     rekurzie vyšších ordinálnych rádov,

     dokazovanie konečnosti iteratívnych programov,

  2. **Nedeterministické programovanie: generuj a testuj,**

     backtracking, logické premenné,

  3. **Technika neúplných dátových štruktúr:**

     rozdiel zoznamov, rozdiel stromov,

  4. **Extra – logické programovanie**

     operátor rezu,

     masážovanie databázy: `assert`, `retract`,

  5. **Manipulácia s termami**

     konverzie termov do zoznamov,

     modifikácie unifikačných algoritmov,

  6. **Definite Clause Grammars a syntaktická analýza,**

  7. **Druhorádové programovanie**

     množinové predikáty: `setof`, `bagof`, `forall`

  8. **Meta – programovanie**

     meta – interpretácia a čiastočná interpretácia logických programov, sledovanie výpočtu (debugging) logických programov,

- **Logické programovanie s ohraničeniami (constraints)**

  základná myšlienka CLP - Constraint Logic Programming,

  úvod do CLP(R), všeobecná schéma CLP(A) systémov,

  spĺňanie ohraničení (constraint satisfaction), hierarchické CLP,

  informácia o iných CLP systémoch - Prolog III, CHIP, Trilogy I,

# Literatúra


1. Sterling,L., Shapiro,E.: *The Art of Prolog - Advanced Programming Techniques*, Second Edition, The MIT Press, 1986,
2. Lloyd,J.W.: *Foundations of Logic Programming*, Springer Verlag, Second Edition, 1987,
3. Apt,K.R.: *Introduction to Logic Programming*, (Revised and Extended Version, Report CS-R8826, Centre for Mathematics and Computer Science, The Netherlands, 1988,
4. Kelemen, J. a kol.: *Základy umelej inteligencie*, Alfa, 1992,
5. Jirků,P.: *Programování v jazyku Prologu*, SNTL, Praha, 1992,
6. Cloksin,W.F., Melish,C.S: *Programming in Prolog*, Second edition, Springer Verlag, 1984,
7. Hogger,J.: *Essentials of Logic Programming*, Imperial College of Science and Technology, London, 1990,
8. Hogger,J.: *Introduction to Logic Programming*, Academic Press, London, 1984,
9. Bratko,I: *Programovací jazyky pro umělou inteligenci*, Sofsem '92,
10. Coelho,H., Otta, J.C, Pereira,L.M.: *How to solve it with Prolog*, Lisboa, Laboratorio Nacional de Engenharia Civil, 1982,
11. Pereira,L.M.: *Prolog by Examples*, Springer Verlag, 1987,
12. Štepánek, P., Štepánková,O. : *Logické programování a Prolog*, in Ročenka výpočetní techniky 1, SNTL, 1989,
13. Girard, J.Y., Lafont, Y., Taylor, P.: *Proofs and Types*, Cambridge University Press, 1989,
14. Matyska, L.: *Constraint Logic Programming*, zborník SOFSEMu '93, VUSEI–AR Bratislava, 1993.

# LOGICKÉ PROGRAMOVANIE

Peter Borovanský,
UI MFF UK, UI-21,
tel.: 720003/253,
borovan@fmph.uniba.sk

PLÁN:

1. Úvod do deklaratívneho a logického programovania,
2. Rekurzívne programy a dátové štruktúry,
3. Logické programy, interpretácie a modely logických programov,
4. Monotónne a spojité zobrazenia na úplných zväzoch, pevné body,
5. Odpoveďová substitúcia, unifikácia, algoritmy unifikácie,
6. SLD–rezolúcia, bezospornosť,
7. Úplnosť SLD rezolúcie, procedurálna sémantika,
8. Vypočítateľnosť,
9. Vzťah logických dôkazov k typom,
10. Advancované techniky programovania v Prolog
11. Rekurzia a iterácia,
12. Nedeterministické programovanie: generuj a testuj,
13. Technika neúplných dátových štruktúr:
14. Extra – logické programovanie
15. Manipulácia s termami
16. Definite Clause Grammars a syntaktická analýza,
17. Druhorádové programovanie
18. Meta – programovanie
19. Logické programovanie s constraintami

# FORMALITY

PODMIENKY:

- písomka po teoretickej časti (15 bodov),
- písomka pred záverom semestra (20 bodov),
- termín písomiek (mimo výuky) bude známy týždeň vopred, neexistuje náhradný termín,
- prémiové úlohy počas semestra (20 bodov),
- prémie: (1-3 body), časovo ohraničené, oznamovanie na nástenke, automaticky testované, zvláštna bonifikácia za opisovanie,
- riadny zápočet získava každý, kto má aspoň 21 bodov na konci zápočtového týždňa,
- na opravný zápočet možno získať body na zápočtovej písomke (10 bodov),
- opravný zápočet získava každý, kto má aspoň 21 bodov po zápočtovej písomke,
- počas semestra možno získať najviac 35 bodov,
- na skúšku môže ísť len ten, kto má zapísaný zápočet,
- skúšku nemusí robiť každý, kto cez semester získa aspoň 30 bodov z písomiek,
- skúška (30 bodov – 10 bodov teoretická časť, 20 bodov praktická časť),
- známka na skúške

| | |
|---|---|
| aspoň 50 bodov | výborný |
| aspoň 40 bodov | veľmi dobrý |
| aspoň 30 bodov | dobrý |
| menej ako 30 bodov | nevyhovel |

# DEKLARATÍVNE PROGRAMOVANIE

## PARADIGMY:

*klasické programovanie* $Algorithms + Data = Programs$,
primárny je *popis algoritmu* a dát, t.j. spôsob 'obrábania' dát,
vznik štruktúrovaných programovacích jazykov procedurálneho typu (Pascal, C, ..., Modula, Ada), zdokonaľovanie dátových štruktúr, modulárne programovanie, polymorfizmus,

*deklaratívne programovanie* časti programu definujú *matematické relácie* (resp. špecifikácie) nad dátovými objektami,
primárne je navrhnúť korektné a konzistentné špecifikácie pre daný problém, implementácia špecifikácií je sekundárny problém,
vznik jazykov, resp. systémov podporujúcich deklaratívne programovanie (funkcionálne a logické jazyky – Lisp, Prolog, pravidlové a term-prepisujúce systémy, jazyk relačných databáz, ...),
*sémantika programov* – akú reláciu počíta konkrétny program,
*rýchle prototypovanie* a *programovanie vo veľkom*,
*syntéza programov z formálnych špecifikácií* – automatické metódy konštrukcie programu, ktorý počíta formálne popísanú reláciu,

*logické programovanie* $Algorithms = Logic + Control$, Kowalski,
$Logic$ – čo algoritmus robí (deklaratívna zložka algoritmu)
$Control$ – ako to robí (procedurálna zložka algoritmu),
primárne je, čo algoritmus robí, sekundárne ako to robí,
vlastnosti riešenia sa deklarujú v *logickom jazyku*, t.j. *logický program* je konečná množina formúl (*logická teória*), *výpočet* $L_P$ zodpovedá dôkazu v tejto teórii,
*rezolučné techniky* – automatické metódy hľadania dôkazov za istých predpokladov kladených na tvar $L_P$

```
var x:integer;
procedure Foo(var i:integer);
begin
  x:=x-2*i-1;
  if x>=0 then
    begin i:=i+1; Foo(i) end
end;
i:=0; Foo(i); writeln(i);
```

```
for(i=0; (x=x-i-i-1)>=0; i++);
```

$\{i = \lfloor\sqrt{x}\rfloor\}$

```
function Foo(x,s,i:integer):integer;
begin
  s:=s+2*i+1;
  if x >= s then
    Foo:=Foo(x,s,i+1)
  else
    Foo:=i
end;
writeln(Foo(x,0,0));
```

$\{x, s, i \geq 0\}\ y := Foo(x, s, i)\ \{y := max\{k : s + \Sigma_{j=i}^{k-1} 2 * j + 1 \leq x\}\}$
$\{x \geq 0\}\ y := Foo(x, 0, 0)\ \{y := max\{k : \Sigma_{j=1}^{k} 2 * j - 1 \leq x\}\}$
$\{x \geq 0\}\ y := Foo(x, 0, 0)\ \{y := max\{k : k^2 \leq x\} = \lfloor\sqrt{x}\rfloor\}$

Deklaratívne programovanie je, keď: pre každú procedúru viem v dostatočne formálnom jazyku popísať reláciu, ktorá platí pre vstupné a výstupné objekty. Preto:

- nepoužívam globálne premenné,
- nepoužívam cykly (ale rekurziu), resp. každý cyklus popíšem invariantom cyklu,
- nepoužívam aliasing,

# JAZYK PRVORÁDOVEJ LOGIKY

## Abeceda prvorádového jazyka

- *konštantné symboly* – $a, b, c, \ldots$,
- *funkčné symboly* – $f, g, h, \ldots$,
- *predikátové symboly* – $p, q, r, \ldots$,
- *premenné* – $X, x, Y, y, Z, z, \ldots$,
- *kvantifikátory, spojky, pomocné symboly* – $\exists, \forall, \vee, \wedge, \neg, \rightarrow, \leftarrow, \equiv, \ldots$

## Term

- konštantný symbol, premenná,
- $f(t_1, t_2, \ldots, t_n)$, ak $t_i$ sú termy a $f$ je n-árny funkčný symbol.

## Atomická formula (atóm)

- $p(t_1, t_2, \ldots, t_n)$, ak $t_i$ sú termy a $p$ je n-árny predikátový symbol.

## Formula

- atomická formula,
- $F \vee G, F \wedge G, \neg F, F \rightarrow G, F \leftarrow G, F \equiv G$,
- $(\exists x)\, F$, $(\forall x)\, F$, ak $F, G$ sú formuly.

## Literál

- $A$, $\neg A$, ak $A$ je atomická formula.

Základný term je term, ktorý neobsahuje premenné.
Základný atóm je atóm, ktorý neobsahuje premenné.

Herbrandove univerzum $U$ je množina všetkých základných termov.
Herbrandova báza $B$ je množina všetkých základných atómov.

# PROGRAMY vs. FORMULY

*Príklad: Protekcie*

Definujte reláciu $protekcia/2$ ako tranzitívny, reflexívny a symetrický uzáver relácie $pozna/2$:

$(\forall X)\ (\forall Y)\ protekcia(X,Y) \equiv (X = Y) \vee protekcia(Y,X) \vee$
$((\exists Z) protekcia(X,Z) \wedge pozna(Z,Y) \vee protekcia(Z,Y) \wedge pozna(X,Z))$

---

*Príklad: Prirodzené čísla*

Konštanta $0$, funkčný symbol $s/1$. Univerzum $U = \{0, s(0), \ldots, s^i(0), \ldots\}$.

$(\forall X)\ odd(X) \equiv X = s(0) \vee (\ (\exists Y)\ X = s(s(Y)) \wedge odd(Y))$

$(\forall X)\ (\forall Y)\ (\forall Z)\ plus(X,Y,Z) \equiv X = 0 \wedge Y = Z \vee$
$(\ (\exists U)\ (\exists W)\ X = s(U) \wedge Z = s(W) \wedge plus(U,Y,W))$

$(\forall X)\ (\forall Y)\ less(X,Y) \equiv X = 0 \wedge \neg Y = 0 \vee$
$(\ (\exists U)\ (\exists W)\ X = s(U) \wedge Y = s(W) \wedge less(U,W))$

---

*Príklad: Binárne stromy*

Konštanta $0$, funkčné symboly $s/1$, $node/3$.

$(\forall X)\ nat(X) \equiv X = 0 \vee (\ (\exists Y)\ X = s(Y) \wedge nat(Y))$

$(\forall X)\ tree(X) \equiv X = 0 \vee$
$(\ (\exists Value)\ (\exists Left)\ (\exists Right)\ X = node(Value, Left, Right) \wedge$
$nat(Value) \wedge tree(Left) \wedge tree(Right))$

$(\forall X)\ (\forall N)\ depth(X,N) \equiv X = N = 0 \vee$
$(\ (\exists Left)\ (\exists Right)\ (\exists Nl)\ (\exists Nr)\ X = node(\ \_\ , Left, Right) \wedge$
$depth(Left, Nl) \wedge depth(Right, Nr) \wedge max(s(Nl), s(Nr), N))$

$(\forall X)\ sorted(X) \equiv ordered(X, -99999, 99999)$
$(\forall X)\ (\forall L)\ (\forall U)\ ordered(X, L, U) \equiv X = 0 \vee$
$(\exists Left)\ (\exists Right)\ (\exists Value)\ x = tree(Value, Left, Right) \wedge$
$less(L, Value) \wedge less(Value, U) \wedge ordered(Left, L, Value) \wedge$
$ordered(Right, Value, U)$

IDEMPOTENCIA $\wedge$ , $\vee$ , $\neg$

$$\begin{aligned} A \wedge A &\equiv A \\ A \vee A &\equiv A \\ \neg\neg A &\equiv A \end{aligned}$$

KOMUTATIVITA $\wedge$ , $\vee$ , $\equiv$

$$\begin{aligned} A \wedge B &\equiv B \wedge A \\ A \vee B &\equiv B \vee A \\ A \equiv B &\equiv B \equiv A \end{aligned}$$

ASOCIATIVITA A DISTRIBUTIVITA $\wedge$ , $\vee$

$$\begin{aligned} A \wedge (B \wedge C) &\equiv (A \wedge B) \wedge C \\ A \vee (B \vee C) &\equiv (A \vee B) \vee C \\ A \vee (B \wedge C) &\equiv (A \vee B) \wedge (A \vee C) \\ A \wedge (B \vee C) &\equiv (A \wedge B) \vee (A \wedge C) \end{aligned}$$

De Morganove pravidlá

$$\begin{aligned} \neg(A \wedge B) &\equiv (\neg A \vee \neg B) \\ \neg(A \vee B) &\equiv (\neg A \wedge \neg B) \\ A \leftarrow B &\equiv A \vee \neg B \\ (A \leftarrow B) \leftarrow C &\equiv A \leftarrow (B \wedge C) \\ A \leftarrow (B \leftarrow C) &\equiv (A \leftarrow B) \wedge (A \vee C) \\ A \leftarrow (B \vee C) &\equiv (A \leftarrow B) \wedge (A \leftarrow C) \\ (A \wedge B) \leftarrow C &\equiv (A \leftarrow C) \wedge (B \leftarrow C) \\ \neg (\forall X)\, A[X] &\equiv (\exists X)\, \neg A[X] \\ \neg (\exists X)\, A[X] &\equiv (\forall X)\, \neg A[X] \\ (\forall X)\,(A[X] \wedge B[X]) &\equiv (\forall X)\, A[X] \wedge (\forall X)\, B[X] \\ (\exists X)\,(A[X] \vee B[X]) &\equiv (\exists X)\, A[X] \vee (\exists X)\, B[X] \\ (\forall X)\ (\forall Y)\, A[X,Y] &\equiv (\forall Y)\ (\forall X)\, A[X,Y] \\ (\exists X)\ (\exists Y)\, A[X,Y] &\equiv (\exists Y)\ (\exists X)\, A[X,Y] \end{aligned}$$

# PREFIXOVÁ NORMÁLNA FORMA

Prefixová normálna forma

- $(Q_1X_1)(Q_2X_2)\dots(Q_nX_n)F$, ak $Q_i$ sú kvantifikátory a $F$ je formula neobsahujúca kvantifikátory.

---

1. *odstránenie ekvivalencií a implikácií*

$$A \equiv B \Rightarrow (A \leftarrow B) \wedge (B \leftarrow A)$$
$$A \leftarrow B \Rightarrow A \vee \neg B$$

2. *pasírovanie negácie*

$$\neg\neg A \Rightarrow A$$
$$\neg(A \vee B) \Rightarrow \neg A \wedge \neg B$$
$$\neg(A \wedge B) \Rightarrow \neg A \vee \neg B$$
$$\neg(\forall X)\, A[X] \Rightarrow (\exists X)\neg A[X]$$
$$\neg(\exists X)\, A[X] \Rightarrow (\forall X)\neg A[X]$$

3. *pasírovanie* $\vee$

$$A \vee (B \wedge C) \Rightarrow (A \vee B) \wedge (A \vee C)$$

4. *pasírovanie* $\forall$, $\exists$

$$A \vee (Q\ X)B[X] \Rightarrow (Q\ X)(A \vee B[X])$$
$$A \wedge (Q\ X)B[X] \Rightarrow (Q\ X)(A \wedge B[X])$$
$$(\forall X)\, A[X] \wedge (\forall X)\, B[X] \Rightarrow (\forall X)\,(A[X] \wedge B[X])$$
$$(\exists X)\, A[X] \vee (\exists X)\, B[X] \Rightarrow (\exists X)\,(A[X] \vee B[X])$$
$$(Q_1\ X)A[X] \wedge (Q_2\ X)B[X] \Rightarrow (Q_1\ X)(Q_2\ Z)(A[X] \wedge B[Z])$$
$$(Q_1\ X)A[X] \vee (Q_2\ X)B[X] \Rightarrow (Q_1\ X)(Q_2\ Z)(A[X] \vee B[Z])$$

---

Veta (o prefixovej normálnej forme) Pre ľubovoľnú formulu $F$ existuje formula $G$ v prefixovej normálnej forme taká, že platí $F \equiv G$.

# SKOLEMOVA NORMÁLNA FORMA

## Skolemova normálna forma

- $(\forall X_1)(\forall X_2)\ldots(\forall X_n)F$, ak $F$ je formula neobsahujúca kvantifikátory.

---

1. *skolemizácia*

$$(\forall X_1)\ldots(\forall X_n)\ (\exists Y)\,A(Y)\ \Rightarrow\ (\forall X_1)\ldots(\forall X_n)\,A(f(X_1,\ldots,X_n))$$
$$(\exists Y)\,A(Y)\ \Rightarrow\ A(c)$$

   $f$ je nová Skolemova funkcia, $c$ je Skolemova konštanta,

2. opakujeme pasírovanie $\forall$ a skolemizáciu.

---

Veta (o skolemovej normálnej forme) Pre ľubovoľnú formulu $F$ existuje formula $G$ v skolemovej normálnej forme taká, že $F$ je *splniteľná práve* vtedy, keď $G$. Taktiež platí, že formula $F$ je *logickým dôsledkom* formuly $G$.

Príklad: $F = (\exists X)\,p(X)$, $G = p(a)$. Domain 0,1, interpretácia $a = 0$,$p = x > 0$.

---

## Klauzula

- $(\forall X_1)\ldots(\forall X_k)\,L_1 \lor \ldots \lor L_n$, kde $L_1, L_2, \ldots, L_n$ sú literály

## Klauzálna forma

- $(\forall X_1)\ldots(\forall X_k)\,A_1 \lor \ldots \lor A_m \leftarrow B_1 \land \ldots \land B_n$
  $A_1, \ldots, A_m \leftarrow B_1, \ldots, B_n$, kde $A_i, B_j$ sú atomické formuly.

Hornova klauzula, ak $m = 1$: $\quad A \leftarrow B_1, \ldots, B_n$.

Fakt, ak $m = 1, n = 0$: $\quad A \leftarrow$ .

Cieľ, ak $m = 0, n > 0$: $\quad \leftarrow B_1, \ldots, B_n$.

Prázdny cieľ, ak $m = n = 0$: $\quad \Box$.

Logický program je konečná množina Hornových klauzúl.

# PODMNOŽINA

PRÍKLAD: použitím predikátu $\in$ definujte predikát podmnožina $s(X,Y) \equiv X \subset Y$. Formulu prepíšte do skolemovej normálnej formy.

---

$$(\forall X)\ (\forall Y)\,(s(X,Y)\ \equiv\ (\forall U)\,(U \in Y\ \leftarrow\ U \in X))$$

PO ODSTRÁNENÍ $\equiv$:

$$(\forall X)\ (\forall Y)\,((s(X,Y)\ \leftarrow\ (\forall U)\,(U \in Y\ \leftarrow\ U \in X))\wedge$$
$$(\,(\forall U)\,(U \in Y\ \leftarrow\ U \in X)\ \leftarrow\ s(X,Y)))$$

PO ODSTRÁNENÍ $\leftarrow$:

$$(\forall X)\ (\forall Y)\,((s(X,Y)\ \vee\ \neg\ (\forall U)\,(U \in Y\ \vee\ \neg\, U \in X))\wedge$$
$$(\,(\forall U)\,(U \in Y\ \vee\ \neg\, U \in X)\ \vee\ \neg\, s(X,Y)))$$

PO PASÍROVANÍ $\neg$:

$$(\forall X)\ (\forall Y)\,((s(X,Y)\ \vee\ (\exists U)\,(\,\neg\, U \in Y\ \wedge\ U \in X))\wedge$$
$$(\,(\forall U)\,(U \in Y\ \vee\ \neg\, U \in X)\ \vee\ \neg\, s(X,Y)))$$

PO PASÍROVANÍ $\vee$:

$$(\forall X)\ (\forall Y)\,(\,(\exists U)\,((s(X,Y)\ \vee\ \neg\, U \in Y)\ \wedge\ (s(X,Y)\ \vee\ U \in X))\wedge$$
$$(\,(\forall U)\,(U \in Y\ \vee\ \neg\, U \in X\ \vee\ \neg\, s(X,Y)))$$

PO PASÍROVANÍ $\forall$:

$$(\forall X)\ (\forall Y)\ (\exists U)\,((s(X,Y)\ \vee\ \neg\, U \in Y)\ \wedge\ (s(X,Y)\ \vee\ U \in X))\wedge$$
$$(\forall X)\ (\forall Y)\ (\forall U)\,(U \in Y\ \vee\ \neg\, U \in X\ \vee\ \neg\, s(X,Y))$$

PO SKOLEMIZÁCII:

$$(\forall X)\,(\forall Y)\,((s(X,Y) \vee \neg\, f(X,Y) \in Y) \wedge (s(X,Y) \vee f(X,Y) \in X))\wedge$$
$$(\forall X)\ (\forall Y)\ (\forall U)\,(U \in Y\ \vee\ \neg\, U \in X)\ \vee\ \neg\, s(X,Y))$$

---

KLAUZULY:

$s(X,Y)\ \leftarrow\ f(X,Y) \in Y$

$s(X,Y), f(X,Y) \in X\ \leftarrow$

$U \in Y\ \leftarrow\ U \in X\ \wedge\ s(X,Y)$

# REKURZÍVNE PROGRAMY

*Príklad: Protekcie*

$$(\forall X)\ (\forall Y)\, protekcia(X,Y) \leftarrow (X = Y) \lor protekcia(Y,X) \lor$$
$$((\exists Z) protekcia(X,Z) \land pozna(Z,Y) \lor protekcia(Z,Y) \land pozna(X,Z))$$

```
protekcia(X,Y)<-X=Y
protekcia(X,Y)<-protekcia(Y,X)
protekcia(X,Y)<-protekcia(X,Z), pozna(Z,Y)
protekcia(X,Y)<-protekcia(Z,Y), pozna(X,Z)
```

---

*Príklad: Prirodzené čísla*

$$(\forall X)\, odd(X) \leftarrow X = s(0) \lor (\, (\exists Y)\, X = s(s(Y)) \land odd(Y))$$

```
odd(s(0))<-
odd(s(s(Y)))<-odd(Y)
```

$$(\forall X)\ (\forall Y)\ (\forall Z)\, plus(X,Y,Z) \leftarrow X = 0 \land Y = Z \lor$$
$$(\, (\exists U)\ (\exists W)\, X = s(U) \land Z = s(W) \land plus(U,Y,W))$$

```
plus(0,Z,Z)<-
plus(s(U),Y,s(W))<-plus(U,Y,W)
```

$$(\forall X)\ (\forall Y)\, less(X,Y) \leftarrow X = 0 \land \neg Y = 0 \lor$$
$$(\, (\exists U)\ (\exists W)\, X = s(U) \land Y = s(W) \land less(U,W))$$

```
less(0,s(_))<-
less(s(U),s(W))<-less(U,W)
```

$$(\forall X)\ (\forall Y)\, equal(X,Y) \leftarrow X = Y = 0 \land$$
$$(\, (\exists U)\ (\exists W)\, X = s(U) \land Y = s(W) \land equal(U,W))$$

```
equal(0,0)<-
equal(s(U),s(W))<-equal(U,W)
```

```
equal(X,X)<-
```

# PRIRODZENÉ ČÍSLA

```
times(0,X,0)<-
times(s(X),Y,Z)<-times(X,Y,W),plus(W,Y,Z)

div2(0,0)<-
div2(s(0),0)<-
div2(s(s(X)),s(Y))<-div2(X,Y)

exp(s(X),0,0)<-
exp(0,s(X),s(0))<-
exp(s(N),X,Y)<-exp(N,X,Z),times(Z,X,Y)

fact(0,s(0))<-
fact(s(N),F)<-fact(N,F1),times(s(N),F1,F)

minimum(N1,N2,N1)<-less(N1,N2)
minimum(N1,N2,N2)<-less(N2,N1)
minimum(N1,N2,N1)<- N2=N1

mod(X,Y,Z)<-less(Z,Y),times(Y,Q,W),plus(W,Z,X)

mod(X,Y,X)<-less(X,Y)
mod(X,Y,Z)<-plus(X1,Y,X),mod(X1,Y,Z)

ackermann(0,N,s(N))
ackermann(s(M),0,Val)<-ackermann(M,s(0),Val)
ackermann(s(M),s(N),Val)<-
  ackermann(s(M),N,Val1),ackermann(M,Val1,Val)

gcd(X,Y,Gcd)<-mod(X,Y,Z),gcd(Y,Z,Gcd)
gcd(X,0,X)<-less(0,X)
```

# CELÉ A RACIONÁLNE ČÍSLA

```
int(X)<-nat(X)
int(m(s(X)))<-nat(X)

add(0,X,X)<-
add(X,0,X)<-
add(s(X),s(Y),s(s(Z)))<-add(X,Y,Z)
add(s(X),m(s(Y)),Z)<-add(X,m(Y),Z)
add(m(s(X)),s(Y),Z)<-add(m(X),Y,Z)
add(m(s(X)),m(s(Y)),m(s(s(Z))))<-add(X,Y,Z)

times(0,X,0)<-
times(X,0,0)<-
times(s(X),Y,Z)<-times(X,Y,W),add(W,Y,Z)
times(m(s(X)),Y,m(Z))<-times(s(X),Y,Z)

lower(0,s(_))<-
lower(m(_),0)<-
lower(m(_),s(_))<-
lower(s(X),s(Y))<-less(X,Y)
lower(m(X),m(Y))<-less(Y,X)

%-------------------------------------------
ratio(over(A,B))<-int(A),positive(B)

positive(s(0))
positive(s(X))<-positive(X)

. . . . . . . . . . . . . . .
```

# PRÍKLADY PROGRAMOV

*Príklad: Puzzle 8*

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

```
sused(1,2)<-    sused(2,3)<-  ...  sused(8,9)<-
sused(1,4)<-    sused(4,7)<-  ...  sused(6,9)<-
sused(X,Y)<-sused(Y,X).

konfiguracia(9,nil).
konfiguracia(X,f(Y,T))<-
  sused(X,Y), konfiguracia(Y,T).
```

---

*Príklad: Databázy:* Definujte základné databázové operácie: zjednotenie, prienik, rozdiel, kartézsky súčin, projekciu, selekciu a spojenie.

```
% --- zjednotenie
r_union_s(X1,...,Xn)<-r(X1,...,Xn)
r_union_s(X1,...,Xn)<-s(X1,...,Xn)
% --- rozdiel
r_diff_s(X1,...,Xn)<-r(X1,...,Xn), not s(X1,...,Xn)
r_diff_s(X1,...,Xn)<-s(X1,...,Xn), not r(X1,...,Xn)
% --- kartezsky sucin
r_x_s(X1,...,Xm,Y1,...,Yn)<-r(X1,...,Xm), s(Y1,...,Yn)
% --- projekcia
r_proj(Xk1,...,Xkm)<-r(X1,...,Xn)
% --- selekcia
r_sel(X1,...,Xn)<-r(X1,...,Xn), podmienka(X1,...,Xn)
% --- prienik
r_x_s(X1,...,Xn)<-r(X1,...,Xn), s(X1,...,Xn)
% --- spojenie
r_join_s(X1,...,Xm,Z1,...,Zk,Y1,...,Yn)<-
  r(X1,...,Xm,Z1,...,Zk), s(Z1,...,Zk,Y1,...,Yn)
```

# PRÍKLADY PROGRAMOV

*Príklad: Binárne stromy*

$(\forall X)\ (\forall N)\, depth(X,N) \leftarrow\ X = N = 0 \vee$
$(\ (\exists Left)\ (\exists Right)\ (\exists Nl)\ (\exists Nr)\, X = node(\ \_\ , Left, Right) \wedge$
$depth(Left, Nl)\ \wedge\ depth(Right, Nr)\ \wedge\ max(s(Nl), s(Nr), N))$

```
depth(0,0)<-
depth(node(Value,Left,Right),s(M))<-
  depth(Left,Nl), depth(Right,Nr), max(Nl,Nr,M)

sorted(X)<-ordered(X,-99999,99999)
ordered(0,_,_)<-
ordered(tree(Value,Left,Right),L,U)<-L<Value, Value<U,
  ordered(Left,L,Value), ordered(Right,Value,U)
```

---

*Príklad: Rodinky*

$rodic/2, zena/1, muz/1, zname\_priezvisko/2$

```
otec(X,Y)<-rodic(X,Y), muz(X)
dedo(X,Y)<-otec(X,Z), otec(Z,Y)
brat(X,Y)<-rodic(Z,X), rodic(Z,Y), muz(X)
brat(X,Y)<-rodic(Z,X), rodic(Z,Y), muz(X), X <> Y

predok(X,Y)<-rodic(X,Y)
predok(X,Y)<-rodic(X,Z), predok(Z,Y)

priezvisko(X,P)<-zname_priezvisko(X,P)
priezvisko(X,P)<-rodic(Y,X),muz(Y),priezvisko(Y,P)
priezvisko(X,P)<-zena(X), rodic(X,D), rodic(Y,D),
  muz(Y), priezvisko(Y,P)

vrstovnik(X,X)<-
vrstovnik(X,Y)<-
  rodic(Xr,X), rodic(Yr,Y), vrstovnik(Xr,Yr)
```

# REKURZÍVNE DÁTOVÉ ŠTRUKTÚRY

## Zoznamy

- $nil \equiv []$
- $.(head, tail) \equiv [head|tail]$
- $.(a_1, .(a_2, \ldots, .(a_n, tail) \ldots) \equiv [a_1, a_2, \ldots, a_n|tail]$
- $.(a_1, .(a_2, \ldots, .(a_n, []) \ldots) \equiv [a_1, a_2, \ldots, a_n]$

---

*Príklad: Konverzie*

```
[a] = .(a,[]) = [a|[]]
[a,b] = .(a,.(b,[])) = [a|[b]] = [a|[b|[]]] = [a,b|[]]
```

*Príklad: Zoznamy*

```
is_list([])<-
is_list([X|Xs])<-is_list(Xs)

member(X,[X|_])<-
member(X,[_|Ys])<-member(X,Ys)

last([X],X)
last([_,Y|Z],L)<-last([Y|Z],L)

length([],0)
length([_|X],s(L))<-length(X,L)

prefix([],_)<-
prefix([X|Xs],[X|Ys])<-prefix(Xs,Ys)

sufix(X,X)<-
sufix(X,[_|Ys])<-sufix(X,Ys)
```

```
append([],X,X)<-
append([X|Xs],Y,[X|Zs])<-append(Xs,Y,Zs)

prefix(X,Y)<-append(X,_,Y)
sufix(X,Y)<-append(_,X,Y)

sublist(X,Y)<-prefix(Z,Y),sufix(X,Z)
sublist(X,Y)<-sufix(Z,Y),prefix(X,Z)
sublist(X,Y)<-append(A,B,Y),append(X,_,B)

last(Xs,X)<-append(_,[X],Xs)

reverse([],[])
reverse([X|Xs],Y)<-
  reverse(Xs,Ys),append(Ys,[X],Y)
```

$reverse(A, B, C) \equiv append(A^R, B, C)$

```
reverse(X,Y)<-reverse(X,[],Y)
reverse([],X,X)
reverse([X|Xs],Y,Z)<-reverse(Xs,[X|Y],Z)

subseq([X|Xs],[X|Ys])<-subseq(Xs,Ys)
subseq(Xs,[Y|Ys])<-subseq(Xs,Ys)
subseq([],Ys)

double([],L,L)
double([X|Xs],Y,[X,X|Zs])<-append(Xs,Y,Zs)

triple(L,LLL)<-append(L,L,LL),append(L,LL,LLL)
triple(L,LLL)<-append(L,LL,LLL),append(L,L,LL)

delete([X|Xs],X,Ys)<-delete(Xs,X,Ys)
delete([X|Xs],Z,[X|Ys])<-X\=Z,delete(Xs,Z,Ys)
delete([],X,[])
```

```
select(X,[X|Xs],Xs)
select(X,[Y|Ys],[Y|Zs])<-select(X,Ys,Zs)

sort(Xs,Ys)<-permutation(Xs,Ys),ordered(Ys)

ordered([_])
ordered([X,Y|Ys])<-X=<Y,ordered([Y|Ys])

permutation([],[])
permutation(Xs,[Z|Zs])<-
  select(Z,Xs,Ys),permutation(Ys,Zs)

permutation([],[])
permutation([X|Xs],Zs)<-
  permutation(Xs,Ys),insert(X,Ys,Zs)

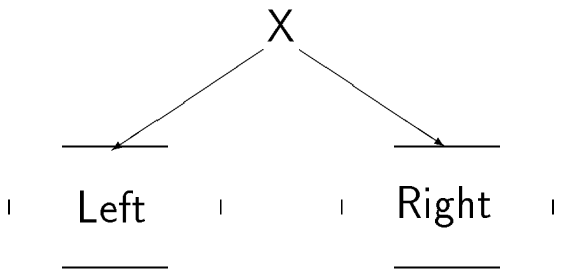
insert(X,Ys,Zs)<-select(X,Zs,Ys)

qsort([],[])
qsort([X|Xs],Sorted)<-
  partition(Xs,X,Small,Large),
  qsort(Small,Small1),
  qsort(Large,Large1),
  append(Small,[X|Large],Sorted)

partition([],_,[],[])
partition([X|Xs],P,[X|Small],Large)<-
  X<P,partition(Xs,P,Small,Large)
partition([X|Xs],P,Small,[X|Large])<-
  X>=P,partition(Xs,P,Small,Large)
```

# BINÁRNE STROMY

Funkčné symboly `nil`, `node(X,Left,Right)`




```
%--- find(X,Y) iff X belongs to the tree Y
find(X,node(X,_,_))
find(X,node(_,Left,_))<- find(X,Left)
find(X,node(_,_,Right))<- find(X,Right)

find(X,node(Value,Left,Right))<-
  X=Value ; find(X,Left) ; find(X,Right)

%--- subtree(X,Y) iff X is a subtree of Y
subtree(X,X)
subtree(X,node(_,L,R))<-
  subtree(X,L) ; subtree(X,R)

%--- iso(X,Y) iff X and Y are isomorphic
iso(nil,nil)
iso(node(X,Left1,Right1),node(X,Left2,Right2))<-
  (iso(Left1,Left2),iso(Right1,Right2)) ;
  (iso(Left1,Right2),iso(Right1,Left2))

%--- subst(X,Y,T,T1) iff T1 = T[X<-Y]
subst(_,_,nil,nil)
subst(X,Y,node(X,L,R),node(Y,L1,R1))<-
  subst(X,Y,L,L1), subst(X,Y,R,R1)
subst(X,Y,node(Z,L,R),node(Z,L1,R1))<-
  X\=Z,subst(X,Y,L,L1), subst(X,Y,R,R1)
```

# USPORIADANÉ BINÁRNE STROMY

```
%--- ordered(X) iff X is a binary search tree
ordered(X)<-ordered(X,-99999,99999)
ordered(nil,_,_)<-
ordered(node(Value,Left,Right),L,U)<-L=<Value, Value<=U,
  ordered(Left,L,Value), ordered(Right,Value,U)

%--- tree_insert(X,T,T1) iff T1 = T+{X}
tree_insert(X,nil,node(X,nil,nil))
tree_insert(X,node(Y,L,R),node(Y,L1,R))<-
  X < Y, tree_insert(X,L,L1)
tree_insert(X,node(Y,L,R),node(Y,L,R1))<-
  X >= Y, tree_insert(X,R,R1)
```

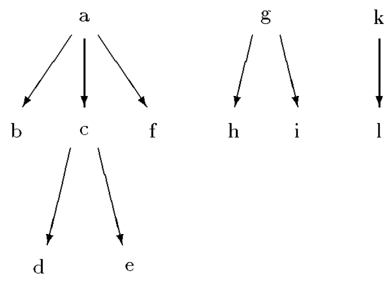
# TRAVERZOVANIE STROMOV

```
%- preorder(X,Y) iff Y=list of nodes of X in preorder
preorder(nil,[])
preorder(node(X,Left,Right),List)<-
  preorder(Left,List_l), preorder(Right,List_r),
  append([X|List_l],List_r,List)

%- inorder(X,Y) iff Y=list of nodes of X in inorder
inorder(nil,[])
inorder(node(X,Left,Right),List)<-
  inorder(Left,List_l), inorder(Right,List_r),
  append(List_l,[X|List_r],List)

%- postorder(X,Y) iff Y=list of nodes of X in postorder
postorder(nil,[])
postorder(node(X,Left,Right),List)<-
  postorder(Left,List_l), postorder(Right,List_r),
  append(List_r,[X],List1), append(List_l,List1,List)
```

# VŠEOBECNÉ STROMY




```
type Forest(t:Type) = List Tree t
type Tree(t:Type) = t,List Tree t

type Forest(t:Type) = List(t,Forest t)

(a,[]) ... tree
[(a,[]),(b,[]),(c,[])] ... forest
(d,[(a,[]),(b,[]),(c,[])]) ... tree
[(d,[(a,[]),(b,[]),(c,[])]),(e,[(f,[]),(g,[])]),(h,[])]

is_tree((_,Forest))<-is_forest(Forest)
is_forest([])
is_forest([(_,Sons)|Broths])<-
  is_forest(Sons), is_forest(Broths)

find_tree(X,(X,_))
find_tree(X,(Y,Forest))<- X\=Y, find_forest(X,Forest)

find_forest(X,[Tree|Forest])<-
  find_tree(X,Tree) ; find_forest(X,Forest)

find_forest(X,[(Y,Sons)|Broths])<-
  X=Y ; find_forest(X,Sons) ; find_forest(X,Broths)

delete(X,[],[])
delete(X,[(X,Sons)|Broths],Broths)
delete(X,[(Y,Sons)|Broths],[(Y,Sons1)|Broths1])<-
  X\=Y,delete(X,Sons,Sons1),delete(X,Broths,Broths1)

delete(X,[(X,Sons)|Broths],Forest)<-
  append(Sons,Broths,Forest)
```

# POLYNÓMY

OPERÁTORY sú funkčné symboly písané v špeciálnej notácii.
DEFINÍCIA OPERÁTORA op(priorita,asociativita,meno).

```
op(10,yfx,^)                      op(51,xfy,'.')
op(11,xfx,mod)                    op(60,fx,not)
op(21,yfx,[*,/])                  op(250,xfy,',')
op(31,yfx,[+,-])                  op(252,xfy,';')
op(40,xfx,[=,<,>,=<,>=,is])       op(255,xfx,'<-')
```

---

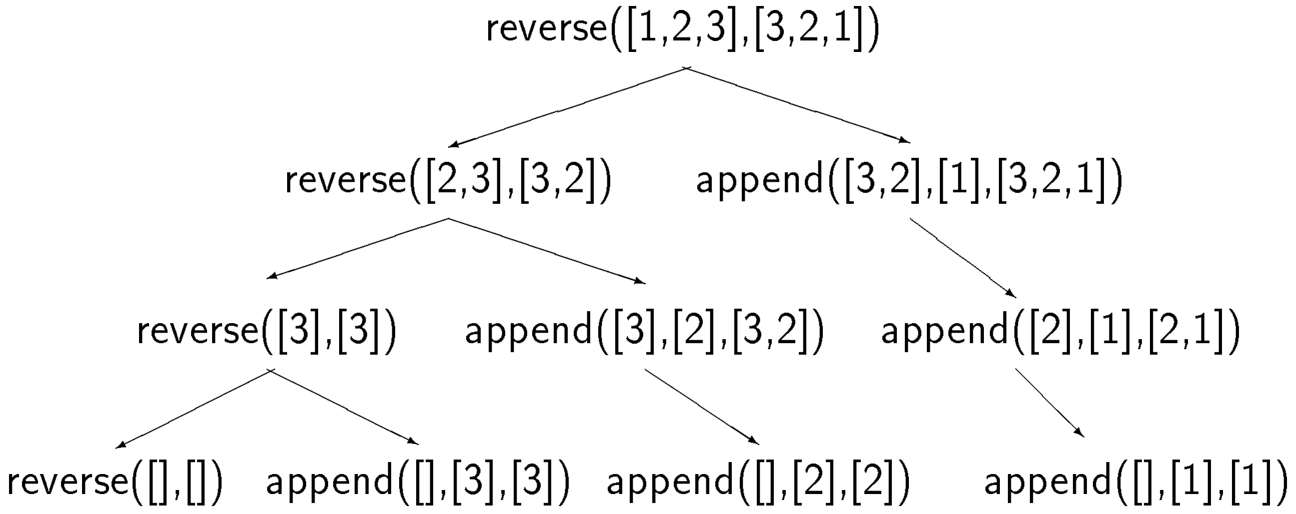
```
%--- polynom(P,X) iff P is polynom over variable X
polynom(X,X)
polynom(Term,X)<-integer(Term)
polynom(Term1+Term2,X)<-
  polynom(Term1,X), polynom(Term2,X)
. . . . . . . . . . . .
polynom(Term1/Term2,X)<-
  polynom(Term1,X), integer(Term2)
polynom(Term1^N,X)<-
  polynom(Term1,X), natural(N)

%--- deriv(P,X,Dp) iff Dp = P dX
deriv(X,X,s(0))                   deriv(sin(X),X,cos(X))
deriv(X^s(N),X,s(N)*X^N)
deriv(F+G,X,DF+DG)<-deriv(F,X,DF), deriv(G,X,DG)
. . . . . . . . . . . .
deriv(F*G,X,F*DG+G*DF)<-
  deriv(F,X,DF), deriv(G,X,DG)
deriv(s(0)/F,X,-DF/(F*F))<-deriv(F,X,DF)
deriv(F/G,X,(G*DF-F*DG)/(G*G))<-
  deriv(F,X,DF), deriv(G,X,DG)
%--- ina reprezentacia: sin(arg)=unary_term(sin,arg)
deriv(unary_term(F,U),X,DF*DU)<-
  deriv(unary_term(F,U),U,DF),deriv(U,X,DU)
```

# STROM DÔKAZU

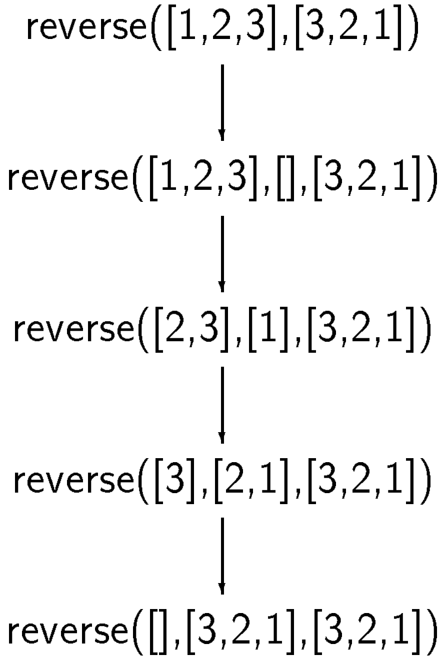
```
reverse([],[])
reverse([X|Xs],Zs)<-reverse(Xs,Ys),append(Ys,[X],Zs)

append([],Ys,Ys)
append([X|Xs],Ys,[X|Zs])<-append(Xs,Ys,Zs)
```




---

```
reverse(Xs,Ys)<-reverse(Xs,[],Ys)
reverse([],Ys,Ys)
reverse([X|Xs],Ys,Zs)<-reverse(Xs,[X|Ys],Zs)
```



# VÝPOČET LOGICKÉHO PROGRAMU

- namiesto dokazovania, že základný atóm $A$ je logickým dôsledkom programu $P$ (sémantická vlastnosť) stačí dokázať, že $A$ je odvoditeľný z programu $P$ (syntaktická vlastnosť) – dôsledok Gödelovej vety o úplnosti PP1,
- namiesto dokazovania, že základný atóm $A$ je logickým dôsledkom programu $P$ dokazujeme, že množina uzavretých formúl $P \cup \{\neg A\}$ je nesplniteľná – dôsledok vety o dedukcii,
- ak program $P$ obsahuje aspoň jeden unárny funkčný symbol, potom problém, či základný atóm $A$ je logickým dôsledkom $P$ je nerozhodnuteľný, len čiastočne rozhodnuteľný – dôsledok Churchovej vety o nerozhodnuteľnosti,

Rezolučný (rezolvenčný) princíp [Robinson, 1965]:

- množina klauzúl $M = \{\ldots, L_1 \vee \ldots \vee L_n \vee A, K_1 \vee \ldots \vee K_m \vee \neg A, \ldots\}$ je nesplniteľná práve vtedy, ak je nesplniteľná množina $M \cup \{L_1 \vee \ldots \vee L_n \vee K_1 \vee \ldots \vee K_m\}$. Klauzula $L_1 \vee \ldots \vee L_n \vee K_1 \vee \ldots \vee K_m$ sa nazýva rezolventou,
- množina klauzúl $M = \{\ldots, L_1 \vee \ldots \vee L_n \vee A, K_1 \vee \ldots \vee K_m \vee \neg B, \ldots\}$, kde $A$, $B$ sú unifikovateľné ($\theta = mgu(A, B)$), je nesplniteľná práve vtedy, ak je nesplniteľná množina $M \cup \{L_1 \vee \ldots \vee L_n \vee K_1 \vee \ldots \vee K_m\}\theta$,
- množina obsahujúca prázdnu klauzula $\Box$ je nesplniteľná.

---

```
P:    p(X)<-q(X)
      q(a)
A:    p(a)
```

$$
\begin{aligned}
M_1 &= \{p(X) \vee \neg q(X), q(a)\} \cup \{\neg p(a)\} \\
\theta &= \{X/a\} \\
M_2 &= M_1 \cup \{\neg q(a)\} \\
M_3 &= M_2 \cup \{\Box\}
\end{aligned}
$$

# PRÍKLAD 'REVERSE'

$P = \{$
$append([], X, X),$
$append([X|Xs], Y, [X|Zs]) \vee \neg append(Xs, Y, Zs),$
$reverse([], []),$
$reverse([X|Xs], Y) \vee \neg reverse(Xs, Ys) \vee \neg append(Ys, [X], Y)\}$

$A = reverse([1, 2, 3], [3, 2, 1])$

---

$$
\begin{aligned}
M_1 &= P \cup \{\neg reverse([1,2,3],[3,2,1])\} \\
M_2 &= M_1 \cup \{\neg reverse([2,3], Ys1) \vee \neg append(Ys1, [1], [3,2,1])\} \\
M_3 &= M_2 \cup \{(\neg reverse([3], Ys2) \vee \neg append(Ys2, [2], Ys1)) \\
&\qquad\qquad \vee \neg append(Ys1, [1], [3,2,1])\} \\
M_4 &= M_3 \cup \{((\neg reverse([], Ys3) \vee \neg append(Ys3, [3], Ys2)) \\
&\qquad\qquad \vee \neg append(Ys2, [2], Ys1)) \\
&\qquad\qquad \vee \neg append(Ys1, [1], [3,2,1])\} \\
M_5 &= M_4 \cup \{(\neg append([], [3], Ys2) \vee \neg append(Ys2, [2], Ys1)) \\
&\qquad\qquad \vee \neg append(Ys1, [1], [3,2,1])\} \\
M_6 &= M_5 \cup \{\neg append([3], [2], Ys1) \\
&\qquad\qquad \vee \neg append(Ys1, [1], [3,2,1])\} \\
M_7 &= M_6 \cup \{\neg append([], [2], Zs1) \\
&\qquad\qquad \vee \neg append([3|Zs1], [1], [3,2,1])\} \\
M_8 &= M_7 \cup \{\neg append([3,2], [1], [3,2,1])\} \\
M_9 &= M_8 \cup \{\neg append([2], [1], [2,1])\} \\
M_{10} &= M_9 \cup \{\neg append([], [1], [1])\} \\
M_{11} &= M_{10} \cup \{\Box\}
\end{aligned}
$$

# SUBSTITÚCIE

SUBSTITÚCIA $\theta = \{v_1/t_1, \dots, v_n/t_n\}$, kde $v_i$ sú rôzne premenné a $v_i \neq t_i$.

PREMENOVACIA SUBSTITÚCIA $\theta$, ak $t_1, \dots, t_n$ sú rôzne premenné.

VÝRAZ $E$ je term, literál alebo konjunkcia, resp. disjunkcia literálov.

INŠTANCIA $E\theta$ výrazu $E$ vznikne simultánnym nahradením každého výskytu premennej $v_i$ za $t_i$ vo výraze $E$.

PRÍKLAD: Ak $E = p(f(X), g(Y), Z)$ a $\theta = \{X/f(X), Y/Z, Z/c\}$, potom $E\theta = p(f(f(X)), g(Z), c)$.

KOMPOZÍCIA $\theta\sigma$ SUBSTITÚCIÍ $\theta = \{u_1/s_1, \dots, u_m/s_m\}$ a $\sigma = \{v_1/t_1, \dots, v_n/t_n\}$ je substitúcia, ktorá vznikne z množiny $\{u_1/s_1\sigma, \dots u_m/s_m\sigma, v_1/t_1, \dots, v_n/t_n\}$ vynechaním dvojíc $u_i/s_i\sigma$, pre ktoré $u_i = s_i\sigma$ a dvojíc $v_j/t_j$ pre $v_j \in \{u_1, \dots, u_m\}$.

PRÍKLAD: Ak $\theta = \{X/f(Y), Y/Z\}$ a $\sigma = \{X/a, Y/b, Z/Y\}$, potom $\theta\sigma = \{X/f(b), Z/Y\}$ a $\sigma\theta = \{X/a, Y/b\}$.

LEMA (O SKLADANÍ SUBSTITÚCIÍ)

- $\theta\epsilon = \epsilon\theta = \theta$,
- $(E\theta)\sigma = E(\theta\sigma)$,
- $(\theta\sigma)\gamma = \theta(\sigma\gamma)$.

VARIANTY sú také dva výrazy $E$, $F$, pre ktoré existujú $\theta$ a $\sigma$, že platí $E = F\theta$ a $F = E\sigma$.

PRÍKLAD: $p(X, f(X,Y), Z)$ a $p(Y, f(Y,Z), U)$ sú varianty, ale $p(X,Y)$ a $p(Z,Z)$ nie sú varianty.

LEMA (O VARIANTOCH): Ak $E$, $F$ sú varianty, potom existujú premenovacie substitúcie $\theta$, $\sigma$ také, že $E = F\theta$ a $F = E\sigma$.

# UNIFIKÁCIA a M.G.U.

UNIFIKÁTOR
Substitúcia $\theta$ je unifikátor konečnej množiny výrazov $S$, ak $S\theta$ má jeden prvok.

NAJVŠEOBECNEJŠÍ UNIFIKÁTOR
Substitúcia $\theta$ je najvšeobecnejší unifikátor konečnej množiny výrazov $S$, ak $\theta$ je unifikátor $S$ a pre ľubovoľný unifikátor $\sigma$ množiny $S$ existuje substitúcia $\gamma$ taká, že $\sigma = \theta\gamma$.

PRÍKLAD: Termy $p(f(X), a)$ a $p(Y, f(W))$ nie sú unifikovateľné; $p(f(X), Z)$ a $p(Y, a)$ sú unifikovateľné a $\sigma = \{Y/f(a), X/a, Z/a\}$ je ich unifikátor. Najvšeobecnejší unifikátor je $\theta = \{Y/f(X), Z/a\}$. Platí tiež, že $\sigma = \theta\{X/a\}$.

LEMA (O M.G.U.) Všetky najvšeobecnejšie unifikátory sa odlišujú v premenovaní premenných, t.j. ak $\theta$ a $\sigma$ sú mgu, potom existuje premenovacia substitúcia $\gamma$ taká, že $\theta = \sigma\gamma$.

---

UNIFIKAČNÝ ALGORITMUS [Martelli,Montanari, 1982]
Pre unifikovanú dvojicu atómov s rovnakým predikátovým symbolom $p(a_1, \ldots, a_n)$, $p(b_1, \ldots, b_n)$ vytvoríme množinu rovností $\{a_1 = b_1, \ldots, a_n = b_n\}$, ktorú nedeterministicky redukujeme pomocou nasledujúcich pravidiel:

- rovnosť $f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)$ nahraď $s_1 = t_1, \ldots, s_n = t_n$,
- pre rovnosť $f(s_1, \ldots, s_n) = g(t_1, \ldots, t_n)$, ak $f \neq g$, fail,
- rovnosť $x = x$ vyhoď,
- rovnosť $t = x$, ak $t$ nie je premenná nahraď $x = t$,
- pre rovnosť $x = t$ a $x \neq t$: ak $x$ sa nachádza v $t$ ("occur check"), potom fail, inak aplikuj substitúciu $\{x/t\}$ na všetky ostatné rovnice

PRÍKLAD: pre množinu $\{f(X) = f(f(Z)), g(a, Y) = g(a, X)\}$ dostaneme $\{X = f(Z), Y = f(Z)\}$.

# ABSTRAKTNÝ INTERPRETER

VETA (O MGU ALGORITME):

- uvedený unfikačný algoritmus pre unifikovateľné termy počíta ich mgu, inak je jeho výsledkom fail,
- výsledná substitúcia $\theta$ je idempotentná, t.j. $\theta\theta = \theta$,
- výsledná substitúcia $\theta$ je relevantná, t.j. premenné, ktoré sa v nej vyskytujú sa nachádzajú tiež v unifikovaných termoch.

POZNÁMKA (O OCCUR CHECKU): occur check je dôležitý (inak by algoritmus prehlásil termy $f(X)$ a $f(f(X))$ za unifikovateľné), ale v reálnych systémoch často chýba (dôvody sú pragmatické).

| | čas | pamäť |
|---|---|---|
| Robinson, 1965 | $exp(n)$ | $exp(n)$ |
| Boyer,Moore, 1972 | $exp(n)$ | $O(n)$ |
| Corbina,Bidoit, 1983 | $O(n^2)$ | $O(n)$ |
| Ružička,Prívara, 1989 | $O(n.\alpha(n))$ | $O(n)$ |
| Martelli,Montanari, 1982 | $O(n+m.\alpha(m))$ | $O(n)$ |
| Huet, 1973 | $O(n.\alpha(n))$ | $O(n)$ |
| Patterson,Wegman, 1978 | $O(n)$ | $O(n)$ |

---

VSTUP je logický program $P$ a cieľ $G = \leftarrow A_1, \ldots, A_n$,

VÝSTUP je inštancia cieľa $G\theta$, ak je táto inštancia odvoditeľná z programu $P$, inak fail,

ALGORITMUS

- inicializuj $R := \{A_1, \ldots, A_n\}$,
- pokiaľ $R \neq \{\}$ opakuj
  - vyber cieľ $A$ z $R$ a (premenovanú) klauzulu $A' \leftarrow B_1, \ldots, B_n$ takú, že $\theta = mgu(A, A')$,
  - nahraď $A$ za $B_1, \ldots, B_n$ v množine $R$ a aplikuj $\theta$ na $R$ a $G$,
- ak $R$ je $\{\}$, výstupom je modifikovaný cieľ $G$, inak fail

# SLD–STROM pre append

```
append([],Ys,Ys)
append([X|Xs],Ys,[X|Zs])<-append(Xs,Ys,Zs)
```

$\leftarrow \underline{append([1,2],[3,4],Zs)}$

$\theta_1 = \{X1/1, Xs1/[2], Ys1/[3,4], Zs/[1|Zs1],\}$

$\leftarrow \underline{append([2],[3,4],Zs1)}$

$\theta_2 = \{X2/2, Xs2/[], Ys2/[3,4], Zs1/[2|Zs2]\}$

$\leftarrow \underline{append([],[3,4],Zs2)}$

$\theta_3 = \{Ys3/[3,4], Zs2/[3,4]\}$

$\square$

Výsledok: $append([1,2],[3,4],Zs)\theta_1\theta_2\theta_3 = append([1,2],[3,4],[1,2,3,4])$

$\leftarrow \underline{append(Xs,[3,4],Zs)}$

$\theta_1 = \{Xs/[X1|Xs1], Ys1/[3,4], Zs/[X1|Zs1]\}$

$\theta_1 = \{Xs/[], Ys1/[3,4], Zs/[3,4]\}$

$\leftarrow \underline{append(Xs1,[3,4],Zs1)}$ $\square$

$append([],[3,4],[3,4])$

$\theta_2 = \{Xs1/[X2|Xs2], Ys2/[3,4], Zs1/[X2|Zs2]\}$

$\theta_2 = \{Xs1/[], Ys2/[3,4], Zs1/[3,4]\}$

$\leftarrow \underline{append(Xs2,[3,4],Zs2)}$ $\square$

$append([X1],[3,4],[X1,3,4])$

$\theta_3 = \{Xs2/[X3|Xs3], Ys3/[3,4], Zs2/[X3|Zs3]\}$

$\theta_3 = \{Xs2/[], Ys3/[3,4], Zs2/[3,4]\}$

$\leftarrow \underline{append(Xs3,[3,4],Zs3)}$ $\square$

$append([X1,X2],[3,4],[X1,X2,3,4])$

nekonečný výpočet

```
append([],Ys,Ys)
append([X|Xs],Ys,[X|Zs])<-append(Xs,Ys,Zs)
```

$\leftarrow \underline{append([1,2],Ys,Zs)}$

$\theta_1 = \{X1/1, Xs1/[2], Ys/Ys1, Zs/[1|Zs1],\}$

$\leftarrow \underline{append([2],Ys1,Zs1)}$

$\theta_2 = \{X2/2, Xs2/[], Ys1/Ys2, Zs1/[2|Zs2]\}$

$\leftarrow \underline{append([],Ys2,Zs2)}$

$\theta_3 = \{Ys2/Zs2\}$

$\Box$

Výsledok: $append([1,2],Ys,Zs)\theta_1\theta_2\theta_3 = append([1,2],Zs2,[1,2|Zs2])$

$\leftarrow \underline{append(Xs,Ys,[1,2,3])}$

$\theta_1 = \{X1/1, Zs1/[2,3], Xs/[1|Xs1], Ys/Ys1\}$

$\theta_1 = \{Xs/[], Ys/[1,2,3], Ys1/[1,2,3]\}$

$\leftarrow \underline{append(Xs1,Ys1,[2,3])}$

$\Box$ $append([],[1,2,3],[1,2,3])$

$\theta_2 = \{X2/2, Zs2/[3], Xs1/[2|Xs2], Ys1/Ys2\}$

$\theta_2 = \{Xs1/[], Ys1/[2,3], Ys2/[2,3]\}$

$\leftarrow \underline{append(Xs2,Ys2,[3])}$

$\Box$ $append([1],[2,3],[1,2,3])$

$\theta_3 = \{X3/3, Zs3/[], Xs2/[3|Xs3], Ys2/Ys3\}$

$\theta_3 = \{Xs2/[], Ys2/[3], Ys3/[3]\}$

$\leftarrow \underline{append(Xs3,Ys3,[])}$

$\Box$ $append([1,2],[3],[1,2,3])$

$\theta_4 = \{Xs3/[], Ys3/[], Ys4/[]\}$

$\Box$ $append([1,2,3],[],[1,2,3])$

# PERMUTÁCIE

```
%--- select(X,Y,Z) iff Z = Y-[X]
select(X,[X|Xs],Xs).
select(X,[Y|Ys],[Y|Zs]):-select(X,Ys,Zs).
%--- perm(X,Y) iff Y is permutation of X
perm([],[]).
perm(Xs,[Z|Zs]):-select(Z,Xs,Ys),perm(Ys,Zs).
%----------------------------------------
?- select(X,[1,2,3],[2,3]).
X = 1 ->;
no
?- select(X,[1,2,3],Z).
X = 1
Z = [2,3] ->;
X = 2
Z = [1,3] ->;
X = 3
Z = [1,2] ->;
no
?- perm([1,2,3],[2,3,1]).
yes
?- perm([1,2,3],Y).
Y = [1,2,3] ->;
Y = [1,3,2] ->;
Y = [2,1,3] ->;
Y = [2,3,1] ->;
Y = [3,1,2] ->;
Y = [3,2,1] ->;
no
?-perm(X,[1,2,3]).
X = [1,2,3] ->;
stack overflow
```

# INTERPRETÁCIE

Interpretácia prvorádového jazyka $L$

- neprázdna množina – domain $D$,
- priradenie všetkým konštantám hodnoty z $D$,
- priradenie všetkým $n$-árnym funkčným symbolom zobrazenia z $D^n \rightarrow D$,
- priradenie všetkým $n$-árnym predikátovým symbolom relácie nad $D^n$.

Priradenie premenným $A$ v danej interpretácii $I$ je priradenie všetkým premenným prvky z $D$.

Hodnota termu v interpretácii $I$ s priradením $A$

- hodnota premennej je daná priradením premenných podľa $A$,
- hodnota konštanty je daná priradením konštánt podľa $I$,
- hodnota termu $f(t_1, \ldots, t_n)$ je $f'(t'_1, \ldots, t'_n)$, ak $t'_i$ je hodnota termu $t_i$ a $n$-árnemu funkčnému symbolu $f$ je priradené zobrazenie $f'$ v $I$.

Pravdivostná hodnota formuly v interpretácii $I$ s priradením premenných $A$

- pre atomickú formulu $p(t_1, \ldots, t_n)$ je $p'(t'_1, \ldots, t'_n)$, ak $t'_i$ je hodnota termu $t_i$ a $n$-árnemu predikátovému symbolu $p$ je priradená relácia $p'$ v interpretácii $I$,
- pre formulu tvaru $F \vee G, F \wedge G, \neg F, F \rightarrow G, F \leftarrow G, F \equiv G$ pravdivostnú hodnotu získame aplikáciou logickej operácie na pravdivostné hodnoty formúl $F, G$,
- pre formulu tvaru $(\exists x) F$ je pravdivostná hodnota $true$, ak existuje $d \in D$, že pravdivostná hodnota formuly $F(x/d)$ je $true$, pričom $F(x/d)$ je formula, ktorá z $F$ vznikne nahradením premennej $x$ za $d$,
- pre formulu tvaru $(\forall x)\ F$ je pravdivostná hodnota $true$, ak pre každé $d \in D$ je pravdivostná hodnota formuly $F(x/d)$ je $true$.

# MODELY

Oblasť pôsobenia kvantifikátora $(\forall x)$ vo formule $(\forall x)\, F$ je $F$.

Premenná $x$ má viazaný výskyt vo formule $F$, ak sa nachádza priamo za nejakým kvantifikátorom, alebo v oblasti pôsobenia kvantifikátora s rovnakou premennou.

Premenná je voľná vo formule $F$, ak nemá viazaný výskyt v $F$.

Nech $I$ je interpretácia a $F$ je formula jazyka $L$.

- $F$ je splniteľná v $I$, ak formula $\exists(F)$ je $true$ v interpretácii $I$,
- $F$ je pravdivá v $I$, ak formula $\forall(F)$ je $true$ v interpretácii $I$,
- $F$ je nesplniteľná v $I$, ak formula $\exists(F)$ je $false$ v interpretácii $I$,
- $F$ je nepravdivá v $I$, ak formula $\forall(F)$ je $false$ v interpretácii $I$.

$F$ je uzavretá formula, ak nemá žiadne voľné premenné.

Interpretácia $I$ je modelom uzavretej formule $F$, ak formula $F$ má pravdivostnú hodnotu $true$ v interpretácii $I$ (splniteľná, pravdivá v $I$).

Interpretácia $I$ je modelom konečnej množiny uzavretých formúl $S$, ak je modelom každej formule $F$ z tejto množiny.

Konečná množina uzavretých formúl $S$ jazyka $L$ je

- splniteľná, ak existuje interpretácia $I$, ktorá je modelom $S$,
- pravdivá, ak každá interpretácia $I$ je modelom $S$,
- nesplniteľná, ak žiadna interpretácia $I$ nie je modelom $S$,
- nepravdivá, ak existuje interpretácia $I$, ktorá nie je modelom $S$,

Uzavretá formula $F$ je logickým dôsledkom konečnej množiny uzavretých formúl $S$,ak každá interpretácia $I$, ktorá je modelom $S$, je modelom aj $F$.

Veta (o dedukcii)
Nech $F$ je uzavretá formula $F$ a $S$ je množina uzavretých formúl. Potom $F$ je logickým dôsledkom $S$ práve vtedy, ak $S \cup \{\neg F\}$ je nesplniteľná.

# HERBRANDOVSKÉ INTERPRETÁCIE

Herbrandove univerzum $U_L$ je množina všetkých základných termov jazyka.

Herbrandova báza $B_L$ je množina všetkých základných atómov jazyka $L$.

Herbrandova interpretácia interpretácia, pre ktorú

- domain $D = U_L$,
- konštanty sú zobrazené na seba,
- $n$-árny funkčným symbol $f$ je zobrazený do $(U_L)^n \to U_L$ tak, že $(t_1, \ldots, t_n) \to f(t_1, \ldots, t_n)$.

Veta (o redukcii na Herbrandove modely)
Ak množina klauzúl $S$ má model, potom $S$ má aj Herbrandov model.

Dôsledok:
Množina klauzúl je nesplniteľná práve vtedy, ak nemá Herbrandov model.

Veta (o prieniku Herbrandových modelov)
Nech $P$ je logický program a $\{M_i\}_{i \in I}$ je neprázdna množina Herbrandových modelov programu $P$. Potom $\cap_{\{i \in I\}} M_i$ je Herbrandov model $P$.

Najmenší Herbrandov model programu $P$ je prienik všetkých Herbrandových modelov $P$, t.j. $M_P = \cap\{I \;:\; I\ je\ Herbrandov\ model\ P\}$.

Veta (o najmenšom Herbrandovom modeli) Nech $P$ je logický program.
$M_P = \{A \in B_P \;:\; A$ je logickým dôsledkom programu $P\}$.

---

Definícia $T_P$:
Pre program $P$ definujme $T_P : 2^{B_P} \to 2^{B_P}$ nasledovne:
$T_P(I) = \{A \in B_P : A \leftarrow B_1, \ldots, B_n$ je základná inštancia klauzuly programu $P$ a $\{B_1, \ldots, B_n\} \subseteq I\}$

---

Veta (o Herbrandovom modeli)
Ak $P$ je program a $I$ je Herbrandova interpretácia, potom $I$ je (Herbrandovým) modelom $P$ práve vtedy, ak $T_P(I) \subseteq I$.

# OPERÁTORY A PEVNÉ BODY

Čiastočné usporiadanie je relácia $R$ definovaná na $S$, pre ktorú platí:

- reflexívna, t.j. $(x, x) \in R$, pre všetky $x \in S$
- antisymetrická, t.j. ak $(x, y) \in R$ a $(y, x) \in R$, potom $x = y$ pre všetky $x, y \in S$,
- tranzitívna ak $(x, y) \in R$ a $(y, z) \in R$, potom $(x, z) \in R$ pre všetky $x, y, z \in S$,

Príklad: Relácia $\subseteq$ na $2^B$ je čiastočné usporiadanie.

Nech $S$ je množina s čiastočným usporiadaním $\leq$.
Horné ohraničenie $X \subseteq S$ je $a \in S$ také, že $x \leq a$ pre všetky $x \in X$.
Dolné ohraničenie $X \subseteq S$ je $a \in S$ také, že $a \leq x$ pre všetky $x \in X$.

Supremom množiny $X \subseteq S$ je také horné ohraničenie $sup(X) = a \in S$, že pre ľubovoľné horné ohraničenie $a'$ množiny $X$ platí, že $a \leq a'$.
Infimom množiny $X \subseteq S$ je také dolné ohraničenie $inf(X) = a \in S$, že pre ľubovoľné dolné ohraničenie $a'$ množiny $X$ platí, že $a' \leq a$.

$(S, \leq)$ je úplný zväz ak každá podmnožina má supremum a infimum.
$\bot = inf(S), \top = sup(S)$, resp. $\bot = inf(2^B) = \emptyset, \top = sup(2^B) = B$.

Nech $(L, \leq)$ je úplný zväz.
Zobrazenie $T : L \to L$ je monotónne, ak $T(x) \leq T(y)$ pre ľubovoľné $x \leq y$ z $L$, t.j. ak $T(X) \subseteq T(Y)$ pre ľubovoľné $X \subseteq Y \in 2^B$.

Zobrazenie $T : L \to L$ je finitárne, ak pre ľubovoľnú nekonečnú postupnosť $x_0 \leq x_1 \leq \ldots$ platí : $T(sup\{x_n : n = 0, 1, \ldots\}) \leq \sup\{T(x_n : n = 0, 1, \ldots)\}$, t.j. pre $I_0 \subseteq I_1 \subseteq \ldots$ platí : $T(\cup_{n=0}^{\infty} I_n) \subseteq \cup_{n=0}^{\infty} T(I_n)$.

Zobrazenie $T : L \to L$ je spojité, ak je monotónne a finitárne.

Dôsledok:
Zobrazenie $T : L \to L$ je spojité, ak pre ľubovoľnú nekonečnú postupnosť $x_0 \leq x_1 \leq \ldots$ platí : $T(sup\{x_n : n = 0, 1, \ldots\}) = \sup\{T(x_n : n = 0, 1, \ldots)\}$, t.j. pre $I_0 \subseteq I_1 \subseteq \ldots$ platí: $T(\cup_{n=0}^{\infty} I_n) = \cup_{n=0}^{\infty} T(I_n)$.

Poznámka:
Existuje úplný zväz $(L, \leq)$ a zobrazenie $T : L \to L$, ktoré je monotónne, ale nie je spojité.

Veta o $T_P$:
Ak $P$ je logický program, potom $T_P$ je spojité zobrazenie na úplnom zväze $(2^{B_P}, \subseteq)$.

# PEVNÉ BODY

Pevný bod (fixpoint) zobrazenia $T : L \to L$ je $x \in L$ také, že $T(x) = x$.
Pre-fixpoint zobrazenia $T : L \to L$ je $x \in L$ také, že $T(x) \leq x$.
Post-fixpoint zobrazenia $T : L \to L$ je $x \in L$ také, že $x \leq T(x)$.

Dôsledok vety o Herbrandovom modeli:
Ak $P$ je program a $I$ je Herbrandova interpretácia, potom $I$ je modelom $P$ práve vtedy, ak $I$ je pre-fixpoint zobrazenia $T_P$.

Najmenším fixpointom zobrazenia $T : L \to L$ je $lfp(T) = x \in L$ také, že $T(x) = x$ a pre ľubovoľný iný fixpoint $b$ $(T(b) = b)$ platí, že $a \leq b$.
Najväčším fixpointom zobrazenia $T : L \to L$ je $gfp(T) = x \in L$ také, že $T(x) = x$ a pre ľubovoľný iný fixpoint $b$ platí, že $b \leq a$.

Veta (o fixpointe –Tarskeho veta)
Nech $(L, \leq)$ je úplný zväz a $T : L \to L$ je monotónne zobrazenie. Potom $T$ má najmenší (najväčší) fixpoint. Navyše platí, že
$lfp(T) = inf\{x : T(x) = x\} = inf\{x : T(x) \leq x\}$ a
$gfp(T) = sup\{x : T(x) = x\} = sup\{x : x \leq T(x)\}$.

Dôsledok:
Nech $(L, \leq)$ je úplný zväz a $T : L \to L$ je monotónne zobrazenie. Ak $a \in L$ a $a \leq T(a)$, potom existuje fixpoint $a'$ taký, že $a \leq a'$.
Analogicky, ak $b \in L$ a $T(b) \leq b$, potom existuje fixpoint $b'$ taký, že $b' \leq b$.

# ORDINÁLNE ČÍSLA

Konštrukcia ordinálnych čísel:

$$
\begin{aligned}
0 &= \emptyset \\
1 &= \{\emptyset\} = \{0\} \\
2 &= \{\emptyset, \{\emptyset\}\} = \{0, 1\} \\
3 &= \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\} = \{0, 1, 2\} \\
\omega &= \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}, \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}, \ldots\} = \{0, 1, 2, 3, \ldots\} \\
\omega + 1 &= \omega \cup \{\omega\}
\end{aligned}
$$

Nasledujúce ordinálne číslo: $\alpha + 1 = \alpha \cup \{\alpha\}$.
Ak $\alpha = \beta + 1$, $\beta$ nazývame predchádzajúce ordinálne číslo pre $\alpha$ a môžeme písať $\beta = \alpha - 1$.
Limitné ordinálne číslo: je také ordinálne číslo, ktoré nemá predchádzajúce ordinálne číslo.

Príklady limitných ordinálnych čísiel:
$0, \omega, \omega + \omega = \omega 2, \omega + \omega + \omega = \omega 3, \ldots, \omega n, \ldots$

$\alpha < \beta$ ak $\alpha \in \beta$.

Princíp transfinitnej indukcie:
Ak platí $P(0) \ \wedge \ (\forall \gamma < \beta)\, P(\gamma) \rightarrow P(\beta)$, potom $(\forall \alpha)\, P(\alpha)$.

Iterácia zobrazenia:
Ak $(L, \leq)$ je úplný zväz a $T : L \rightarrow L$ je monotónne zobrazenie. Definujeme:

$$
\begin{aligned}
T \uparrow 0 &= \bot, \\
T \uparrow \alpha &= T(T \uparrow \beta),\ ak\ \beta = \alpha - 1, \\
T \uparrow \alpha &= sup(T \uparrow \beta : \beta < \alpha),\ \alpha\ je\ limit.\ ordinal \\
T \downarrow 0 &= \top, \\
T \downarrow \alpha &= T(T \downarrow \beta),\ ak\ \beta = \alpha - 1, \\
T \downarrow \alpha &= inf(T \downarrow \beta : \beta < \alpha),\ \alpha\ je\ limit.\ ordinal
\end{aligned}
$$

# PRÍKLAD $T_P \uparrow \alpha(\emptyset), T_P \downarrow \alpha(B_P)$

```
p(a)<-p(X),q(X)
p(f(X))<-p(X)
q(b)<-
q(f(X))<-q(X)
```

$$
\begin{aligned}
T_P \uparrow 0(\emptyset) &= \bot = \{\} \\
T_P \uparrow 1(\emptyset) &= \{q(b)\} \\
T_P \uparrow 2(\emptyset) &= \{q(b), q(f(b))\} \\
T_P \uparrow 3(\emptyset) &= \{q(b), q(f(b)), q(f(f(b)))\} \\
&\dots \\
T_P \uparrow n(\emptyset) &= \{q(f^i(b)) : i < n\} \\
&\dots \\
T_P \uparrow \omega(\emptyset) &= \{q(f^i(b)) : i < \omega\} = lfp(T_P)
\end{aligned}
$$

$$
\begin{aligned}
T_P \downarrow 0(B_P) &= \top = B_P \\
T_P \downarrow 1(B_P) &= \{q(f^{i+1}(a)), q(f^i(b)), p(f^i(a)), p(f^{i+1}(b)) : i < \omega\} \\
T_P \downarrow 2(B_P) &= \{q(f^{i+2}(a)), q(f^i(b)), p(f^i(a)), p(f^{i+2}(b)) : i < \omega\} \\
T_P \downarrow 3(B_P) &= \{q(f^{i+3}(a)), q(f^i(b)), p(f^i(a)), p(f^{i+3}(b)) : i < \omega\} \\
&\dots \\
T_P \downarrow n(B_P) &= \{q(f^{i+n}(a)), q(f^i(b)), p(f^i(a)), p(f^{i+n}(b)) : i < \omega\} \\
&\dots \\
T_P \downarrow \omega(B_P) &= \{q(f^i(b)), p(f^i(a)) : i < \omega\} \\
T_P \downarrow (\omega+1)(B_P) &= \{q(f^i(b)), p(f^{i+1}(a)) : i < \omega\} \\
T_P \downarrow (\omega+2)(B_P) &= \{q(f^i(b)), p(f^{i+2}(a)) : i < \omega\} \\
&\dots \\
T_P \downarrow (\omega+n)(B_P) &= \{q(f^i(b)), p(f^{i+n}(a)) : i < \omega\} \\
&\dots \\
T_P \downarrow (\omega+\omega)(B_P) &= \{q(f^i(b)) : i < \omega\} = gfp(T_P)
\end{aligned}
$$

# VLASTNOSTI MOCNÍN: $T \uparrow \alpha, T \downarrow \alpha$

$T \uparrow \alpha = T \uparrow \alpha(\bot)$
$T \downarrow \alpha = T \downarrow \alpha(\top)$

Veta
Ak $(L, \leq)$ je úplný zväz a $T : L \to L$ je monotónne zobrazenie.

- pre ľubovoľné $\alpha$ platí $T \uparrow \alpha \leq lfp(T)$ (resp. $T \downarrow \alpha \geq gfp(T)$),
- pre ľubovoľné $\alpha$ platí $T \uparrow \alpha \leq T \uparrow (\alpha{+}1)$ (resp. $T \downarrow \alpha \geq T \downarrow (\alpha{+}1)$),
- pre ľubovoľné $\alpha < \beta$ platí $T \uparrow \alpha \leq T \uparrow \beta$ (resp. $T \downarrow \alpha \geq T \downarrow \beta$),
- pre ľubovoľné $\alpha < \beta$, ak platí $T \uparrow \alpha = T \uparrow \beta$, potom $T \uparrow \alpha = lfp(T)$ (resp. $T \downarrow \alpha = gfp(T)$),
- existuje $\beta$ taká, že ak $\beta \leq \alpha$, potom $T \uparrow \alpha = lfp(T)$ (resp. $T \downarrow \alpha = gfp(T)$)

$\alpha$ je dolný uzáver zobrazenia $T$, ak je to najmenší ordinál taký, že $T \uparrow \alpha = lfp(T)$

$\alpha$ je horný uzáver zobrazenia $T$, ak je to najmenší ordinál taký, že $T \downarrow \alpha = gfp(T)$

Veta (Kleeneho)
Ak $(L, \leq)$ je úplný zväz a $T : L \to L$ je spojité zobrazenie. Potom $lfp(T) = T \uparrow \omega$.

---

Zobrazenie $T : L \to L$ je zdola spojité, ak pre ľubovoľnú nekonečnú postupnosť $x_0 \geq x_1 \geq \ldots$ platí : $T(inf\{x_n : n = 0, 1, \ldots\}) = \inf\{T(x_n : n = 0, 1, \ldots)\}$, t.j. pre $I_0 \supseteq I_1 \supseteq \ldots$ platí: $T(\cap_{n=0}^{\infty} I_n) = \cap_{n=0}^{\infty} T(I_n)$.

Veta (Kleeneho)
Ak $(L, \leq)$ je úplný zväz a $T : L \to L$ je zdola spojité monotónne zobrazenie. Potom $gfp(T) = T \downarrow \omega$.

Veta Nech $P$ je logický program. Potom $M_P = lfp(T_P) = T_P \uparrow \omega$.

---

Korektná odpoveďová substitúcia pre logický program $P$ a cieľ $G = \leftarrow A_1, \ldots, A_n$ je substitúcia $\theta$, ak platí, že $\forall((A_1 \wedge \ldots \wedge A_n)\theta)$ je logickým dôsledkom $P$.

Veta: Nech $P$ je logický program, $G = \leftarrow A_1, \ldots, A_n$ je cieľ a $\theta$ je substitúcia taká, že $(A_1 \wedge \ldots \wedge A_n)\theta$ je základná formula. Potom nasledujúce tvrdenia sú ekvivalentné:

- $\theta$ je korektná odpoveďová substitúcia,
- $(A_1 \wedge \ldots \wedge A_n)\theta$ je $true$ vo všetkých Herbrandových modeloch $P$,
- $(A_1 \wedge \ldots \wedge A_n)\theta$ je $true$ v najmenšom Herbrandovom modeli $M_P$,

---

# CVIČENIA

---

1. nájdi $P$ taký, že $gfp(T_P) \neq T_P \downarrow \omega$,
2. nájdi $P$ taký, že $gfp(T_P) \neq lfp(T_P)$,
3. nájdi $P$ taký, že $gfp(T_P) = T_P \downarrow (\omega n)$, $gfp(T_P) \neq T_P \downarrow (\omega(n-1))$,
4. nájdi $P$ taký, že $T_P$ nie je zdola spojité,
5. ak $P$ neobsahuje funkčné symboly, potom $gfp(T_P) = T_P \downarrow \omega$,
6. ak sa v každej klauzule logického programu $P$ všetky premenné nachádzajúce sa na pravej strane (tele) nachádzajú aj na ľavej strane (hlave) implikácie, potom $gfp(T_P) = T_P \downarrow \omega$,
7. ak sa v každej klauzule logického programu $P$ všetky premenné nachádzajúce sa na pravej strane (tele) nachádzajú aj na ľavej strane (hlave) implikácie, alebo množiny premenných tela a hlavy klauzuly sú disjunktné, potom existuje $n$ také, že $gfp(T_P) = T_P \downarrow \omega n$ (pričom $n$ závisí len od programu $P$),
8. nájdi monotónne ale nie spojité zobrazenie $T$ také, že $lfp(T) \neq T \uparrow \omega$,
9. nájdi program $P$ taký, že $\omega\omega = \omega^2$ je horný uzáver zobrazenia $T_P$.

# SLD–ODVODENIE

Krok SLD–odvodenia
Nech $G = \leftarrow A_1, \ldots, A_m, \ldots, A_k$ a $C$ je klauzula tvaru $A \leftarrow B_1, \ldots, B_q$. Cieľ $G'$ je odvodený z cieľa $G$ pomocou klauzuly $C$ a substitúcie $\theta$, ak platí:

- $A_m$ je zvolený atóm v $G$,
- $\theta$ je najvšeobecnejší unifikátor $A_m$ a $A$,
- rezolventa $G'$ má tvar $\leftarrow (A_1, \ldots, A_{m-1}, B_1, \ldots, B_q, A_{m+1}, \ldots, A_k)\theta$.

SLD–odvodenie $P \cup \{G\}$ je (konečná alebo nekonečná) postupnosť cieľov $G = G_0, G_1, \ldots$, postupnosť variantov klauzúl programu $P$: $C_1, C_2, \ldots$ a postupnosť substitúcií $\theta_1, \theta_2, \ldots$ taká, že platí: cieľ $G_{i+1}$ je odvodený z $G_i$ pomocou klauzuly $C_{i+1}$ a substitúcie $\theta_{i+1}$.

SLD–zamietnutie $P \cup \{G\}$ je také konečné SLD–odvodenie, ktoré končí prázdnou klauzulou $\Box$. Ak $G_n = \Box$, hovoríme, že dĺžka odvodenia je $n$.

Neobmedzené SLD–zamietnutie $P \cup \{G\}$ je SLD–zamietnutie, v ktorom nahradíme podmienku, že $\theta_i$ sú najvšeobecnejšie unifikátory podmienkou, že $\theta_i$ sú unifikátory.

SLD–strom pre program $P$ a cieľ $G$ je strom spĺňajúci nasledujúce podmienky:

- každý vrchol stromu je cieľ (eventuálne prázdny),
- koreňom je cieľ $G$,
- ak $\leftarrow A_1, \ldots, A_m, \ldots, A_k$ a $(k > 0)$ je nejaký vrchol v SLD–strome a $A_m$ je zvolený atóm v cieli, potom pre každý variant klazuly $A \leftarrow B_1, \ldots, B_q$ programu $P$ takej, že $\theta$ je najvšeobecnejší unifikátor $A_m$ a $A$, je vrchol $\leftarrow (A_1, \ldots, A_{m-1}, B_1, \ldots, B_q, A_{m+1}, \ldots, A_k)\theta$ nasledovníkom vrchola $\leftarrow A_1, \ldots, A_m, \ldots, A_k$. (Poznámka: na hranu píšeme variant použitej klauzuly a substitúciu $\theta$)).
- vrcholy tvorené prázdnym cieľom nemajú nasledovníkov.

# BEZOSPORNOSŤ SLD–ODVODENIA

Úspešná množina $SS_P$ programu $P$ je množina všetkých $A \in B_P$ takých, že $P \cup \{\leftarrow A\}$ má SLD–zamietnutie.

Vypočítaná odpoveďová substitúcia pre program $P$ a cieľ $G$ je taká substitúcia $\theta$, ktorú získame z kompozície $\theta_1\theta_2\ldots\theta_n$ reštrikciou na premenné cieľa $G$ ($\theta = \theta_1\theta_2\ldots\theta_n/Var(G)$), pričom $\theta_1, \theta_2, \ldots, \theta_n$ je postupnosť najvšeobecnejších unifikátorov nejakého SLD–zamietnutia $P \cup \{G\}$.

Veta (Bezospornosť SLD–rezolúcie)
Nech $P$ je logický program a $G$ je cieľ. Potom každá vypočítaná odpoveďová substitúcia pre $P \cup \{G\}$ je tiež korektná odpoveďová substitúcia pre $P \cup \{G\}$.

Dôsledok
Nech $P$ je logický program a $G$ je cieľ. Ak existuje SLD–zamietnutie pre $P \cup \{G\}$, potom $P \cup \{G\}$ je nesplniteľná.

Dôsledok
Nech $P$ je logický program. Potom $SS_P \subseteq M_P$.

Veta
Nech $P$ je logický program a $G = \leftarrow A_1, \ldots, A_k$ je cieľ. Ak existuje SLD–zamietnutie pre $P \cup \{G\}$ dĺžky $n$, kde $\theta_1, \theta_2, \ldots \theta_n$ je postupnosť najvšeobecnejších unifikátorov, potom $\cup_{j=1}^{k}[A_j\theta_1\theta_2\ldots\theta_n] \subseteq T_P \uparrow n$.

Poznámka
$[A] = \{A\theta \in B_P : \text{pre nejakú substitúciu } \theta\}$.

---

## Occur check – kontrola výskytu

---

```
test1 <- p(X,X)
p(X,f(X)) <-
```

```
test2 <- p(X,X)
p(X,f(X)) <- p(X,X)
```

```
test3 <- append(X-X,Y-Y,[1|Z]-Z)
append(A-B,B-C,A-C) <-
```

# Úplnosť SLD–rezolúcie

Lema (Mgu)

Nech $P$ je logický program a $G$ je cieľ. Ak existuje neohraničené SLD–zamietnutie pre $P \cup \{G\}$ dĺžky $n$, potom existuje SLD–zamietnutie pre $P \cup \{G\}$ rovnakej dĺžky. Navyše platí, že ak $\theta_1, \theta_2, \ldots \theta_n$ je postupnosť unifikátorov neohraničeného SLD–zamietnutia a $\theta'_1, \theta'_2, \ldots \theta'_n$ je postupnosť najvšeobecnejších unifikátorov SLD–zamietnutia, potom existuje substitúcia $\gamma$ taká, že $\theta_1, \theta_2, \ldots \theta_n = \theta'_1, \theta'_2, \ldots \theta'_n \gamma$.

Lema (Lifting)

Nech $P$ je logický program, $G$ je cieľ a $\theta$ je substitúcia. Ak existuje SLD–zamietnutie pre $P \cup \{G\theta\}$ dĺžky $n$, potom existuje SLD–zamietnutie pre $P \cup \{G\}$ rovnakej dĺžky. Navyše platí, že ak $\theta_1, \theta_2, \ldots \theta_n$ je postupnosť najvšeobecnejších unifikátorov SLD–zamietnutia $P \cup \{G\theta\}$ a $\theta'_1, \theta'_2, \ldots \theta'_n$ je postupnosť najvšeobecnejších unifikátorov SLD–zamietnutia $P \cup \{G\}$, potom existuje substitúcia $\gamma$ taká, že $\theta\theta_1, \theta_2, \ldots \theta_n = \theta'_1, \theta'_2, \ldots \theta'_n \gamma$.

Veta (Úplnosť SLD–rezolúcie)

Nech $P$ je logický program. Potom platí, že $SS_P = M_P$.

Veta

Nech $P$ je logický program a $G$ je cieľ. Ak $P \cup \{G\}$ je nesplniteľná, potom existuje SLD–zamietnutie pre $P \cup \{G\}$.

Lema

Nech $P$ je logický program a $A$ je atóm taký, že $\forall(A)$ je logickým dôsledkom $P$. Potom existuje SLD-zamietnutie pre $P \cup \{\leftarrow\ A\}$ s $\{\}$ vypočítanou odpoveďovou substitúciou.

Veta (Úplnosť SLD–rezolúcie)

Nech $P$ je logický program a $G$ je cieľ. Pre každú korektnú odpoveďovú substitúciu $\theta$ pre $P \cup \{G\}$ existuje vypočítaná odpoveďová substitúcia $\sigma$ pre $P \cup \{G\}$ taká, že $\theta = \sigma\gamma$, pre nejaké $\gamma$.

# Nezávislosť výpočtového pravidla

Výpočtové pravidlo R je funkcia z množiny cieľov do množiny atómov. Ak $G = \leftarrow\ A_1, \ldots, A_{m-1}, A_m, A_{m+1}, \ldots, A_n$ a $R(G) = A_m$, hovoríme, že $A_m$ je zvolený atóm v cieli $G$.

Nezávislosť výpočtového pravidla
Nech $P$ je logický program a $G$ je cieľ. Ak existuje SLD–zamietnutie pre $P \cup \{G\}$ s vypočítanou odpoveďovou substitúciou $\sigma$. Potom pre ľubovoľné výpočtové pravidlo $R$ existuje SLD–zamietnutie $P \cup \{G\}$ s vypočítanou odpoveďovou substitúciou $\sigma'$ a platí, že $G\sigma'$ je variant $G\sigma$.

$SS_P^R$ je množina $A \in B_P$ takých, že existuje SLD–zamietnutie s výpočtovým pravidlom $R$ pre $P \cup \{G\}$.

Veta: $SS_P^R = M_P$.

Veta
Nech $P$ je logický program, $R$ je výpočtové pravidlo a $G$ je cieľ. Ak $P \cup \{G\}$ je nesplniteľná, potom existuje SLD–zamietnutie pre $P \cup \{G\}$ s výpočtovým pravidlom $R$.

Veta (Úplnosť SLD–rezolúcie)
Nech $P$ je logický program, $R$ je výpočtové pravidlo a $G$ je cieľ. Pre každú korektnú odpoveďovú substitúciu $\theta$ pre $P \cup \{G\}$ existuje $R$-vypočítaná odpoveďová substitúcia $\sigma$ pre $P \cup \{G\}$ taká, že $\theta = \sigma\gamma$, pre nejaké $\gamma$.

---

## VYPOČÍTATEĽNOSŤ

---

Veta (Sila logických programov)
Nech $f$ je $n$-árna čiastočne rekurzívna funkcia. Potom existuje logický program s $n+1$-árnym predikátovým symbolom $p_f$ taký, že všetky vypočítané odpoveďové substitúcie pre $P \cup \{\leftarrow\ p_f(s^{k_1}(0), \ldots, s^{k_n}(0), X)\}$ majú tvar $\{x/s^k(0)\}$, práve vtedy, ak $f(k_1, \ldots, k_n) = k$ pre ľubovoľné nezáporné $k_1, \ldots, k_n$.

# Rekurzia a iterácia

Iteratívna klauzula je Hornova klauzula obsahujúca v tele práve jedno rekurzívne volanie, a to ako posledný podcieľ.
Iteratívna procedúra obsahuje len fakty a iteratívne klauzuly.

---

```
%--- fact(N,F) akk F = N! (rekurzivna verzia)
fact(0,0).
fact(N,F):-N>0,N1 is N-1,fact(N1,F1),F is F1*N.

fact(N,F):-factorial(0,N,1,F). (iterativna verzia-1)
factorial(I,N,T,F):-
  I<N,I1 is I+1,T1 is T*I1,factorial(I1,N,T1,F).
factorial(N,N,F,F).

fact(N,F):-factorial(N,1,F).   (iterativna verzia-2)
factorial(N,T,F):-
  N>0,T1 is T*N,N1 is N-1,factorial(N1,T1,F).
factorial(0,F,F).
```

---

```
%--- product(X,Y,P) akk P je skalarny sucin vektorov X,Y
product([],[],0).
product([X|Xs],[Y|Ys],P):-product(Xs,Ys,P1),P is P1+X*Y.

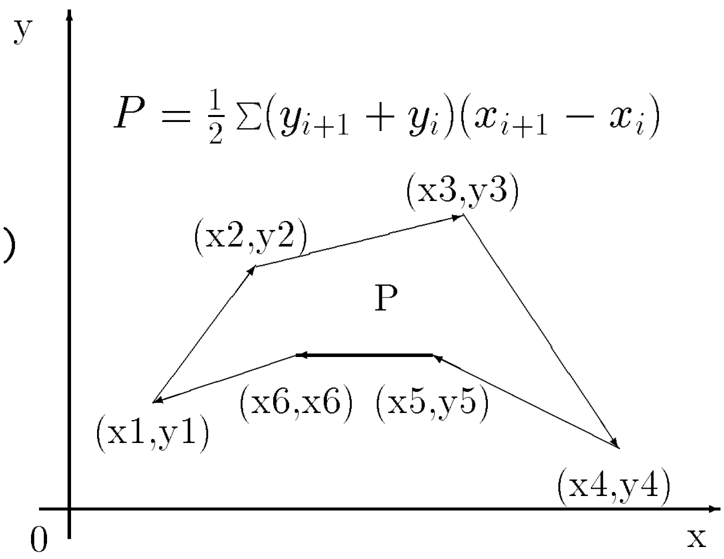
%--- product(X,Y,P) akk P je skalarny sucin vektorov X,Y
product(X,Y,P):-prod(X,Y,0,P).
prod([],[],P,P).
prod([X|Xs],[Y|Ys],P1,P):-P2 is P1+X*Y,prod(Xs,Ys,P2,P).
```

---

```
%--- between(I,J,K) akk K je z intervalu <I;J>
between(I,J,I):-I=<J.
between(I,J,K):-I<J,I1 is I+1,between(I1,J,K).
```

# Technika akumulátorov

Príklad (Polygón):




```
%--- area(Polygon,P) akk
     P je plocha Polygonu
area([_],0).
area([(X1,Y1),(X2,Y2)|Z],P)
  :-area([(X2,Y2)|Z],P1),
  P is (X1*Y2-Y1*X2)/2+P1.

%--- iterativna verzia
area(Polygon,P):-area(Polygon,0,P)
area([_],P,P).
area([(X1,Y1),(X2,Y2)|XYs],P1,P):-
  P2 is (X1*Y2-Y1*X2)/2+P1,area([X2,Y2)|XYs],P2,P).
```

---

Príklad (Reverse):

```
%--- iterativna verzia reverse
reverse(X,Y):-reverse(X,[],Y).
reverse([],Y1,Y).
reverse([X|Xs],Y1,Y):-reverse(Xs,[X|Y1],Y).
```

---

# STROMOVÁ REKURZIA

---

Príklad (Fibonacci): $0, 1, 1, 2, 3, 5, 8, \ldots$

```
%--- fib(N,F) akk F je N-te fibonacciho cislo
fib(N,N):-N<2.
fib(N,F):-N>1,N1 is N-1,N2 is N-2,
          fib(N1,F1),fib(N2,F2),F is F1+F2.
%--- fib(N,F) akk F je N-te fibonacciho cislo
fib(N,F):-fib(N,0,1,F).
fib(0,F1,F2,F1).
fib(N,F1,F2,F):-N>0,N1 is N-1,F3 is F1+F2,fib(N1,F2,F3,F).
```

# SPLOŠŤOVANIE

```
%--- flat(Xs,Ys) akk Ys is list of the elements of Xs
flat([X|Xs],Ys):-flat(X,Ys1),flat(Xs,Ys2),
                 append(Ys1,Ys2,Ys).
flat(X,[X]):-atomic(X),X \= [].
flat([],[]).

%--- flat(Xs,Ys,Ys1) akk Ys1-Ys is list of elements of Xs
flat(X,Y):-flat(X,[],Y).
flat([X|Xs],Ys1,Ys):-flat(X,Ys1,Ys2),flat(Xs,Ys2,Ys).
flat(X,Ys,[X|Ys]):-atomic(X),X \= [].
flat([],Ys,Ys).
%--- using stack
flat(Xs,Ys):-flat(Xs,[],Ys).
flat([X|Xs],S,Ys):-ifthenelse(X=[_|_],flat(X,[Xs|S],Ys),
                                    (Ys=[X|Zs],flat(Xs,S,Zs))).
flat([],[X|S],Ys):-flat(X,S,Ys).
flat([],[],[]).
%--- Rene's solution
flat([],[]).
flat([[]|Xs],Ys):-flat(Xs,Ys).
flat([X|Xs],[X|Ys]):-atomic(X), X \= [], flat(Xs,Ys).
flat([[Z|Zs]|Xs],Y):-flat([Z|[Zs|Xs]],Y).
```

---

Miery:

1. $m_1([X|Xs]) = m_1(X) + m_1(Xs), m_1(atomic - []) = 1, m_1([]) = 0,$
2. $m_2(X) = m_1(X),$
3. ak $S = [y_1, y_2, \ldots, y_m]$ a $A = [X|Xs]$, potom $m_3(A, S) =$lexikograf$(m_1(y_m), \ldots, m_1(y_2), m_1(y_1), m_1(A))$.
4. $m_4([X|Xs]) = 2{*}m_4(X){+}m_4(Xs), m_4(atomic{-}[]) = 1, m_4([]) = 0.$

# ACKERMANNOVA FUNKCIA

$ackermann(0, N) = N + 1$
$ackermann(M, 0) = ackermann(M - 1, 1)$
$ackermann(M, N) = ackermann(M - 1, ackermann(M, N - 1))$

| M : N | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 2 | 3 | 5 | 7 | 9 | 11 |
| 3 | 5 | 13 | | | |
| 4 | 13 | | | | |

```
%--- ackermann(M,N,A) akk A=Ackermann(M,N) (rekurzivne)
ackermann(0,N,s(N)).
ackermann(s(M),0,Val):-ackermann(M,s(0),Val).
ackermann(s(M),s(N),Val):-
   ackermann(s(M),N,Val1), ackermann(M,Val1,Val).
```

Iteratívne riešenie:
$(N, [M|\alpha]) \ \rightarrow \ (Ackermann(M, N), \alpha)$

1. $(N, [0|\alpha]) \ \rightarrow \ (Ackermann(0, N), \alpha) = (s(N), \alpha)$,
2. $(0, [s(M)|\alpha]) \ \rightarrow \ (s(0), [M|\alpha])$,
3. $(s(N), [s(M)|\alpha]) \ \rightarrow \ (N, [s(M), M, |\alpha])$.

```
%--- ackermann(M,N,A) akk A=Ackermann(M,N)
ackerman(M,N,Ack):-ack(N,[M],Ack).
ack(Ack,[],Ack).
ack(N,[0|S],Ack):-ack(s(N),S,Ack).
ack(0,[s(M)|S],Ack):-ack(s(0),[M|S],Ack).
ack(s(N),[s(M)|S],Ack):-ack(N,[s(M),M|S],Ack).
```

# LOGICKÉ PREMENNÉ

Kombinačné čísla:

```
komb(N,0,1).
komb(N,N,1).
komb(N,K,C):-N>0,N\=K,N1 is N-1,K1 is K-1,
             komb(N1,K1,C1),komb(N1,K,C2),
             C is C1+C2.
```

Iteratívne riešenie:

1. $[komb(N,0,C)|\alpha] \rightarrow \alpha$ a $C=1$,
2. $[komb(N,N,C)|\alpha] \rightarrow \alpha$ a $C=1$,
3. $[komb(N,K,C)|\alpha]$ $a$ $(N>0, K \neq N) \rightarrow$ $[komb(N-1,K-1,C1), komb(N-1,K,C2), plus(C1,C2,C)|\alpha]$

Interpreter pre kombinačné čísla:

```
int_komb([]).
int_komb([komb(N,0,C)|Stack]):-C=1,int_komb(Stack).
int_komb([komb(N,N,C)|Stack]):-C=1,int_komb(Stack).
int_komb([komb(N,K,C)|Stack]):-N>0,K\=N,
                     N1 is N-1,K1 is K-1,
                     int_komb([komb(N1,K1,C1),
                               komb(N1,K,C2),
                               plus(C1,C2,C)|Stack]).
int_komb([plus(C1,C2,C)|Stack]):-C is C1+C2,
                     int_komb(Stack).

komb(N,K,C):-int_komb([komb(N,K,C)]).
```

# Problémy na cvičenie:

- definujte iteratívnu verziu predikátu $power(X, N, V)$, ktorý platí práve vtedy, ak $V = X^N$. Uvažujte o zložitosti algoritmu (sublineárna zložitosť), nájdite klesajúcu mieru pre iteratívne riešenie.
- definujte iteratívnu verziu predikátu $comb(N, K, C)$, ktorý platí práve vtedy, ak $C$ je kombinačné číslo $N$ nad $K$. Uvažujte o zložitosti algoritmu, nájdite klesajúcu mieru pre iteratívne riešenie.
- nájdite klesajúcu mieru pre iteratívne riešenie ackermannovej funkcie.
- Nech $a, b, c, d, g, p, r$ sú ľubovoľné predikáty. Definujte iteratívnu verziu nasledujúcej definície:

```
f(X,Y,Result):-
   ifthenelse(p(X,Y),
              r(X,Y,Result),
              (a(X,Xa),b(Y,Yb),f(Xa,Yb,XaYb),
               c(X,Xc),d(Y,Yd),f(Xc,Yd,XcYd),
               g(XaYb,XcYd,Result))).
```

- Definujte binárny predikát $otoc(cislo, obraz)$ ako rekurzívny a iteratívny proces. Príklad: $otoc(34060, 6043)$.
- Fibonacciho čísla sú definované ako

$$Fib(n) = \begin{cases} 0 & \text{ak n=0} \\ 1 & \text{ak n=1} \\ Fib(n-1) + Fib(n-2) & \text{inak} \end{cases}$$

  Definujte binárny predikát fibonacci(N,F), ktorý platí akk $F$ je $N$-té Fibonacciho číslo, ako a) rekurzívny, b) iteratívny proces.
- Definujte euklidov algoritmus NSD ako iteratívny proces.
- Rozmieňanie mincí. Definujte predikát $pocet\_rozmeneni(Suma, Pocet)$. Predpokladajte, že hodnoty mincí máte dané (nie sú vstupom programu). Z každej mince máte dostatočný počet kusov. Rozmenenia, ktoré sa líšia len poradím mincí sa počítajú iba raz! Funkciu definujte ako a) rekurzívny, b) iteratívny proces (pozn. ťažké).
- Definujte predikát $polovica(N, Polovica)$, ktorý počíta $\lceil N/2 \rceil$ bez použitia delenia (rekurzívne aj iteratívne).
- Ako iteratívny proces definujte preklad daného nezáporného čísla do dvojkovej sústavy, napr. (1 0 0 1).

# MANIPULÁCIA S TERMAMI

SYSTÉMOVÉ PREDIKÁTY:

- $functor(Term, Name, Arity)$ ak $Term$ nie je premenná, unifikuje $Name$ s menom funkčného symbola termu $Term$ a $Arity$ s jeho aritou. Ak $Term$ je premenná, unifikuje $Term$ s $f(X_1, \ldots, X_n)$,
- $arg(N, Term, X)$ unifikuje $X$ s hodnotou $N$-tého argumentu termu $Term$,
- $arg0(N, Term, X)$ unifikuje $X$ s hodnotou $(N + 1)$-vého argumentu termu $Term$,
- $Term = ..List$ prevádza term $Term$ do zoznamu $List$. Prvý prvok tohoto listu je funkčný symbol termu $Term$, za ktorým nasledujú argumenty termu $Term$. Taktiež prevádza zoznam do termu inverzným spôsobom.

## PRÍKLADY

```
%--- subterm(T1,T2) akk T1 is subterm T2
subterm(T,T).
subterm(T1,T2):-functor(T2,F,N), subterm(N,T1,T2).
subterm(N,T1,T2):-(arg(N,T2,Arg), subterm(T1,Arg) ;
        N1 is N-1, N1 >= 1, subterm(N1,T1,T2)).

subterm(T,T).
subterm(T1,T2):-T2 =.. [F|Args],subterm1(T1,Args).
subterm1(T1,[Arg|Args]):-
        subterm(T1,Arg) ; subterm1(T1,Args).

%--- converts term to list
term_to_list(X,[F|Y]):-term_to_list(1,X,Y).
term_to_list(N,X,Ys1):-ifthenelse(arg(N,X,XX),
        (N1 is N+1,Ys1=[XX|Ys],term_to_list(N1,X,Ys),
        Ys1 = []).
```

# MANIPULÁCIA S TERMAMI

```
list_to_term([F|Args],Y):-length(Args,N),
        functor(Y,F,N),list_to_term(Args,1,Y).
list_to_term([],_,_).
list_to_term([Arg|Args],N,Y):-arg(N,Y,Arg),
        N1 is N+1, list_to_term(Args,N1,Y).
```

---

CVIČENIA:

- definujte predikát $pocet\_vyskytov(Sub, Term, N)$, ktorý platí, ak sa term $Sub$ $N$-krát vyskytuje v terme $Term$.
- definujte predikát $position(Subterm, Term, Path)$, kde $Path$ je zoznam pozícií argumentov od koreňa k termu $Subterm$ v terme $Term$. Príklad: $position(a, foo(b, goo(a, c)), [2, 1])$.
- definujte predikáty $functor$ a $arg$ pomocou (=..).
- opraviť derivovanie.

---

## Manipulácia s premennými

---

SYSTÉMOVÉ PREDIKÁTY:

- $var(X)$ determines whether $X$ is an uninstantiated variable,
- $nonvar(X)$ determines whether $X$ is not an uninstantiated variable,
- `X = Y` determines whether $X$ and $Y$ are unifiable,
- `X \ = Y` determines whether $X$ and $Y$ are not unifiable,
- `X == Y` determines whether $X$ and $Y$ are equivalent,
- `X \ == Y` determines whether $X$ and $Y$ are not equivalent.

```
%--- ground(Term) akk Term je zakladny term
ground(Term):-nonvar(Term),(atomic(Term) ;
                      functor(Term,F,N),ground(N,Term)).
ground(N,Term):-nonvar(Term),ifthen(N>0,(arg(N,Term,Arg),
                ground(Arg),N1 is N-1,ground(N1,Term))).
```

---

TRIVIÁLNA DEFINÍCIA (využíva zabudovanú prologovskú unifikáciu, problém s occur-checkom):

```
unify(X,X).
```

2.VERZIA (nevyužíva zabudovanú prologovskú unifikáciu, stále problém s occur-checkom):

```
unify(X,Y):-(var(X) ; var(Y)),!,X=Y.
unify(X,Y):-X=..[F|Xs],Y=..[F|Ys],unify_list(Xs,Ys).
unify_list([],[]).
unify_list([X|Xs],[Y|Ys]):-unify(X,Y),unify_list(Xs,Ys).
```

3.VERZIA (implementácia occur-checku):

```
%--- not_in(X,Y) akk premenna X sa nenachadza v terme Y
not_in(X,Y):-var(Y),!,X \== Y.
not_in(X,Y):-atomic(Y),!.
not_in(X,Y):-functor(Y,F,N),N>0,not_in(X,N,Y).
not_in(X,N,Y):-ifthen(N>0,(arg(N,Y,YY),not_in(X,YY),
                          N1 is N-1,not_in(X,N1,Y))).
unify(X,Y):-var(X), var(Y),!,X=Y.
unify(X,Y):-nonvar(X),var(Y),!,not_in(Y,X),X=Y.
unify(X,Y):-nonvar(Y),var(X),!,not_in(X,Y),X=Y.
unify(X,Y):-functor(X,F,N),functor(Y,F,N),!,
            unify_args(N,X,Y).
unify_args(N,X,Y):-ifthen(N>0,(arg(N,X,Xs),arg(N,Y,Ys),
            unify(Xs,Ys),N1 is N-1,unify_args(N1,X,Y))).
```

```
%--- subst(Old,New,Term,STerm) akk STerm = Term[Old<-New]
.subst(Old,New,Old,New).
.subst(Old,New,Term,Term):-atomic(Term),Term\=Old.
.subst(Old,New,Term,New):-ground(Term),Old=Term.
subst(Old,New,Var,Var):-var(Var).
subst(Old,New,Term,STerm):-functor(Term,F,N),N>0,
                           subst(N,Old,New,Term,STerm).
subst(N,Old,New,Term,STerm):-ifthenelse(N>0,
   (arg(N,Term,X),subst(Old,New,X,SX),      % then
      argrep(Term,N,SX,STerm1),N1 is N-1,
      subst(N1,Old,New,STerm1,STerm)),
   Term=STerm).                              % else
```

---

- $freeze(Term, Frozen)$ vytvorí kópiu termu $Term$ so zamrznutými premennými (unikátne konštanty), zamrznutý term je základný term. Dve zamrznuté premenné sú unifikovateľné, akk sú identické.

```
occurs_in1(X,Term):-subterm(Sub,Term), X == Sub.
occurs_in2(X,Term):-freeze(X,Xf),freeze(Y,Termf),
                    subterm(Xf,Yf).
```

- $melt(Frozen, Thawed)$ zamrznutý term $Frozen$ je späť rozmrazený na (nezákladný) term $Thawed$. $Thawed$ je kópia $Frozen$, pričom zamrznuté premenné sú nahradené späť. Ak sa počas hybernácie na zamrznuté premenné aplikuje nejaká substitúcia, potom sa tiež aplikuje na term $Frozen$ počas rozmrazovania.

```
non_ground_subst(X,Y,Old,New):-freeze(X,Xf),
   freeze(Old,Old1),subst(Xf,Y,Old,Old1),melt(Old1,New).
```

- $melt\_new(Frozen, Thawed)$ – zamrznuté premenné sú nahradené za nové premenné.

```
copy(Term,Copy):-freeze(Term,Frozen),
                 melt_new(Frozen,Copy).
```

# NEÚPLNÉ DÁTOVÉ ŠTRUKTÚRY

NEÚPLNÝ ZOZNAM:

$[1,2,3]$ je rozdiel zoznamov $[1,2,3,8]$ a $[8]$, $[1,2,3]$ a $[]$, $[1,2,3,4,5]$ a $[4,5]$ – vo všeobecnosti $[1,2,3|Xs]$ a $Xs$.

ROZDIEL ZOZNAMOV je term tvaru $As - Bs$.

ZREŤAZENIE dvoch zoznamov reprezentovaných ako rozdiel zoznamov:

```
append_dl(Xs-Ys,Ys-Zs,Xs-Zs).
```

IMPLEMENTÁCIA FRONTU:

```
queue(S):-queue(S,Queue-Queue). % odstartuj s prazdnou fr.
%--- Queue loop
queue([enqueue(X)|Xs],Queue):-  % priadj na koniec
   enqueue(X,Queue,Queue1),queue(Xs,Queue1).
queue([dequeue(X)|Xs],Queue):-  % vyber zo zaciatku
   dequeue(X,Queue,Queue1),queue(Xs,Queue1).
queue([],Queue):-write(Queue).  % koniec direktiv

enqueue(X,As-[X|Bs],As-Bs).     % pridaj na koniec
.dequeue(X,[X|As]-Bs,As-Bs).    % vyber zo zaciatku
.dequeue(X,Queue,As-Bs):-
.   not empty(Queue),Queue=[X|As]-Bs.
.empty([]-[]).                  % test, ci je prazdna
```

QUICK-SORT:

```
qsort([X|Xs],Y-Z):-partition(X,Xs,L,B),
        qsort(L,Y-[X|Q]),qsort(B,Q-Z).
qsort([],L-L).

partition([],_,[],[]).
partition([X|Xs],P,[X|Small],Large):-
        X<P,!,partition(Xs,P,Small,Large).
partition([X|Xs],P,Small,[X|Large]):-
        partition(Xs,P,Small,Large).
```

# PRÍKLADY

REVERSE:
```
reverse(X,Y):-reverse_dl(X,Y-[]).
reverse_dl([X|Xs],Y-Z):-reverse_dl(Xs,Y-[X|Z]).
reverse_dl([],Y-Y).
```

PALINDROMY: $palindrom(X) : -reverse(X, X)$.
```
palindrom(X,Y):-pal_dl(X,Y-[]).
pal_dl([X|Xs],[X|Y]-Z):-pal_dl(Xs,Y-[X|Z]).
pal_dl([],X-X).
```

DUTCH FLAG PROBLEM:
```
%--- 1.verzia s appendom
dutch(X,RWB):-distribute(X,R,W,B),
        append(W,B,WB),append(R,WB,RWB).
distribute([red(X)|Xs],[red(X)|R],W,B):-
        distribute(Xs,R,W,B).
distribute([white(X)|Xs],R,[white(X)|W],B):-
        distribute(Xs,R,W,B).
distribute([blue(X)|Xs],R,W,[blue(X)|B]):-
        distribute(Xs,R,W,B).
distribute([],[],[],[]).

%--- 2.verzia bez appendu
dutch(X,RWB):-dutch_dl(X,RWB-WB,WB-B,B-[]).
dutch_dl([red(X)|Xs],[red(X)|R1]-R,W,B):-
        dutch_dl(Xs,R1-R,W,B).
dutch_dl([white(X)|Xs],R,[white(X)|W1]-W,B):-
        dutch_dl(Xs,R,W1-W,B).
dutch_dl([blue(X)|Xs],R,W,[blue(X)|B1]-B):-
        dutch_dl(Xs,R,W,B1-B).
dutch_dl([],R-R,W-W,B-B).
```

# NEDETERMINISTICKÉ PROGRAMOVANIE

## Základná myšlienka 'generate-and-test' techniky:

```
find(X):-generate(X),test(X).

sort(Xs,Ys):-permutation(Xs,Ys),ordered(Ys).
```

## Problém "SEND MORE MONEY":

```
s(S,E,N,D,M,O,R,N,Y):-
  c(D),
  c(E),D\=E,
  Y is (D+E) mod 10,
  Y\=E,Y\=D,
  Pr1 is (D+E) // 10,
  c(N),N\=D,N\=E,N\=Y,
  c(R),R\=D,R\=E,R\=Y,R\=N,
  E is (N+R+Pr1) mod 10,
  Pr2 is (N+R+Pr1) // 10,
  c(O), O\=D,O\=E,O\=Y,O\=N,O\=R,
  N is (E+O+Pr2) mod 10,
  Pr3 is (E+O+Pr2) // 10,
  c(S), S\=D,S\=E,S\=Y,S\=N,S\=R,S\=O,
  c(M), M\=O,M\=D,M\=E,M\=Y,M\=N,M\=R,M\=O,M\=S,
  O is (S+M+Pr3) mod 10,
  M is (S+M+Pr3) // 10,
  write(' '),write(S),write(E),write(N),write(D), nl,
  write('+'),write(M),write(O),write(R),write(E), nl,
  write(M),write(O),write(N),write(E),write(Y),nl.

c(0). c(1). c(2). c(3). . . . . . c(7). c(8). c(9).
```

# GENERUJ A TESTUJ

## Problém 8 dám:

```
queens:-queens(8,[]).

queens(0,X):- !,write(X),nl,fail.
queens(N,Qs):-q(Q),safe(Q,Q,Q,Qs),N1 is N-1,
              queens(N1,[Q|Qs]).
q(1). q(2). q(3). q(4). q(5). q(6). q(7). q(8).

safe(_,_,_,[]).
safe(A,B,C,[Q|Qs]):-A1 is A+1,C1 is C-1,
              A1\=Q,B\=Q,C1\=Q,safe(A1,B,C1,Qs).
```

## Birthday Boys:

Eddie has bought a video cassette tape (VT), a compact disk (CD), a cassette tape (CT) and a record token (RT) as presents for his brothers' birthdays. Given the following information, what will each brother get?

Alan will not get the CD unless Barry gets the VT and David gets the CT. Barry will not get the CT unless Carl gets the RT and Alan gets the VT. David will not get the RT unless Alan gets the CD and Barry gets the VT. Alan will not get the VT unless Carl gets the RT and David gets the CT. Barry will not get the VT unless Alan gets the RT and David gets the CT. Alan will not get the RT unless Barry gets the CD and Carl gets the VT. Carl will not get the RT unless Barry gets the CD and Alan gets the CT. Alan will not get the CT unless Barry gets the RT and David gets the CD. Carl will not get the VT unless David gets the CD.

## S-P problem:

The problem is as follows: There are two integer numbers $M$ and $N$ such that $1 < M < N, M + N < 100$. Mister $P$ is told their product and mister $S$ is told their sum and both know they were told so. Neither of them knows $M$ or $N$. The following dialogue takes place between them:

P: “I don't know the numbers.”

S: “I knew you wouldn't know them. I don't know them either.”

P: “Now I know them!”

S: “Now I know them, too!”

What are the sum $S$ and the product $P$ and the numbers $M$ and $N$? Try to define predicate $sp(S, P)$ or $mn(M, N)$ which generates all solutions of $S - P$ problem. The upper bound for $M, N$ should be used as a paramerer in your program.

# CONSTRAINT and GENERATE

- $constraint(X, Ciel)$ s premennou $X$ je zviazaný cieľ $Ciel$, ktorý sa vyhodnotí, keď sa inštancuje premenná $X$ (keď $nonvar(X)$). Unifikácia $X$ za term uspeje len, ak sa podarí splniť cieľ $Ciel$.

---

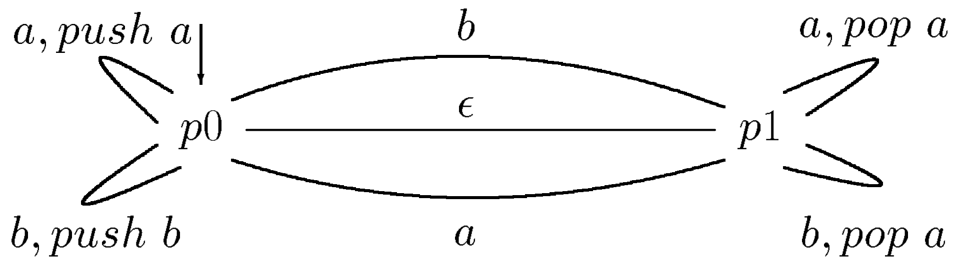
```
algebrogram([S,E,N,D],[M,O,R,E],[M,O,N,E,Y]) :-
        constrain(S,0<S), /* Send nesmie zacat 0 */
        constrain(M,0<M), /* More a Money nesmu zacat 0 */
constrain(E,sum(C1,Y,D,E,0)),
constrain(R,sum(C2,E,N,R,C1)),
constrain(O,sum(C3,N,E,O,C2)),
constrain(M,sum( M,O,S,M,C3)),
        sub_permutacie([0,1,2,3,4,5,6,7,8,9],
                       [Y,D,E,N,R,O,S,M]).
%--- Pomocne definicie
%--- Prenos smerom hore,sucet,X,Y,prenos zdola
sum(0,Sum,X,Y,C):-Sum is  X+Y+C.
sum(1,Sum,X,Y,C) :-Sum1 is X+Y+C, Sum1 is Sum+10.

sub_permutacie(_,[]) :- !.
sub_permutacie(Vzor,[X|Zvysok]) :-
vyber(Vzor,X,Vzor_bez_X),
        sub_permutacie(Vzor_bez_X,Zvysok).

vyber([X|Zvysok],X,Zvysok).
vyber([X|Zvysok],Y,[X|Zvysok_bez_Y]):-
        vyber(Zvysok,Y,Zvysok_bez_Y).

?- algebrogram(X,Y,Z).
X = ....,
Y = ....,
Z = ....
```

# ANALÝZA BEZKONTEXOVÝCH GRAMATÍK




PALINDROMY: $palindrom(X) : -reverse(X, X).$

```
%--- Nondeterministic push-down automat
palindrom(X):-palindrom0(X,[]).
palindrom0([X|Xs],S):-palindrom0(Xs,[X|S]).
palindrom0([X|Xs],S):-palindrom1(Xs,S).
palindrom0(Xs,S):-palindrom1(Xs,S).
palindrom1([X|Xs],[X|S]):-palindrom1(Xs,S).
palindrom1([],[]).
```

GRAMATIKA: $P \rightarrow \epsilon \mid x \mid x\ P\ x$, kde $x \in Terminals.$

```
palindrom(X):-palindrom(X,[]).
palindrom([X|Xs],Y):-palindrom(Xs,[X|Y]).
palindrom([X|Xs],Xs).
palindrom(Xs,Xs).
```

PRÉMIA '.GRM': $P \rightarrow \epsilon \mid x\ P\ x \mid P\ P$, kde $x \in Terminals.$

```
p1(X):-p1(X,[]).
p1([X|Xs],Y):-p1(Xs,[X|Y]).
p1(Xs,Y):-p1(Xs,Z),p1(Z,Y).     % lavo-rekurzivna gramatika
p1(Xs,Xs).
```

$P \rightarrow \epsilon \mid x\ P\ x \mid x\ P\ x\ P$, kde $x \in Terminals.$

```
p2(X):-p2(X,[]).
p2(X,X).
p2([S|X],Y):-Terminals(S),p2(X,[S|Y]).
p2([S|X],Y):-Terminals(S),p2(X,[S|Z]),p2(Z,Y).
```

# TINY PASCAL–SYNTAKTICKÁ ANALÝZA

$$
\begin{array}{ll}
Expr \rightarrow & Term \mid Term\ [+,-]\ Expr \\
Term \rightarrow & Factor \mid Factor\ [*,/]\ Term \\
Factor \rightarrow & Ident \mid Number \mid (\ Expr\ ) \\
Number \rightarrow & integer \\
Ident \rightarrow & atom
\end{array}
$$

```
ident([X|Xs],Xs,ident(X)):-atom(X).
number([X|Xs],Xs,numb(X)):-integer(X).
factor(X1,X,F):-ident(X1,X,F).
factor(X1,X,F):-number(X1,X,F).
factor([ '(' |X1],X,F):-expr(X1,[ ')' |X],F).
term(X1,X,T):-factor(X1,X,T).
term(X1,X,mult(F,T)):-
        factor(X1,['*'|X2],F),term(X2,X,T).
term(X1,X,div(F,T)):-
        factor(X1,['/'|X2],F),term(X2,X,T).
expr(X1,X,E):-term(X1,X,E).
expr(X1,X,add(T,E)):-
        term(X1,['+'|X2],T),expr(X2,X,E).
expr(X1,X,minus(T,E)):-
        term(X1,['-'|X2],T),expr(X2,X,E).
```

$$
\begin{array}{ll}
Statement \rightarrow & Ident\ := Expr \\
 & begin\ Statements\ end \\
Statements \rightarrow & Statement\ Statements \\
 & Statement
\end{array}
$$

```
body(S):-statement(S,[],E),write(E),nl.
statement(X1,X,assign(Var,Expr)):-
        ident(X1,[':='|X2],Var),expr(X2,X,Expr).
statement([begin|X1],X,S):-statements(X1,[end|X],S).
statements(X1,X,[S|Ss]):-
        statement(X1,[';'|X2],S),statements(X2,X,Ss).
statements(X1,X,[S]):-statement(X1,X,S).
```

# DEFINITE CLAUSE GRAMMARS

```
digit(1)-->[one].   digit(2)-->[two].   digit(3)-->[three].
digit(4)-->[four].  digit(5)-->[five].  digit(6)-->[six].
digit(7)-->[seven]. digit(8)-->[eight]. digit(9)-->[nine].

teen(10)-->[ten].                teen(11)-->[eleven].
teen(12)-->[twelve].             teen(13)-->[thirteen].
teen(14)-->[fourteen].           teen(15)-->[fifteen].
teen(16)-->[sixteen].            teen(17)-->[seventeen].
teen(18)-->[eighteen].           teen(19)-->[nineteen].

tens(20)-->[twenty].             tens(30)-->[thirty].
tens(40)-->[forty].              tens(50)-->[fifty].
tens(60)-->[sixty].              tens(70)-->[seventy].
tens(80)-->[eighty].             tens(90)-->[ninety].

%--- najviac dve cifry
xx(N)-->digit(N).
xx(N)-->teen(N).
xx(N)-->tens(T),rest_xx(N1),{N is T+N1}.
rest_xx(0)-->[].
rest_xx(N)-->digit(N).

%--- prave tri cifry
xxx(N)-->digit(D),[hundred],rest_xxx(N1),{N is 100*D+N1}.
rest_xxx(0)-->[].
rest_xxx(N)-->[and],xx(N).

%--- cislo z intervalu <0;999>
number(0)-->[zero].
number(N)-->xx(N).
number(N)-->xxx(N).
```

# EXPAND_TERM

SYSTÉMOVÝ PREDIKÁT ARITY-PROLOGU:
*expand_term*($Term, Newterm$) prekladá term $Term$ z jazyka "definite clause grammars" do jazyka Prolog ($Newterm$).

PRÍKLAD:

```
expand_term((xx(N)-->tens(T),rest_xx(N1),{N is T+N1}),
            (xx(N,X1,X):-tens(T,X1,X2),rest_xx(N1,X2,X),
            N is T+N1))
```

ZJEDNODUŠENÁ VERZIA *expand_term*:

```
expand_term((H-->B),(Head:-Body)):-!,
  expand_term(H,Head,X,Y),expand_term(B,Body,X,Y).
expand_term((A,B),P,S1,S):-!,
  expand_term(A,A1,S1,S2),expand_term(B,B1,S2,S),
  and(A1,B1,P).
expand_term({P},P,S,S):-!.
expand_term(!,!,S,S):-!.
expand_term(A,A1,S1,S):-non_terminal(A),!,
  A=..[F|Args],append(Args,[S1,S],Args1),A1=..[F|Args1].
%-alternativa:  functor(A1,A,2),arg(1,A1,S1),arg(2,A1,S).
expand_term(A,true,S1,S):-terminals(A),!,sequence(A,S1,S).
expand_term((A;B),(A1;B1),S1,S):-!,
  or(A,A1,S1,S2),or(B,B1,S2,S).
%--- pomocne predikaty
and(P,Q,R):-P=true,!,R=Q;Q=true,!,R=P;R=(P,Q).
or(X,S0,S,P):-expand_term(X,Pa,S0a,S),
              (var(S0a),S0a==S,!,S0=S0a,P=Pa;
                                   P=(S0=S0a,Pa)).
non_terminal(A):-atom(A).
terminals([_|_]).
sequence([X|Xs],[X|S1],S):-sequence(Xs,S1,S).
```

```
sequence([],S,S).
```

# PRÁCA S DATABÁZOU (PROGRAMOM)

Systémové predikáty:

- *asserta(Clause)* pridá klauzulu *Clause* na začiatok procedúry pre daný predikát,
- *assertz(Clause)*, *assert(Clause)* pridá klauzulu *Clause* na koniec procedúry pre daný predikát,
- *clause(Head, Body)* unifikuje hlavu *Head* a telo *Body* klauzuly s hlavou a telom klauzuly v databáze (programe). Hlava nesmie byť neinštanciovaná premenná,
- *current_predicate(Predicate)* cez backtracking vracia všetky predikáty v programe v tvare *name* alebo *name/arity*.
- *retract(Clause)* odstráni prvý výskyt špecifikovanej klauzuly z databázy,
- *abolish(Name/Arity)* odstráni všetky klauzuly procedúry so špecifikovaným menom a aritou z databázy,

Príklad: `copy(T1,T2):-assert(foo(T1)),retract(foo(T2)).`

Tabelizácia:

```
lemma(P):-clause(P,'!'), ! ; P,asserta((P :- ! )).

fib(0,0):-!.
fib(1,1):-!.
fib(N,F):-N>1,N1 is N-1,N2 is N-2,lemma(fib(N1,F1)),
          lemma(fib(N2,F2)),F is F1+F2.
```

Pikoška:

```
foo(X):- (X,(retract(db(fooo,_)),!,fail;
             asserta(db(fooo,X)), fail) );
          retract(db(fooo,X)).
```

## Systémové predikáty Arity-Prologu:

- $recorda(Key, Term, Ref)$ pridá term $Term$ podľa kľúča $Key$ na začiatok predikátu a vráti referenciu $Ref$ na tento pridaný term,
- $recordz(Key, Term, Ref)$ pridá term $Term$ podľa kľúča $Key$ na koniec predikátu a vráti referenciu $Ref$ na tento pridaný term,
- $record\_after(Ref, Term, Newref)$ pridá term $Term$ za term referovaný referenciou $Ref$ a vráti novú referenciu $Newref$,
- $recorded(Key, Term, Ref)$ cez backtracking vracia termy a asociované referencie pre daný kľúč,
- $instance(Ref, Term)$ vráti term $Term$ asociovaný s denou referenciou $Ref$,
- $ref(Ref)$ uspeje, ak $Ref$ je referencia,
- $pref(Ref, Prev)$ vráti predchádzajúcu referenciu $Prev$ k referencii $Ref$ (ak existuje),
- $nref(Ref, Next)$ vráti nasledujúcu referenciu $Next$ k referencii $Ref$ (ak existuje),
- $nth\_ref(Key, N, Ref)$ vráti refereciu $Ref$ na $N$-tý term v reťazci podľa kľúča $Key$,
- $key(Key, Ref)$ vráti referenciu na prvý term pre kľúč $Key$,
- $keys(Key)$ vráti meno a aritu všetkých kľúčov v momentálnej databáze,
- $replace(Ref, Term)$ nahradí term referovaný referenciou $Ref$ za term $Term$,
- $erase(Ref)$ odstráni term, na ktorý referuje referencia $Ref$,
- $eraseall(Key)$ odstráni všetky termy podľa kľúča $Key$.
- $expunge$ odstráni informáciu o vymazaných termoch,
- $hard\_erase(Ref)$ nezachová stopu po vymazanom terme $Ref$.

# DRUHO-RÁDOVÉ PROGRAMOVANIE

## Množinové predikáty:

- $bagof(Term, Goal, Bag)$ vráti neusporiadaný zoznam $Bag$ všetkých inštancií termu $Term$, ak cieľ $Goal$ je splnený,
- $setof(Term, Goal, Set)$ vráti zoznam $Set$ všetkých inštancií (bez opakovania) termu $Term$, ak cieľ $Goal$ je splnený,
- $findall(Term, Goal, List)$ vráti neusporiadaný zoznam $List$ všetkých inštancií termu $Term$, ak cieľ $Goal$ je splnený. O všetkých voľných premenných v cieli $Goal$ sa predpokladá, že sú existenčne kvantifikované.

```
between(I,J,I):- I=<J.
between(I,J,X):-I < J,I1 is I+1,between(I1,J,X).

gen(N,List):-findall(X,between(1,N,X),List).
gen1(N,List):-between(1,N,M),gen(M,List).
gengen1(N,ListList):-findall(List,gen1(N,List),ListList).
%--- verzia 1
findall(Term,Goal,_):-Goal,assertz(bof(Term)),fail.
findall(_,_,List):-loop(List),!.
loop([X|List]):-retract(bof(X)),!,loop(List).
loop([]).
%--- verzia 2
findall(Term,Goal,List):-asserta(bof(end)),Goal,
                         asserta(bof(Term)),fail.
findall(Term,Goal,List):-retract(bof(X)),loop(X,[],List),!.
loop(end,L,L):-!.
loop(X,L1,L):-retract(bof(Y)),!,loop(Y,[X|L1],L).
%--- setof
setof(Term,Goal,List):-asserta(bof(end)),Goal,
                       aserta(bof(Term)),fail.
aserta(X):- X,! ; asserta(X).
setof(Term,Goal,List):-retract(bof(X)),loop(X,[],List).
```

# DRUHO-RÁDOVÉ PROGRAMOVANIE

```
%--- forall
for_all(Goal,Cond):-setof(Cond,Goal,Inst),check(Inst).
check([X|Xs]):-X,check(Xs).
check([]).
```

PRÍKLAD: SPLNITEĽNOSŤ FORMULY PP1 V KONEČNOM DOMAINE:

```
%--- domain
domain(0). domain(1). domain(2). domain(3). domain(4).
%--- operacie
plus(X,Y,Z):-Z is (X+Y) mod 5.
krat(X,Y,Z):-Z is (X*Y) mod 5.
%--- interpreter formul PP1
solve(and(A,B)):-!,solve(A),solve(B).
solve(not(A)):-!,not solve(A).
solve(imply(A,B)):-!,solve(or(not(A),B)).
solve(or(A,B)):-!,solve(A);solve(B).
solve(exists(X,A)):-!,domain(X),solve(A).
solve(all(X,A)):-!,for_all(domain(X),solve(A)).
solve(X):-domain(X).
solve(X):- X.
```

---

# PREHĽADÁVANIE GRAFOV

POPIS GRAFU:

```
h(a,b). h(a,c). h(a,d). h(b,c). h(c,d). . . . . .
graph(G):-setof((Node,Successors),
                setof(S,h(Node,S),Successors),G).
```

DEPTH-FIRST STRATEGY:

```
%--- verzia 1-depth-first na DAGu
connected(X,X).
connected(X,Y):-h(X,Z),connected(Z,Y).
```

# PREHĽADÁVANIE GRAFOV

```
%--- depth-first na na vseobecnom grafe
connected(X,Y):-connected(X,Y,[X]).
connected(X,X,_).
connected(X,Y,Visited):-h(X,Z),not member(Z,Visited),
                        connected(Z,Y,[Z|Visited]).
```

Breadth-first strategy:

```
%--- verzia 1-breadth-first na DAGu
connected(X,Y):-connected_bfs([X|Xs]-Xs,Y).
connected_bfs([]-[],Y):-!,fail.
connected_bfs([X|Xs]-Ys,X).
connected_bfs([X|Xs]-Ys,Y):-findall_dl(N,h(X,N),Ys-Zs),
                            connected_bfs(Xs-Zs,Y).
findall_dl(Term,Goal,_):-asserta(bof(end)),Goal,
                         asserta(bof(Term)),fail.
findall_dl(_,_,Xs-Ys):-retract(bof(X)),loop(X,Xs-Ys),!.
loop(end,L-L):-!.
loop(X,Xs-Ys):-retract(bof(Y)),!,loop(Y,Xs-[X|Ys]).

%--- verzia 2-breadth-first na vseobecnom grafe
connected(X,Y):-connected([X|Xs]-Xs,Y,[X]).
connected([]-[],Y,Visited):-!,fail.
connected([A|Xs]-Ys,A,Visited).
connected([A|Xs]-Ys,B,Visited):-
        setof(N,h(A,N),Ns),
        filter(Ns,Visited,Visited1,Xs-Ys,Xs1),
        connected(Xs1,B,Visited1).
filter([N|Ns],Visited,Visited1,Xs-Ys,Xs1-Ys1):-
        member(N,Visited),!,
        filter(Ns,Visited,Visited1,Xs-Ys,Xs1-Ys1).
filter([N|Ns],Visited,Visited1,Xs-[N|Ys],Xs1-Ys1):-
        filter(Ns,[N|Visited],Visited1,Xs-Ys,Xs1-Ys1).
filter([],V,V,Xs-Ys,Xs-Ys).
```

# META-PROGRAMOVANIE

META-INTERPRETER – VERZIA 1: seba-referencia

```
solve(A):-A.        resp.   solve(A):-call(A).
```

META-INTERPRETER – VERZIA 2: "čistý Prolog"

```
solve(true):-!.
solve((A,B)):-!,solve(A),solve(B).
solve(A):-clause(A,B),solve(B).
```

META-INTERPRETER – VERZIA 2A: vytváranie stromu dôkazu

```
solve(true,true):-!.
solve((A,B),(Proofa,Proofb)):-!,
        solve(A,Proofa),solve(B,Proofb).
solve(A,(A:-Proof)):-clause(A,B),solve(B,Proof).

?-solve(rev([1,2],R),Proof_tree).

R=[2,1]
Proof_tree=rev([1,2],[2,1]):-
             (rev([2],[2]) :-
               (rev([],[]):-true),
               (app([],[2],[2]):-true)),
             (app([2],[1],[2,1]):-
               (app([],[1],[1]):-true))
```

META-INTERPRETER – VERZIA 2B: systémové predikáty

```
solve(true):-!.
solve((A,B)):-!,solve(A),solve(B).
solve(A):-clause(A,B),solve(B).
solve(A):-system(A),!,A.

system(integer(_)).      system(atom(_)).
system(atomic(_)).       system(clause(_,_)).
system(_ =.. _).         system(_ = _).
system(_ < _).           system(_ > _).
```

# TRASOVANIE

META-INTERPRETER – VERZIA 2C: improved version

```
solve(true).
solve((A,B)):-solve(A),solve(B).
solve((A;B)):-solve(A);solve(B).
solve(not A):-not solve(A).
solve(setof(A,B,C)):-setof(A,solve(B),C).
solve(A):-system(A),!,A.
solve(A):-reduce(A).
reduce(A):-clause(A,B),solve(B).
```

META-TRACER – VERZIA 3: tracer

```
trace(G):-trace(G,0).
trace(true,_):-!.
trace((A,B),D):-!,trace(A,D),trace(B,D).
trace((A;B),D):-!,(trace(A,D);trace(B,D)).
trace(not A,D):-!,not trace(A,D).
trace(A,D):-system(A),!,show(A,D),A.
trace(A,D):-clause(A,B),D1 is D+1,show(A,D),
        trace(B,D1).
show(A,D):-nl,tab(D),tab(D),write(A).
```

```
?-trace(rev([1,2,3],X)).       |  app([],X,X).
rev([1,2,3],_0038)             |  app([X|Xs],Y,[X|Zs]):-
  rev([2,3],_017C)             |    app(Xs,Y,Zs).
    rev([3],_02D4)             |  rev([],[]).
      rev([],[])               |  rev([X|Xs],Y):-
      app([],[3],[3])          |    rev(Xs,Ys),app(Ys,[X],Y)
    app([3],[2],[3|_0734])     |
      app([],[2],[2])          |
  app([3,2],[1],[3|_091C])     |
    app([2],[1],[2|_0A24])     |
      app([],[1],[1])          |
```

# TRASOVANIE

META-TRACER – VERZIA 3A: tracer

```
trace(G):-trace(G,0).
trace(true,_):-!.
trace((A,B),D):-!,trace(A,D),trace(B,D).
trace((A;B),D):-!,(trace(A,D);trace(B,D)).
trace(not A,D):-!,not trace(A,D).
trace(A,D):-system(A),!,in1(A,D),A,out1(A,D).
trace(A,D):-clause(A,B),D1 is D+1,in1(A,D),
            trace(B,D1),out1(A,D).
show(A,B,D):-nl,tab(D),tab(D),write(A),write(B).
in1(A,D):- show('call ',A,D); show('fail ',A,D),fail.
out1(A,D):-show('exit ',A,D); show('redo ',A,D),fail.
```

```
?-trace(rev([1,2],X)).        | ?-trace(rev([1,2],[2,3])).
call rev([1,2],_038)          | call rev([1,2],[2,3])
  call rev([2],_18C)          |   call rev([2],_0188)
    call rev([],[])           |     call rev([],[])
    exit rev([],[])           |     exit rev([],[])
    call app([],[2],[2])      |     call app([],[2],[2])
    exit app([],[2],[2])      |     exit app([],[2],[2])
  exit rev([2],[2])           |   exit rev([2],[2])
  call app([2],[1],[2|_9AC])|   call app([2],[1],[2,3])
    call app([],[1],[1])      |   fail app([2],[1],[2,3])
    exit app([],[1],[1])      |   redo rev([2],[2])
  exit app([2],[1],[2,1])     |     redo app([],[2],[2])
exit rev([1,2],[2,1])         |     fail app([],[2],[2])
                              |     redo rev([],[])
                              |     fail rev([],[])
                              |   fail rev([2],_0188)
                              | fail rev([1,2],[2,3])
```

## Meta-tracer – verzia 4: depth-breadth first strategy in Prolog

```
go(A):-solve([(0,A)|E]-E).
solve([]-[]).
solve([(D,A)|H]-T):- clause(A,B),D1 is D+1,
        show((A:-B),D),
        schedule(D1,B,H-T,H1-T1),solve(H1-T1).
solve([(D,A)|H]-T):- system(A),!,show(A,D),A,
        solve(H-T).
show(A,D):-nl,tab(D),tab(D),tab(2),write(A).
schedule(_,true,H-T,H-T):-!.
schedule(D,(A,B),H1-T1,H-T):-depth_first,!,
        schedule(D,B,H1-T1,H2-T2),
        schedule(D,A,H2-T2,H-T).
schedule(D,(A,B),H1-T1,H-T):-breadth_first,!,
        schedule(D,A,H1-T1,H2-T2),schedule(D,B,H2-T2,H-T).
schedule(D,A,H-T,[(D,A)|H]-T):-depth_first,!.
schedule(D,A,H-[(D,A)|T],H-T):-breadth_first,!.
```

```
?-go(rev(X,[a])).  // depth_first   |?-go(rev(X,[a])).  // breadth_first
rev([_1C8|_1CC],[a]):-              |rev([_1C8|_1CC],[a]):-
  . rev(_1CC,_1E4),                 |  . rev(_1CC,_1E4),
  . app(_1E4,[_1C8],[a])            |  . app(_1E4,[_1C8],[a])
  rev([],[]):-true                  |  rev([],[]):-true
  app([],[a],[a]):-true             |  app([],[a],[a]):-true
X=[a]->;                            |X=[a]->;
  rev([_434|_438],_1E4):-           |  rev([_434|_438],_1E4):-
    . rev(_438,_450),               |    . rev(_438,_450),
    . app(_450,[_434],_1E4)         |    . app(_450,[_434],_1E4)
    rev([],[]):-true                |  app([],[a],[a]):-true
    app([],[_434],[_434]):-true     |    rev([],[]):-true
  app([a],[_1C8],[a]):-             |    rev([_808|_80C],_450):-
    . app([],[_1C8],[])             |      . rev(_80C,_824),
    rev([_6A0|_6A4],_450):-         |      . app(_824,[_808],_450)
      . rev(_6A4,_6BC),             |  app([a|_6A8],[_1C8],[a]):-
      . app(_6BC,[_6A0],_450)       |    . app(_6A8,[_1C8],[])
      rev([],[]):-true              |    rev([],[]):-true
      app([],[_6A0],[_6A0]):-true   |    app([],[a],[a]):-true
    app([_6A0],[_434],[_6A0|_BE8]):-|    rev([_850|_854],_450):-
      . app([],[_434],_BE8)         |      . rev(_854,_86C),
      app([],[_434],[_434]):-true   |      . app(_86C,[_850],_450)
                                    |    app([],[a],[a]):-true
                                         app([a|_AC4],[_434],[a|_6A8]):-
   zacykli sa                                      . app(_AC4,[_434],_6A8)
```

# TRASOVANIE

Meta-tracer – verzia 5: auto debugging

```
solve(_,0,[]):-!.
solve(true,_,no).
solve((A,B),D,Over):-solve(A,D,O1),
        ifthenelse(O1=no,solve(B,D,O),O=O1).
solve(A,D,O):-clause(A,B),D1 is D-1,solve(B,D1,O1),
        ifthenelse(O1=no,O=O1,O=[A|O1]).
solve(A,D,no):-system(A),!,A.
```

Meta-tracer – verzia 6: false clauses - bottom-up

```
solve(true,true).
solve((A,B),(Proofa,Proofb)):-
        solve(A,Proofa),solve(B,Proofb).
solve(A,(A:-Proof)):-clause(A,B),solve(B,Proof).
false_sol(A,Cl):-solve(A,Proof),false_cl(Proof,Cl).
false_cl(true,ok):-!.
false_cl((A,B),Cl):-!,false_cl(A,Cla),
        (Cla=ok,!,false_cl(B,Cl); Cl=Cla).
false_cl((A:-B),Cl):-!,false_cl(B,Clb),
       (Clb=ok,!,query(A,Answ),check_answ(Answ,A,B,Cl);
        Cl=Clb).
query(A,yes):-system(A),!.
query(A,Answ):-write('Is '),write(A),
        write(' true ? '),read(Answ).
check_answ(yes,_,_,ok).
check_answ(no,A,B,(A:-B1)):-extract_body(B,B1).
extract_body(true,true).
extract_body((A:-B),A).
extract_body(((A:-B),Bs),(A,As)):-extract_body(Bs,As).
```

# TRASOVANIE

META-TRACER – VERZIA 6: false clauses - top-down

```
false_sol(A,Cl):-solve(A,Proof),false_goal(Proof,Cl).
false_goal((A:-B),Cl):-false_conjunction(B,Cl),!.
false_goal((A:-B),(A:-B1)):-extract_body(B,B1).
false_conjunction(((A:-B),Bs),Cl):-query(A,no),!,
        false_goal((A:-B),Cl).
false_conjunction((A:-B),Cl):-query(A,no),!,
        false_goal((A:-B),Cl).
false_conjunction((_,As),Cl):-false_conjunction(As,Cl).
```

META-TRACER – VERZIA 6: missing solutions

```
% missing(A,G) iff A is false, then G is ...
missing((A,B),G):- !,(not A,missing(A,G);
                      A,missing(B,G)).
missing(A,G):-clause(A,B),query_clause((A:-B)),!,
        missing(B,G).
missing(A,A):-not system(A).
query_clause(Cl):-
        write('Enter true instance of the clause '),
        write(Cl), nl,
        write(' or no., if there is no such clause'),nl,
        read(Answ),!,check_answ(Answ,Cl).
check_answ(no,Cl):-!,fail.
check_answ(Cl,Cl):-!.
check_answ(Answ,Cl):-write('Illegal answer '),nl,!,
        query_clause(Cl).
```

# Prémia: '.BIN'

## BINÁRNA SÚSTAVA

Unárna sústava kódujúca prirodzené čísla do termov tvaru $s(s(...s(0)...))$ je ťažkopádna. Preto zaveďme binárnu sústavu:

```
0  ...  0
1  ...  i(0)
2  ...  o(i(0))
3  ...  i(i(0))
4  ...  o(o(i(0)))
5  ...  i(o(i(0)))
 . . . . . . . . . . .
```

Term $0$ kóduje číslo $0$; ak term $t$ kóduje číslo $kod(t) = C$, potom term $i(t)$ kóduje číslo $kod(i(t)) = 2*C+1$ a term $o(t)$ kóduje číslo $kod(o(t)) = 2*C$. Definujte nasledujúce predikáty:

- $incr(X, Y)$ platí práve vtedy, ak $kod(Y) = kod(X) + 1$,
- $plus(X, Y, Z)$ platí práve vtedy, ak $kod(Z) = kod(X) + kod(Y)$,
- $krat(X, Y, Z)$ platí práve vtedy, ak $kod(Z) = kod(X) * kod(Y)$,
- $max(X, Y, Z)$ platí práve vtedy, ak $kod(Z) = max(kod(X), kod(Y))$

## PRÍKLAD:

- $incr(i(i(0)), o(o(i(0))))$ platí,
- $plus(i(i(0)), o(o(i(0))), i(i(i(0))))$ platí,
- $krat(i(i(0)), o(o(i(0))), o(o(i(i(0)))))$ platí,
- $max(i(i(0)), o(o(i(0))), o(o(i(0))))$ platí

---

POČET BODOV: 2
TERMÍN: 31.3.1993
FILE NAME: <PRIEZVISKO>.BIN

# Prémia: '.GRM'

GRAMATIKA

Uvažujme nasledujúcu bezkontextovú gramatiku
$P \rightarrow \epsilon \mid P\ P \mid a\ P\ a \mid b\ P\ b$. Slová budeme kódovať do zoznamov prirodzeným spôsobom: zoznam $[x_1, x_2, \ldots, x_n]$ reprezentuje slovo $x_1 x_2 \ldots x_n$.

Definujte predikát $patri(X)$, ktorý platí práve vtedy, keď slovo reprezentované zoznamom $X$ patrí do jazyka generovaného touto gramatikou.

PRÍKLAD:

- $patri([a, b, b, a])$ platí,
- $patri([a, b, b, a, b, a, a, a, a, b])$ platí,
- $patri([a, b, a])$ neplatí

---

POČET BODOV: 2
TERMÍN: 31.3.1993
FILE NAME: <PRIEZVISKO>.GRM

# Prémia: '.AIR'

Letecká spoločnosť $\Sigma$-air plánuje realizovať malý informačný systém poskytujúci najzákladnejšie informácie často žiadané jej pasažiermi. Jej databáza obsahuje údaje o letoch kódované v tvare termov
$flight(Flight\_number, From, To, Departure, Arrival, Price)$.
$Flight\_number$ je jednoznačné číslo letu, $From, To$ sú mená miest a $Departure, Arrival$ sú časy odletu a príletu lietadla. $Price$ je cena letu.

- Definujte predikát $route(From, To, Departure, Arrival, Path)$, ktorý pre zadané mestá $From$, $To$ nájde letecké spojenie z mesta $From$ do mesta $To$, ktoré z mesta $From$ neodlieta skôr ako $Departure$ a do mesta $To$ nepriletí neskôr ako $Arrival$. $Path$ je zoznam $Flight\_numbers$ týchto letov v poradí od mesta $From$ do $To$.
- Definujte predikát $cheapest(From, To, Path, Total)$, ktorý pre zadané mestá $From$, $To$ nájde najlacnejšie spojenie $Path$ v rámci jedného dňa letmi spoločnosti $\Sigma$-air. $Total$ je celková cena letu.

Príklad:

```
flight(79,bratislava,malacky,8.10,9.15,400).
flight(80,bratislava,malacky,12.15,13.35,450).
flight(81,malacky,bratislava,19.15,20.35,350).
flight(82,bratislava,kosice,21.06,23.56,700).
```

- jedno z riešení $route(bratislava, kosice, 7.00, 24.00, Path)$ je aj $Path = [79, 81, 82]$,
- riešenie pre $cheapest(bratislava, kosice, Path, Price)$ je $Path = [82]$, $Price = 700$.

---

Počet bodov: 3 (za obe časti), 1 (za prvú časť)
Termín: do Veľkej noci
File name: <priezvisko>.AIR

# Prémia: '.PUZ'

PUZZLE

Algebrogram je matematický hlavolam, v ktorom sú nad sebou napísané sčítance dávajúce spolu súčet. Niektoré (alebo všetky) cifry v súčte sú nahradené písmenami (premennými). Tie treba uhádnuť a dosadiť tak, aby výsledný súčet bol korektný. Príklad (legendárny):

```
     SEND           solution:    9567
   + MORE                       +1085
    =====                       =====
    MONEY                       10652
```

Pre účely tohoto príkladu budeme algebrogramy kódovať do zoznamu tvaru $[[a_1, a_2, \ldots, a_n], [b_1, b_2, \ldots, b_m], \ldots, [u_1, u_2, \ldots, u_p], [z_1, z_2, \ldots, z_r]]$, kde $a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m, \ldots, u_1, u_2, \ldots, u_p, z_1, z_2, \ldots, z_r$ sú cifry alebo premenné. Za rôzne premenné treba dosadiť rôzne cifry tak, aby platilo: $(a_1, a_2, \ldots, a_n) + (b_1, b_2, \ldots, b_m) + \ldots + (u_1, u_2, \ldots, u_p) = (z_1, z_2, \ldots, z_r)$, pričom $(c_1, c_2, \ldots, c_q)$ je hodnota čísla s ciframi $c_1, c_2, \ldots, c_q$ v desiatkovej sústave. Horeuvedený algebrogram sa zakóduje: $[[S, E, N, D], [M, O, R, E]], [M, O, N, E, Y]]$

Definujte predikát $puzzle(A)$, ktorý nájde **všetky** riešenia zakódovaného algebrogramu $A$.

Príklad:

$? - puzzle([[S, E, N, D], [M, O, R, E]], [M, O, N, E, Y]]).$
$S = 9, E = 5, N = 6, D = 7,$
$M = 1, O = 0, R = 8, Y = 2- >;$
$no$

---

Počet bodov: 3
Termín: 30.4.1993
File name: <priezvisko>.PUZ

# Prémia: '.EQU'

## EKVIVALENTNE VÝRAZY

Aritmetické výrazy sú termy generované nasledujúcim programom:

```
expr(X):-atomic(X),X\=[].
expr((A+B)):-expr(A),expr(B).
expr((A-B)):-expr(A),expr(B).
expr((A*B)):-expr(A),expr(B).
```

- a) Definujte predikát $equiv1(X, Y)$, ktorý je pravdivý práve vtedy, ak aritmetické výrazy $X$ a $Y$ sú ekvivalentné v tradičnej aritmetickej interpretácii.
- b) Uvažujte ďalšiu klauzulu:

  ```
  expr((A/B)):-expr(A),expr(B).
  ```

  Definujte predikát $equiv2(X, Y)$, ktorý je pravdivý práve vtedy, ak aritmetické výrazy $X$ a $Y$ sú ekvivalentné v prieniku ich definičných oborov.

PRÍKLAD:
$equiv1((a + b) * (a + b), (a * a + b * b + 2 * a * b))$
$equiv1((a + b) * (a - b), (a * a + b * b - a * b - a * b + 0))$
$equiv2((1/a - 1/b), (b - a)/a * b)$

---

POČET BODOV: 3 (ČASŤ A) + 3 (ČASŤ B)
TERMÍN: KONIEC SEMESTRA
FILE NAME: <PRIEZVISKO>.EQU

# Prémia: '.DIR'

DIRECTORY

Definované sú dva predikáty: $find\_first(Path, File, Attr)$ s módom $(+,-,-)$ a $find\_next(File, Attr)$ s módom $(-,-)$, ktoré pracujú analogicky ako dve služby MS DOSu. $find\_first$ pre zadanú cestu $Path$ vráti prvú položku v tomto adresári. $find\_next$ vráti nasledujúcu položku v adresári, ktorý bol predtým nastavený pomocou $find\_first$.
Pod položkou rozumieme súbor (potom $File$ je meno súboru a $Attr = f$), adresár (potom $File$ je meno podadresára a $Attr = d$) alebo informáciu o tom, že taká položka neexistuje (potom $File ='' $ a $Attr = no\_more$).
Definujte predikát $dir(Path, Directory)$ pracujúci v móde $(+,-)$, ktorý pre zadanú cestu vráti adresárový les v premennej $Directory$.
Adresárový les je
zoznam adresárov. Adresár je dvojica: $(Polozka, Adresarovy\ podles)$, pričom $Polozka$ je dvojica $(File, Attr)$.

---

Počet bodov: 2
Termín: 30.4.1993
File name: <priezvisko>.DIR

# Prémia: '.CAN'

## CANNIBALS and MISSIONARIES

$M$ missionaries and $C$ cannibals seek to cross a river $(M \geq C > 0)$. A boat is available which holds two people, and which can be navigated by any combination of missionaries and cannibals involving one or two people. If the missionaries on either bank of the river, or 'en route' in the river, are outnumbered at any time by the cannibals, the cannibals will indulge their anthropophagic tendencies and do away with the missionaries. Define a predicate $schedule(M, C)$ to find the (simplest) schedule of crossing that will permit all the missionaries and cannibals to cross the river safely.

PRÍKLAD:

pre $m = c = 3$:

| ľavý breh | lodička | pravý breh |
|---|---|---|
| (m m m k k k) | | () |
| (m m k k) | (m k)$->$ | (m k) |
| (m m m k k) | $<-$(m) | (k) |
| (m m m) | (k k)$->$ | (k k k) |
| (m m m k) | $<-$(k) | (k k) |
| (m k) | (m m)$->$ | (m m k k) |
| (m m k k) | $<-$(m k) | (m k) |
| (k k) | (m m)$->$ | (m m m k) |
| (k k k) | $<-$(k) | (m m m) |
| (k) | (k k)$->$ | (m m m k k) |
| (k k) | $<-$(k) | (m m m k) |
| () | (k k)$->$ | (m m m k k k) |

---

POČET BODOV: 3

TERMÍN: KONIEC SEMESTRA

FILE NAME: <PRIEZVISKO>.CAN

# Semestrálna písomka 1

31.3.1993-LOPRO1

## Zoznamy (4 body)

*Nepresný číslený zoznam* je ľubovoľný hladký zoznam čísel (integer) a atómov **hocico** (napríklad $[1, 2, hocico, 3, hocico]$).
Dva nepresné číselné zoznamy sú *porovnateľné*, ak po nahradení všetkých výskytov atómu **hocico** za rôzne premenné (napr. anonymné premenné) v oboch zoznamoch, dostaneme unifikovateľné zoznamy.
Hovoríme, že nepresný číselný zoznam $x$ sa nachádza v inom nepresnom číselnom zozname $y$, ak existuje súvislý úsek $z$ zoznamu $y$ (t.j. existujú $u, z, v$, že $y = uzv$), ktorý je porovnateľný s $x$.
Definujte predikát $nachadza\_sa(X, Y)$, ktorý platí práve vtedy, ak sa nepresný číselný zoznam $X$ nachádza v nepresnom číselnom zozname $Y$.
Príklad:
$nachadza\_sa([2, 3], [1, 2, 3, 4, 5])$,
$nachadza\_sa([2, hocico, 4], [1, 2, 3, 4, 5])$,
$nachadza\_sa([2, hocico, 4], [1, 2, 3, hocico, 5])$,
$nachadza\_sa([2, hocico, 4], [1, 2, hocico, hocico, 5])$.

## Adresár (4 body)

*Adresár* je zoznam položiek adresára. Položka adresára je buď term **file(Menosúboru)**, alebo term **dir(Menopodadresára,Podadresár)**.
Príklad:

```
[file(config),file(autoexec),file(command),
   dir(dos,[file(arj),file(backup),file(restore)]),
   dir(api,[file(api),file(apc),
            dir(example,[file(prvy),file(druhy),file(treti)])])]
```

*Cesta* k súboru $file$ je zoznam $[adr_1, adr_2, \ldots, adr_n, file]$, kde $adr_1, adr_2, \ldots, adr_n$ sú mená podadresárov vedúcich k súboru $file$. Definujte predikát $existuje(Cesta, Adresar)$, ktorý je pravdivý práve vtedy, ak v adresári $Adresar$ cesta $Cesta$ smeruje k nejakému súboru.
Príklad: Nech $A$ je horeuvedený adresár,
$existuje([config], A)$ platí,
$existuje([dos, arj], A)$ platí,
$existuje([api, example, druhy], A)$ platí,
$existuje([api, treti], A)$ neplatí.

## $T_P \uparrow \alpha, T_P \downarrow \alpha$ (4 body)

Vypočítajte $T_P \uparrow \alpha, T_P \downarrow \alpha$, pre $\alpha = n, \omega, \omega + n, \omega + \omega$, kde $n \in \omega$ a program $P$ má tvar:

```
p(f(X)) <- p(X)                     q(f(X)) <- q(X)
p(a) <- p(X),q(X)                   q(b) <-
```

## Modely (3 body)

Nájdite interpretáciu formuly

$$
\begin{array}{ll}
( & (\forall X)\, p(X, X) \wedge \\
 & (\forall X, Y, Z)\, [(p(X, Y) \ \wedge\ p(Y, Z)) \ \rightarrow\ p(X, Z)] \wedge \\
 & (\forall X, Y)\, [p(X, Y) \ \vee\ p(Y, X)] \\
) & \rightarrow\ (\exists Y)\ (\forall X)\, p(Y, X)
\end{array}
$$

ktorá nie je modelom. Dokážte, že neexistuje interpretácia tejto formuly s konečným domainom $D$, ktorá by nebola jej modelom.

# Semestrálna písomka 2

5.5.1993-LOPRO2

## Uhádni a dokáž (5 bodov)

Dokážte, resp. vyvráťte nasledujúce tvrdenie:
Nech $A$ je atóm a $P$ je logický program. $\forall(A)$ je logickým dôsledkom $P$ práve vtedy, ak existuje $n$, že $[A] \subseteq T_P \uparrow n$.

## Variant (5 bodov)

Definujte predikát $variant(E, F)$, ktorý platí práve vtedy, ak $E$, $F$ sú varianty. Pozor: termy $E$, $F$ môžu obsahovať premenné.

## Magické číslo (5 bodov)

Číslo $X$ je *magické*, ak prvých $i$ cifier v desiatkovom zápise tohoto čísla tvorí (v desiatkovom zápise) číslo deliteľné $i$, pre každé $1 \leq i \leq$ počet cifier magického čísla. Príklad: 381 je magické, lebo
3 je deliteľné 1,
38 je deliteľné 2,
381 je deliteľné 3.
Definujte predikát $magic(X)$, ktorý nájde všetky magické čísla obsahujúce všetky cifry $1 \ldots 9$ a každú práve raz. Pri hodnotení záleží najmä na efektívnosti hľadania týchto čísel.

## Zrkadlo iteratívne (5 bodov)

Definujte binárny predikát $zrkadlo(X, Y)$, ktorý daný zoznam $X$ 'otočí' na všetkých úrovniach, t.j. každý zoznam nachádzajúci sa v $X$ musí byť v $Y$ v opačnom poradí. Príklad namiesto presnej definície:

$zrkadlo([1, 2, 3, [4, 5, [6], 7], 8, 9], [9, 8, [7, [6], 5, 4], 3, 2, 1])$.

Všetky klauzuly v tomto riešení musia byť iteratívne.

# Riešenia semestrálnej písomky 2

5.5.1993-LOPRO2 – riešenia

```
%--- Riesenie VARIANT

variant(E,F):-v(E,F,[],Z).

v(E,F,Z1,[(E,F)|Z1]):-var(E),var(F),!,not_in(E,F,Z1).
v(E,F,Z1,Z):-nonvar(E),nonvar(F),!,
     E=..[Fu|Eargs],F=..[Fu|Fargs],vs(Eargs,Fargs,Z1,Z).
v(E,F,Z,Z):-atomic(E),atomic(F),!,E=F.

vs([],[],Z,Z).
vs([Ea|Eargs],[Fa|Fargs],Z1,Z):-v(Ea,Fa,Z1,Z2),vs(Eargs,Fargs,Z2,Z).

not_in(E,F,[]).
not_in(E,F,[(X,Y)|Zs]):-(E==X,F==Y;E\==X,F\==Y),not_in(E,F,Zs).

%--- Riesenie ZRKADLO/1
zrkadlo(X,Y):-z(X,[],Y,[]).
z([],Y,Y,[]).
z([X|Xs],Y1,Y,S) :- atomic(X),!, z(Xs,[X|Y1],Y,S).
z([X|Xs],Y1,Y,S) :- z(Xs,[V|Y1],Y,[(X,V)|S]).
z([],Y,Y,[(X,V)|S]) :- z(X,[],V,S).

%--- Ine riesenie ZRKADLO/2

zrkadlo(Z1,Z2):-r(Z1,[],[[]],Z2).
% r(zoz-in,este-urobit,konstr-vysl,vysledok).
r([],[],[V],V).
r([],[V|S1],[R,Vr|S2],Z):-r(V,S1,[[R|Vr]|S2],Z).
r([A|B],S1,[R|S2],Z):-atomic(A),!,r(B,S1,[[A|R]|S2],Z).
r([L|B],S1,S2,Z):-r(L,[B|S1],[[]|S2],Z).

%--- Riesenie MAGIC

magic(Y):-d(0,[1,2,3,4,5,6,7,8,9],Y,0).
d(I,Xs,[X|Y],C):-extract(X,Xs,Xs1),C1 is 10*C+X,I1 is I+1,
                  0 is C1 mod I1,d(I1,Xs1,Y,C1).
d(_,[],[],_).

extract(X,[X|Xs],Xs).
extract(X,[Y|Xs],[Y|Xs1]):-extract(X,Xs,Xs1).

%--- Riesenie UHÁDNI A DOKÁŽ

Neplati, napr. pre program P={p(a)<-} a ciel G=<-p(X).
```

# Semestrálna písomka 3.

19.5.1993-LOPRO3

## Super zoznam (5 bodov)

Definujte predikát $super\_zoznam(Y)$, ktorý je pravdivý, ak $Y$ je zoznam všetkých podmnožín nejakej množiny, pre ktorý navyše platí, že každé dva po sebe idúce prvky tohoto zoznamu sa líšia v práve jednom prvku (resp. symetrický rozdiel po sebe idúcich podmnožín je jednoprvkový). Nestačí vedieť vygenerovať prvé riešenie, treba vedieť otestovať každé správne.

## $\infty$ (3 body)

Definujte predikát, pre ktorý existuje nekonečne veľa SLD-zamietnutí.

## Puzzle (4 body)

Nájdite postupnosť desiatkových cifier $d_1, d_2 \ldots d_n$ takých, že platí:

$$< d_1 d_2 d_3 \ldots d_n 5 > *5 =< 5 d_1 d_2 d_3 \ldots d_n >$$

Poznámka: $< abc >$ je desiatkový zápis čísla $100 * a + 10 * b + c$.

## Lesy (3 body)

Les $Les$ obsahuje vo vrcholoch prologovské atómy, ktoré vypísané v preorderovom poradí tvoria postupnosť $a_1, a_2, \ldots, a_n$. Definujte predikát $find(Index, Les, Prvok)$, ktorý vráti $Prvok = a_{Index}$ pre daný les $Les$, ak sa v ňom nachádza. Ak nie, zlyhá.

# Skúška 1.

14.5.1993-SKÚŠKA1

## Naprogramujte $T_P$ operátor (5 bodov)

Definuje predikát $tp(P, I, J)$, ktorý pre zadaný program $P$ a Herbrandovskú interpretáciu $I$ (podmnožinu $B_P$), vypočíta hodnotu $J = T_P(I)$. Program $P$ je zoznam klauzúl v tvare $if(A, [B_1, \ldots, B_n])$, pričom $Var(A) \subseteq Var(B_1, \ldots, B_n)$, t.j. všetky premenné nachádzajúce sa na ľavej strane sa nachádzajú aj na pravej strane implikácie.
Príklad: $tp([if(p(a), []), if(p(f(X)), [p(X)])], [p(a), p(f(a))], J)$, kde $J = [p(a), p(f(a)), p(f(f(a)))]$.

## Počet výskytov (5 bodov)

Definujte predikát $pocet\_vyskytov(Sub, Term, N)$, ktorý platí práve vtedy, ak sa term $Sub$ $N$-krát nachádza v terme $Term$. Termy $Sub$, $Term$ nemusia byť základné, t.j. môžu obsahovať premenné.

## $gfp(T_P) = T_P \downarrow \omega$ (5 bodov)

Ak sa v každej klauzule logického programu $P$ všetky premenné nachádzajúce sa na pravej strane (tele) nachádzajú aj na ľavej strane (hlave) implikácie, potom $gfp(T_P) = T_P \downarrow \omega$.

## Lesy (5 bodov)

Les $Les$ obsahuje celočíselné hodnoty, ktoré vypísané v preorderovom poradí tvoria postupnosť $a_1, a_2, \ldots, a_n$. Definujte predikát $bratia(Index1, Index2, Les)$, ktorý platí, ak $Index1$ a $Index2$ sú bratia v lese $Les$. Dva vrcholy sú bratia, ak majú rovnakého predka.

---

---

### Riešenie 1. príkladu

```
tp(P,I,J):-findall(A,tpa(P,I,A),J).

tpa(P,I,A):-member(C,P),tca(C,I,A).

tca(if(H,B),I,H):-unif_list(B,I).

unif_list([],I).
unif_list([A|B],I):-member(A,I).

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

# Skúška 2.

19.5.1993-SKÚŠKA2

## Puzzle (4 body)

Nájdite postupnosť desiatkových cifier $d_1, d_2 \ldots d_n$ takých, že platí: $< d_1 d_2 d_3 \ldots d_n 5 > *5 =< 5 d_1 d_2 d_3 \ldots d_n >$. Poznámka: $< abc >$ je desiatkový zápis čísla $100 * a + 10 * b + c$.

## Dokážte (5 bodov)

Nech $X$ je podmnožina úplného zväzu herbrandovských interpretácií programu $P$, pre ktorú platí, že každá konečná podmnožina $X$ má horné ohraničenie v $X$.
Dokážte, že $\{A_1, \ldots, A_n\} \subseteq sup(X)$ práve vtedy, ak pre nejaké $I \epsilon X$ platí $\{A_1, \ldots, A_n\} \subseteq I$

## Patterny (6 bodov)

Pattern je term tvaru:

```
Pattern ::= str(retazec alebo premenna) | any(retazec) |
            and(zoznam patternov) | or(zoznam patternov)
```

$y[I \ldots O]$ je podreťazec reťazca $y$ s indexami $I \ldots O-1$, $I < O$. $y[I]$ je znak s indexom $I$.
Interpretácia jednotlivých variantov je nasledujúca:

- $str(x)$, ak $x$ je reťazec, je porovnateľné s podreťazcom $y[I \ldots O]$, ak $y[I \ldots O] = x$
  match('abcdef',str('cde'),I,O)  I,O = 2,5
  match('ababc',str('ab'),I,O)  I,O = 0,2  I,O = 2,4
  match('abcef',str('cde'),I,O)  zlyhá
  $str(x)$, ak $x$ je premenná, je porovnateľné s podreťazcom $y[I \ldots O]$, a premenná $x$ nadobudne hodnotu podreťazca $y[I \ldots O]$
  match('ab',str(V),I,O)  V,I,O = 'a',0,1  V,I,O = 'b',1,2  V,I,O = 'ab',0,2
- $any(x)$ je porovnateľné so znakom $y[I]$, ak sa vyskytuje v reťazci $x$ a $O = I + 1$
  match('abcba',any('bcd'),I,O)  I,O = 1,2  I,O = 2,3
- $and([p_1, p_2, \ldots, p_n])$ je porovnateľné s $y[I \ldots O]$, ak
  $p_1$ je porovnateľné s $y[I \ldots I_1]$, $p_2$ je porovnateľné s $y[I_1 \ldots I_2]$ ... a t. ď.
  $p_n$ je porovnateľné s $y[I_{n-1} \ldots O]$
- $or([p_1, p_2, \ldots, p_n])$ je porovnateľné s $y[I \ldots O]$, ak platí aspoň jeden z dole uvedených riadkov:
  $p_1$ je porovnateľné s $y[I \ldots O]$, $p_2$ je porovnateľné s $y[I \ldots O]$ ... a t. ď.
  $p_n$ je porovnateľné s $y[I \ldots O]$

Definujte predikát $match(re\check{t}azec, pattern, I, O)$, ktorý vygeneruje všetky možnosti, ako možno pattern $pattern$ porovnať s podreťazcom $re\check{t}azec[I \ldots O]$ a vráti hodnoty $I$ a $O$.

## Lesy (5 bodov)

Les $Les$ obsahuje celočíselné hodnoty, ktoré vypísané v preorderovom poradí tvoria postupnosť $a_1, a_2, \ldots, a_n$. Definujte dvojicu predikátov $Out(Les, List)$ a $In(List, Les)$, ktoré prevádzajú $Les$ do vonkajšej formy $List$ a z vonkajšej formy $List$ do vnútornej formy $Les$ tak, že platí:

pre každý les $Les$ existuje práve jeden list $List$, že platí $Out(Les, List)$,

pre každý list $List$ existuje práve jeden les $Les$, že platí $In(List, Les)$,

$Out(Les, List)$ & $member(X, List) \longrightarrow integer(X)$,

$Out(Les, List) \equiv In(List, Les)$.

# Skúška 3.

26.5.1993-SKÚŠKA3

## Podzoznamy (4 body)

Nech $X = [a_1, a_2, ..., a_n]$ je zoznam rôznych prvkov (povedzme množina). **Vygenerujte** zoznam všetkých podzoznamov zoznamu $X$ (resp. podmnožín množiny $X$) usporiadaný tak, že každé dva po sebe idúce prvky tohoto zoznamu sa odlišujú v práve jednom prvku (resp. symetrický rozdiel po sebe idúcich podmnožín je jednoprvkový).
Definujte predikát $subsets(X, Y)$, kde $X$ je vstupný zoznam $X$ a $Y$ je výstupný zoznam podzoznamov s hore popísanou vlastnosťou.

## Manželia (6 bodov)

Napíšte program na riešenie problému stabilných manželstiev:
Daná je množina mužov a množina žien, ktorí sa chcú zobrať. Pozn.: sobáše sú medzi dvojicou muž a žena.
Každý muž aj žena si stanovili nejaké preferencie (požiadavky) na partnerov vo forme zoznamu priorít partnerov. Problém stabilných manželstiev spočíva v tom, že treba nájsť množinu manželstiev $< muz, zena >$ pričom platí, že neexistujú v nej dvaja nezosobášení ľudia opačného pohlavia, ktorí by sa navzájom preferovali pred svojimi manželskými partnermi.

| | | | | | |
|---|---|---|---|---|---|
| adam | cyntia | tamara | zuza | renata | sylvia |
| benjamín | zuza | cyntia | renata | sylvia | tamara |
| ctibor | cyntia | renata | tamara | sylvia | zuza |
| david | zuza | renata | cyntia | sylvia | tamara |
| emil | tamara | renata | cyntia | zuza | sylvia |

| | | | | | |
|---|---|---|---|---|---|
| zuza | emil | adam | david | benjamín | ctibor |
| cyntia | david | emil | benjamín | adam | ctibor |
| renata | adam | david | benjamín | ctibor | emil |
| sylvia | ctibor | benjamín | david | adam | emil |
| tamara | david | benjamín | ctibor | emil | adam |

Definujte predikát $manzelia(Preferencie, Dvojice)$, ktorý pre zadané preferecie $Preferencie$ (vo vašom formáte) nájde riešenie problému stabilných manželstiev v tvare zoznamu dvojíc $<$ $muz, zena >$.

## Špinavosti (5 bodov)

Nájdite logické vysvetlenie toho, čo "znamená" nasledujúci predikát:

```
goo(X) :-
  X,
  ( retract(db(gooo,_)),! ;
    asserta(db(gooo,1)), fail ) ;
  ( retract(db(gooo,_)), fail ) .
```

Definujte predikát $goo(X)$ bez predikátov $assert$ a $retract$.

## Dokáž (5 bodov)

Nech $\theta$ je unifikátor konečnej množiny termov $S$. Dokážte, že $\theta$ je najvšeobecnejší unifikátor a $\theta\theta = \theta$ práve vtedy, ak pre každý unifikátor $\sigma$ množiny $S$ platí $\sigma = \theta\sigma$

# Skúška 4.

2.6.1993-SKÚŠKA4

## Syntax (5+7 bodov)

Nasledujúca gramatika popisuje funkcionálny jazyk Fun:

| | |
|---|---|
| *číslo* | `integer` |
| *premenná* | `var` |
| *procedúra* | `atóm` |
| *výraz* | *číslo* \| *premenná* \|<br>*výraz* + *výraz* \| *výraz* − *výraz* \| *výraz* ∗ *výraz* \| *výraz* / *výraz* \|<br>( *výraz* ) \| *procedúra* ( *výrazy* )<br>if *podmienka* then *výraz* else *výraz* |
| *výrazy* | $\varepsilon$ \| *výraz* *výrazy* |
| *premenné* | $\varepsilon$ \| *premenná* *premenné* |
| *podmienka* | *výraz* = *výraz* \| *výraz* ≠ *výraz* \| *výraz* < *výraz* \| *výraz* > *výraz* \| |
| *definícia* | *procedúra* ( *premenné* ) is *výraz* |

- Napíšte syntaktický analyzátor pre neterminál *definícia*. Predpoklady: na vstupe je zoznam lexikálnych symbolov pre definíciu v tvare: `[ fact,'(',N,')',is,if,N,'<',2,then,1,else,N,'*',fact,'(',N,'-',1,')']` . Syntaktický analyzátor vytvára vnútornú formu. Vnútorná forma pre aritmetický výraz je aritmetický strom, pre výrazy je zoznam vnútorných foriem pre jednotlivé výrazy (podobne pre premenné), pre if then else je `ifthenelse(podmienka,thenpart,elsepart)` a pre definíciu je `is(procedura,premenne,vyraz)`. Príklad definície: `fact(N) is if N < 2 then 1 else N*fact(N-1)` (5 bodov)

- definujte predikát `interpreter/0`, ktorý v cykle číta *definície* a *výrazy*. Ak načítal definíciu, zanalyzuje ju a vnútorný tvar uloží do databázy. Ak načítal výraz vyhodnotí ho a vypíše výsledok. (7 bodov)

## Hanoj (5 bodov)

Hanojská konfigurácia je konfigurácia kotúčov na troch označených stĺpikoch. Je popísaná termom, napr. $hanoi([1,3,7],[2,4,5],[6])$. Definujte predikát $path(From, To, Moves)$, ktorý nájde nejakú hanojskú cestu (v tvare zoznamu ťahov $Moves$) z konfigurácie $From$ do konfigurácie $To$. Ťah je dvojprvkový zoznam $[cislo_tyce, cislo_tyce]$. Ak takáto cesta neexistuje predikát zlyhá.

## Occur-check (3 body)

Nájdite cieľ, pre ktorý prologovský interpreter **bez occur checku** s nasledujúcim programom dá nekorektnú odpoveď:

```
listy(node(nil,X,nil),[X|Y]-Y).
listy(node(L,X,R),X-Y):-listy(L,X-Z),listy(R,Z-Y).
```

# Skúška 5.

25.6.1993-SKÚŠKA5

## Unifikácia (5 bodov)

Hovoríme, že dva termy $t_1, t_2$ sú *slabo unifikovateľné*, ak existujú substitúcie $\theta_1, \theta_2$ také, že $t_1\theta_1 = t_2\theta_2$. Termy $t_1, t_2$ sú *unifikovateľné*, ak existuje substitúcia $\theta$ taká, že $t_1\theta = t_2\theta$. Termy $t_1, t_2$ sú *jednostranne unifikovateľné*, ak existuje substitúcia $\theta$ taká, že $t_1\theta = t_2$.
Vyjadrite množinové vzťahy medzi nasledujúcimi množinami dvojíc termov:

- $M = \{(t_1, t_2) : t_1$ a $t_2$ sú jednostranne unifikovateľné $\}$,
- $U = \{(t_1, t_2) : t_1$ a $t_2$ sú unifikovateľné $\}$,
- $W = \{(t_1, t_2) : t_1$ a $t_2$ sú slabo unifikovateľné $\}$.

## Puzzle (5 bodov)

Čo počíta nasledujúci program, resp. akú reláciu definuje predikát $p$:

```
p([]).
p([X|Y]):-l([X|Y], X, Z),p(Z).

l([A],A,[]).
l([A|B],A,C):-a(C,[A],B).

a([],L,L).
a([U|M],L,[U|N]):-a(M,L,N).
```

## Záhrada (4 body)

Máme strom $S$. Predpokladajte, že v databáze je definovaný predikát $z\_do(x, y)$, ktorý platí práve vtedy, keď v strome $S$ je vrchol $y$ synom vrchola $x$. Definujte predikát $koren(V)$, ktorý platí práve vtedy, keď vrchol $V$ je koreňom stromu $S$. Predikát $koren(V)$ sa môže volať aj s premennou!

## Osem (6 bodov)

Konfigurácia hry 8 je rozostavenie ôsmich označených kameňov na šachovnici rozmerov $3 \times 3$. Definujte predikát $path(From, To, Moves)$, ktorý nájde nejakú cestu (v tvare zoznamu ťahov $Moves$) z konfigurácie $From$ do konfigurácie $To$. Ak takáto cesta neexistuje predikát zlyhá.

# Skúška 6.

25.6.1993-SKÚŠKA6

## Počet výskytov (5 bodov)

Definujte predikát $pocet\_vyskytov(Sub, Term, N)$, ktorý platí, ak sa term $Sub$ $N$-krát nachádza v terme $Term$. Termy $Sub$, $Term$ nemusia byť základné, t.j. môžu obsahovať premenné.

## Zlý sen (6 bodov)

Ak sa v každej klauzule logického programu $P$ všetky premenné nachádzajúce sa na pravej strane (tele) nachádzajú aj na ľavej strane (hlave) implikácie, alebo množiny premenných tela a hlavy klauzuly sú disjunktné, potom existuje $n$ také, že $gfp(T_P) = T_P \downarrow \omega n$ (pričom $n$ závisí len od programu $P$).

Definujte predikát, pre ktorý existuje nekonečne veľa SLD-zamietnutí.

## Uhádni a dokáž (7 bodov)

Dokážte, resp. vyvráťte nasledujúce tvrdenie:
Ak $A \varepsilon Tp \uparrow n$ pre nejaké $n \varepsilon \omega$, potom existuje SLD-zamietnutie pre $P \cup \{\leftarrow A\}$ dĺžky najviac $n$.

## Uhádni a dokáž (5 bodov)

Dokážte, resp. vyvráťte nasledujúce tvrdenie:
Nech $A$ je atóm a $P$ je program. $\forall(A)$ je logickým dôsledkom $P$ práve vtedy, ak existuje konečné $n$, že $[A] \subseteq T_P \uparrow n$.

## Puzzle (3 body)

Prečo nasledujúci program nepočíta korektne v Arity Prologu:

```
flatt(X,Y):-flatt_q(X,Q-Q,Y).
flatt_q([],Q-Q,[]):-!.
flatt_q(X,Q-Q,[X]):-atomic(X),X\=[],!.
flatt_q(X,[F|Q1]-Q,[X|Y]):-atomic(X),X\=[],!,flatt_q(F,Q1-Q,Y).
flatt_q([X|Xs],Q1-[X|Q],Y):-flatt_q(Xs,Q1-Q,Y).
flatt_q([],[F|Q1]-Q,Y):-flatt_q(F,Q1-Q,Y).
```

## Debugger (8 bodov)

Definujte predikát $debug(P, Q, Args)$, ktorý zistí, či predikáty $P$ a $Q$ sa rovnako správajú vzhľadom na argumenty $Args$. Predpokladajte, že predikáty $P$,$Q$ nerobia side-efekty.

# Skúška 7.

## Puzzle (2+3+4 body)

Nájdite logické vysvetlenie toho, čo znamenajú nasledujúce predikáty:

- ```
  foo(X) :-
    ( X, ( retract(db(fooo,_)), !, fail ; asserta(db(fooo,X)), fail ) ) ;
    retract(db(fooo,X)).
  ```
- ```
  goo(X) :-
    X,
    (retract(db(gooo,_)),! ; asserta(db(gooo,1)),fail) ; retract(db(gooo,_)), fail.
  ```
- ```
  hoo(X,Y) :- X,
    ( not Y, no_record,!,fail ; update_record, fail) ; retract(db(hooo,2)).

  no_record :- retract(db(hooo,_)) ; true.

  update_record :-
     ( db(hooo,2) ;
       retract(db(hooo,1)), asserta(db(hooo,2)) ; asserta(db(hooo,1)) ), !.
  ```

## Uhádni a dokáž (7 bodov)

Dokážte, resp. vyvráťte nasledujúce tvrdenie:

Ak $A \varepsilon Tp \uparrow n$ pre nejaké $n \varepsilon \omega$, potom existuje SLD-zamietnutie pre $P \cup \{\leftarrow A\}$ dĺžky najviac $n$.
  ```

# Témy na projekty.

1. *Constrainty*
   Vstupom je sústava lineárnych diofantických rovníc tvaru $a(1,1)*x(1)+...a(1,m)*x(m) = b(1)$, $a(2,1)*x(1)+...a(2,m)*x(m) = b(2)$,
   $a(n,1)*x(1)+...a(n,m)*x(m) = b(n)$. pričom všetky $a(i,j)$,$b(i)$ sú celé. Nájdite všetky riešenia v obore celých čísel (resp. parametrickú formu). Neskôr uvažujte intervalové ohraničenia na premenné typu $l(i) \leq x(i) \leq u(i)$. Neskôr uvažujte tiež o sústave diofantických nerovníc.

2. *Reformátovač*
   Vstupom je ľubovoľný prologovský program.

   - zostrojiť graf volania predikátov,
   - v rozumnej forme prezentovať cross reference table,
   - zostrojiť graf nestovania predikátov špecifikovať reláciu: predikát $p$ je vnorený do predikátu $q$
   - preformátovať program podľa grafu nestovania

   Výstupom je preformátovaný program a tabuľka referencií

3. *Interpreter Concurrent Prologu*

   - read-only unifikácia,
   - interpreter CP,
   - trasovač programov.

   Literatúra: bude.

4. *Grafové algoritmy*

   - najkratšia cesta medzi vrcholmi v orientovanom nezáporne ohodnotenom grafe,
   - generátor všetkých minimálnych kostier,
   - generátor všetkých maximálnych tokov v grafe,
   - hľadanie rovinného nakreslenia grafu.

   Literatúra: Kučera,L.: Kombinatorické algoritmy.

5. *Flow analyzátor - groudness analysis*
   Vstupom je ľubovoľný prologovský program s informáciou o vstupno-výstupných módoch niektorých predikátov. Mód + znamená vstupný základný term, $-$ znamená výstupný základný term. Príklad: `mode(append,[+,+,+]).` - test na append `mode(append,[+,+,-]).` - zretaz dva zoznamy `mode(append,[-,-,+]).` - pre dané $z$ hladaj $x$,$y$ tak, že $append(x,y,z)$ `mode(append,[+,-,+]).` - pre dané $z$,$x$ hladaj $y$ tak, že $append(x,y,z)$ `mode(append,[-,+,+]).` - pre dané $z$,$y$ hladaj $x$ tak, že $append(x,y,z)$ Zadaný je tiež mód cieľa. Zistite, či program je používaný konzistentne vzhľadom na uvedené módy.

6. *Dokazovač*
   Program pre zadanú formulu výrokového počtu zostrojí jej dôkaz.
   Literatúra: existuje, treba si nájsť.