IB013 Logické programovanie: zbierka príkladov

Hana Rudová
Fakulta informatiky, Masarykova univerzita

11. marca 2013

Obsah

Podekovanie

Materiály v zbierke boli prvotne pripravené Markom Kĺučárom.

Časť I
Zadania úloh

1 Unifikácia

1. Pre každú z nasledujúcich unifikácií rozhodnite, či uspeje. Ak áno, zapíšte výslednú substitúciu

   (a) `pondelok = pondelok`
   (b) `'Pondelok' = pondelok`
   (c) `'pondelok' = pondelok`
   (d) `Pondelok = pondelok`
   (e) `pondelok = utorok`
   (f) `den(pondelok) = pondelok`
   (g) `den(pondelok) = X`
   (h) `den(X) = den(pondelok)`
   (i) `den(pondelok,X) = den(Y,utorok)`
   (j) `den(pondelok,X,streda) = den(Y,utorok,X)`
   (k) `den(pondelok,X,streda) = den(Y,stvrtok)`
   (l) `den(D) = D`
   (m) `vikend(den(sobota),den(nedela)) = vikend(X,Y)`
   (n) `vikend(den(sobota),X) = vikend(X,den(nedela))`

2 Backtracking, rez

Fibonacciho postupnosť je postupnosť čísel, ktorej prvé dva členy sú rovné 1 a každý ďalší člen je súčtom dvoch predchádzajúcich ($fib(1) = 1, fib(2) = 1, fib(n) = fib(n-1) + fib(n-2), n > 2$).

1. Nižšie uvedený logický program obsahuje predikát `fib/2`, ktorý vypočíta n-tý člen Fibonacciho postupnosti. Program však nie je úplne korektný. Správnu hodnotu síce vypočíta, no ak si po jej nájdení vyžiadame pomocou `;` ďalšie riešenia, program sa zacyklí (správne by mal skončiť s odpoveďou `no`). Pridajte do programu na správne miesta operátory rezu tak, aby program v takýchto prípadoch necyklil.

```
fib(N,1) :- N =< 2.
fib(N,X) :- N1 is N-2, fib(N1,1,1,X).
fib(0,_A,X,X).
fib(N,A,B,X) :- N1 is N-1, C is A+B, fib(N1,B,C,X).
```

2. Zamyslite sa a zdôvodnite, prečo nasledujúci logický program, ktorý počíta n-tý člen Fibonacciho postupnosti priamo podľa definície, nie je vhodný (napriek tomu, že výsledok vypočíta vždy korektne)

```
fib(1,1).
fib(2,1).
fib(N,X) :- N>2, N1 is N-1, N2 is N-2, fib(N1,Y),fib(N2,Z),X is Y+Z.
```

3. Napíšte predikát `convert/2`, ktorý prevedie svoj prvý argument (prirodzené číslo) na reprezentáciu daného čísla pomocou následníkov. Napr.:
   ```
   ?-convert(3,X).
   X = s(s(s(0)))
   ```

4. Napíšte predikát `geq/2`, ktorý ako argumenty dostane 2 čísla zadané pomocou následníkov a uspeje, ak prvé číslo je väčšie alebo rovné druhému. Napr.:
   ```
   ?-geq(s(0),s(s(0))).
   no
   ```

5. Napíšte predikát `sucet/3`, ktorý vypočíta súčet dvoch čísel zadaných pomocou následníkov. Napr.:
   ```
   ?- sucet(s(0),s(s(0)),X).
   X = s(s(s(0)))
   ```

6. Napíšte predikát `sucin/3`, ktorý vypočíta súčin dvoch čísel zadaných pomocou následníkov. Napr.:
   ```
   ?- sucin(s(s(0)),s(s(0)),X).
   X = s(s(s(s(0))))
   ```

7. Zadaná je databáza faktov v tvare `priamacesta(a,b)`, ktoré hovoria o tom, že existuje priama (jednosmerná) cesta z miesta `a` do miesta `b`. Napíšte predikát `cesta/2` tak, že `cesta(a,b).` uspeje, ak existuje nejaká cesta (ktorá môže prechádzať cez ľubovoľný počet miest) z miesta `a` do miesta `b`. Môžete predpokladať, že cesty netvoria cykly (tzn. ak raz odídete z nejakého miesta, už sa tam nedá vrátiť)
   Rozšírenie 1: Upravte vaše riešenie tak, že `cesta(a,b,S).` uspeje, ak existuje nejaká cesta a S je zoznam miest, cez ktoré táto cesta prechádza (ideálne v poradí, v akom ich prechádzame)
   Rozšírenie 2: Upravte vaše riešenie tak, aby fungovalo aj v prípade, že cesty tvoria cykly.

3 Aritmetika

1. Rozhodnite, ktoré z nasledujúcich dotazov uspejú a ktoré nie:

   ```
   15  is  3*5.
   14  =\=  3*5.
   15  =  3*5.
   15  ==  3*5.
   15  =\=  3*5.
   15  =:=  3*5.
   3*5 == 3*5.
   3*5 == 5*3.
   3*5  =:=  3*5.
   10-3  =\=  9-2.
   ```

2. Napíšte predikát `mocnina/1`, ktorý uspeje ak číslo zadané ako argument je mocninou dvojky (využite fakt, že mocniny dvojky je možné opakovane bez zvyšku deliť dvomi až kým nedostaneme jednotku)

3. Napíšte predikát `sucet/2`, ktorý spočíta súčet prvých N prirodzených čísel. Napr.:
   ```
   ?- sucet(5,X).
   X = 15
   ```

4. Napíšte predikát `nsd/3`, ktorý vypočíta najväčšieho spoločného deliteľa dvoch čísel. Napr.:
   ```
   ?- nsd(21,49,X).
   X = 7
   ```
   Využite Euklidov algoritmus, ktorý je založený na pozorovaní, že najväčší spoločný deliteľ čísel a, b kde $a > b$ je rovnaký ako najväčší spoločný deliteľ čísel $(a - b), b$

5. Napíšte predikát `cifsucet/2`, ktorý spočíta ciferný súčet zadaného čísla. Napr.:
   ```
   ?- cifsucet(12345,X).
   X = 15
   ```

4 Zoznamy

1. Ktoré z nasledujúcich zápisov predstavujú korektný zápis zoznamu? Aký počet prvkov majú v týchto prípadoch príslušné zoznamy?

   ```
   [1|[2,3,4]]
   [1,2,3|[]]
   [1|2,3,4]
   [1|[2|[3|[4]]]]
   [1,2,3,4|[]]
   [[]|[]]
   [[1,2]|4]
   [[1,2],[3,4]|[5,6,7]]
   ```

2. Napíšte predikát `dvojnasobok/2`, ktorý uspeje ak prvky v druhom zozname sú dvojnásobkami prvkov prvého zoznamu.
   napr. `dvojnasobok([5,3,2],[10,6,4])` uspeje, ale `dvojnasobok([1,2,3],[2,4,5])` neuspeje

3. Napíšte predikát `filter/2`, ktorý zo zoznamu odstráni všetky nečíselné hodnoty, napr.:
   ```
   ?- filter([5,s(0),3,a,2,A,7],X).
   X = [5,3,2,7]
   ```

4. Napíšte predikát `cifry/2`, ktorý zo zoznamu cifier vybuduje číslo, napr.:
   ```
   ?- cifry([2,8,0,7],X).
   X = 2807
   ```

5. Napíšte predikát `nty/3`, ktorý vráti n-tý prvok zoznamu, napr.:
   ```
   ?- nty(4,[5,2,7,8,0],N).
   N = 8
   ```

6. Napšte predikát `rovnaju_sa/2`, ktorý uspeje, ak sa dva zadané zoznamy rovnajú. V prípade, že sa zoznamy nerovnajú, vypíšte dôvod, prečo je to tak (jeden zoznam je kratší ako druhý, výpis prvých 2 prvkov, ktoré sa nerovnajú...) Môžete predpokladať, že zoznamy neobsahujú premenné. Napríklad:
   ```
   ?- rovnaju_sa([5,3,7],[5,3,7]).
   yes

   ?- rovnaju_sa([5,3,7],[5,3,7,2]).
   1. zoznam je kratsi
   no

   ?- rovnaju_sa([5,3,7],[5,2,3,7]).
   3 sa nerovna 2
   no
   ```

7. Napíšte predikát `priemer/2`, ktorý vypočíta aritmetický priemer zadaného zoznamu. Napr.:
   ```
   ?- priemer([1,2,3,4,5,6],X).
   X = 3.5
   ```

8. Napíšte predikát `zip/3`, ktorý z dvoch zoznamov vytvorí nový spojením príslušných prvkov do dvojíc. Napr.:
   ```
   zip([1,2,3],[4,5,6],S).
   S = [1-4,2-5,3-6]
   ```

9. Napíšte predikát `insert/3`, ktorý do usporiadaného zoznamu čísel vloží ďalšie číslo tak, aby aj výsledný zoznam bol usporiadaný. Napr.:
   ```
   ?- insert(7,[1,2,3,10,12],X).
   X = [1,2,3,7,10,12]
   ```

10. Napíšte predikát `merge/3`, ktorý spojí 2 vzostupne usporiadané zoznamy čísel tak, aby výsledný zoznam bol opäť vzostupne usporiadaný.

5 Vstup a výstup

1. Napíšte predikát `wordcount/2`, ktorý ako prvý parameter dostane názov textového súboru obsahujúci slová vo formáte `slovo.` a ako svoj druhý argument vráti zoznam slov a počet ich výskytov v texte. Každé slovo sa má vo výslednom zozname nachádzať práve raz. Výsledný zoznam teda môže vyzerať napríklad takto: `[a-10,ahoj-4,...]`.

2. Napíšte predikát `lettercount/2`, ktorý ako prvý parameter dostane názov textového súboru a ako svoj druhý argument vráti zoznam obsahujúci počet výskytov jednotlivých písmen abecedy (pre jednoduchosť predpokladajme, že súbor obsahuje len malé písmená). Ostatné znaky by mali byť ignorované. (formát výstupného zoznamu si zvoľte podľa vlastného uváženia)

6 Dekompozícia termu

1. Napíšte predikát `map/3` tak, že zavolanie `map(Funkcia,Z1,Z2)` aplikuje funkciu `Funkcia` postupne na všetky prvky zoznamu `Z1` a výsledky vloží do zoznamu `Z2`. Ak napríklad máme `dvojnasobok(X,Y) :- Y is 2*X.` potom po zavolaní `map(dvojnasobok,[5,3,2],S).` dostaneme `S = [10,6,4]`. Pri definícii využite konštrukciu pomocou `=..` a predikát `call/1`.

2. Napíšte predikát `nahrad/4` tak, že `nahrad(T1,T2,T3,T4)` (kde `T1,T2,T3,T4` sú termy bez premenných) uspeje práve ak `T4` vznikol z `T1` nahradením každého výskytu `T2` termom `T3`. Napríklad:
   ```
   ?- nahrad(a(a(a)),a,b,X).
   X = a(a(b))

   ?- nahrad([sin(pi/4),sqrt(sin(pi/2)*2+4),2*sin(pi/2)],sin(pi/2),1,X).
   X = [sin(pi/4),sqrt(1*2+4),2*1]
   ```

3. Napíšte predikát `position(Subterm,Term,Position)` kde `Position` je zoznam identifikujúci prvú pozíciu termu `Subterm` v terme `Term`. Napríklad:
   ```
   ?- position(5,2*f(5),X).
   X = [2,1]
   ```
   pretože `f(5)` je druhý argument pre operátor `*` a `5` je prvý argument `f`

4. Napíšte predikát `copy/2`, tak, že `copy(T1,T2)` uspeje ak term `T2` je kópiou termu `T1`. Tzn. termy majú rovnakú štruktúru, ale používajú rôzne premenné (`f(X,g(X))` je kópiou

`f(Y,g(Y))` ale nie je kópiou `f(X,g(X))` ani `f(U,g(V))`). Snažte sa, aby sa chovanie predikátu `copy` čo najviac podobalo na chovanie zabudovaného predikátu `copy_term/2`.
Nápoveda: použite pomocný zoznam na zapamätanie si premenných v pôvodnom terme a v kópii.

7 Všetky riešenia

1. Uvažujme nasledujúcu databázu faktov:

```
f(a,b).
f(a,c).
f(a,d).
f(e,c).
f(g,h).
f(g,b).
f(i,a).
```

   Zistite bez použitia Prologu aká bude odpoveď na dotaz:

   (a) `findall(X,f(a,X),List).`
   (b) `findall(X,f(X,b),List).`
   (c) `findall(X,f(X,Y),List).`
   (d) `bagof(X,f(X,Y),List).`
   (e) `setof(X,Y^f(X,Y),List).`

2. Napíšte predikát `subsets/2`, ktorý pre danú množinu vygeneruje množinu všetkých jej podmnožín (množinou rozumieme zoznam, v ktorom sa prvky neopakujú). Na poradí podmnožín nezáleží. Môžete si skúsiť napísať aj predikát `isset/1` ktorý uspeje ak zadaný zoznam korektne reprezentuje množinu, tzn. neobsahuje duplicitné prvky.

3. Napíšte predikát `permutations/2`, ktorý pre daný zoznam vygeneruje zoznam všetkých jeho permutácií **v lexikografickom poradí**. Napr.:
```
:- permutations([1,2,3],S).
S = [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]
```

   Môžete použiť zabudovaný predikát `permutation/2` z `library(lists)`

4. Napíšte predikát `variations(+S,+K,-Z)` tak, že `Z` bude obsahovať všetky `K`-prvkové variácie zoznamu `S`.

8 Rezolúcia

1. Nájdite (ak existuje) najobecnejší unifikátor σ nasledujúcich literálov:

   (a) $p(X)$ a $p(f(Y))$
   (b) $p(X, f(X))$ a $p(f(X), Y)$
   (c) $p(X, Y)$ a $p(Z, Z)$
   (d) $p(X, 8)$ a $p(Y, Y)$
   (e) $p(f(X))$ a $p(Z, Y)$
   (f) $p(f(Z), g(X))$ a $p(Y, g(Y))$
   (g) $p(X, g(Z), X)$ a $p(f(Y), Y, W)$

2. Na literáloch C_1, C_2 vykonajte rezolúciu a nájdite rezolventu, jednotlivé kroky (premenovanie premenných, nájdenie najobecnejšieho unifikátora, rezolučný princíp) rozpíšte:

   (a) $C_1 = \{q(X), \neg r(Y), p(f(Z), f(Z))\}$ a $C_2 = \{n(Y), \neg r(W), \neg p(f(W), f(W))\}$

   (b) $C_1 = \{q(X), \neg r(Y), p(X, Y)\}$ a $C_2 = \{n(Y), \neg r(W), \neg p(f(W), f(W))\}$

3. Pomocou lineárnej rezolúcie vyvráťte

   $S = \{\{\neg t(X)\}, \{s(1)\}, \{c(1), c(2), \neg s(1)\}, \{t(0), \neg c(1)\}, \{t(1), \neg c(1)\}, \{t(0), \neg c(2)\}, \{t(2), \neg c(2)\}\}$

4. Prepíšte nasledujúci program v prologovskej notácii ako množinu klauzulí. Následne na cieli `?- a.` demonštrujte lineárnu rezolúciu.

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

5. Pomocou rezolučného princípu vyvráťte $S = \{\{a, b\}, \{c, \neg b\}, \{\neg c\}, \{\neg a, d\}, \{\neg e\}, \{e, \neg d\}\}$. Existuje viacero možných riešení, skúste nájsť čo najviac z nich.

6. Uvážme nasledujúci Prologovský program a dotaz `?- a.` Nakreslite zodpovedajúci SLD strom.

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

7. Uvážme nasledujúci Prologovský program a dotaz `?- p(X,X).` Nakreslite zodpovedajúci SLD strom.

```
p(X,Y) :- q(X,Z), r(Z,Y).
p(X,X) :- s(X).
q(X,b).
q(b,a).
q(X,a) :- r(a,X).
r(b,a).
s(X) :- t(X,a).
s(X) :- t(X,b).
s(X) :- t(X,X).
t(a,b).
t(b,a).
```

8. Do programu z predchádzajúcej úlohy pridajte 1 operátor rezu, tak aby výsledný program uspel práve dvakrát (pomôžte si nakresleným stromom z predchádzajúcej úlohy).

9 Negácia v logickom programovaní

1. Rozhodnite, či nasledujúce programy sú stratifikované. Ak áno, nájdite nejakú ich stratifikáciu:

   (a)
   ```
   p(X):-q(X), \+ r(X).
   p(X):-q(X), \+ t(X).
   r(X):-s(X), \+ t(X).
   t(a).
   s(a).
   s(b).
   q(a).
   ```

   (b)
   ```
   p :- q, \+ r
   r :- s, \+ p
   q.
   s.
   ```

   (c)
   ```
   p :- \+ q, \+ r
   q :- \+ s
   r :- \+ t
   s.
   t.
   ```

2. Uvážme logický program:

   ```
   f(X) :- \+ g(X).
   g(X) :- \+ f(X).
   g(5).
   ```

   Pre ciele `:- f(5)` a `:-` ¬ `f(5)` rozhodnite, či existuje ich odvodenie

   (a) podľa closed world assumption

   (b) podľa negation as failure

3. Uvažujme nasledujúci logický program:

   ```
   a(X) :- c(X), Y is X+1, \+ d(Y).
   a(X) :- c(X), \+ d(X).
   b(X) :- X > 0.
   c(X) :- d(X), Y is X+1, \+ c(Y).
   d(1).
   d(2).
   ```

   a dotaz `:- a(P).`
   Napíšte množinu SLDNF odvodení pre tento dotaz a uveďte o aký typ odvodenia sa jedná.

10 Logické programovanie s obmedzujúcimi podmienkami

1. Napíšte predikát `labeling/1` ktorý ako aktuálnu premennú vyberá premennú s najmenšou veľkosťou domény a hodnoty priraďuje vzostupne (pomocou `indomain`)

2. Zistite aké budú domény `A,B,C` po AC-3 pre `domain([A,B,C],1,10), A#=2*B+1, C#<A`

3. Ako sa zmenia domény z predchádzajúcej úlohy ak miesto AC-3 použijeme konzistenciu medzí?

4. Ako vyzerá stavový priestor pre obmedzenia `A,B,C in 1..4, A#=B+1, C#<B` pri usporiadaní premenných `A,B,C` a použití:

   (a) Backtrackingu

   (b) Kontroly dopredu (forward checking)

   (c) Pohľadu dopredu (looking ahead)

5. Vyriešte pomocou Prologu nasledujúci algebrogram:

```
KC +  I  = OK
 +    +    +
 A +  A  = KM
----------------------------
OL + KO  = LI
```

6. Napíšte program pre hľadanie riešení „zmenšenej" verzie Sudoku (hracie pole má rozmery 4×4 a je rozdelené na 4 štvorce 2×2, v každom riadku, stĺpci a štvorci sa musí každé z čísel 1 až 4 nachádzať práve raz.). Ako rozšírenie môžete pridať aj pekne formátovaný výpis nájdeného riešenia.

7. Vyriešte pomocou Prologu nasledujúcu variáciu (jednu z mnoha) známej Einsteinovej hádanky: Je rada piatich domov, pričom každý má inú farbu. V týchto domoch žije päť ľudí rôznych národností. Každý z nich chová iné zviera, rád pije iný nápoj a fajčí iné cigarety.

   I. Brit býva v červenom dome.

   II. Švéd chová psa.

   III. Dán pije čaj.

   IV. Zelený dom stojí hneď vľavo od bieleho.

   V. Majiteľ zeleného domu pije kávu.

   VI. Ten, kto fajčí Pall Mall, chová vtáka.

   VII. Majiteľ žltého domu fajčí Dunhill.

   VIII. Človek z prostredného domu pije mlieko.

   IX. Nór býva v prvom dome.

   X. Ten, kto fajčí Blend, býva vedľa toho, kto chová mačku.

   XI. Ten, kto chová kone, býva vedľa toho, kto fajčí Dunhill.

   XII. Ten, kto fajčí Blue Master, pije pivo.

   XIII. Nemec fajčí Prince.

   XIV. Nór býva vedľa modrého domu.

   XV. Ten, kto fajčí Blend, má suseda, ktorý pije vodu.

   Kto chová rybičky?

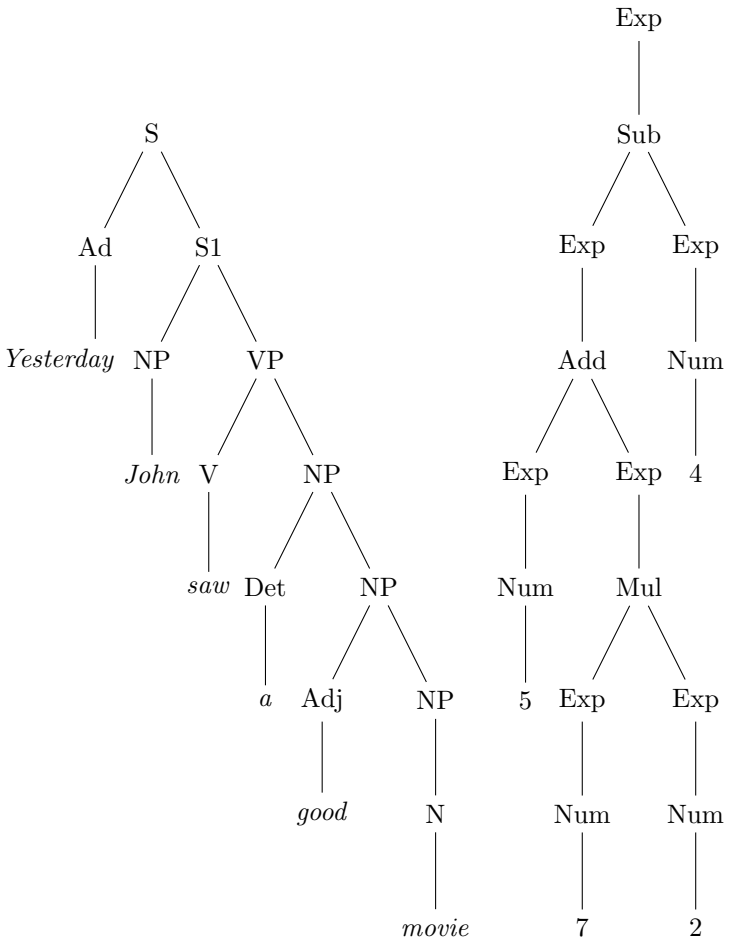
   Úloha má jediné riešenie, je možné jednoznačne určiť všetky informácie, nielen informáciu o tom, kto chová rybičky. Najzložitejšou časťou úlohy teda je zvoliť si správnu reprezentáciu zadaných faktov, následne je zvyšok relatívne jednoduchý. Ak sa vám to nedarí, pri riešení v závere zbierky nájdete nápovedu.

11 DC gramatiky

1. Napíšte DCG pre rozpoznávanie párnych (=sudých) binárnych čísel.
2. Napíšte DCG rozpoznávajúcu palindrómy pozostávajúce z písmen a, b (palindróm je slovo, ktoré sa číta rovnako od začiatku aj od konca. Napr. `[]`, `[a,b,a]`, `[a,b,b,a]` sú palindrómy, ale `[a,b,b]` palindróm nie je)
3. Napíšte DCG pre jazyk, ktorý obsahuje slová v tvare $a^n b^m$ kde $m \geq n$ (pomôcka: vyskúšajte najprv jazyk slov v tvare $a^n b^n$)
4. Napíšte DCG pre jazyk slov v tvare $a^n b^{2n}$.
5. Napíšte DCG pre jazyk slov v tvare $a^n b^{n+1} c^{n+2}$.
6. Preveďte nasledujúcu gramatiku na predikáty s rozdielovými klauzulami:

```
s --> np, vp.
np --> det, n.
vp --> tv, np.
vp --> v.
det --> [the].
det --> [a].
det --> [every].
n --> [man].
n --> [woman].
n --> [park].
tv --> [loves].
tv --> [likes].
v --> [walks].
```

7. Nakreslite derivačný strom pre nejakú vetu rozpoznávanú gramatikou z predchádzajúcej úlohy.
8. Napíšte, aké termy zodpovedajú nasledujúcim derivačným stromom:



9. Pre prvý strom z predchádzajúcej úlohy zostrojte zodpovedajúcu gramatiku s vytváraním derivačného stromu (tzn. tak aby derivačný strom vety "Yesterday John saw a good movie" zodpovedal znázornenému - overte si to v Prologu). Nevadí, ak vaša gramatika bude generovať aj iné, nezmyselné vety.

12 Rôzne

1. Zistite čo robí nasledujúci zámerne škaredo napísaný predikát a prepíšte ho tak, aby bol zrozumiteľnejší (pri zachovaní logickej ekvivalencie s pôvodnou definíciou). Pri zisťovaní, čo program robí, využite trasovanie v Prologu (predikát `trace/0`).

```
foo(X) :- (\+ (X = 1)),(X=2;X=3;foo(X,X)).
foo(X,Y) :-
  (Y=X -> (Z is X-1,foo(X,Z)); (Y=1 ; (P is (X // Y) * Y, \+(P=X), Z is Y-1, foo(X,Z)))).
```

Časť II
Riešenia úloh

1 Unifikácia

1. výsledky si ľahko overíte v prologu

2 Backtracking, rez

1. ```
   fib(N,1) :- N =< 2, !.
   fib(N,X) :- N1 is N-2, fib(N1,1,1,X).
   fib(0,_A,X,X) :- !.
   fib(N,A,B,X) :- N1 is N-1, C is A+B, fib(N1,B,C,X).
   ```

2. Program je veľmi neefektívny (až exponenciálna zložitosť), pretože hodnoty predchádzajúcich členov postupnosti sa počítajú opakovane. Najlepšie to uvidíte, ak si nakreslíte strom výpočtu pre nejaké malé `N`, napr. 6. Takto zapísaný program navyše neumožňuje optimalizáciu posledného volania.

3. ```
   convert(0,0).
   convert(X,s(Y)) :- X1 is X-1, convert(X1,Y).
   ```

4. ```
   geq(_,0).
   geq(s(X),s(Y)) :- geq(X,Y).
   ```

5. ```
   sucet(0,X,X).
   sucet(s(X),Y,s(Z)):-sucet(X,Y,Z).
   ```

6. ```
   sucin(_X,0,0).
   sucin(X,s(Y),Z) :- sucin(X,Y,Z1), sucet(X,Z1,Z).
   ```

7. Riešenie, ktoré zahŕňa obe rozšírenia:

   ```
   (1) :- dynamic navstivene/1.

   (2) cesta(A,A,[A]).
   (3) cesta(A,B,[A|S]) :-
   (4)        assert(navstivene(A)),
   (5)        priamacesta(A,X),
   (6)        \+ navstivene(X),
   (7)        cesta(X,B,S).

   (8) cesta(A,_B,_S) :- retract(navstivene(A)), fail.
   ```

   Pre riešenie bez druhého rozšírenia stačí vynechať riadky 1,4,6,8. Alernatívne by sme mohli druhé rozšírenie riešiť pridaním ďalšieho argumentu - zoznamu už navštívených vrcholov. Ak by sme nechceli riešiť ani prvé rozšírenie, môžeme z predikátov odstrániť tretí argument. Stojí za zmienku, že úloha bez rozšírení je v podstate ekvivalentá úlohe z cvičení s faktami v tvare `rodic(x,y).` a s predikátom `potomek/2`.

3 Aritmetika

1. Výsledky si ľahko overíte v prologu.

2. ```
   mocnina(1).
   mocnina(X) :-  X mod 2 =:= 0, X1 is X // 2, mocnina(X1).
   ```

3. Funkčné riešenie:

   ```
   sucet(N,S) :- sucet(N,0,S).
   sucet(0,S,S) :- !.
   sucet(N,X,S) :- N1 is N-1, X1 is X+N, sucet(N1,X1,S).
   ```

Lepšie riešenie (využíva vzorec pre súčet aritmetickej postupnosti):

```
sucet(N,S) :- S is (N*(N+1)) // 2.
```

4. ```
   nsd(X,X,X).
   nsd(A,B,X) :- A>B, A1 is A-B, nsd(A1,B,X).
   nsd(A,B,X) :- B>A, B1 is B-A, nsd(A,B1,X).
   ```
   Dodajme, že existuje efektívnejšia verzia využívajúca zvyšky po delení.

5. ```
   cifsucet(N,CS) :- cifsucet(N,0,CS).
   cifsucet(0,CS,CS) :- !.
   cifsucet(N,X,CS) :- X1 is X + N mod 10, N1 is N //10, cifsucet(N1,X1,CS).
   ```

4 Zoznamy

1. Výsledky si ľahko overíte v Prologu, na zistenie počtu prvkov zoznamu môžete použiť `length/2`

2. ```
   dvojnasobok([],[]).
   dvojnasobok([X1|T1],[X2|T2]) :- X2 is 2*X1, dvojnasobok(T1,T2).
   ```

3. ```
   filter([],[]).
   filter([X|T1],[X|T2]) :- number(X), !, filter(T1,T2).
   filter([_X|T1],T2) :- filter(T1,T2).
   ```

4. ```
   cifry(S,X) :- cifry(S,0,X).
   cifry([],X,X).
   cifry([A|T],P,X) :- P1 is 10*P + A, cifry(T,P1,X).
   ```

5. ```
   nty(N,[X|_T],Y) :- N =< 1, !,  X = Y.
   nty(N,[_X|T],Nty) :- N1 is N-1, nty(N1,T,Nty).
   ```

6. ```
   rovnaju_sa([],[]).
   rovnaju_sa([X|T1],[X|T2]) :- !, rovnaju_sa(T1,T2).
   rovnaju_sa([],_S) :- write('1. zoznam je kratsi'), !, fail.
   rovnaju_sa(_S,[]) :- write('2. zoznam je kratsi'), !, fail.
   rovnaju_sa([X|_T1],[Y|_T2]) :- write(X), write(' sa nerovna '), write(Y), !, fail.
   ```
   Pre úplnosť dodajme, že ak by sme nepožadovali výpis dôvodu nerovnosti, vystačili by sme si s faktom `rovnaju_sa(S,S).` (ak S neobsahuje premenné)

7. ```
   priemer(S,X) :- priemer(S,0,0,X).
   priemer([],Sum,Count,X) :- X is Sum/Count.
   priemer([H|T],Sum,Count,X) :- Sum1 is Sum+H, Count1 is Count+1, priemer(T,Sum1,Count1,X).
   ```

8. ```
   zip([H1|T1],[H2|T2],[H1-H2|T]) :- zip(T1,T2,T), !.
   zip(_,_,[]).    %aspon 1 zoznam je prazdny
   ```

9. ```
   insert(X,[],[X]).
   insert(X,[H|T],[X,H|T]) :- X =< H, !.
   insert(X,[H|T],[H|T1]) :- insert(X,T,T1).
   ```

10. ```
    merge([],X,X) :- !.  %bez rezu by merge([],[],S). uspel dvakrat
    merge(X,[],X).
    merge([H1|T1],[H2|T2],[H1|T]) :- H1 =< H2, !, merge(T1,[H2|T2],T).
    merge([H1|T1],[H2|T2],[H2|T]) :- merge([H1|T1],T2,T).
    ```
    ```

5 Vstup a výstup

1. Jedno z možných riešení s využitím dynamickej modifikácie databázy:

```
:- dynamic slovo/2.

wordcount(File,S) :-
        retractall(slovo(_,_)),
        seeing(Current),
        see(File),
        repeat,
                read(C),
                (slovo(C,X) -> (Y is X+1, retract(slovo(C,X))); Y is 1),
                assert(slovo(C,Y)),
                C == end_of_file,
                retract(slovo(end_of_file,1)),
        !,
        seen,
        see(Current),
        findall(Z-Count,slovo(Z,Count),S).
```

2.
```
:- dynamic znak/2.

lettercount(File,S) :-
        retractall(znak(_,_)),
        seeing(Current),
        see(File),
        repeat,
                get_char(C),
                (znak(C,X) -> (Y is X+1, retract(znak(C,X))); Y is 1),
                (member(C,[a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z])
                    -> assert(znak(C,Y));
                    true),
                C == end_of_file,
        !,
        seen,
        see(Current),
        setof(Z-Count,znak(Z,Count),S).
```

6 Dekompozícia termu

1.
```
map(_Funkcia, [], []).
map(Funkcia, [H1|T1], [H2|T2]) :-
    F =.. [Funkcia, H1, H2], call(F), map(Funkcia, T1, T2).
```

2.
```
nahrad(X,_,_,X) :- var(X),!.
nahrad(Co,Co,Cim,Cim) :- !.
nahrad(S,_,_,S) :- atomic(S), !.
nahrad([X|T],Co,Cim,[X1|T1]) :- nahrad(X,Co,Cim,X1),nahrad(T,Co,Cim,T1),!.
nahrad(X,Co,Cim,X1) :- compound(X), X =.. [H|T], nahrad(T,Co,Cim,T1), X1 =.. [H|T1].
```

3.
```
position(S,S,[]) :- !.
position(S,[H|_],[1|T]) :- position(S,H,T),!.
position(S,[_|T],[X1|T1]) :- position(S,T,[X2|T1]),X1 is X2+1, !.
position(S,T,L) :- compound(T), T =..[_H|A], position(S,A,L).
```

4.
```
copy(A,B,Z):-cp(A,[],B,Z).

cp(A,Vars,A,Vars):-
    atomic(A).
cp(V,Vars,NV,NVars):-
    var(V),register_var(V,Vars,NV,NVars).
cp(Term,Vars,NTerm,NVars):-
    compound(Term),
    Term=..[F|Args],    % decompose term
    cp_args(Args,Vars,NArgs,NVars),
    NTerm=..[F|NArgs].  % construct copy term

cp_args([H|T],Vars,[NH|NT],NVars):-
    cp(H,Vars,NH,SVars),
    cp_args(T,SVars,NT,NVars).
cp_args([],Vars,[],Vars).

register_var(V,[X/H|T],N,[X/H|NT]):-
    V\==X,          % different variables
    register_var(V,T,N,NT).
register_var(V,[X/H|T],H,[X/H|T]):-
    V==X.           % same variables
register_var(V,[],N,[V/N]).
```

7 Všetky riešenia

1. Výsledky si ľahko overíte v Prologu
2.
```
subsets(X,S) :- isset(X), findall(Y,subset(X,Y),S).

isset([]).
isset([X|T]) :- \+ member(X,T), isset(T).

subset([],[]).
subset([_X|T],S) :- subset(T,S).
subset([X|T],[X|S]) :- subset(T,S).
```
3.
```
:- use_module(library(lists)).
permutations(S,Z) :- setof(X, permutation(S,X), Z).
```
4.
```
:- use_module(library(lists)).
variations(S,K,Z) :- findall(Y, (subset(S,X), length(X,K), permutation(X,Y)), Z)
```

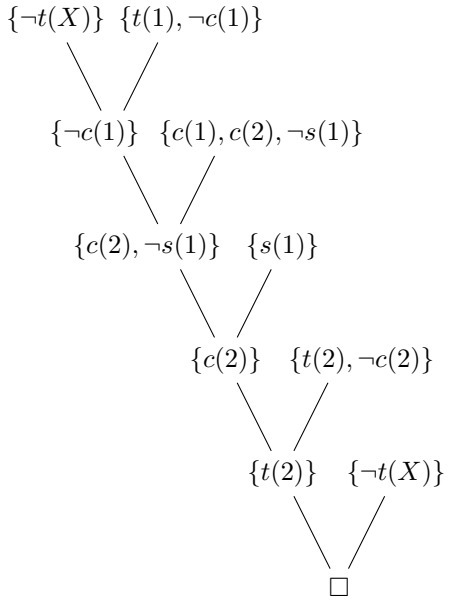
8 Rezolúcia

1. (a) $\sigma = [X/f(Y)]$
   (b) neexistuje
   (c) $\sigma = [X/Z, Y/Z]$
   (d) $\sigma = [X/8, Y/8]$
   (e) neexistuje
   (f) $\sigma = [X/f(Z), Y/f(Z)]$
   (g) $\sigma = [W/f(g(Z)), X/f(g(Z)), Y/g(Z)]$

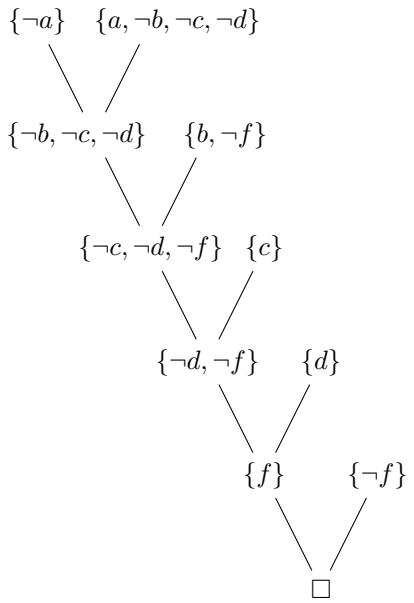
2. (a) Premenovanie premenných $\rho = [Y/A]$
   $C_1 = \{q(X), \neg r(A), p(f(Z), f(Z))\}$, $C_2 = \{n(Y), \neg r(W), \neg p(f(W), f(W))\}$
   Najobecnejší unifikátor $\sigma = [Z/W]$
   $C_1 = \{q(X), \neg r(A), p(f(W), f(W))\}$, $C_2 = \{n(Y), \neg r(W), \neg p(f(W), f(W))\}$
   Z rezolučného princípu dostávame $C = \{q(X), \neg r(A), n(Y), \neg r(W)\}$

   (b) Premenovanie premenných $\rho = [Y/A]$
   $C_1 = \{q(X), \neg r(A), p(X, A)\}$, $C_2 = \{n(Y), \neg r(W), \neg p(f(W), f(W))\}$
   Najobecnejší unifikátor $\sigma = [X/f(W), A/f(W)]$
   $C_1 = \{q(f(W)), \neg r(f(W)), p(f(W), f(W))\}$, $C_2 = \{n(Y), \neg r(W), \neg p(f(W), f(W))\}$
   Z rezolučného princípu $\{q(f(W)), \neg r(f(W)), n(Y), \neg r(W)\}$

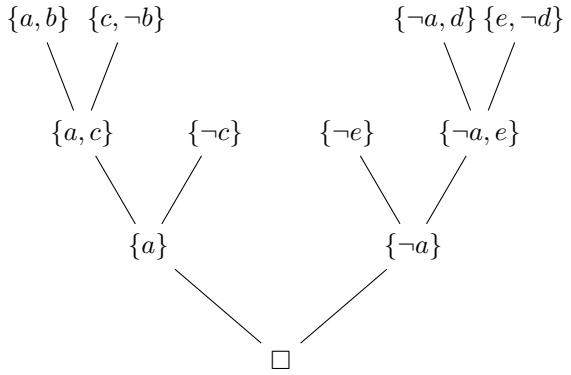
3. Jeden z možných postupov



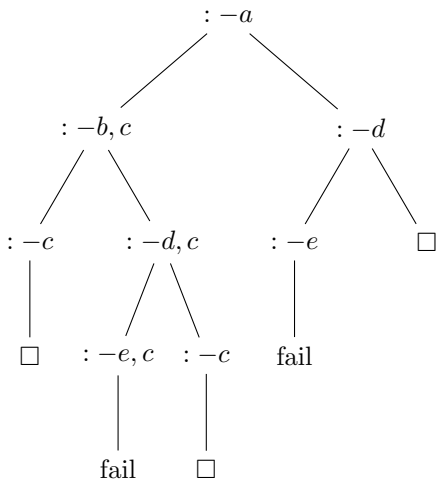
4. $P = \{P_1, P_2, P_3, P_4, P_5\}$
   $P_1 = \{a, \neg b, \neg c, \neg d\}, P_2 = \{b, \neg f\}, P_3 = \{c\}, P_4 = \{d\}, P_5 = \{f\}$
   $C = \{\neg a\}$



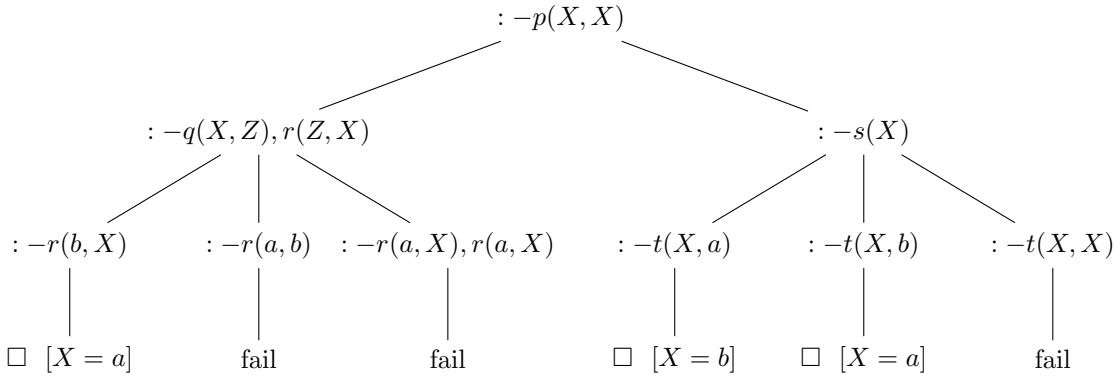
5. Jedno z možných riešení:



6.

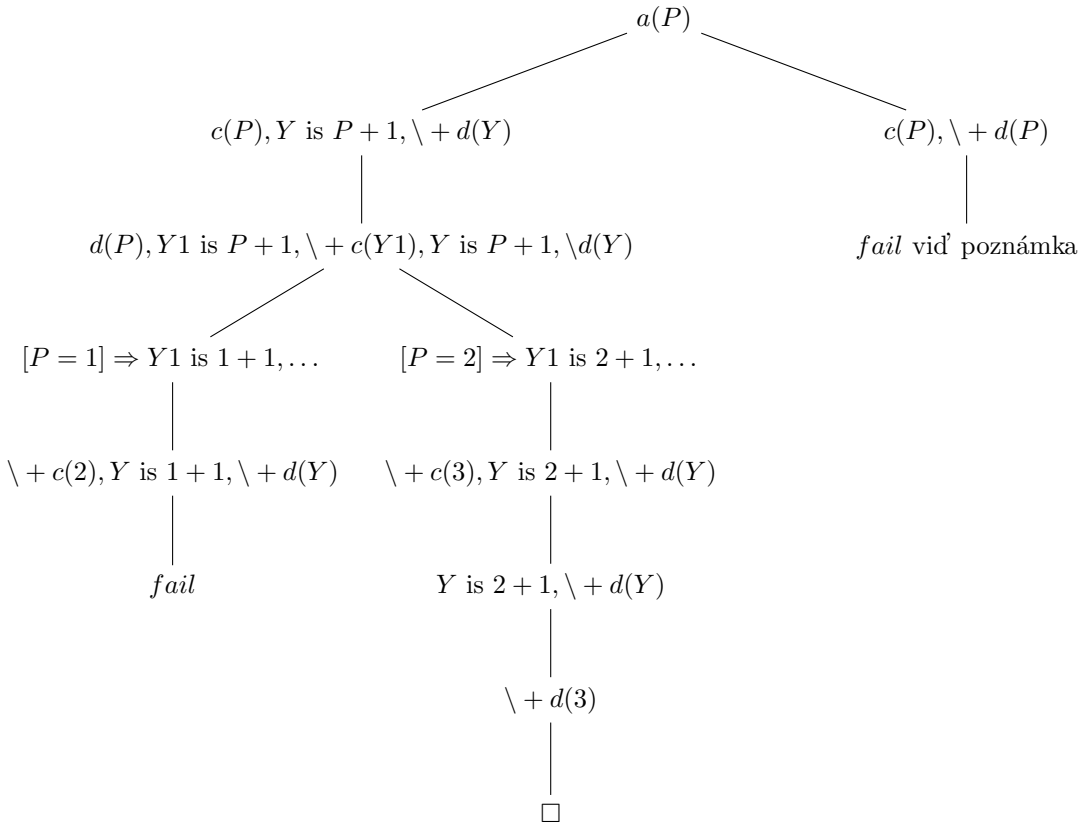


7.



8. `s(X) :- t(X,a).` nahradíme za `s(X) :- t(X,a),!.`

9 Negácia v logickom programovaní

1. (a) Áno, napríklad: S_0 obsahuje fakty S_1 tretie pravidlo, S_2 prvé dve pravidlá.
   (b) Nie
   (c) Áno, napríklad: S_0 obsahuje fakty, S_1 tretie pravilo, S_2 druhé pravidlo, S_3 prvé pravidlo.
2. (a) Existuje pre `:- ¬ f(5)`
   (b) Odvodenie neexistuje ani pre jeden z cieľov, odvodenia sú blokované. Problémom je, že program nie je stratifikovaný.
3. 

$c(2)$

$d(2), Y \text{ is } 2+1, \backslash + c(Y)$

$Y \text{ is } 2+1, \backslash + c(Y)$

$\backslash + c(3)$

$\square \Rightarrow \backslash + c(2)$ neúspešné

$c(3)$

$d(3), Y \text{ is } 3+1, \backslash + c(Y)$

$fail \Rightarrow \backslash + c(3)$ úspešné

Poznámka: časť stromu bola pre nedostatok miesta vynechaná. V príslušnej vetve ale nikdy nenastane úspech, keďže v ďalšom kroku dostaneme cieľ v tvare `d(P), ..., \+d(P)`. Vo vlastnom záujme si však chýbajúcu časť stromu nakreslite.

10 Logické programovanie s obmedzujúcimi podmienkami

1. ```
   labeling([]).
   labeling(Vars) :-
           sortByDomainSize(Vars,SortedVars),
           SortedVars=[First|Rest],
           indomain(First),
           label(Rest).
   ```

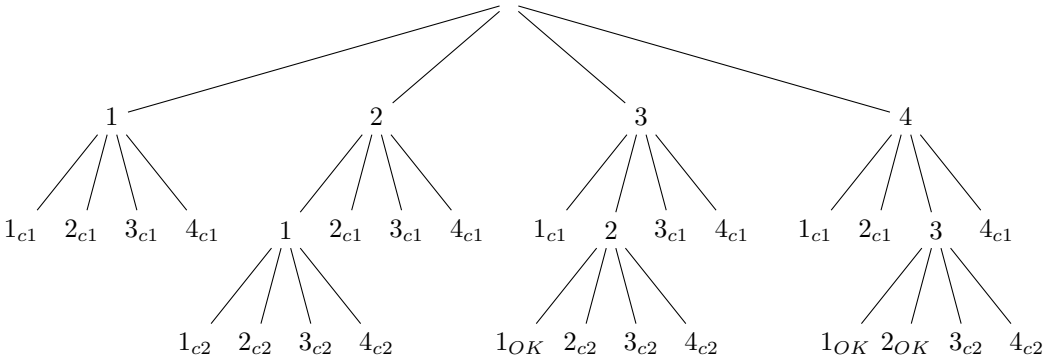
   Kde `sortByDomainSize` je nejaký radiaci algoritmus, ktorý namiesto porovnania `X>Y` používa `fd_size(X,SizeX), fd_size(Y,SizeY), SizeX>SizeY`

2. `A in {3,5,7,9}, B in 1..4, C in 1..8`

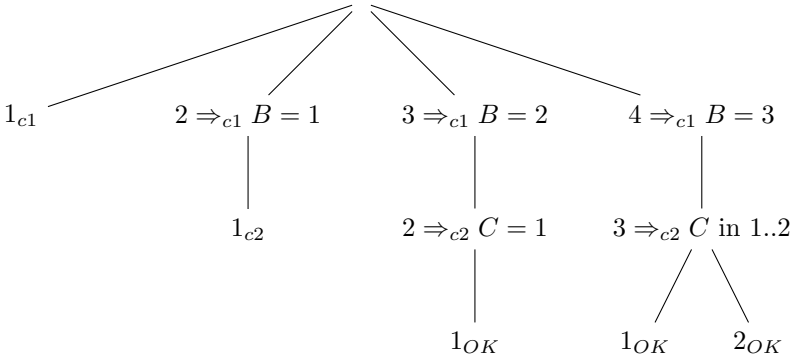
3. `A in 3..9, B in 1..4, C in 1..8`

4. Označme si dané odmedzenia postupne $c1$ a $c2$. Dolný index pri hodnote označuje obmedzenie, ktoré bolo porušené, prípadne je OK ak bolo nájdené splňujúce priradenie

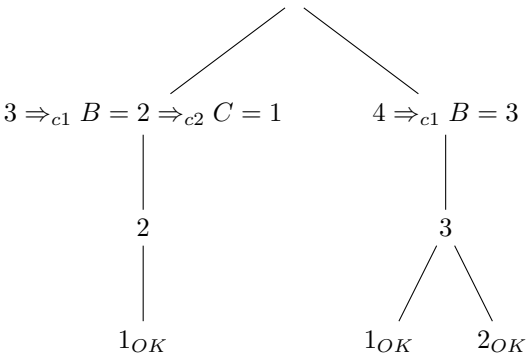
   (a)
   ```



(b)



(c) Pred samostatným prehľadávaním zaistíme hranovú konzistenciu pomocou AC3 - dostávame `A in 3..4, B in 2..3, C in 1..2`



5.
```
alg([K,C,I,O,A,M,L]) :-
        domain([K,I,O,A,L],1,9),
        domain([C,M],0,9),
        all_distinct([K,C,I,O,A,M,L]),
        10*K + C + I #= 10*O + K,
        A + A #= 10*K + M,
        10*O + L + 10*K + O #= 10*L + I,
        10*K + C + A #= 10*O + L,
        I + A #= 10*K + O,
        10*O + K + 10*K + M #= 10*L + I,
        labeling([],[K,C,I,O,A,M,L]).
```

6. Riešenie bez výpisu:

```
sudoku4([A1,A2,A3,A4,
```

```
B1,B2,B3,B4,
C1,C2,C3,C4,
D1,D2,D3,D4]) :-
          Vals = [A1,A2,A3,A4,B1,B2,B3,B4,C1,C2,C3,C4,D1,D2,D3,D4],
          domain(Vals,1,4),
          all_distinct([A1,A2,A3,A4]),
          all_distinct([B1,B2,B3,B4]),
          all_distinct([C1,C2,C3,C4]),
          all_distinct([D1,D2,D3,D4]),
          all_distinct([A1,B1,C1,D1]),
          all_distinct([A2,B2,C2,D2]),
          all_distinct([A3,B3,C3,D3]),
          all_distinct([A4,B4,C4,D4]),
          all_distinct([A1,A2,B1,B2]),
          all_distinct([A3,A4,B3,B4]),
          all_distinct([C1,C2,D1,D2]),
          all_distinct([C3,C4,D3,D4]),
          labeling([],Vals).
```

7. **Nápoveda**: Domy si môžeme očíslovať zľava doprava číslami 1 až 5. Pre každú národnosť, farbu, zviera, nápoj a cigarety si vytvoríme premennú, ktorá bude nadobúdať hodnoty 1 až 5 podľa príslušného domu. Následne môžeme napríklad obmedzenie "človek z prostredného domu pije mlieko" zapísať ako `Mlieko #= 3` a obmedzenie "Brit býva v červenom dome" ako `Brit #= Cerveny`.

   Kompletné riešenie:

```
einstein([Nor,Brit,Sved,Dan,Nemec,
     Cerveny,Zeleny,Zlty,Modry,Biely,
     Caj,Kava,Mlieko,Pivo,Voda,
     Pes,Vtak,Macka,Kon,Ryba,
     Pallmall,Dunhill,Blend,Bluemaster,Prince]):-
   Vals = [Nor,Brit,Sved,Dan,Nemec,
           Cerveny,Zeleny,Zlty,Modry,Biely,
           Caj,Kava,Mlieko,Pivo,Voda,
           Pes,Vtak,Macka,Kon,Ryba,
           Pallmall,Dunhill,Blend,Bluemaster,Prince],

   domain(Vals,1,5),

   all_distinct([Nor,Brit,Sved,Dan,Nemec]),
   all_distinct([Cerveny,Zeleny,Zlty,Modry,Biely]),
   all_distinct([Caj,Kava,Mlieko,Pivo,Voda]),
   all_distinct([Pes,Vtak,Macka,Kon,Ryba]),
   all_distinct([Pallmall,Dunhill,Blend,Bluemaster,Prince]),

   %Brit byva v cervenom dome.
   Brit #= Cerveny,
   %Sved chova psa.
   Sved #= Pes,
   %Dan pije caj.
   Dan #= Caj,
   %Zeleny dom stoji hned vlavo od bieleho.
   Zeleny #= Biely - 1,
```

```
%Majitel zeleneho domu pije kavu.
Zeleny #= Kava,
%Ten, kto fajci Pall Mall, chova vtaka.
Pallmall #= Vtak,
%Majitel zlteho domu fajci Dunhill.
Zlty #= Dunhill,
%clovek z prostredneho domu pije mlieko.
Mlieko #= 3,
%Nor byva v prvom dome.
Nor #= 1,
%Ten, kto fajci Blend, byva vedla toho, kto chova macku.
(Blend #= Macka+1 ; Blend #=Macka-1),
%Ten, kto chova kone, byva vedla toho, kto fajci Dunhill.
(Kon #= Dunhill+1 ; Kon#=Dunhill-1),
%Ten, kto fajci Blue Master, pije pivo.
Bluemaster #= Pivo,
%Nemec fajci Prince.
Nemec #= Prince,
%Nor byva vedla modreho domu.
(Nor #= Modry + 1 ; Nor #= Modry-1 ),
%Ten, kto fajci Blend, ma suseda, ktory pije vodu.
(Blend #= Voda + 1; Blend#= Voda-1),

labeling([],Vals).
```

11 DC gramatiky

1. Párne čísla sú práve tie, ktorých binárny zápis končí nulou. Daná DCG môže vyzerať napríklad nasledovne:

   ```
   parne --> [0].
   parne --> [1], parne.
   parne --> [0], parne.
   ```

2. Výsledná gramatika môže vyzerať takto:

   ```
   palindrom --> [].
   palindrom --> [a].
   palindrom --> [b].
   palindrom --> [X],palindrom,[X], {member(X,[a,b])}.
   ```

   (posledný riadok je možné nahradiť dvoma rozpísaním možných hodnôt `X`)

3. Pre jazyk $a^n b^n$ napíšeme gramatiku

   ```
   an_bn --> [].
   an_bn --> [a],an_bn,[b].
   ```

   Gramatiku pre jazyk $a^n b^m$ kde $m \geq n$ potom môžeme zadefinovať nasledujúcim spôsobom:

   ```
   an_bm --> an_bn, same(b).
   ```

   Pričom `same` definujeme ako

```
same(_X) --> [].
same(X) --> [X],same(X).
```

(teda trochu inak ako bol definovaný na cvičeniach)

4. ```
   an_b2n --> [].
   an_b2n --> [a],an_bn,[b],[b].
   ```

5. Jedno z možných riešení je veľmi podobné riešeniu pre jazyk $a^n b^n c^n$, ktoré bolo uvedené na cvičeniach. Nižšie uvedené riešenie je založené na rovnakej myšlienke (tzn. pomocou následníkov), avšak je zapísané stručnejším spôsobom pomocou pomocného neterminálu `repeat`:

   ```
   abc --> repeat(a,X), repeat(b,s(X)), repeat(c,s(s(X))).
   repeat(_,0) --> [].
   repeat(A,s(X)) --> [A],repeat(A,X).
   ```

6. ```
   s(S0,S) :- np(S0,S1), vp(S1,S).
   np(S0,S) :- det(S0,S1), n(S1,S).
   vp(S0,S) :- tv(S0,S1), np(S1,S).
   vp(S0,S) :- v(S0,S).
   det([the|S],S).
   det([a|S],S).
   det([every|S],S).
   n([man|S],S).
   n([woman|S],S).
   n([park|S],S).
   tv([loves|S],S).
   tv([likes|S],S).
   v([walks|S],S).
   ```

7. závisí od vybranej vety

8. S (Ad (*Yesterday*), S1 (NP (N (*John*)), VP (V (*saw*), NP (Det (*a*), NP (Adj (*good*), NP (N(*movie*)))))))
   Exp(Sub(Exp(Add(Expr(Num(5)),Exp(Mul(Expr(Num(7)),Exp(Num(2)))))),Exp(Num(4)))

9. ```
   sentence(s(Ad,S)) --> adverb(Ad), sentence1(S).
   adverb(ad(yesterday)) --> [yesterday].
   sentence1(s1(NP,VP)) --> noun_phrase(NP), verb_phrase(VP).
   noun_phrase(np(N)) --> noun(N).
   noun_phrase(np(Det,NP)) --> determiner(Det), noun_phrase(NP).
   noun_phrase(np(Adj,NP)) --> adjective(Adj), noun_phrase(NP).
   determiner(det(a)) --> [a].
   adjective(adj(good)) --> [good].
   noun(n(john)) --> [john].
   noun(n(movie)) --> [movie].
   verb_phrase(vp(V,NP)) --> verb(V), noun_phrase(NP).
   verb(v(saw)) --> [saw].
   ```

12 Rôzne

1. `foo(X).` uspeje práve ak `X` je prvočíslo. Program môžeme prepísať ako:
   ```

```
prime(X) :- X =\= 1, X1 is X-1, test(X,X1).
test(_X,1) :- !.
test(X,Y) :- X mod Y > 0, Y1 is Y-1, test(X,Y1).
```