Úlohy s ohraničeniami

OBSAH PREDNÁŠKY

- Hlavné aplikačné oblasti
- Základné definície
- Metódy riešenia
- Logické programovanie ohraničení
- Jazyk ECLiPSe

Hlavné aplikačné oblasti

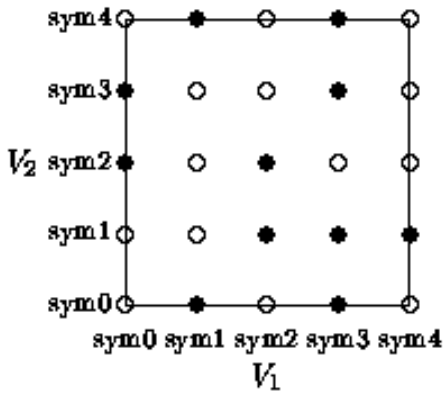
- Rozvrhovanie
  - časové
    - výrobné,
    - záťaž procesorov,
    - rozvrh hodín a pod.
  - priestorové (rozmiestňovanie skladov, pobočiek, strojov ...)
- Plánovanie
- Rôzne optimalizačné úlohy
  - investičná,
  - distribučná,
  - dopravná a pod.
- Konfiguračný návrh

Základné definície

- Úloha s ohraničeniami (CSP) je vo všeobecnosti definovaná pomocou 3 množín:
  - Množina **premenných** $V = \{V_1, V_2, \ldots, V_n\}$
  - Množina **domén** $D = \{D_1, D_2, \ldots, D_n\}$
  - Množina **ohraničení** $C = \{C^1, \ldots, C^n, C^{1,2}, \ldots, C^{n-1,n}, \ldots, C^{1,2,\ldots,n}\}$

- Domény môžu byť:
  - ***Konečné*** (boolovské, ohraničené celočíselné, vymenované)
  - ***Nekonečné*** (neohraničené celočíselné, racionálne, reálne)
- Ohraničenia môžu byť:
  - ***Unárne*** (C^i), ***binárne*** ($C^{i,j}$), ..., ***n-árne*** ($C^{1,2,\ldots,n}$)
  - Definované ***vymenovaním*** alebo ***predpisom*** (pozri ďalej)

Definície ohraničení

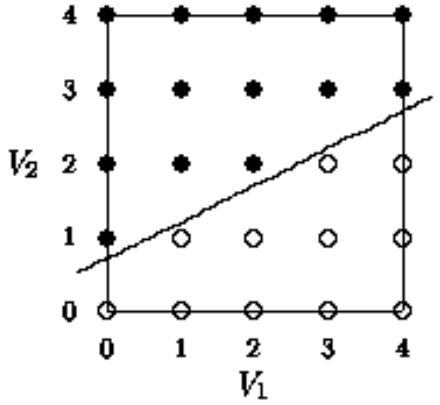


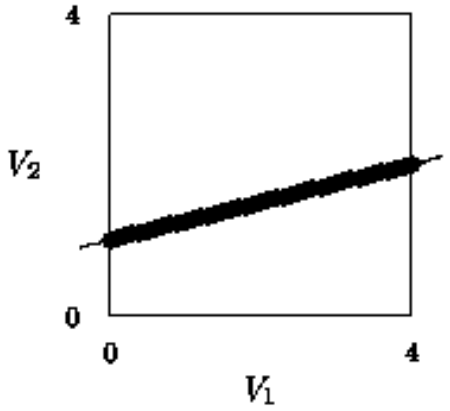
Ohraničenia definované vymenovaním povolených (zakázaných) kombinácií hodnôt,

napr. $C^{1,2}$ = *{(sym0,sym2), (sym0,sym3), (sym1,sym0), …}*

Ohraničenia definované predpisom

Napr. nerovnosť pre celočíselné premenné $C^{1,2}$: $V_2 > 0.5V_1 + 0.75$



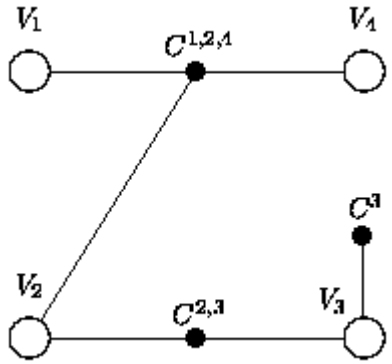


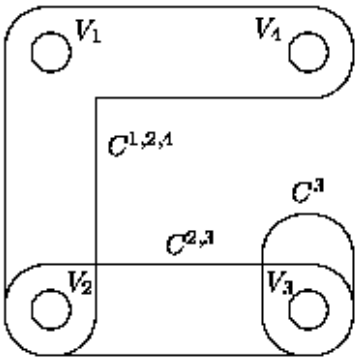
alebo rovnosť pre reálne premenné $C^{1,2}$: $V_2 = 0.25V_1 + 1$

Grafická reprezentácia úloh s ohraničeniami

Sieť ohraničení

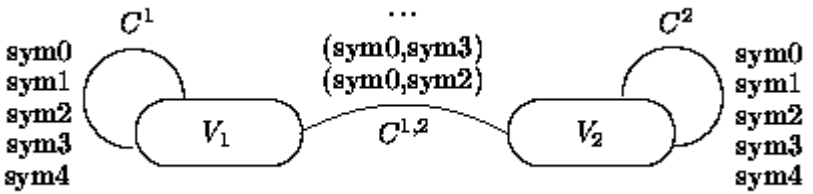
a) Neorientovaný graf s dvoma typmi uzlov (pre premenné a pre ohraničenia), hrany sú možné len medzi uzlami rôznych typov





b) Hypergraf s jedným typom uzlov (premenné) a hyperhranami (ohraničenia)

c) Jednoduchšie znázornenie pre binárne CSP

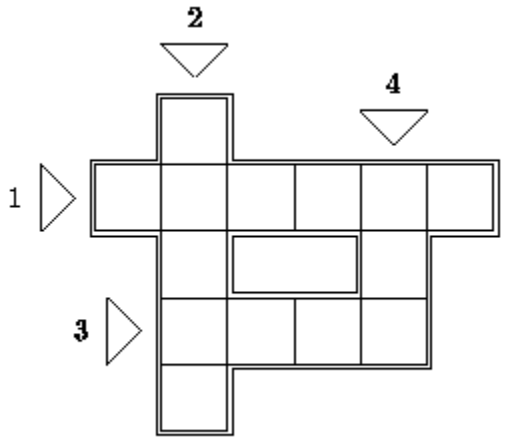


Čo znamená riešiť úlohu s ohraničeniami?

- **Priestor prehľadávania P** je definovaný ako kartézsky súčin domén všetkých premenných, t.j. $D_1 \times D_2 \times \ldots \times D_n$
- **Riešiť úlohu s ohraničeniami** znamená jednu z nasledujúcich úloh:
  - Dokázať existenciu alebo neexistenciu riešenia
  - Zistiť počet riešení
  - **Nájsť jedno riešenie**
  - Nájsť všetky riešenia
- Ak je definovaná **funkcia nákladov** $f_c(P) \rightarrow \mathbf{R}$ nad P, potom je možné riešiť aj optimalizačnú verziu CSP, t.j.
  - **Nájsť optimálne riešenie (ktoré minimalizuje f_c)**

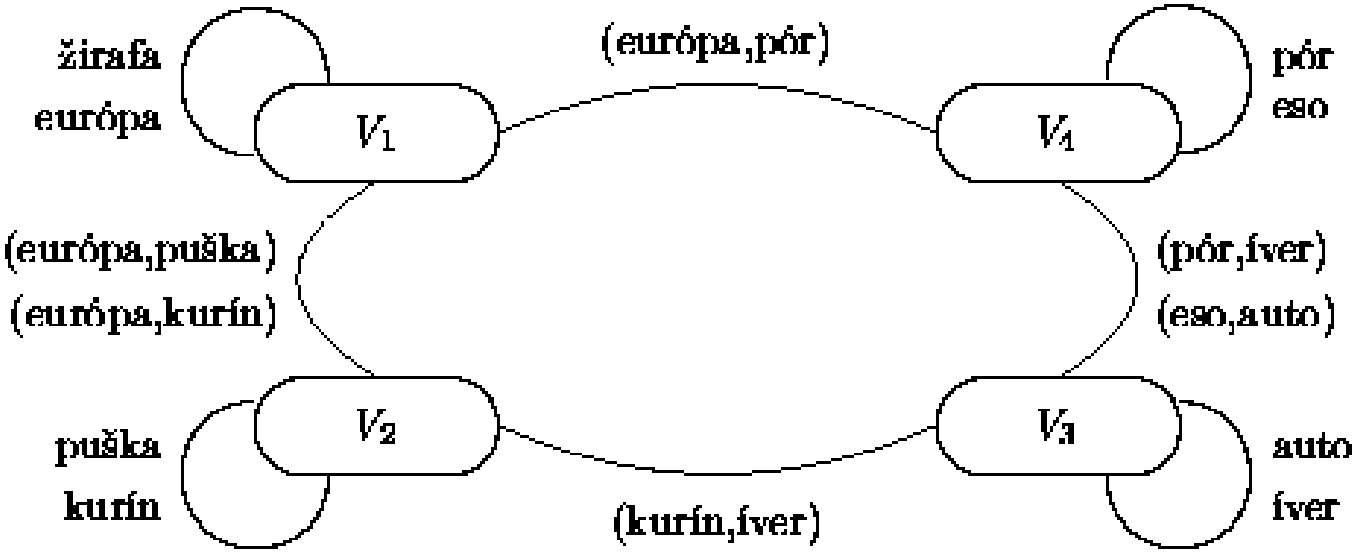
Príklad 1. (doplňovačka)

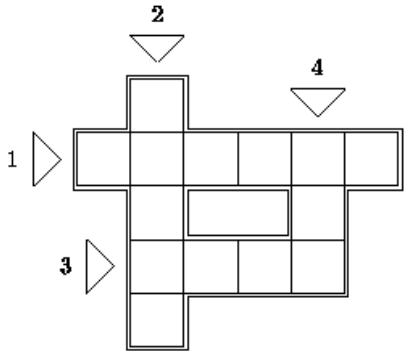
- Nasledovné *slová* sú k dispozícii pre vloženie do danej doplňovačky:
  - 6 písmen: žirafa, európa
  - 5 písmen: puška, kurín
  - 4 písmená: auto, íver
  - 3 písmená: pór, eso



- Transformácia na úlohu s ohraničeniami:
  - $V = \{V_1, V_2, V_3, V_4\}$
  - $D_i = \{$žirafa, európa, puška, kurín, auto, íver, pór, eso$\}$ $\forall$ i = 1 .. 4
  - $C = \{C^1, C^2, C^3, C^4, C^{1,2}, C^{1,4}, C^{2,3}, C^{3,4}\}$
  - Napr. $C^1 = \{$(žirafa), (európa)$\}$, $C^{1,2} = \{$(žirafa,žirafa), …, (eso, eso), …, (európa, puška), …$\}$

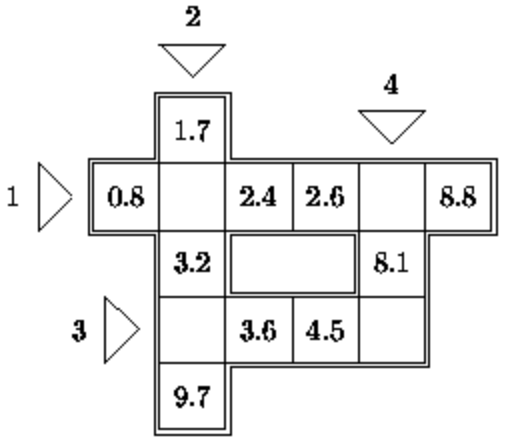
Sieť ohraničení pre príklad 1 (redukovaná verzia)





Príklad 2. (numerická doplňovačka)

- Doplňte 4 kladné *reálne čísla* do voľných políčok tak, aby
  - súčet čísiel v každom riadku a stĺpci bol 20
  - sa čísla zvyšovali v smere zľava doprava
  - sa čísla zvyšovali v smere zhora nadol



- Transformácia na úlohu s ohraničeniami:
  - $V = \{V_{12}, V_{14}, V_{32}, V_{34}\}$
  - $D_i = \mathbf{R} \quad \forall\ i$
  - $C = \{C^{12}, C^{14}, C^{32}, C^{34}, C^{12,14}, C^{12,32}, C^{32,34}, C^{14,34}\}$
  - Napr. $\mathbf{C^{12}}$**:** $\{V^{12} > 1.7, V^{12} < 2.4\}$,

    $\mathbf{C^{12,14}}$ **:** $\{V^{12} < V^{14}, 0.8 + V^{12} + 2.4 + 2.6 + V^{14} + 8.8 = 20\}$, atď.

Prístupy k riešeniu úloh s ohraničeniami nad **konečnými doménami**

1. **Redukčné algoritmy** (sa snažia zredukovať P, t.j. domény premenných a/alebo ohraničenia tak aby nájdenie riešenia bolo čo najjednoduchšie)
   - Zabezpečenie silnej *k*-konzistencie je výpočtovo náročné, ale následné nájdenie riešenia je veľmi jednoduché
2. **Prehľadávacie algoritmy** (prehľadávajú P za účelom nájdenia riešení)
   - Jednoduché algoritmy ale musia prehľadávať obrovské časti P, čo spôsobuje exponenciálnu časovú zložitosť
3. **Kombinované algoritmy** (kombinujú prístupy 1 a 2 v snahe využiť výhody oboch prístupov)
   - Pred každým krokom prehľadávania sa aplikuje (jednoduchší) redukčný algoritmus

Redukčné algoritmy 1.

- ***k*-konzistencia** (*k* je z intervalu <1, n>) siete ohraničení vlastne znamená konzistenciu všetkých podmnožín množiny premenných **V** s kardinalitou *k*
- Formálna definícia *k*-konzistencie:

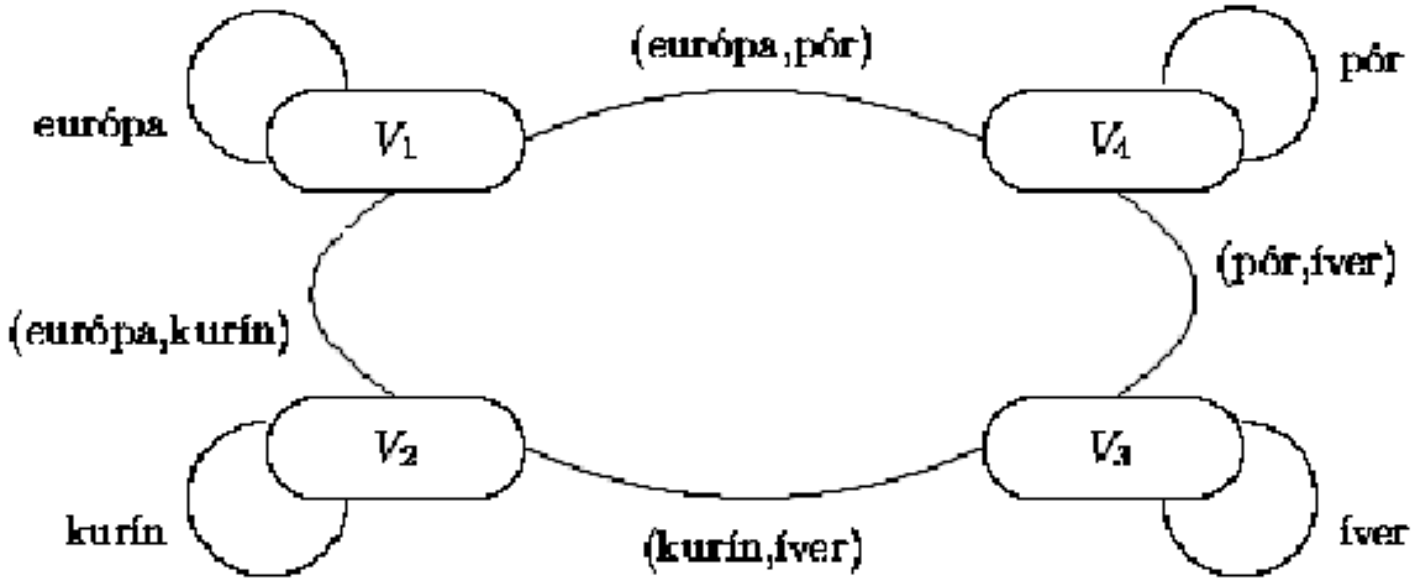
  Nech ľubovoľným *k-1* premenným sú priradené také hodnoty z ich domén, aby všetky ohraničenia definované nad touto (*k-1*)-ticou premenných boli splnené. Pre ľubovoľnú ďalšiu *k*-tu premennú je možné vybrať takú hodnotu z jej domény, že všetky ohraničenia definované nad vzniknutou *k*-ticou premenných sú splnené.
- **Silná *k*-konzistencia** znamená, že sieť ohraničení je *j*-konzistentná pre všetky *j* z intervalu <1, *k*>

Redukčné algoritmy 2.

- 1-konzistenciu (uzlovú konzistenciu, NC – *node consistency*) je veľmi jednoduché zabezpečiť, stačí porovnať každú doménu D_i s unárnym ohraničením C^i a odstrániť všetky nekonzistentné hodnoty z D_i
- 2-konzistencia (hranová konzistencia, AC – *arc consistency*) znamená, že hodnoty v ľubovoľnej doméne D_i sa porovnávajú s hodnotami inej premennej D_j a ak pre ľubovoľnú hodnotu v_m^i z D_i neexistuje konzistentná hodnota z D_j (s ohľadom na $C^{i,j}$), táto hodnota v_m^i sa odstráni z domény D_i
- Existujú rôzne verzie algoritmu AC s rôznou časovou a priestorovou zložitosťou (AC2 .. AC7)
- Zložitosť algoritmov pre zabezpečenie *k*-konzistencie *k*>2 je exponenciálna, preto sa používajú iba veľmi zriedkavo (niekedy sa používa 3-konzistencia nazývaná tiež konzistencia po ceste, PC – *path consistency*)

Redukčné algoritmy 3.

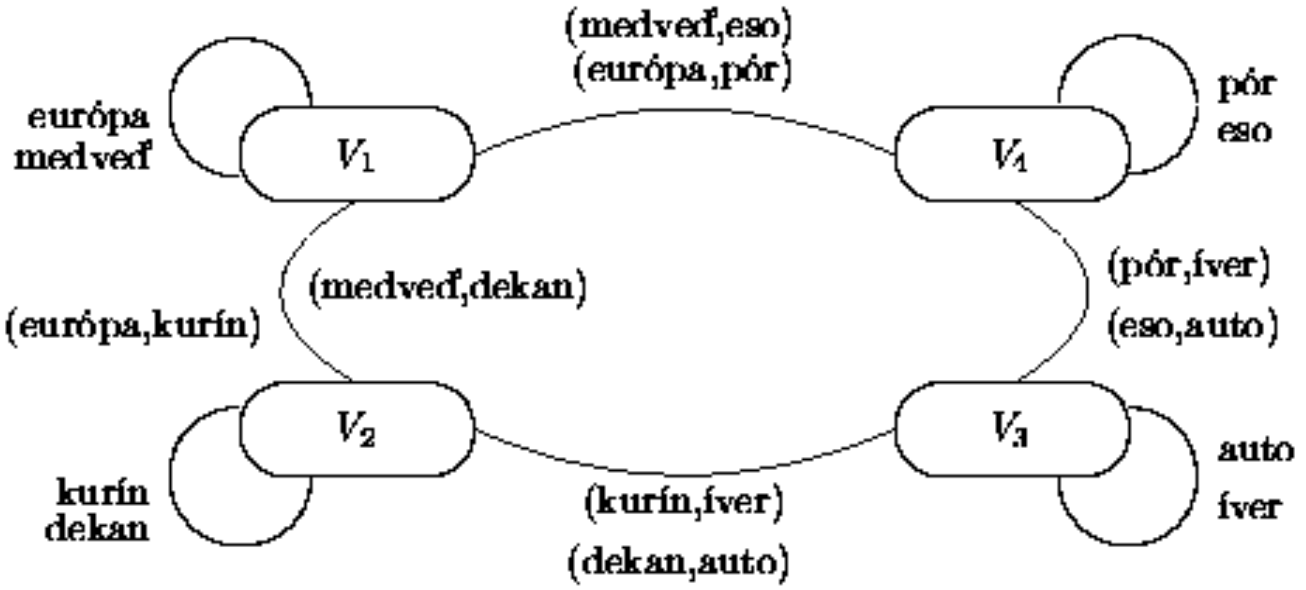
- Aplikujme teraz algoritmus AC na úlohu z príkladu 1 pri danom poradí premenných: V_1, V_2, V_4, a V_3



- V tomto prípade redukčný algoritmus vedie k redukcii P na jediný bod, ktorý predstavuje jediné riešenie tohto CSP

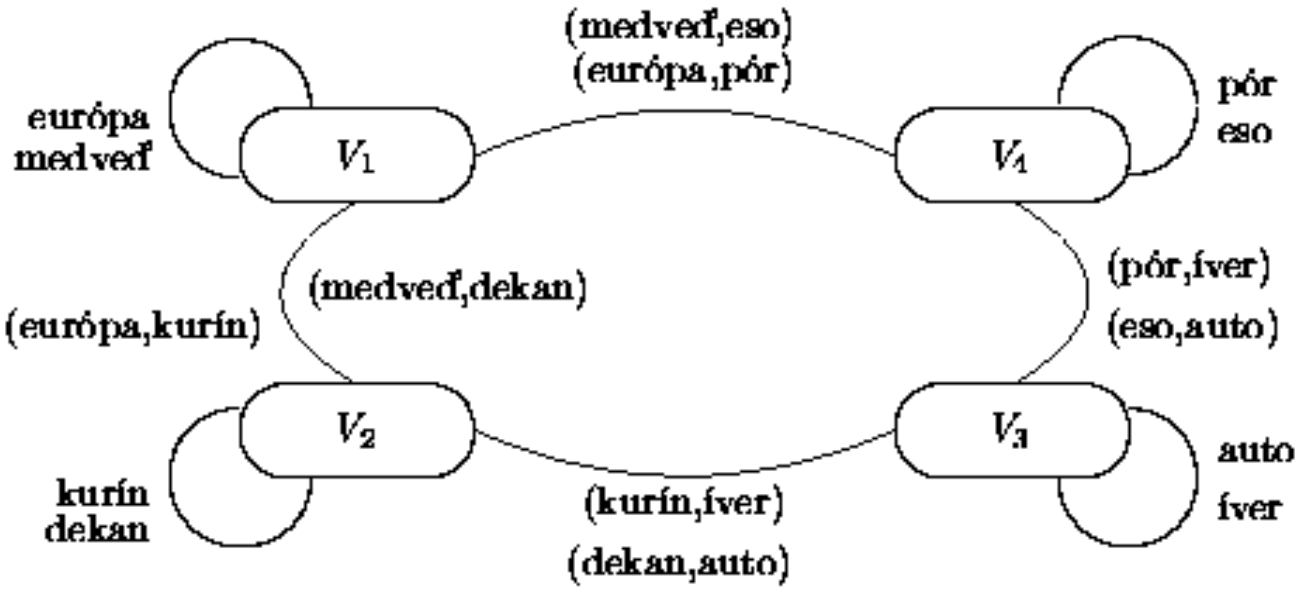
Redukčné algoritmy 4.

- Aplikujme teraz algoritmus AC na úlohu z príkladu 1 modifikovanú takým spôsobom, že pridáme 2 nové možné slová, a síce **medveď** a **dekan**
- Poradie premenných sa nezmení: V_1, V_2, V_4, a V_3



Redukčné algoritmy 5.

- Žiadna ďalšia redukcia použitím algoritmu AC nie je možná

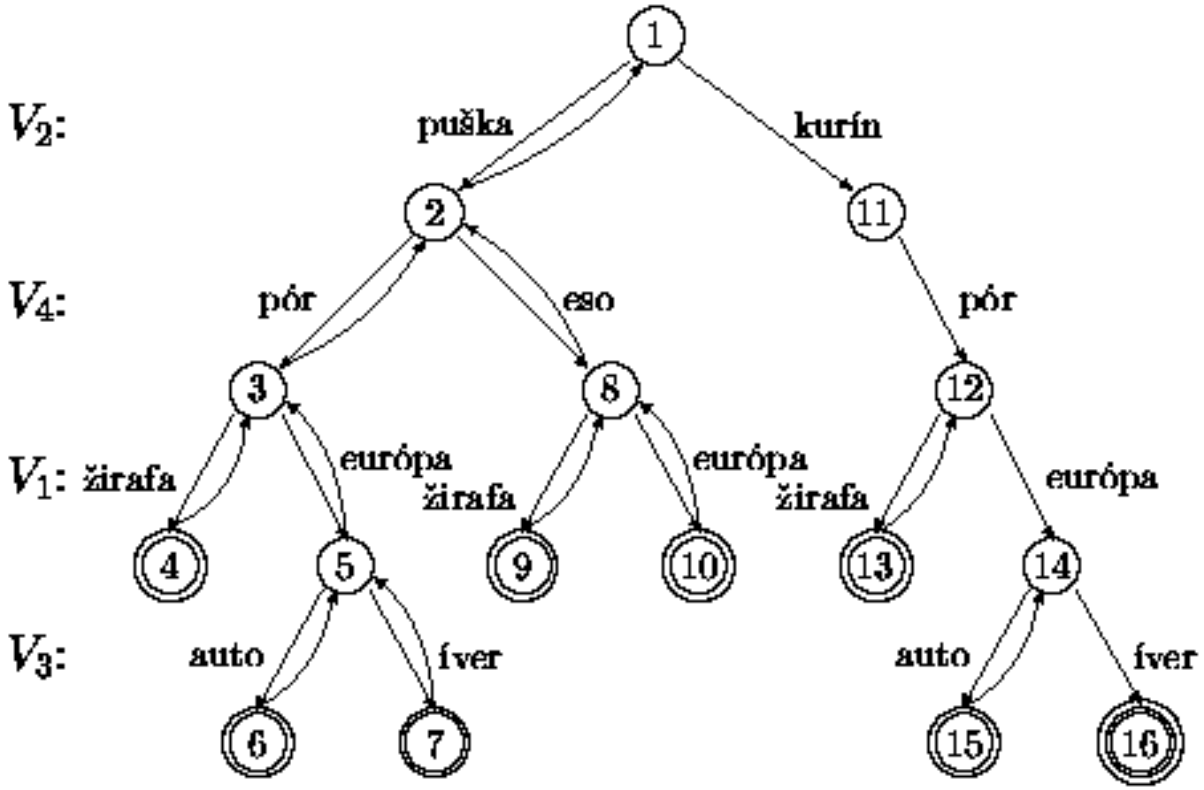


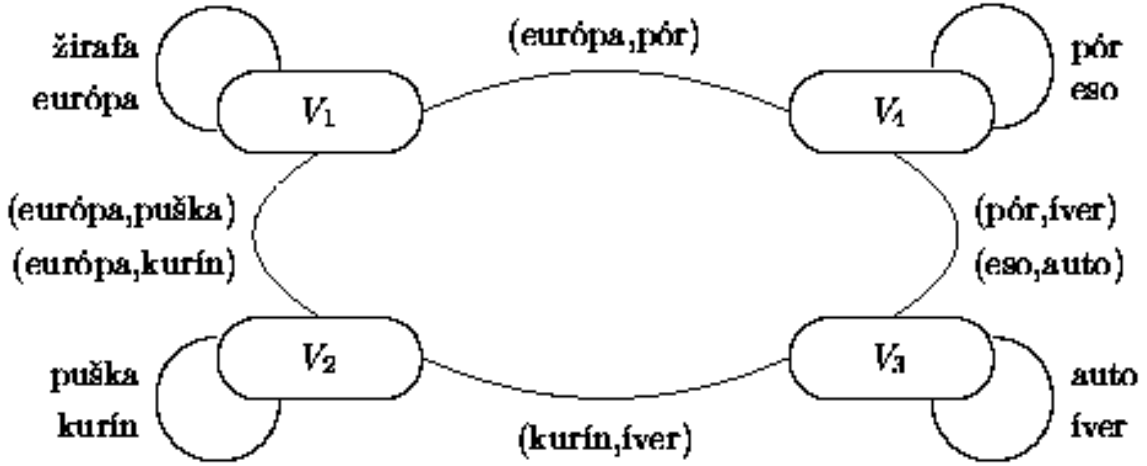
- Pre CSP s *n* premennými vo všeobecnosti iba silná *n*-konzistencia zabezpečí že následné prehľadávanie nemôže skončiť v slepej uličke!

Prehľadávacie algoritmy 1.

- Základným je algoritmus **spätného navracania** (BT - *backtracking*), ktorý vlastne realizuje slepé prehľadávanie P do hĺbky
  - Premenným sa priraďujú hodnoty sekvenčne
  - Každé ohraničenie sa testuje hneď ako je to možné (t.j. akonáhle sú priradené hodnoty všetkým premenným, nad ktorými je ohraničenie definované)
  - Ak je ohraničenie porušené, BT sa vráti k premennej, ktorej bola priradená hodnota naposledy a pokúsi sa jej priradiť novú hodnotu z jej domény
  - Zvykne dochádzať k „trashing“-u (opakovanie tej istej príčiny neúspechu v rôznych častiach priestoru prehľadávania)
- Lepšie (ale výpočtovo náročnejšie) prehľadávacie algoritmy sú *spätné navracanie riadené závislosťami*, *algoritmus spätného skoku* a mnohé ďalšie

- Strom prehľadávania generovaný algoritmom BT pre dané poradie premenných (V_2, V_4, V_1, V_3) a usporiadanie hodnôt (viď. redukovaná sieť ohraničení dole)



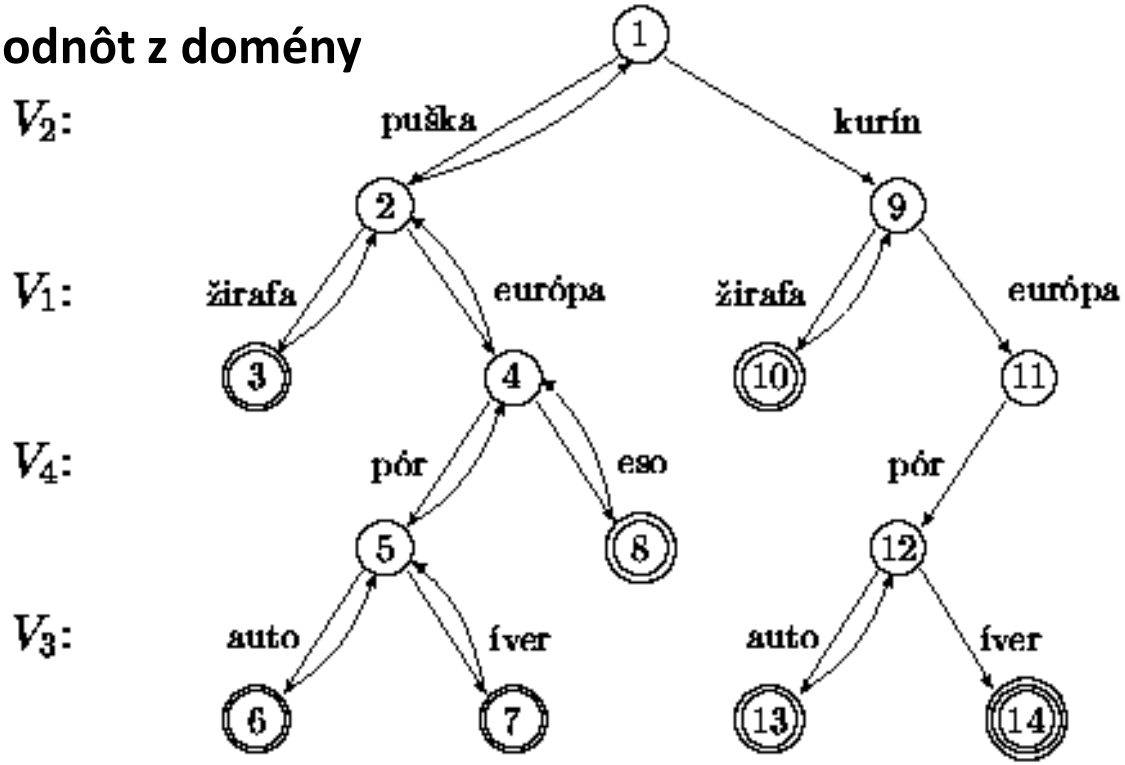


- Trashing možno pozorovať v uzloch 6 a 15: tá istá príčina neúspechu – $C^{3,4}$ nekompatibilita slov pór a auto

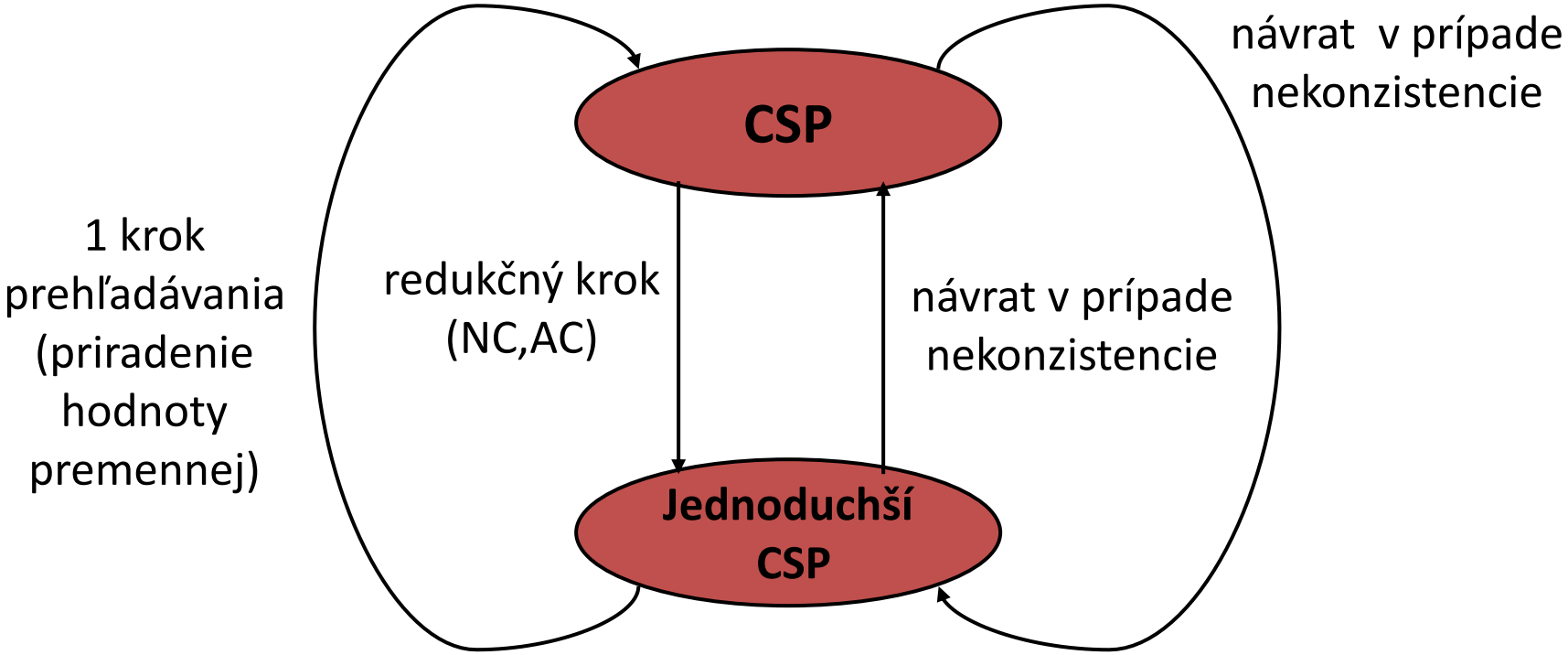
Prehľadávacie algoritmy 3.

- Efektívnosť procesu prehľadávania je daná nielen použitým algoritmom prehľadávania ale aj:
  - **Usporiadaním premenných** (viď. napr. strom prehľadávania algoritmu BT pre príklad 1 so zmeneným poradím premenných na V_2, V_1, V_4, V_3)
  - **Poradie priradzovania hodnôt z domény**

(napr. ak zmeníme poradie hodnôt v D_1 a D_3 ušetríme ďalšie dva uzly v strome prehľadávania, a síce uzly s číslami 10 a 13)



Kombinované algoritmy 1.

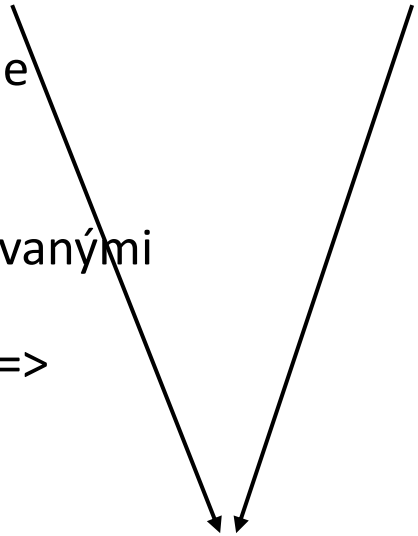


Logické programovanie

- Deklaratívne programovanie
- Logické premenné
- Unifikácia => syntaktická manipulácia s neinterpretovanými štruktúrami
- Prehľadávacia stratégia BT => „generuj a testuj“

Spĺňanie ohraničení

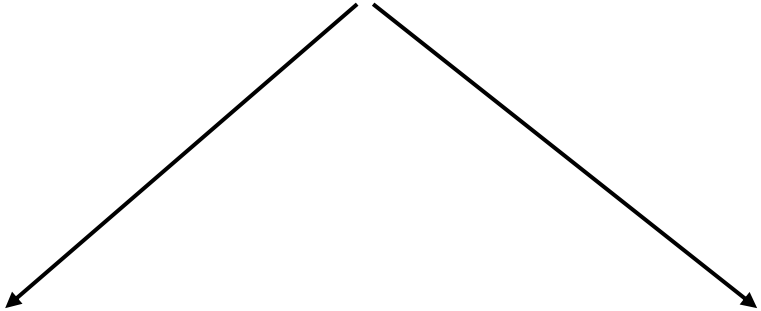
- Deklaratívna formulácia úlohy ako:
  - Premenné s počiatočnými doménami
  - Ohraničenia
- Redukčné algoritmy (určitý stupeň konzistencie)



Logické programovanie ohraničení (CLP)

- Deklaratívne programovanie
- Premenné s atribútmi => rozšírená unifikácia
- Sémantické objekty (kvôli špecifickým typom domén)
- Kombinovaná stratégia prehľadávania => „ohranič a generuj“
- Integrované optimalizačné algoritmy
- 2 rôzne technológie

Logické programovanie ohraničení



Konečné domény

- Neúplné redukčné algoritmy
- Samostatný algoritmus pre každý typ ohraničení (lokálna propagácia)
- Optimalizácia metódou „vetvenia a medzí“

Nekonečné domény

- Úplné redukčné algoritmy
- Jeden algoritmus pre celú množinu ohraničení
- Optimalizácia použitím simplexového algoritmu

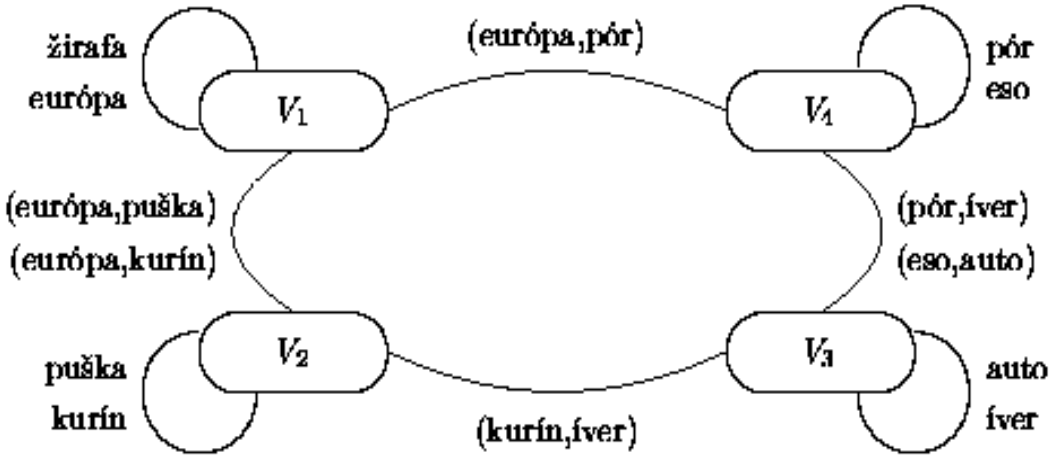
Jazyk ECLiPSe

- ECLiPSe (ECLiPSe Common Logic Programming System) je logický programovací jazyk (rozšírený Prolog) ktorého cieľom je poskytovať platformu pre integráciu rôznych rozšírení logického programovania, najmä logické programovanie ohraničení (CLP)
- Ďalšie informácie: http://eclipseclp.org/
- Príklady na nasledujúcich stranách boli naprogramované v ECLiPSe verzii 5.3

Príklad 1 – riešenie v jazyku ECLiPSe

```
:- lib(fd).

doplňovačka(Zoznam) :-
        % PREMENNÉ
    Zoznam = [V1,V2,V3,V4],
        % DOMÉNY
    V1 :: [žirafa,európa],
    V2 :: [puška,kurín],
    V3 :: [auto,íver],
    V4 :: [pór,eso],
        % OHRANIČENIA
    V1 #= európa #<=> (V2 #= puška #\/ V2 #= kurín), %C12
    V2 #= kurín #<=> V3 #= íver,                     %C23
    V1 #= európa #<=> V4 #= pór,                     %C14
    ((V3 #= íver #<=> V4 #= pór) #\/
     (V3 #= auto #<=> V4 #= eso)) ,                  %C34
        % PREHĽADÁVANIE
    labeling(Zoznam).
```



Implementuje metódu vetvenia a medzí

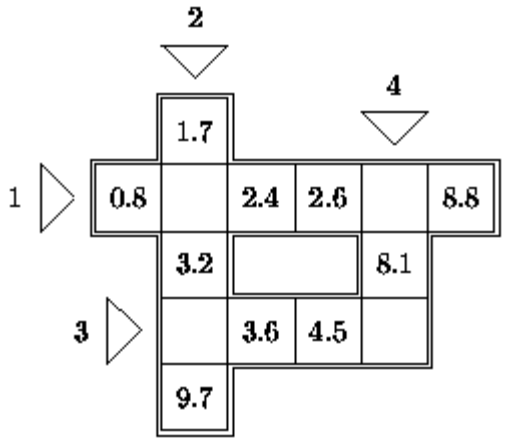
Príklad 1 – výsledok

```
?- doplňovačka(Z).
Z = [európa, kurín, íver, pór]
Yes (0.00s cpu)
```

Príklad 2 – riešenie v jazyku ECLiPSe

```
:- lib(clpr).

numerická_doplňovačka(Zoznam) :-
        % PREMENNÉ
    Zoznam = [V12,V14,V23,V34],
        % OHRANIČENIA
    {
     1.7 < V12, V12 < 2.4, % C^12
     2.6 < V14, V14 < 8.1, % C^14
     3.2 < V23, V23 < 3.6, % C^23
     8.1 < V34,            % C^34
     0.8 + V12 + 2.4 + 2.6 + V14 + 8.8 = 20, % C^12,14
     1.7 + V12 + 3.2 + V23 + 9.7 = 20,       % C^12,23
     V23 + 3.6 + 4.5 + V34 = 20,             % C^23,34
     V14 + 8.1 + V34 = 20                    % C^14,34
    }.
```



Príklad 2 – výsledky

```
?- numerická_doplňovačka(Zoznam).
Zoznam = [V12, V14, V23, V34]
Yes (0.00s cpu)

% Linear constraints:
{
  V12 < 2.2,
  V14 = 5.3999999999999986 - V12,
  V23 = 5.4 - V12,
  V34 = 6.5 + V12,
  V12 > 1.8000000000000003
}
```

To znamená, že existuje nekonečne veľa riešení, V12 je nezávislá premenná a musí byť z intervalu (1.8, 2.2). Nech napr. V12 = 2.0

```
?- numerická_doplňovačka([V12,
   V14, V23, V34]), V12 = 2.0.
V14 = 3.3999999999999986
V23 = 3.4000000000000004
V34 = 8.5
V12 = 2.0
Yes (0.00s cpu)
```

Príklad 2 – riešenie s absolútnou presnosťou

```
:- lib(clpq).

numerická_doplňovačka(Zoznam) :-
        % PREMENÉ
    Zoznam = [V12,V14,V23,V34],
        % OHRANIČENIA
      {
       4/5 < V12, V12 < 12/5, 17/10 < V12, V12 < 16/5,  % C^12
        13/5 < V14, V14 < 44/5, V14 < 81/10,            % C^14
        V23 > 16/5, V23 < 18/5, V23 < 97/10,            % C^23
        V34 > 9/2, V34 > 81/10,                         % C^34
        4/5 + V12 + 12/5 + 13/5 + V14 + 44/5 = 20,      % C^12,14
        17/10 + V12 + 16/5 + V23 + 97/10 = 20,          % C^12,23
        V23 + 18/5 + 9/2 + V34 = 20,                    % C^23,34
        V14 + 81/10 + V34 = 20                          % C^14,34
      }.
```

Príklad 2 – výsledky pre racionálne domény

```
?- numerická_doplňovačka(Zoznam).
Zoznam = [V12, V14, V23, V34]
Yes (0.00s cpu)

% Linear constraints:
{
  V12 < 11 / 5,
  V14 = 27 / 5 - V12,
  V23 = 27 / 5 - V12,
  V34 = 13 / 2 + V12,
  V12 > 9 / 5
}
```

znamená, že existuje nekonečne veľa riešení, V12 je nezávislá premenná a musí byť z intervalu (9/5, 11/5), nech napr. V12 = 2

```
?- numerická_doplňovačka([2, V14,
   V23, V34]).
V14 = 17 / 5
V23 = 17 / 5
V34 = 17 / 2
Yes (0.01s cpu)
```

Príklad 2 – optimalizačná verzia (1)

```
:- lib(clpr).

numerická_doplňovačka(Zoznam, Vážený_súčet) :-
        % PREMENNÉ
    Zoznam = [V12,V14,V23,V34],
        % KRITERIÁLNA FUNKCIA
     Vážený_súčet = 5*V12 + 3*V14 + 2*V23 + V34,
        % OHRANIČENIA
     {
      1.7 < V12, V12 < 2.4, % C^12
      2.6 < V14, V14 < 8.1, % C^14
      3.2 < V23, V23 < 3.6, % C^23
      8.1 < V34,            % C^34
      0.8 + V12 + 2.4 + 2.6 + V14 + 8.8 = 20, % C^12,14
      1.7 + V12 + 3.2 + V23 + 9.7 = 20,       % C^12,23
      V23 + 3.6 + 4.5 + V34 = 20,             % C^23,34
      V14 + 8.1 + V34 = 20                    % C^14,34
     },
        % OPTIMALIZÁCIA
    maximize(Vážený_súčet).
```

Implementuje simplexový algoritmus

Príklad 2 – optimalizačná verzia (2)

```
?- numerická_doplňovačka(L,Suma).
No (0.00s cpu)
```

To znamená, že v tomto prípade riešenie neexistuje (v dôsledku ostrých nerovností), ale môžeme vypočítať supremum kriteriálnej funkcie

Namiesto **volania predikátu:**

`maximize(Vážený_súčet)`

použijeme volanie:

`sup(Vážený_súčet, Supremum)`

a výsledok bude**:**

```
?- numerická_doplňovačka(L, Sup).
L = [V12, V14, V23, V34]
Sup = 35.699999999999996
Yes (0.01s cpu)

% Linear constraints:
{
  V14 = 5.3999999999999986 - V12,
  V23 = 5.4 - V12,
  V34 = 6.5 + V12,
  V12 =< 2.2,
  V12 >= 1.8000000000000003
}
```

Príklad 3 – Investičná úloha

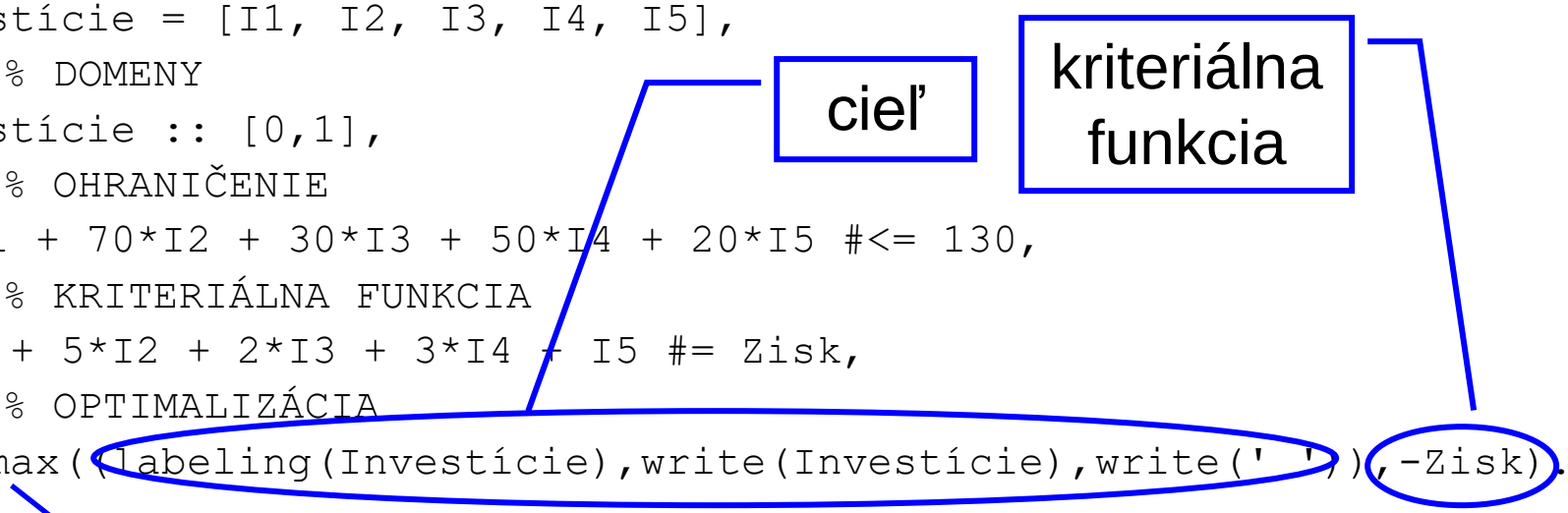
Spoločnosť by chcela zrealizovať päť rôznych investičných akcií (1 až 5) s danými nákladmi a očakávanými ziskami (viď. tabuľka) v priebehu nasledujúceho roka. Ale existuje limit investícií pre nasledujúci rok vo výške 130 peňažných jednotiek. Ktoré investičné akcie majú byť vybrané, aby sa maximalizoval ročný zisk?

Investičná akcia	1	2	3	4	5
Náklady	40	70	30	50	20
Očakávaný ročný zisk	4	5	2	3	1

Príklad 3 – riešenie v jazyku ECLiPSe

```
:- lib(fd).

investičný(Investície) :-
        % PREMENNÉ
    Investície = [I1, I2, I3, I4, I5],
        % DOMENY
    Investície :: [0,1],
        % OHRANIČENIE
    40*I1 + 70*I2 + 30*I3 + 50*I4 + 20*I5 #<= 130,
        % KRITERIÁLNA FUNKCIA
    4*I1 + 5*I2 + 2*I3 + 3*I4 + I5 #= Zisk,
        % OPTIMALIZÁCIA
    min_max(labeling(Investície),write(Investície),write(' ')),-Zisk).
```



Program je možné napísať aj všeobecne tak, aby riešil ľubovoľnú investičnú úlohu veľkosti N.

Príklad 3 – výsledky

```
?- investicny(Investicie).

[0, 0, 0, 0, 0] Found a solution with cost 0
[0, 0, 0, 0, 1] Found a solution with cost -1
[0, 0, 0, 1, 0] Found a solution with cost -3
[0, 0, 0, 1, 1] Found a solution with cost -4
[0, 0, 1, 1, 0] Found a solution with cost -5
[0, 0, 1, 1, 1] Found a solution with cost -6
[0, 1, 0, 1, 0] Found a solution with cost -8
[1, 0, 1, 1, 0] Found a solution with cost -9
[1, 1, 0, 0, 1] Found a solution with cost -10

Investicie = [1, 1, 0, 0, 1]
Yes (0.02s cpu)
```

Termíny skúšok z RaL

Vždy **UTOROK od 9:00 v miestnosti V4_V010**

- 2.5.2023 – Predtermín 1.
- 9.5.2023 – Predtermín 2.
- 16.5.2023 – RaL 3.
- 30.5.2023 – RaL 4.
- 6.6.2023 – RaL 5.
- 13.6.2023 – RaL 6.
- 27.6.2023 – RaL 7. – len opravné termíny