

# CLP( $FD$ ) - prohledávání

# Vestavěné predikáty pro labeling

- Instanciaci proměnné Variable hodnotami v její doméně

`indomain( Variable )`

hodnoty jsou instanciovány při backtrackingu ve vzrůstajícím pořadí

```
?- X in 4..5, indomain(X).
```

```
  X = 4 ? ;
```

```
  X = 5 ?
```

# Vestavěné predikáty pro labeling

- Instanciaci proměnné `Variable` hodnotami v její doméně

`indomain( Variable )`

hodnoty jsou instanciovány při backtrackingu ve vzrůstajícím pořadí

```
?- X in 4..5, indomain(X).
```

```
    X = 4 ? ;
```

```
    X = 5 ?
```

```
labeling( [] ).
```

```
labeling( [Var|Rest] ) :- % výběr nejlevější proměnné k instanciaci
    indomain( Var ),      % výběr hodnot ve vzrůstajícím pořadí
    labeling( Rest ).
```

# Vestavěné predikáty pro labeling

- Instanciaci proměnné `Variable` hodnotami v její doméně

`indomain( Variable )`

hodnoty jsou instanciovány při backtrackingu ve vzrůstajícím pořadí

```
?- X in 4..5, indomain(X).
```

```
    X = 4 ? ;
```

```
    X = 5 ?
```

```
labeling( [] ).
```

```
labeling( [Var|Rest] ) :-      % výběr nejlevější proměnné k instanciaci
    indomain( Var ),          % výběr hodnot ve vzrůstajícím pořadí
    labeling( Rest ).
```

- `labeling( Options, Variables )`

```
?- A in 0..2, B in 0..2, B#< A, labeling([], [A,B]).
```

# Uspořádání hodnot a proměnných

- Při prohledávání je rozhodující **uspořádání hodnot a proměnných**
- Určují je **heuristiky výběru hodnot a výběru proměnných**

```
labeling( [] ).
```

```
labeling( Variables ) :-  
    select_variable(Variables,Var,Rest),  
    select_value(Var,Value),
```

# Uspořádání hodnot a proměnných

- Při prohledávání je rozhodující **uspořádání hodnot a proměnných**
- Určují je **heuristiky výběru hodnot a výběru proměnných**

```
labeling( [] ).
```

```
labeling( Variables ) :-  
    select_variable(Variables,Var,Rest),  
    select_value(Var,Value),  
    ( Var #= Value,  
      labeling( Rest )
```

# Uspořádání hodnot a proměnných

- Při prohledávání je rozhodující **uspořádání hodnot a proměnných**
- Určují je **heuristiky výběru hodnot a výběru proměnných**

```
labeling( [] ).
```

```
labeling( Variables ) :-
```

```
    select_variable(Variables,Var,Rest),
```

```
    select_value(Var,Value),
```

```
    ( Var #= Value,
```

```
      labeling( Rest )
```

```
    ;
```

```
    Var #\= Value ,           % nemusí dojít k instanciaci Var
```

```
    labeling( Variables ) % proto pokračujeme se všemi proměnnými včetně Var
```

```
).
```

# Uspořádání hodnot a proměnných

- Při prohledávání je rozhodující **uspořádání hodnot a proměnných**
- Určují je **heuristiky výběru hodnot a výběru proměnných**

```
Labeling( [] ).
```

```
Labeling( Variables ) :-
```

```
    select_variable(Variables,Var,Rest),
```

```
    select_value(Var,Value),
```

```
    ( Var #= Value,
```

```
      Labeling( Rest )
```

```
    ;
```

```
    Var #\= Value ,           % nemusí dojít k instanciaci Var
```

```
    Labeling( Variables ) % proto pokračujeme se všemi proměnnými včetně Var
```

```
    ).
```

- **Statické uspořádání:** určeno už před prohledáváním
- **Dynamické uspořádání:** počítá se během prohledávání



# Výběr hodnoty

- Obecný princip výběru hodnoty: **první úspěch** (*succeed first*)
  - volíme pořadí tak, abychom výběr nemuseli opakovat
  - ?- `domain([A,B,C],1,2), A#B+C.`

# Výběr hodnoty

- Obecný princip výběru hodnoty: **první úspěch** (*succeed first*)
  - volíme pořadí tak, abychom výběr nemuseli opakovat
  - ?- `domain([A,B,C],1,2), A#B+C`. optimální výběr  $A=2, B=1, C=1$  je bez backtrackingu



# Výběr hodnoty

- Obecný princip výběru hodnoty: **první úspěch** (*succeed first*)
  - volíme pořadí tak, abychom výběr nemuseli opakovat
  - ?- `domain([A,B,C],1,2), A#=#B+C`. optimální výběr  $A=2, B=1, C=1$  je bez backtrackingu
- Parametry `Labeling/2` ovlivňující výběr hodnoty    př. `labeling([down], Vars)`
  - **up**: doména procházena ve vzrůstajícím pořadí (default)
  - **down**: doména procházena v klesajícím pořadí
- Parametry `Labeling/2` řídící, jak je výběr hodnoty realizován
  - **step**: volba mezi  $X \# = M$ ,  $X \# \setminus = M$  (default)
    - viz dřívější příklad u "Uspořádání hodnot a proměnných"
  - **enum**: vícenásobná volba mezi všemi hodnotami v doméně
    - podobně jako při `indomain/1`
  - **bisect**: volba mezi  $X \# = < Mid$ ,  $X \# > Mid$ 
    - v jednom kroku `labelingu` nedochází nutně k instanciaci proměnné

# Výběr proměnné

- Obecný princip výběru proměnné: *first-fail*
  - výběr proměnné, pro kterou je nejobtížnější nalézt správnou hodnotu  
pozdější výběr hodnoty pro tuto proměnnou by snadněji vedl k failu
  - vybereme proměnnou **s nejmenší doménou**
  - ?- `domain([A,B,C],1,3), A#<3, A#=B+C.`

# Výběr proměnné

- Obecný princip výběru proměnné: *first-fail*
  - výběr proměnné, pro kterou je nejobtížnější nalézt správnou hodnotu  
pozdější výběr hodnoty pro tuto proměnnou by snadněji vedl k failu
  - vybereme proměnnou **s nejmenší doménou**
  - ?-  $\text{domain}([A,B,C],1,3), A \neq 3, A \neq B+C.$  nejlépe je začít s výběrem A

# Výběr proměnné

## ● Obecný princip výběru proměnné: *first-fail*

- výběr proměnné, pro kterou je nejobtížnější nalézt správnou hodnotu  
pozdější výběr hodnoty pro tuto proměnnou by snadněji vedl k failu
- vybereme proměnnou **s nejmenší doménou**

● ?- domain([A,B,C],1,3), A#<3, A#=#B+C.

nejlépe je začít s výběrem A

## ● Parametry Labeling/2 ovlivňující výběr proměnné

● **leftmost**: nejlevější

(default)

- **ff**: s (a) nejmenší velikostí domény  
(b) nejlevější z nich

fd\_size(Var,Size)

# Výběr proměnné

## ● Obecný princip výběru proměnné: *first-fail*

- výběr proměnné, pro kterou je nejobtížnější nalézt správnou hodnotu  
pozdější výběr hodnoty pro tuto proměnnou by snadněji vedl k failu
- vybereme proměnnou **s nejmenší doménou**

● ?- domain([A,B,C],1,3), A#<3, A#=#B+C. nejlépe je začít s výběrem A

## ● Parametry Labeling/2 ovlivňující výběr proměnné

- **leftmost**: nejlevější (default)
- **ff**: s (a) nejmenší velikostí domény fd\_size(Var,Size)  
(b) nejlevější z nich
- **ffc**: s (a) nejmenší velikostí domény fd\_degree(Var,Size)  
(b) největším množstvím omezením „čekajících“ na proměnné  
(c) nejlevější z nich
- **min/max**: s (a) nejmenší/největší hodnotou v doméně proměnné fd\_min(Var,Min)/fd\_max(Var,Max)  
(b) nejlevnější z nich



# Hledání optimálního řešení

(předpokládejme minimalizaci)

- Parametry Labeling/2 pro optimalizaci: `minimize(F)/maximize(F)`
  - Cena  $\# = A+B+C$ , `labeling([minimize(Cena)], [A,B,C])`
- **Metoda větví a mezí (*branch&bound*)**
  - uvažujeme nejhorší možnou cenu řešení  $UB$  (např. cena už nalezeného řešení)
  - počítáme dolní odhad  $LB$  ceny částečného řešení  
 $LB$  je tedy nejlepší možná cena pro rozšíření tohoto řešení
  - procházíme strom a vyžadujeme, aby prozkoumávaná větev měla cenu  $LB < UB$   
pokud je  $LB \geq UB$ , tak víme, že v této větvi není lepší řešení a odřízneme ji

# Hledání optimálního řešení

(předpokládejme minimalizaci)

- Parametry Labeling/2 pro optimalizaci: `minimize(F)/maximize(F)`
  - Cena  $\# = A+B+C$ , `labeling([minimize(Cena)], [A,B,C])`
- **Metoda větví a mezí (*branch&bound*)**
  - uvažujeme nejhorší možnou cenu řešení  $UB$  (např. cena už nalezeného řešení)
  - počítáme dolní odhad  $LB$  ceny částečného řešení  
 $LB$  je tedy nejlepší možná cena pro rozšíření tohoto řešení
  - procházíme strom a vyžadujeme, aby prozkoumávaná větev měla cenu  $LB < UB$   
pokud je  $LB \geq UB$ , tak víme, že v této větvi není lepší řešení a odřízneme ji

# Algoritmy pro řešení problému splňování podmínek (CSP)

# Grafová reprezentace CSP

## ● Reprezentace podmínek

- intenzionální (matematická/logická formule)
- extenzionální (výčet k-tic kompatibilních hodnot, 0-1 matice)

# Grafová reprezentace CSP

## ● Reprezentace podmínek

- intenzionální (matematická/logická formule)
- extenzionální (výčet k-tic kompatibilních hodnot, 0-1 matice)

## ● **Graf:** vrcholy, hrany (hrana spojuje dva vrcholy)

## ● **Hypergraf:** vrcholy, hrany (hrana spojuje množinu vrcholů)

## ● Reprezentace CSP pomocí **hypergrafu podmínek**

- vrchol = proměnná, hyperhrana = podmínka

# Grafová reprezentace CSP

## ● Reprezentace podmínek

- intenzionální (matematická/logická formule)
- extenzionální (výčet k-tic kompatibilních hodnot, 0-1 matice)

## ● Graf: vrcholy, hrany (hrana spojuje dva vrcholy)

## ● Hypergraf: vrcholy, hrany (hrana spojuje množinu vrcholů)

## ● Reprezentace CSP pomocí hypergrafu podmínek

- vrchol = proměnná, hyperhrana = podmínka

## ● Příklad

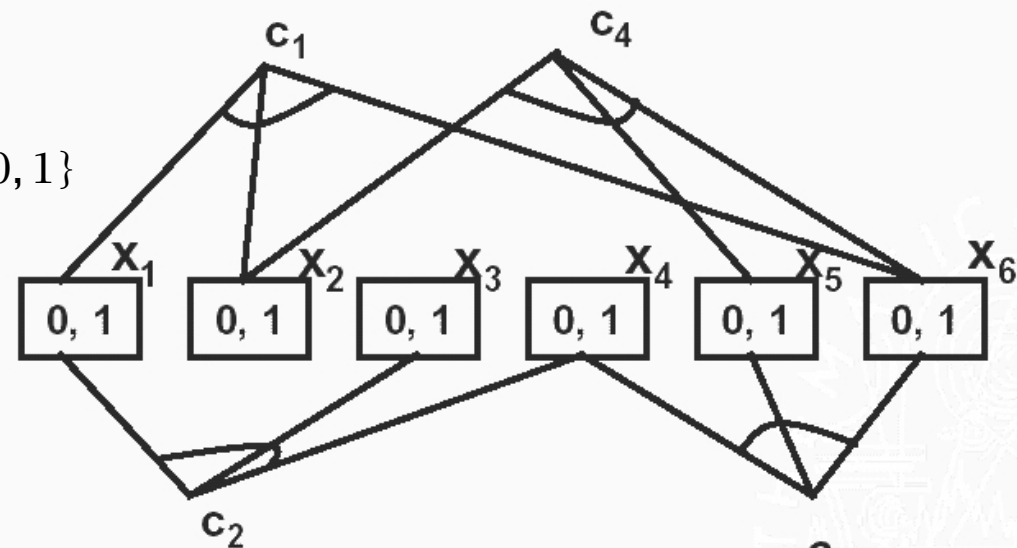
- proměnné  $x_1, \dots, x_6$  s doménou  $\{0, 1\}$

- omezení  $c_1 : x_1 + x_2 + x_6 = 1$

$$c_2 : x_1 - x_3 + x_4 = 1$$

$$c_3 : x_4 + x_5 - x_6 > 0$$

$$c_4 : x_2 + x_5 - x_6 = 0$$



# Binární CSP

## ● Binární CSP

- CSP, ve kterém jsou pouze binární podmínky
- unární podmínky zakódovány do domény proměnné

## ● Graf podmínek pro binární CSP

- není nutné uvažovat hypergraf, stačí graf (podmínka spojuje pouze dva vrcholy)

# Binární CSP

## ● Binární CSP

- CSP, ve kterém jsou pouze binární podmínky
- unární podmínky zakódovány do domény proměnné

## ● Graf podmínek pro binární CSP

- není nutné uvažovat hypergraf, stačí graf (podmínka spojuje pouze dva vrcholy)

## ● Každý CSP lze transformovat na "korespondující" binární CSP

## ● Výhody a nevýhody binarizace

- získáváme unifikovaný tvar CSP problému, řada algoritmů navržena pro binární CSP
- bohužel ale značné zvětšení velikosti problému



# Binární CSP

## ● Binární CSP

- CSP, ve kterém jsou pouze binární podmínky
- unární podmínky zakódovány do domény proměnné

## ● Graf podmínek pro binární CSP

- není nutné uvažovat hypergraf, stačí graf (podmínka spojuje pouze dva vrcholy)

## ● Každý CSP lze transformovat na "korespondující" binární CSP

## ● Výhody a nevýhody binarizace

- získáváme unifikovaný tvar CSP problému, řada algoritmů navržena pro binární CSP
- bohužel ale značné zvětšení velikosti problému

## ● Nebinární podmínky

- složitější propagační algoritmy
- lze využít jejich sémantiky pro lepší propagaci
  - příklad: `all_different` vs. množina binárních nerovností

# Vrcholová a hranová konzistence

## ● Vrcholová konzistence (*node consistency*) NC

- každá hodnota z aktuální domény  $V_i$  proměnné splňuje všechny unární podmínky s proměnnou  $V_i$

# Vrcholová a hranová konzistence

## ● Vrcholová konzistence (*node consistency*) NC

- každá hodnota z aktuální domény  $V_i$  proměnné splňuje všechny unární podmínky s proměnnou  $V_i$

## ● Hranová konzistence (*arc consistency*) AC pro **binární CSP**

- **hrana**  $(V_i, V_j)$  **je hranově konzistentní**, právě když pro každou hodnotu  $x$  z aktuální domény  $D_i$  existuje hodnota  $y$  tak, že ohodnocení  $[V_i = x, V_j = y]$  splňuje všechny binární podmínky nad  $V_i, V_j$ .

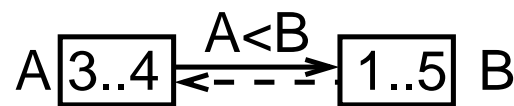
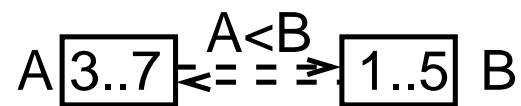
# Vrcholová a hranová konzistence

## ● Vrcholová konzistence (*node consistency*) NC

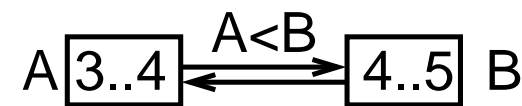
- každá hodnota z aktuální domény  $V_i$  proměnné splňuje všechny unární podmínky s proměnnou  $V_i$

## ● Hranová konzistence (*arc consistency*) AC pro **binární CSP**

- **hrana**  $(V_i, V_j)$  je **hranově konzistentní**, právě když pro každou hodnotu  $x$  z aktuální domény  $D_i$  existuje hodnota  $y$  tak, že ohodnocení  $[V_i = x, V_j = y]$  splňuje všechny binární podmínky nad  $V_i, V_j$ .
- hranová konzistence je **směrová**
  - konzistence hrany  $(V_i, V_j)$  nezaručuje konzistenci hrany  $(V_j, V_i)$



konzistence (A,B)



konzistence (A,B) i (B,A)

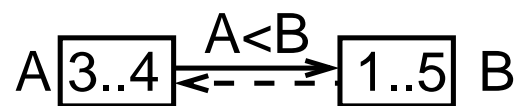
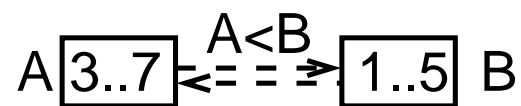
# Vrcholová a hranová konzistence

## ● Vrcholová konzistence (*node consistency*) NC

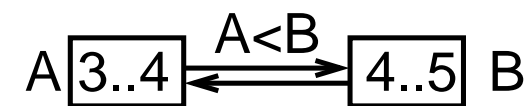
- každá hodnota z aktuální domény  $V_i$  proměnné splňuje všechny unární podmínky s proměnnou  $V_i$

## ● Hranová konzistence (*arc consistency*) AC pro **binární CSP**

- **hrana**  $(V_i, V_j)$  je **hranově konzistentní**, právě když pro každou hodnotu  $x$  z aktuální domény  $D_i$  existuje hodnota  $y$  tak, že ohodnocení  $[V_i = x, V_j = y]$  splňuje všechny binární podmínky nad  $V_i, V_j$ .
- hranová konzistence je **směrová**
  - konzistence hrany  $(V_i, V_j)$  nezaručuje konzistenci hrany  $(V_j, V_i)$



konzistence (A,B)



konzistence (A,B) i (B,A)

- **CSP** je **hranově konzistentní**, právě když

jsou všechny jeho hrany (v obou směrech) hranově konzistentní

# Je hranová konzistence dostatečná?

- Použitím AC odstraníme mnoho nekompatibilních hodnot
  - Dostaneme potom řešení problému? NE
  - Víme alespoň zda řešení existuje? NE

# Je hranová konzistence dostatečná?

- Použitím AC odstraníme mnoho nekompatibilních hodnot
  - Dostaneme potom řešení problému? NE
  - Víme alespoň zda řešení existuje? NE
- $\text{domain}([X, Y, Z], 1, 2), X \neq Y, Y \neq Z, Z \neq X$ 
  - hranově konzistentní
  - nemá žádné řešení

# Je hranová konzistence dostatečná?

- Použitím AC odstraníme mnoho nekompatibilních hodnot
  - Dostaneme potom řešení problému? NE
  - Víme alespoň zda řešení existuje? NE
- $\text{domain}([X,Y,Z],1,2)$ ,  $X \neq Y$ ,  $Y \neq Z$ ,  $Z \neq X$ 
  - hranově konzistentní
  - nemá žádné řešení
- Jaký je tedy význam AC?
  - někdy dá řešení přímo
    - nějaká doména se vyprázdní  $\Rightarrow$  řešení neexistuje
    - všechny domény jsou jednoprvkové  $\Rightarrow$  máme řešení
  - v obecném případě se alespoň zmenší prohledávaný prostor



# Řešení nebinárních podmínek

- S  $n$ -árními podmínkami se pracuje přímo
- Podmínka je **obecně hranově konzistentní (GAC)**, právě když pro každou proměnnou  $V_i$  z této podmínky a každou hodnou  $x \in D_i$  existuje ohodnocení zbylých proměnných v podmínce tak, že podmínka platí
  - $A + B = C$ ,  $A \text{ in } 1..3$ ,  $B \text{ in } 2..4$ ,  $C \text{ in } 3..7$  je obecně hranově konzistentní

# Řešení nebinárních podmínek

- S n-árními podmínkami se pracuje přímo
- Podmínka je **obecně hranově konzistentní (GAC)**, právě když pro každou proměnnou  $V_i$  z této podmínky a každou hodnou  $x \in D_i$  existuje ohodnocení zbylých proměnných v podmínce tak, že podmínka platí
  - $A + B = C$ ,  $A \in 1..3$ ,  $B \in 2..4$ ,  $C \in 3..7$  je obecně hranově konzistentní
- Využívá se sémantika podmínek
  - speciální typy konzistence pro globální omezení
    - viz `all_distinct`
  - konzistence mezí
    - propagace pouze při změně nejmenší a největší hodnoty v doméně proměnné
- Pro různé podmínky lze použít různý druh konzistence
  - $A \# < B$ : hranová konzistence, konzistence mezí

# Konzistenční algoritmus pro nebinární podmínky

- Algoritmus s **frontou proměnných** (někdy též nazýván AC-8)

- opakovaně se provádí revize podmínek, dokud se mění domény

```
procedure Nonbinary-AC-3-with-Variables(Q)
```

```
while Q non empty do
```

```
    vyber a smaž  $V_j \in Q$ 
```

```
    for  $\forall C$  takové, že  $V_j \in scope(C)$  do
```

```
         $W := revise(V_j, C)$ 
```

```
        //  $W$  je množina proměnných jejichž, doména se změnila
```

```
        if  $\exists V_i \in W$  taková, že  $D_i = \emptyset$  then return fail
```

```
         $Q := Q \cup \{W\}$ 
```

```
end Non-binary-consistency
```

- **rozsah omezení  $scope(C)$** : množina proměnných, na nichž je  $C$  definováno

# Konzistenční algoritmus pro nebinární podmínky

## ● Algoritmus s **frontou proměnných** (někdy též nazýván AC-8)

- opakovaně se provádí revize podmínek, dokud se mění domény

```
procedure Nonbinary-AC-3-with-Variables(Q)
```

```
while Q non empty do
```

```
    vyber a smaž  $V_j \in Q$ 
```

```
    for  $\forall C$  takové, že  $V_j \in scope(C)$  do
```

```
         $W := revise(V_j, C)$ 
```

```
        //  $W$  je množina proměnných jejichž, doména se změnila
```

```
        if  $\exists V_i \in W$  taková, že  $D_i = \emptyset$  then return fail
```

```
         $Q := Q \cup \{W\}$ 
```

```
    end Non-binary-consistency
```

- **rozsah omezení**  $scope(C)$ : množina proměnných, na nichž je  $C$  definováno

## ● Implementace

- u každé proměnné je seznam **vybraných podmínek** pro propagaci

- REVISE procedury pro tyto podmínky definuje uživatel v závislosti na typu podmínky

# Revize podmínky pro hranovou konzistenci

- Jak udělat podmínku  $c(V_j, V_i)$  na hraně  $(V_j, V_i)$  hranově konzistentní vůči  $V_j$ ?
- Z domény  $D_j$  vyřadím takové hodnoty  $x$ , které nejsou konzistentní s aktuální doménou  $D_i$  (pro  $x$  neexistuje žádná hodnota  $y$  v  $D_i$  tak, aby ohodnocení  $V_j = x$  a  $V_i = y$  splňovalo binární podmínku  $c(V_j, V_i)$  mezi  $V_j$  a  $V_i$ )

# Revize podmínky pro hranovou konzistenci

- Jak udělat podmínku  $c(V_j, V_i)$  na hraně  $(V_j, V_i)$  hranově konzistentní vůči  $V_j$ ?
- Z domény  $D_j$  vyřadím takové hodnoty  $x$ , které nejsou konzistentní s aktuální doménou  $D_i$  (pro  $x$  neexistuje žádná hodnota  $y$  v  $D_i$  tak, aby ohodnocení  $V_j = x$  a  $V_i = y$  splňovalo binární podmínku  $c(V_j, V_i)$  mezi  $V_j$  a  $V_i$ )

● `procedure revise( $V_j, c(V_j, V_i)$ )`

`Deleted := false`

`for  $\forall x$  in  $D_j$  do`

`if neexistuje  $y \in D_i$  takové, že  $(x, y)$  je konzistentní`

`then  $D_j := D_j - \{x\}$`

`Deleted := true`

`end if`

`return Deleted`

`end revise`

# Revize podmínky pro hranovou konzistenci

- Jak udělat podmínku  $c(V_j, V_i)$  na hraně  $(V_j, V_i)$  hranově konzistentní vůči  $V_j$ ?
- Z domény  $D_j$  vyřadím takové hodnoty  $x$ , které nejsou konzistentní s aktuální doménou  $D_i$  (pro  $x$  neexistuje žádná hodnota  $y$  v  $D_i$  tak, aby ohodnocení  $V_j = x$  a  $V_i = y$  splňovalo binární podmínku  $c(V_j, V_i)$  mezi  $V_j$  a  $V_i$ )
- `procedure revise( $V_j, c(V_j, V_i)$ )`  
Deleted := false  
for  $\forall x$  in  $D_j$  do  
    if neexistuje  $y \in D_i$  takové, že  $(x, y)$  je konzistentní  
    then  $D_j := D_j - \{x\}$   
        Deleted := true  
    end if  
return Deleted  
end revise
- `domain([ $V_1, V_2$ ], 2, 4),  $V_1 \# < V_2$`

# Revize podmínky pro hranovou konzistenci

- Jak udělat podmínku  $c(V_j, V_i)$  na hraně  $(V_j, V_i)$  hranově konzistentní vůči  $V_j$ ?
- Z domény  $D_j$  vyřadím takové hodnoty  $x$ , které nejsou konzistentní s aktuální doménou  $D_i$  (pro  $x$  neexistuje žádná hodnota  $y$  v  $D_i$  tak, aby ohodnocení  $V_j = x$  a  $V_i = y$  splňovalo binární podmínku  $c(V_j, V_i)$  mezi  $V_j$  a  $V_i$ )

● `procedure revise( $V_j, c(V_j, V_i)$ )`

`Deleted := false`

`for  $\forall x$  in  $D_j$  do`

`if neexistuje  $y \in D_i$  takové, že  $(x, y)$  je konzistentní`

`then  $D_j := D_j - \{x\}$`

`Deleted := true`

`end if`

`return Deleted`

`end revise`

● `domain([ $V_1, V_2$ ], 2, 4),  $V_1 \# < V_2$     revise( $V_1, V_1 \# < V_2$ ) smaže 4 z  $D_1$ ,`



# Revize podmínky pro hranovou konzistenci

- Jak udělat podmínku  $c(V_j, V_i)$  na hraně  $(V_j, V_i)$  hranově konzistentní vůči  $V_j$ ?
- Z domény  $D_j$  vyřadím takové hodnoty  $x$ , které nejsou konzistentní s aktuální doménou  $D_i$  (pro  $x$  neexistuje žádná hodnota  $y$  v  $D_i$  tak, aby ohodnocení  $V_j = x$  a  $V_i = y$  splňovalo binární podmínku  $c(V_j, V_i)$  mezi  $V_j$  a  $V_i$ )

● `procedure revise( $V_j, c(V_j, V_i)$ )`

`Deleted := false`

`for  $\forall x$  in  $D_j$  do`

`if neexistuje  $y \in D_i$  takové, že  $(x, y)$  je konzistentní`

`then  $D_j := D_j - \{x\}$`

`Deleted := true`

`end if`

`return Deleted`

`end revise`

- `domain([ $V_1, V_2$ ], 2, 4),  $V_1 \# < V_2$       revise( $V_1, V_1 \# < V_2$ ) smaže 4 z  $D_1, D_2$  se nezmění`

# Konzistence mezí

- **Bounds consistency BC**: slabší než obecná hranová konzistence
  - podmínka má **konzistentní meze (BC)**, právě když pro každou proměnnou  $V_j$  z této podmínky a každou hodnou  $x \in D_j$  existuje ohodnocení zbylých proměnných v podmínce tak, že je podmínka splněna a pro vybrané ohodnocení  $y_i$  proměnné  $V_i$  platí  $\min(D_i) \leq y_i \leq \max(D_i)$

# Konzistence mezí

- **Bounds consistency BC**: slabší než obecná hranová konzistence
  - podmínka má **konzistentní meze (BC)**, právě když pro každou proměnnou  $V_j$  z této podmínky a každou hodnou  $x \in D_j$  existuje ohodnocení zbylých proměnných v podmínce tak, že je podmínka splněna a pro vybrané ohodnocení  $y_i$  proměnné  $V_i$  platí  $\min(D_i) \leq y_i \leq \max(D_i)$
  - stačí propagace pouze při **změně minimální nebo maximální hodnoty (při změně mezí)** v doméně proměnné
- **Konzistence mezí pro nerovnice**
  - $A \#> B \Rightarrow \min(A) = \min(B)+1, \max(B) = \max(A)-1$
  - příklad:  $A \text{ in } 4..10, B \text{ in } 6..18, A \#> B$   
 $\min(A) = 6+1 \Rightarrow A \text{ in } 7..10$   
 $\max(B) = 10-1 \Rightarrow B \text{ in } 6..9$

# Konzistence mezí

- **Bounds consistency BC**: slabší než obecná hranová konzistence
  - podmínka má **konzistentní meze (BC)**, právě když pro každou proměnnou  $V_j$  z této podmínky a každou hodnou  $x \in D_j$  existuje ohodnocení zbylých proměnných v podmínce tak, že je podmínka splněna a pro vybrané ohodnocení  $y_i$  proměnné  $V_i$  platí  $\min(D_i) \leq y_i \leq \max(D_i)$
  - stačí propagace pouze při **změně minimální nebo maximální hodnoty (při změně mezí)** v doméně proměnné
- **Konzistence mezí pro nerovnice**
  - $A \#> B \Rightarrow \min(A) = \min(B)+1, \max(B) = \max(A)-1$
  - příklad:  $A \text{ in } 4..10, B \text{ in } 6..18, A \#> B$   
 $\min(A) = 6+1 \Rightarrow A \text{ in } 7..10$   
 $\max(B) = 10-1 \Rightarrow B \text{ in } 6..9$
  - podobně:  $A \#< B, A \#>= B, A \#=< B$

# Konzistence mezi a aritmetická omezení

●  $A \# B + C \Rightarrow \min(A) = \min(B) + \min(C), \max(A) = \max(B) + \max(C)$   
 $\min(B) = \min(A) - \max(C), \max(B) = \max(A) - \min(C)$   
 $\min(C) = \min(A) - \max(B), \max(C) = \max(A) - \min(B)$

# Konzistence mezi a aritmetická omezení

- $A \# = B + C \Rightarrow \min(A) = \min(B) + \min(C), \max(A) = \max(B) + \max(C)$   
 $\min(B) = \min(A) - \max(C), \max(B) = \max(A) - \min(C)$   
 $\min(C) = \min(A) - \max(B), \max(C) = \max(A) - \min(B)$
- změna  $\min(A)$  vyvolá pouze změnu  $\min(B)$  a  $\min(C)$
- změna  $\max(A)$  vyvolá pouze změnu  $\max(B)$  a  $\max(C)$ , ...

# Konzistence mezi a aritmetická omezení

●  $A \# = B + C \Rightarrow \min(A) = \min(B) + \min(C), \max(A) = \max(B) + \max(C)$   
 $\min(B) = \min(A) - \max(C), \max(B) = \max(A) - \min(C)$   
 $\min(C) = \min(A) - \max(B), \max(C) = \max(A) - \min(B)$

● změna  $\min(A)$  vyvolá pouze změnu  $\min(B)$  a  $\min(C)$

● změna  $\max(A)$  vyvolá pouze změnu  $\max(B)$  a  $\max(C)$ , ...

● Příklad:  $A \text{ in } 1..10, B \text{ in } 1..10, A \# = B + 2, A \# > 5, A \# \setminus = 8$

$A \# = B + 2 \Rightarrow$

# Konzistence mezi a aritmetická omezení

●  $A \# = B + C \Rightarrow \min(A) = \min(B) + \min(C), \max(A) = \max(B) + \max(C)$   
 $\min(B) = \min(A) - \max(C), \max(B) = \max(A) - \min(C)$   
 $\min(C) = \min(A) - \max(B), \max(C) = \max(A) - \min(B)$

● změna  $\min(A)$  vyvolá pouze změnu  $\min(B)$  a  $\min(C)$

● změna  $\max(A)$  vyvolá pouze změnu  $\max(B)$  a  $\max(C)$ , ...

● Příklad:  $A \text{ in } 1..10, B \text{ in } 1..10, A \# = B + 2, A \# > 5, A \# \setminus = 8$

$A \# = B + 2 \Rightarrow \min(A) = 1 + 2, \max(A) = 10 + 2 \Rightarrow A \text{ in } 3..10$

$\Rightarrow \min(B) = 1 - 2, \max(B) = 10 - 2 \Rightarrow B \text{ in } 1..8$



# Konzistence mezi a aritmetická omezení

●  $A \# = B + C \Rightarrow \min(A) = \min(B) + \min(C), \max(A) = \max(B) + \max(C)$   
 $\min(B) = \min(A) - \max(C), \max(B) = \max(A) - \min(C)$   
 $\min(C) = \min(A) - \max(B), \max(C) = \max(A) - \min(B)$

- změna  $\min(A)$  vyvolá pouze změnu  $\min(B)$  a  $\min(C)$
- změna  $\max(A)$  vyvolá pouze změnu  $\max(B)$  a  $\max(C)$ , ...

● Příklad:  $A \text{ in } 1..10, B \text{ in } 1..10, A \# = B + 2, A \# > 5, A \# \setminus = 8$

$A \# = B + 2 \Rightarrow \min(A) = 1 + 2, \max(A) = 10 + 2 \Rightarrow A \text{ in } 3..10$   
 $\Rightarrow \min(B) = 1 - 2, \max(B) = 10 - 2 \Rightarrow B \text{ in } 1..8$

$A \# > 5 \Rightarrow \min(A) = 6 \Rightarrow A \text{ in } 6..10$   
 $\Rightarrow \min(B) = 6 - 2 \Rightarrow B \text{ in } 4..8$

(nové vyvolání  $A \# = B + 2$ )

# Konzistence mezi a aritmetická omezení

●  $A \# = B + C \Rightarrow \min(A) = \min(B) + \min(C), \max(A) = \max(B) + \max(C)$   
 $\min(B) = \min(A) - \max(C), \max(B) = \max(A) - \min(C)$   
 $\min(C) = \min(A) - \max(B), \max(C) = \max(A) - \min(B)$

- změna  $\min(A)$  vyvolá pouze změnu  $\min(B)$  a  $\min(C)$
- změna  $\max(A)$  vyvolá pouze změnu  $\max(B)$  a  $\max(C)$ , ...

● Příklad:  $A \text{ in } 1..10, B \text{ in } 1..10, A \# = B + 2, A \# > 5, A \# \setminus = 8$

$A \# = B + 2 \Rightarrow \min(A) = 1 + 2, \max(A) = 10 + 2 \Rightarrow A \text{ in } 3..10$   
 $\Rightarrow \min(B) = 1 - 2, \max(B) = 10 - 2 \Rightarrow B \text{ in } 1..8$

$A \# > 5 \Rightarrow \min(A) = 6 \Rightarrow A \text{ in } 6..10$   
 $\Rightarrow \min(B) = 6 - 2 \Rightarrow B \text{ in } 4..8$

(nové vyvolání  $A \# = B + 2$ )

$A \# \setminus = 8 \Rightarrow A \text{ in } (6..7) \setminus (9..10)$  (meze stejné, k propagaci  $A \# = B + 2$  nedojde)

# Konzistence mezi a aritmetická omezení

●  $A \# = B + C \Rightarrow \min(A) = \min(B) + \min(C), \max(A) = \max(B) + \max(C)$   
 $\min(B) = \min(A) - \max(C), \max(B) = \max(A) - \min(C)$   
 $\min(C) = \min(A) - \max(B), \max(C) = \max(A) - \min(B)$

- změna  $\min(A)$  vyvolá pouze změnu  $\min(B)$  a  $\min(C)$
- změna  $\max(A)$  vyvolá pouze změnu  $\max(B)$  a  $\max(C)$ , ...

● Příklad:  $A \text{ in } 1..10, B \text{ in } 1..10, A \# = B + 2, A \# > 5, A \# \setminus = 8$

$A \# = B + 2 \Rightarrow \min(A) = 1 + 2, \max(A) = 10 + 2 \Rightarrow A \text{ in } 3..10$

$\Rightarrow \min(B) = 1 - 2, \max(B) = 10 - 2 \Rightarrow B \text{ in } 1..8$

$A \# > 5 \Rightarrow \min(A) = 6 \Rightarrow A \text{ in } 6..10$

$\Rightarrow \min(B) = 6 - 2 \Rightarrow B \text{ in } 4..8$  (nové vyvolání  $A \# = B + 2$ )

$A \# \setminus = 8 \Rightarrow A \text{ in } (6..7) \setminus (9..10)$  (meze stejné, k propagaci  $A \# = B + 2$  nedojde)

● Vyzkoušejte si:  $A \# = B - C, A \# > = B + C$

# Globální podmínky

- Propagace je lokální
  - pracuje se s jednotlivými podmínkami
  - interakce mezi podmínkami je pouze přes domény proměnných
- Jak dosáhnout více, když je silnější propagace drahá?
- Seskupíme několik podmínek do jedné tzv. **globální podmínky**
- Propagaci přes globální podmínku řešíme speciálním algoritmem navrženým pro danou podmínku
- Příklady:
  - `all_different` omezení: hodnoty všech proměnných různé
  - `serialized` omezení:  
rozvržení úloh zadaných startovním časem a dobou trvání tak, aby se nepřekrývaly

# Propagace pro all\_distinct

●  $U = \{X_2, X_4, X_5\}$ ,  $\text{dom}(U) = \{2, 3, 4\}$ :

$\{2, 3, 4\}$  nelze pro  $X_1, X_3, X_6$

učitel	min	max
Jan	3	6
Petr	3	4
Anna	2	5
Ota	2	4
Eva	3	4
Marie	1	6

# Propagace pro all\_distinct

●  $U = \{X2, X4, X5\}$ ,  $\text{dom}(U) = \{2, 3, 4\}$ :

$\{2, 3, 4\}$  nelze pro  $X1, X3, X6$

$X1 \text{ in } 5..6, X3 = 5, X6 \text{ in } \{1\} \ \vee \ (5..6)$

učitel	min	max
Jan	3	6
Petr	3	4
Anna	2	5
Ota	2	4
Eva	3	4
Marie	1	6

# Propagace pro all\_distinct

●  $U = \{X_2, X_4, X_5\}$ ,  $\text{dom}(U) = \{2, 3, 4\}$ :

$\{2, 3, 4\}$  nelze pro  $X_1, X_3, X_6$

$X_1$  in  $5..6$ ,  $X_3 = 5$ ,  $X_6$  in  $\{1\} \setminus (5..6)$

● **Konzistence:**  $\forall \{X_1, \dots, X_k\} \subset V : \text{card}\{D_1 \cup \dots \cup D_k\} \geq k$

učitel	min	max
Jan	3	6
Petr	3	4
Anna	2	5
Ota	2	4
Eva	3	4
Marie	1	6

# Propagace pro all\_distinct

●  $U = \{X_2, X_4, X_5\}$ ,  $\text{dom}(U) = \{2, 3, 4\}$ :

$\{2, 3, 4\}$  nelze pro  $X_1, X_3, X_6$

$X_1 \text{ in } 5..6, X_3 = 5, X_6 \text{ in } \{1\} \ \vee \ (5..6)$

● **Konzistence:**  $\forall \{X_1, \dots, X_k\} \subset V : \text{card}\{D_1 \cup \dots \cup D_k\} \geq k$

stačí hledat **Hallův interval**  $I$ : velikost intervalu  $I$  je rovna počtu proměnných, jejichž doména je v  $I$

učitel	min	max
Jan	3	6
Petr	3	4
Anna	2	5
Ota	2	4
Eva	3	4
Marie	1	6



# Propagace pro all\_distinct

- $U = \{X_2, X_4, X_5\}$ ,  $dom(U) = \{2, 3, 4\}$ :

$\{2, 3, 4\}$  nelze pro  $X_1, X_3, X_6$

$X_1$  in  $5..6$ ,  $X_3 = 5$ ,  $X_6$  in  $\{1\} \setminus (5..6)$

- **Konzistence:**  $\forall \{X_1, \dots, X_k\} \subset V : card\{D_1 \cup \dots \cup D_k\} \geq k$

stačí hledat **Hallův interval**  $I$ : velikost intervalu  $I$  je rovna počtu proměnných, jejichž doména je v  $I$

- **Inferenční pravidlo**

- $U = \{X_1, \dots, X_k\}$ ,  $dom(U) = \{D_1 \cup \dots \cup D_k\}$

- $card(U) = card(dom(U)) \Rightarrow \forall v \in dom(U), \forall X \in (V - U), X \neq v$

učitel	min	max
Jan	3	6
Petr	3	4
Anna	2	5
Ota	2	4
Eva	3	4
Marie	1	6

# Propagace pro all\_distinct

- $U = \{X_2, X_4, X_5\}$ ,  $dom(U) = \{2, 3, 4\}$ :

$\{2, 3, 4\}$  nelze pro  $X_1, X_3, X_6$

$X_1$  in  $5..6$ ,  $X_3 = 5$ ,  $X_6$  in  $\{1\} \setminus (5..6)$

- **Konzistence:**  $\forall \{X_1, \dots, X_k\} \subset V : card\{D_1 \cup \dots \cup D_k\} \geq k$

stačí hledat **Hallův interval**  $I$ : velikost intervalu  $I$  je rovna počtu proměnných, jejichž doména je v  $I$

- **Inferenční pravidlo**

- $U = \{X_1, \dots, X_k\}$ ,  $dom(U) = \{D_1 \cup \dots \cup D_k\}$

- $card(U) = card(dom(U)) \Rightarrow \forall v \in dom(U), \forall X \in (V - U), X \neq v$

- hodnoty v Hallově intervalu jsou pro ostatní proměnné nedostupné

učitel	min	max
Jan	3	6
Petr	3	4
Anna	2	5
Ota	2	4
Eva	3	4
Marie	1	6

# Propagace pro all\_distinct

- $U = \{X_2, X_4, X_5\}$ ,  $dom(U) = \{2, 3, 4\}$ :

$\{2, 3, 4\}$  nelze pro  $X_1, X_3, X_6$

$X_1$  in  $5..6$ ,  $X_3 = 5$ ,  $X_6$  in  $\{1\} \setminus (5..6)$

- **Konzistence:**  $\forall \{X_1, \dots, X_k\} \subset V : card\{D_1 \cup \dots \cup D_k\} \geq k$

stačí hledat **Hallův interval**  $I$ : velikost intervalu  $I$  je rovna počtu proměnných, jejichž doména je v  $I$

- **Inferenční pravidlo**

- $U = \{X_1, \dots, X_k\}$ ,  $dom(U) = \{D_1 \cup \dots \cup D_k\}$

- $card(U) = card(dom(U)) \Rightarrow \forall v \in dom(U), \forall X \in (V - U), X \neq v$

- hodnoty v Hallově intervalu jsou pro ostatní proměnné nedostupné

- **Složitost:**  $O(2^n)$  – hledání všech podmnožin množiny  $n$  proměnných (naivní)

učitel	min	max
Jan	3	6
Petr	3	4
Anna	2	5
Ota	2	4
Eva	3	4
Marie	1	6

# Propagace pro all\_distinct

- $U = \{X_2, X_4, X_5\}$ ,  $dom(U) = \{2, 3, 4\}$ :

$\{2, 3, 4\}$  nelze pro  $X_1, X_3, X_6$

$X_1$  in  $5..6$ ,  $X_3 = 5$ ,  $X_6$  in  $\{1\} \setminus (5..6)$

- **Konzistence:**  $\forall \{X_1, \dots, X_k\} \subset V : card\{D_1 \cup \dots \cup D_k\} \geq k$

stačí hledat **Hallův interval**  $I$ : velikost intervalu  $I$  je rovna počtu proměnných, jejichž doména je v  $I$

- **Inferenční pravidlo**

- $U = \{X_1, \dots, X_k\}$ ,  $dom(U) = \{D_1 \cup \dots \cup D_k\}$

- $card(U) = card(dom(U)) \Rightarrow \forall v \in dom(U), \forall X \in (V - U), X \neq v$

- hodnoty v Hallově intervalu jsou pro ostatní proměnné nedostupné

- **Složitost:**  $O(2^n)$  – hledání všech podmnožin množiny  $n$  proměnných (naivní)

$O(n \log n)$  – kontrola hraničních bodů Hallových intervalů (1998)

učitel	min	max
Jan	3	6
Petr	3	4
Anna	2	5
Ota	2	4
Eva	3	4
Marie	1	6

# Prohledávání + konzistence

- Splňování podmínek **prohledáváním** prostoru řešení
  - podmínky jsou užívány pasivně jako test
  - přiřazuji hodnoty proměnných a zkouším co se stane
  - vestavěný prohledávací algoritmus Prologu: **backtracking**, triviální: **generuj & testuj**

# Prohledávání + konzistence

- Splňování podmínek **prohledáváním** prostoru řešení
  - podmínky jsou užívány pasivně jako test
  - přiřazuji hodnoty proměnných a zkouším co se stane
  - vestavěný prohledávací algoritmus Prologu: **backtracking**, triviální: **generuj & testuj**
  - úplná metoda (nalezneme řešení nebo dokážeme jeho neexistenci)
  - zbytečně pomalé (exponenciální): procházím i „evidentně“ špatná ohodnocení

# Prohledávání + konzistence

- Splňování podmínek **prohledáváním** prostoru řešení
  - podmínky jsou užívány pasivně jako test
  - přiřazuji hodnoty proměnných a zkouším co se stane
  - vestavěný prohledávací algoritmus Prologu: **backtracking**, triviální: **generuj & testuj**
  - úplná metoda (nalezneme řešení nebo dokážeme jeho neexistenci)
  - zbytečně pomalé (exponenciální): procházím i „evidentně“ špatná ohodnocení
- **Konzistenční (propagační) techniky**
  - umožňují odstranění nekonzistentních hodnot z domény proměnných
  - neúplná metoda (v doméně zůstanou ještě nekonzistentní hodnoty)
  - relativně rychlé (polynomiální)

# Prohledávání + konzistence

## ● Splňování podmínek **prohledáváním** prostoru řešení

- podmínky jsou užívány pasivně jako test
- přiřazuji hodnoty proměnných a zkouším co se stane
- vestavěný prohledávací algoritmus Prologu: **backtracking**, triviální: **generuj & testuj**
- úplná metoda (nalezneme řešení nebo dokážeme jeho neexistenci)
- zbytečně pomalé (exponenciální): procházím i „evidentně“ špatná ohodnocení

## ● **Konzistenční (propagační) techniky**

- umožňují odstranění nekonzistentních hodnot z domény proměnných
- neúplná metoda (v doméně zůstanou ještě nekonzistentní hodnoty)
- relativně rychlé (polynomiální)

## ● Používá se **kombinace obou metod**

- postupné přiřazování hodnot proměnným
- po přiřazení hodnoty odstranění nekonzistentních hodnot konzistenčními technikami



# Prohledávání s navracením

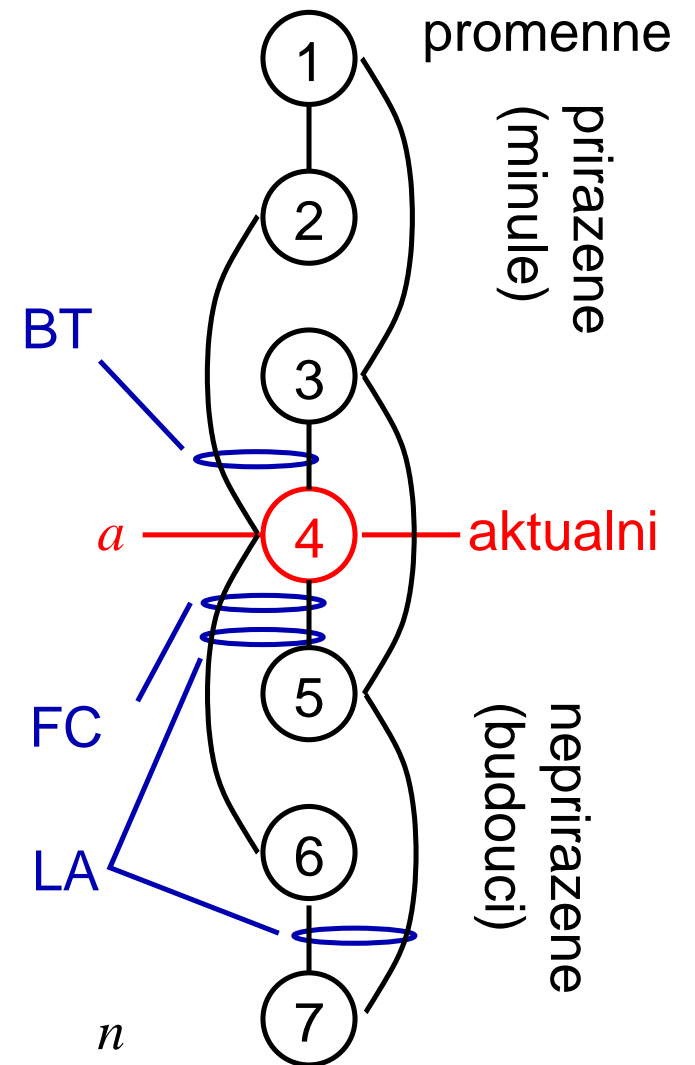
- Základní prohledávací algoritmus pro problémy splňování podmínek
- **Prohledávání stavového prostoru do hloubky**
- Dvě fáze prohledávání s navracením
  - **dopředná fáze**: proměnné jsou postupně vybírány, rozšiřuje se částečné řešení přiřazením konzistentní hodnoty (pokud existuje) další proměnné
    - po vybrání hodnoty testujeme konzistenci
  - **zpětná fáze**: pokud neexistuje konzistentní hodnota pro aktuální proměnnou, algoritmus se vrací k předchozí přiřazené hodnotě

# Prohledávání s navracením

- Základní prohledávací algoritmus pro problémy splňování podmínek
- **Prohledávání stavového prostoru do hloubky**
- Dvě fáze prohledávání s navracením
  - **dopředná fáze**: proměnné jsou postupně vybírány, rozšiřuje se částečné řešení přiřazením konzistentní hodnoty (pokud existuje) další proměnné
    - po vybrání hodnoty testujeme konzistenci
  - **zpětná fáze**: pokud neexistuje konzistentní hodnota pro aktuální proměnnou, algoritmus se vrací k předchozí přiřazené hodnotě
- Proměnné dělíme na
  - **minulé** – proměnné, které už byly vybrány (a mají přiřazenu hodnotu)
  - **aktuální** – proměnná, která je právě vybrána a je jí přiřazována hodnota
  - **budoucí** – proměnné, které budou vybrány v budoucnosti

# Přehled algoritmů

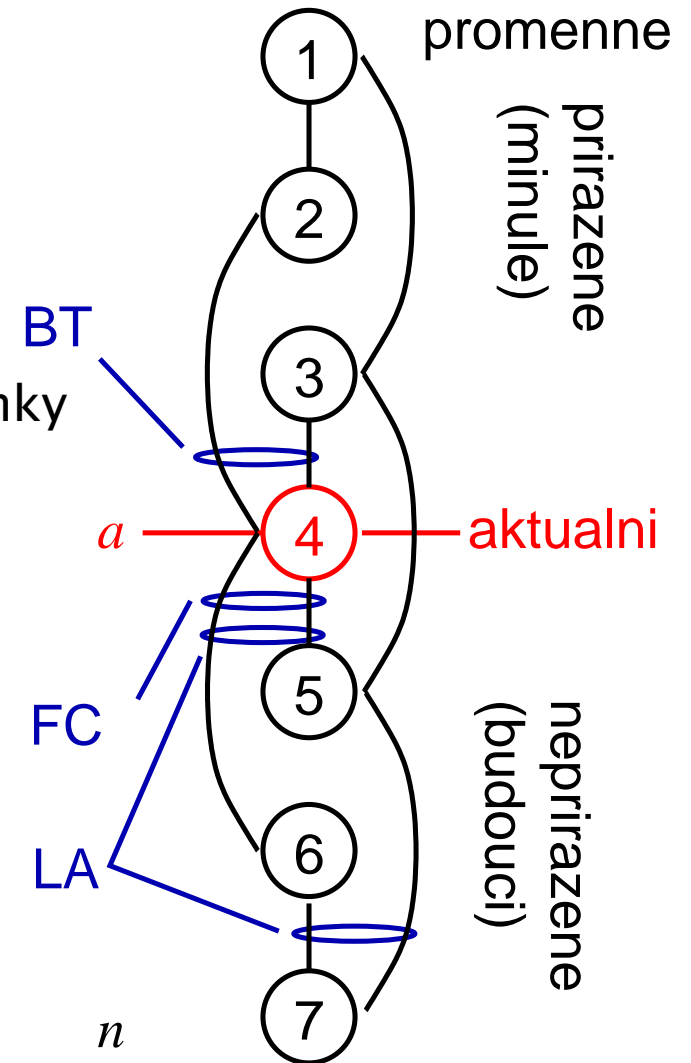
- **Backtracking (BT)** kontroluje v kroku  $a$  podmínky  $c(V_1, V_a), \dots, c(V_{a-1}, V_a)$  z minulých proměnných do aktuální proměnné



# Přehled algoritmů

- **Backtracking (BT)** kontroluje v kroku  $a$  podmínky  $c(V_1, V_a), \dots, c(V_{a-1}, V_a)$  z minulých proměnných do aktuální proměnné

- **Kontrola dopředu (FC)** kontroluje v kroku  $a$  podmínky  $c(V_a, V_{a+1}), \dots, c(V_a, V_n)$  z aktuální proměnné do budoucích proměnných



# Přehled algoritmů

- **Backtracking (BT)** kontroluje v kroku  $a$  podmínky

$$c(V_1, V_a), \dots, c(V_{a-1}, V_a)$$

z minulých proměnných do aktuální proměnné

- **Kontrola dopředu (FC)** kontroluje v kroku  $a$  podmínky

$$c(V_a, V_{a+1}), \dots, c(V_a, V_n)$$

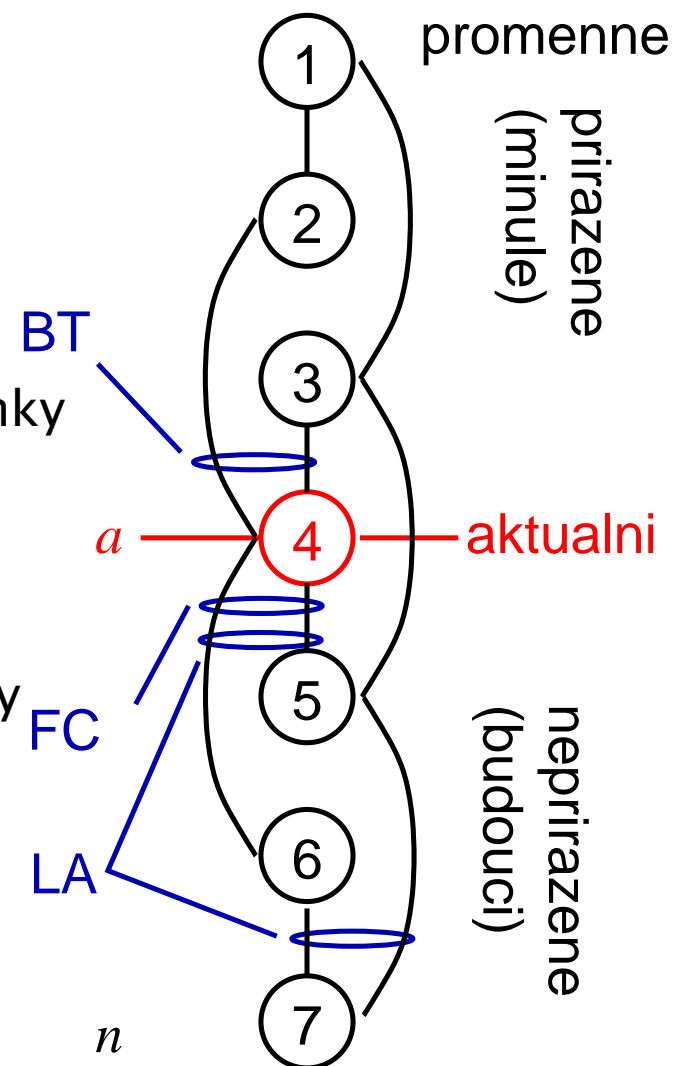
z aktuální proměnné do budoucích proměnných

- **Pohled dopředu (LA)** kontroluje v kroku  $a$  podmínky

$$\forall l(a \leq l \leq n), \forall k(a \leq k \leq n), k \neq l: c(V_k, V_l)$$

z aktuální proměnné do budoucích proměnných

a mezi budoucími proměnnými



# Základní algoritmus prohledávání s navracením

- Pro jednoduchost proměnné očíslováme a ohodnocujeme je v daném pořadí
- Na začátku voláno jako `Labeling(G, 1)`

```
procedure labeling(G, a)
```

```
if  $a > |\text{uzly}(G)|$  then return uzly(G)
```

```
for  $\forall x \in D_a$  do
```

```
    if consistent(G, a) then % consistent(G, a) je nahrazeno FC, LA, ...
```

```
        R := labeling(G, a + 1)
```

```
        if R  $\neq$  fail then return R
```

```
return fail
```

```
end labeling
```

Po přiřazení všech proměnných vrátíme jejich ohodnocení

- Procedure `consistent` uvedeme pouze pro binární podmínky

# Backtracking (BT)

- Backtracking ověřuje v každém kroku konzistenci podmínek vedoucích z minulých proměnných do aktuální proměnné
- Backtracking tedy zajišťuje konzistenci podmínek
  - na všech minulých proměnných
  - na podmínkách mezi minulými proměnnými a aktuální proměnnou

# Backtracking (BT)

- Backtracking ověřuje v každém kroku konzistenci podmínek vedoucích z minulých proměnných do aktuální proměnné
- Backtracking tedy zajišťuje konzistenci podmínek
  - na všech minulých proměnných
  - na podmínkách mezi minulými proměnnými a aktuální proměnnou

● procedure  $BT(G, a)$

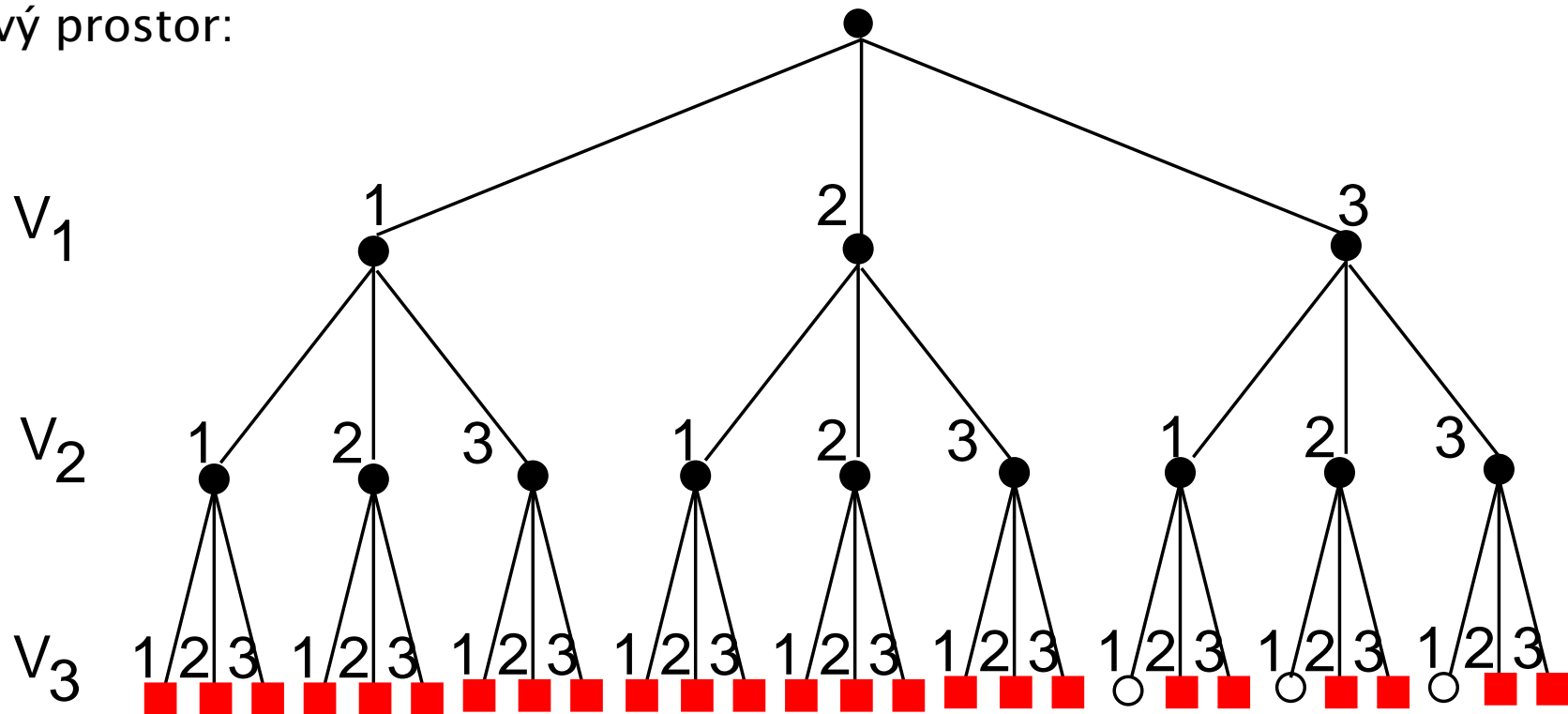
```
Q := {(Vi, Va) ∈ hrany(G), i < a}           % hrany vedoucí do minulých proměnných
Consistent := true
while Q není prázdná ∧ Consistent do
    vyber a smaž libovolnou hranu (Vk, Vm) z Q
    Consistent := not revise(Vk, Vm)       % pokud vyřadíme prvek, bude doména prázdná
return Consistent
end BT
```



# Příklad: backtracking

● Omezení:  $V_1, V_2, V_3$  in  $1 \dots 3$ ,  $V_1 \neq 3 \times V_3$

● Stavový prostor:



- červené čtverečky: chybný pokus o instanciaci, řešení neexistuje
- nevyplněná kolečka: nalezeno řešení
- černá kolečka: vnitřní uzel, máme pouze částečné přiřazení

# Kontrola dopředu (*FC – forward checking*)

- FC je rozšíření backtrackingu
- FC navíc zajišťuje konzistenci mezi aktuální proměnnou a budoucími proměnnými, které jsou s ní spojeny dosud nesplněnými podmínkami

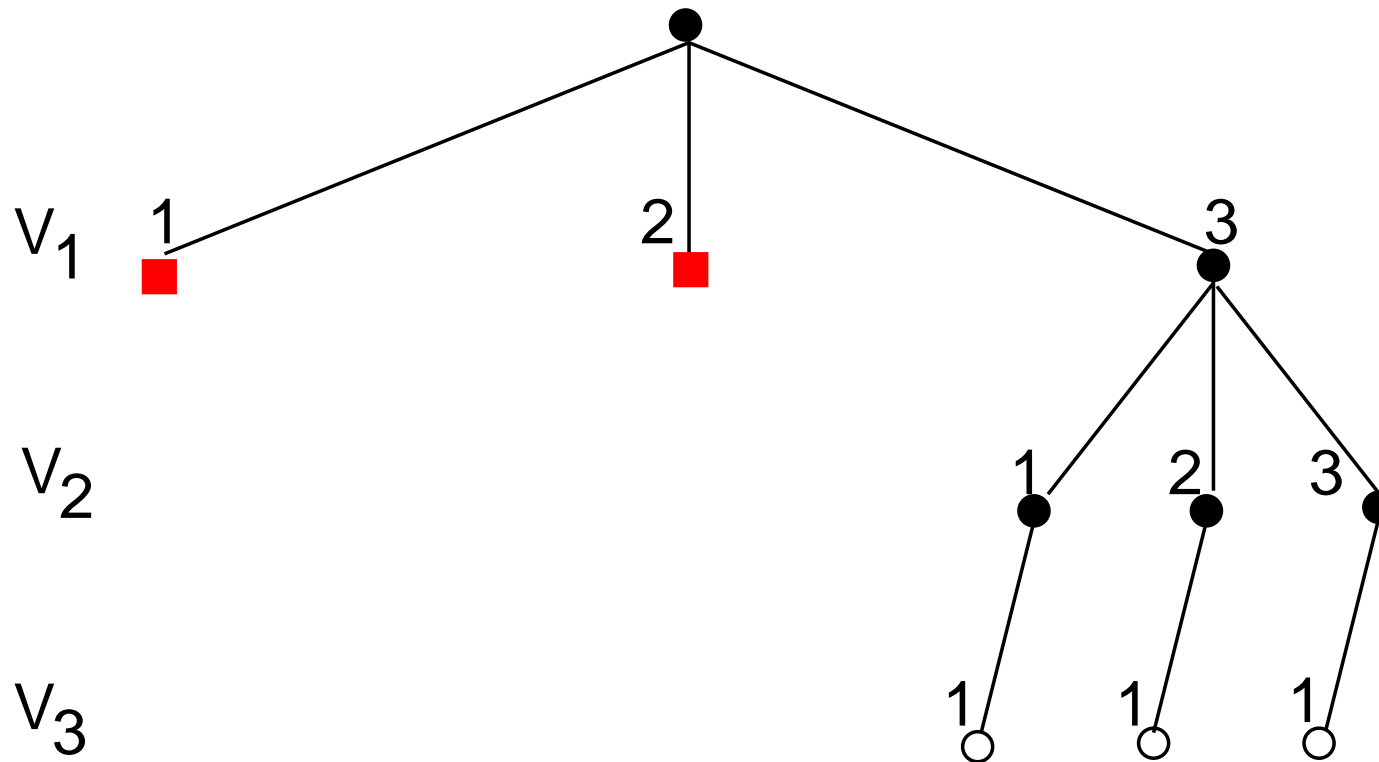
# Kontrola dopředu (*FC – forward checking*)

- FC je rozšíření backtrackingu
- FC navíc zajišťuje konzistenci mezi aktuální proměnnou a budoucími proměnnými, které jsou s ní spojeny dosud nesplněnými podmínkami
- `procedure FC(G, a)`  
`Q := {(Vi, Va) ∈ hrany(G), i > a}`    % přidání hran z aktuální proměnné do budoucích prom.  
`Consistent := true`  
`while Q není prázdná ∧ Consistent do`  
    `vyber a smaž libovolnou hranu (Vk, Vm) z Q`  
    `if revise((Vk, Vm)) then`  
        `Consistent := (|Dk| > 0)`    % vyprázdnění domény znamená nekonzistenci  
`return Consistent`  
`end FC`
- Hran z aktuální proměnné do minulých proměnných není nutno testovat

# Příklad: kontrola dopředu

● Omezení:  $V_1, V_2, V_3$  in  $1 \dots 3$ ,  $V_1 \neq 3 \times V_3$

● Stavový prostor:



# Pohled dopředu (*LA – looking ahead*)

- LA je rozšíření FC, LA zajišťuje hranovou konzistenci
- LA navíc ověřuje i konzistenci všech hran mezi budoucími proměnnými

● procedure  $LA(G, a)$

$Q := \{(V_i, V_a) \in \text{hrany}(G), i > a\}$       % začínáme s hranami do  $a$

Consistent := true

while  $Q$  není prázdná  $\wedge$  Consistent do

    vyber a smaž libovolnou hranu  $(V_k, V_m)$  z  $Q$

    if revise( $(V_k, V_m)$ ) then

$Q := Q \cup \{(V_i, V_k) \mid (V_i, V_k) \in \text{hrany}(G), i \neq k, i \neq m, i > a\}$

        Consistent :=  $(|D_k| > 0)$

return Consistent

end LA

# Pohled dopředu (*LA – looking ahead*)

- LA je rozšíření FC, LA zajišťuje hranovou konzistenci
- LA navíc ověřuje i konzistenci všech hran mezi budoucími proměnnými

● procedure  $LA(G, a)$

$Q := \{(V_i, V_a) \in \text{hrany}(G), i > a\}$       % začínáme s hranami do  $a$

$\text{Consistent} := \text{true}$

while  $Q$  není prázdná  $\wedge$   $\text{Consistent}$  do

    vyber a smaž libovolnou hranu  $(V_k, V_m)$  z  $Q$

    if  $\text{revise}((V_k, V_m))$  then

$Q := Q \cup \{(V_i, V_k) \mid (V_i, V_k) \in \text{hrany}(G), i \neq k, i \neq m, i > a\}$

$\text{Consistent} := (|D_k| > 0)$

return  $\text{Consistent}$

end  $LA$

- Hrany z aktuální proměnné do minulých proměnných opět netestujeme
- Tato LA procedura je založena na AC-3, lze použít i jiné AC algoritmy

# Příklad: pohled dopředu

● Omezení:  $V_1, V_2, V_3$  in  $1 \dots 4$ ,  $V_1\# > V_2$ ,  $V_2\# = 3 \times V_3$

● Stavový prostor:

