

# 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 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 (1) nejmenší velikostí domény `fd_size(Var,Size)`  
(2) (pokud s nejmenší velikostí domény více, tak) nejlevější z nich

# 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 (1) nejmenší velikostí domény fd\_size(Var,Size)  
(2) (pokud s nejmenší velikostí domény více, tak) nejlevější z nich
  - **ffc**: s (1) nejmenší velikostí domény fd\_degree(Var,Size)  
(2) největším množstvím omezením „čekajících“ na proměnné  
(3) nejlevější z nich
  - **min/max**: s (1) nejmenší/největší hodnotou v doméně proměnné fd\_min(Var,Min)/fd\_max(Var,Max)  
(2) 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*)**
  - algoritmus, který implementuje proceduru pro minimalizaci (duálně pro maximalizaci)
  - 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
  - přidává se tedy inkrementálně omezení  $LB \# < UB$  pro snižující se  $UB$  tak, jak nalézáme kvalitnější řešení

# 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)

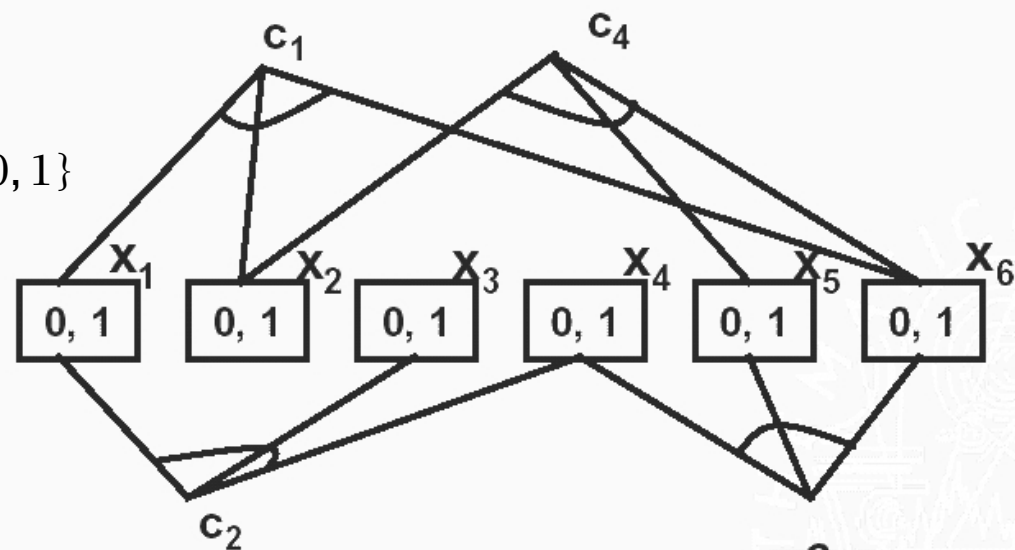
## ● Reprezentace CSP pomocí hypergrafu podmínek

- 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_distinct` 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  $V_i, V_j$ .

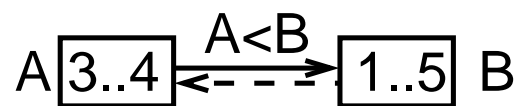
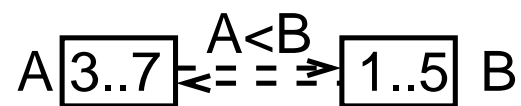
# Vrcholová a hranová konzistence

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

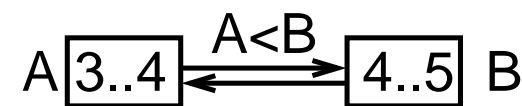
- každá hodnota z aktuální domény  $V_i$  proměnné splňuje všechny unární podmínky s proměnnou  $V_i$

## ● 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)

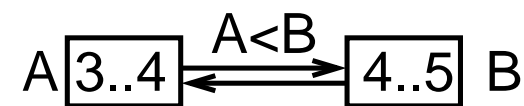
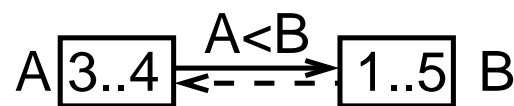
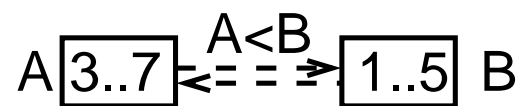
# Vrcholová a hranová konzistence

## ● 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

# Algoritmus revize hrany

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

# Algoritmus revize hrany

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

● `procedure revise((i,j))`

`Deleted := false`

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

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

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

`Deleted := true`

`end if`

`return Deleted`

`end revise`

# Algoritmus revize hrany

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

● `procedure revise((i,j))`

`Deleted := false`

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

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

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

`Deleted := true`

`end if`

`return Deleted`

`end revise`

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

# Algoritmus revize hrany

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

● `procedure revise((i,j))`

`Deleted := false`

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

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

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

`Deleted := true`

`end if`

`return Deleted`

`end revise`

- `domain([V1, V2], 2, 4), V1 #< V2    revise((1,2)) smaže 4 z  $D_1$ ,`



# Algoritmus revize hrany

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

● `procedure revise((i,j))`

`Deleted := false`

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

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

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

`Deleted := true`

`end if`

`return Deleted`

`end revise`

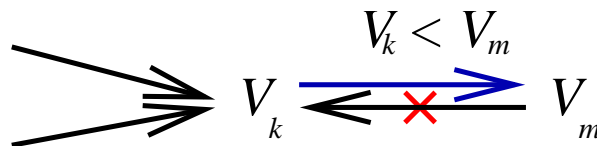
- `domain([V1, V2], 2, 4), V1 #< V2`     `revise((1,2))` smaže 4 z  $D_1, D_2$  se nezmění

# Dosažení hranové konzistence problému

- Jak udělat CSP hranově konzistentní?
  - revize je potřeba opakovat, dokud se mění doména nějaké proměnné
  - efektivnější: opakování revizí můžeme dělat pomocí **fronty**
    - přidáváme do ní hrany, jejichž konzistence mohla být narušena zmenšením domény

# Dosažení hranové konzistence problému

- Jak udělat CSP hranově konzistentní?
  - revize je potřeba opakovat, dokud se mění doména nějaké proměnné
  - efektivnější: opakování revizí můžeme dělat pomocí **fronty**
    - přidáváme do ní hrany, jejichž konzistence mohla být narušena zmenšením domény
- Jaké hrany přesně revidovat po zmenšení domény?
  - ty, jejichž konzistence může být zmenšením domény proměnné narušena jsou to hrany  $(i, k)$ , které vedou do proměnné  $V_k$  se zmenšenou doménou



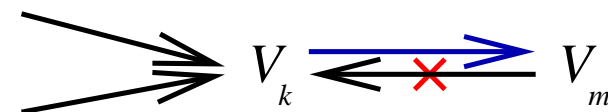
- hranu  $(m, k)$  vedoucí z proměnné  $V_m$ , která zmenšení domény způsobila, není třeba revidovat (změna se jí nedotkne)

# Algoritmus AC-3



```
● procedure AC-3(G)
  Q := {(i,j) | (i,j) ∈ hrany(G), i ≠ j} % seznam hran pro revizi
  while Q non empty do
    vyber a smaž (k,m) z Q
    if revise((k,m)) then % pridani pouze hran, ktere
      Q := Q ∪ {(i,k) ∈ hrany(G), i ≠ k, i ≠ m} % dosud nejsou ve fronte
  end while
end AC-3
```

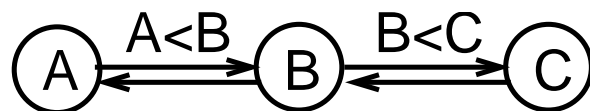
# Algoritmus AC-3



```

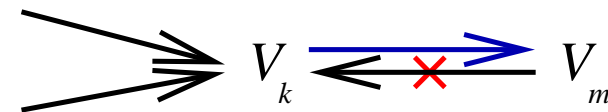
● procedure AC-3(G)
  Q := {(i,j) | (i,j) ∈ hrany(G), i ≠ j}      % seznam hran pro revizi
  while Q non empty do
    vyber a smaž (k,m) z Q
    if revise((k,m)) then                       % pridani pouze hran, ktere
      Q := Q ∪ {(i,k) ∈ hrany(G), i ≠ k, i ≠ m} % dosud nejsou ve fronte
  end while
end AC-3
  
```

● Příklad:



$A < B, B < C: (3..7, 1..5, 1..5) \xrightarrow{AB} (3..4, 1..5, 1..5) \xrightarrow{BA}$   
 $(3..4, 4..5, 1..5) \xrightarrow{BC} (3..4, 4, 1..5) \xrightarrow{CB} (3..4, 4, 5)$   
 $\xrightarrow{AB} (3, 4, 5)$

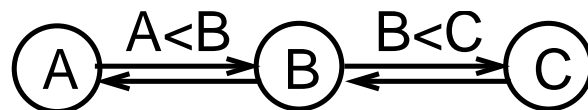
# Algoritmus AC-3



```

● procedure AC-3(G)
  Q := {(i,j) | (i,j) ∈ hrany(G), i ≠ j}      % seznam hran pro revizi
  while Q non empty do
    vyber a smaž (k,m) z Q
    if revise((k,m)) then                       % pridani pouze hran, ktere
      Q := Q ∪ {(i,k) ∈ hrany(G), i ≠ k, i ≠ m} % dosud nejsou ve fronte
  end while
end AC-3
  
```

● Příklad:



$A < B, B < C: (3..7, 1..5, 1..5) \xrightarrow{AB} (3..4, 1..5, 1..5) \xrightarrow{BA}$   
 $(3..4, 4..5, 1..5) \xrightarrow{BC} (3..4, 4, 1..5) \xrightarrow{CB} (3..4, 4, 5)$   
 $\xrightarrow{AB} (3, 4, 5)$

● Technika AC-3 je dnes asi nejpoužívanější, ale stále není optimální

● Jaké budou domény A, B, C po AC-3 pro:  $\text{domain}([A, B, C], 1, 10)$ ,  $A \neq B + 1$ ,  $C \# < B$

# 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

# k-konzistence

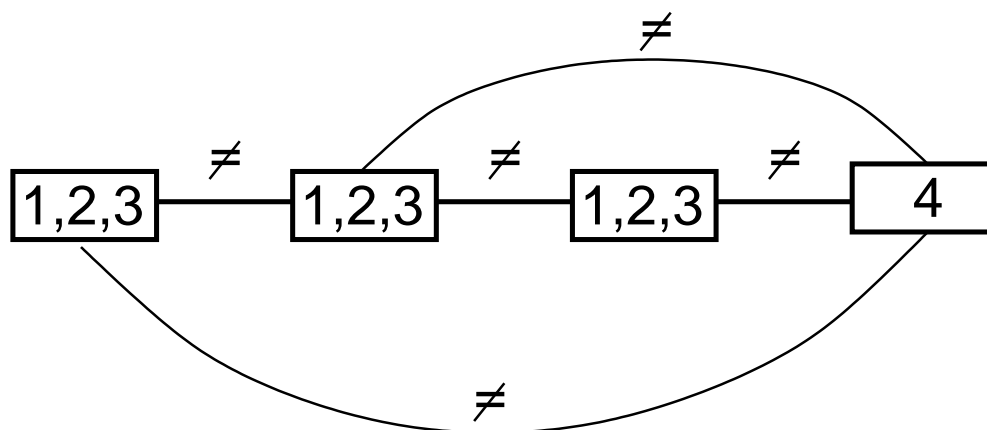
- Mají NC a AC něco společného?
  - NC: konzistence jedné proměnné
  - AC: konzistence dvou proměnných
  - ... můžeme pokračovat

# k-konzistence

- Mají NC a AC něco společného?
  - NC: konzistence jedné proměnné
  - AC: konzistence dvou proměnných
  - ... můžeme pokračovat
- CSP je **k-konzistentní** právě tehdy, když můžeme libovolné konzistentní ohodnocení  $(k-1)$  různých proměnných rozšířit do libovolné  $k$ -té proměnné

# k-konzistence

- Mají NC a AC něco společného?
  - NC: konzistence jedné proměnné
  - AC: konzistence dvou proměnných
  - ... můžeme pokračovat
- CSP je **k-konzistentní** právě tehdy, když můžeme libovolné konzistentní ohodnocení  $(k-1)$  různých proměnných rozšířit do libovolné  $k$ -té proměnné

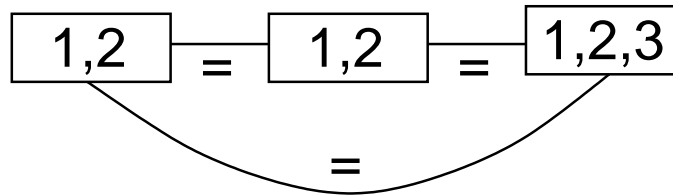


4-konzistentní graf

- Pro obecné CSP, tedy i pro nebinární podmínky

# Silná k-konzistence

3-konzistentní graf



není 2-konzistentní

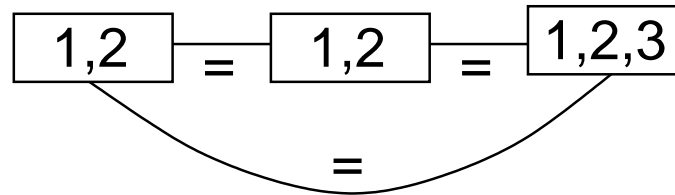
# Silná k-konzistence

3-konzistentní graf

(1, 1) lze rozšířit na (1, 1, 1)

(2, 2) lze rozšířit na (2, 2, 2)

(1, 3) ani (2, 3) nejsou konzistentní dvojice (nerozšiřujeme je)



není 2-konzistentní

(3) nelze rozšířit

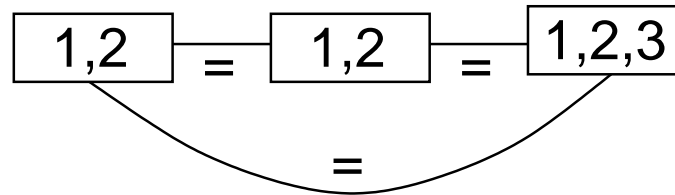
# Silná k-konzistence

3-konzistentní graf

(1, 1) lze rozšířit na (1, 1, 1)

(2, 2) lze rozšířit na (2, 2, 2)

(1, 3) ani (2, 3) nejsou konzistentní dvojice (nerozšiřujeme je)



není 2-konzistentní

(3) nelze rozšířit

● CSP je **silně k-konzistentní** právě tehdy, když je j-konzistentní pro každé  $j \leq k$

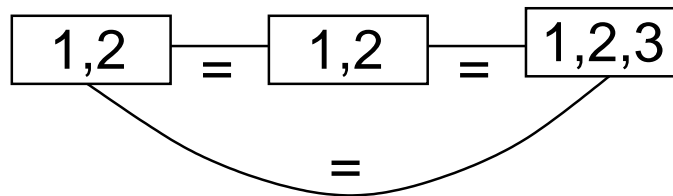
# Silná k-konzistence

3-konzistentní graf

(1, 1) lze rozšířit na (1, 1, 1)

(2, 2) lze rozšířit na (2, 2, 2)

(1, 3) ani (2, 3) nejsou konzistentní dvojice (nerozšiřujeme je)



není 2-konzistentní

(3) nelze rozšířit

- CSP je **silně k-konzistentní** právě tehdy, když je j-konzistentní pro každé  $j \leq k$
- Silná k-konzistence  $\Rightarrow$  k-konzistence
- Silná k-konzistence  $\Rightarrow$  j-konzistence  $\forall j \leq k$
- k-konzistence  $\not\Rightarrow$  silná k-konzistence



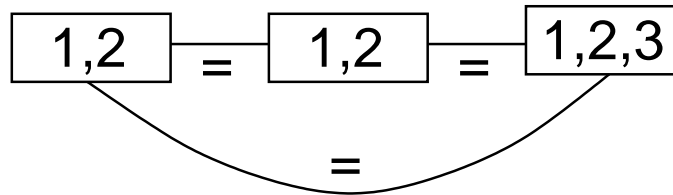
# Silná k-konzistence

3-konzistentní graf

(1, 1) lze rozšířit na (1, 1, 1)

(2, 2) lze rozšířit na (2, 2, 2)

(1, 3) ani (2, 3) nejsou konzistentní dvojice (nerozšiřujeme je)



není 2-konzistentní

(3) nelze rozšířit

- CSP je **silně k-konzistentní** právě tehdy, když je j-konzistentní pro každé  $j \leq k$
- Silná k-konzistence  $\Rightarrow$  k-konzistence
- Silná k-konzistence  $\Rightarrow$  j-konzistence  $\forall j \leq k$
- k-konzistence  $\not\Rightarrow$  silná k-konzistence
- NC = silná 1-konzistence = 1-konzistence
- AC = (silná) 2-konzistence

# Konzistence pro nalezení řešení

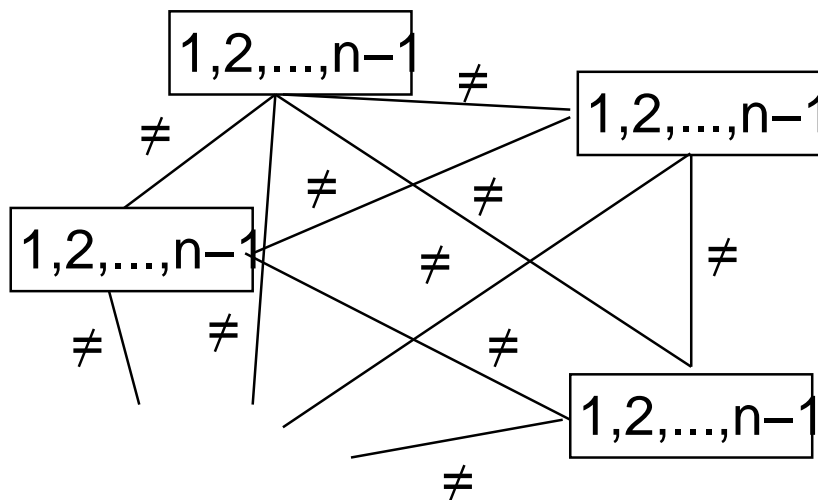
- Máme-li graf s  $n$  vrcholy, jak silnou konzistenci potřebujeme, abychom přímo našli řešení?
- silná  $n$ -konzistence je nutná pro graf s  $n$  vrcholy

# Konzistence pro nalezení řešení

- Máme-li graf s  $n$  vrcholy, jak silnou konzistenci potřebujeme, abychom přímo našli řešení?
- silná  $n$ -konzistence je nutná pro graf s  $n$  vrcholy
  - $n$ -konzistence nestačí (viz předchozí příklad)

# Konzistence pro nalezení řešení

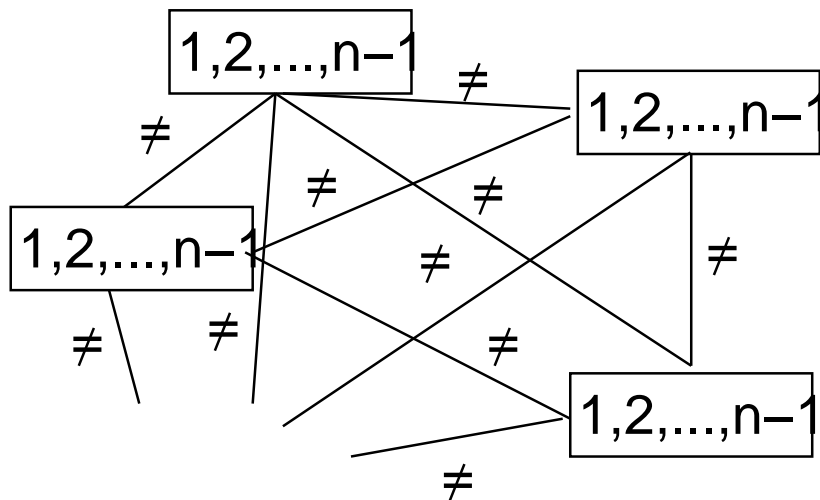
- Máme-li graf s  $n$  vrcholy, jak silnou konzistenci potřebujeme, abychom přímo našli řešení?
- silná  $n$ -konzistence je nutná pro graf s  $n$  vrcholy
  - $n$ -konzistence nestačí (viz předchozí příklad)
  - silná  $k$ -konzistence pro  $k < n$  také nestačí



graf s  $n$  vrcholy  
domény  $1..(n-1)$

# Konzistence pro nalezení řešení

- Máme-li graf s  $n$  vrcholy, jak silnou konzistenci potřebujeme, abychom přímo našli řešení?
- silná  $n$ -konzistence je nutná pro graf s  $n$  vrcholy
  - $n$ -konzistence nestačí (viz předchozí příklad)
  - silná  $k$ -konzistence pro  $k < n$  také nestačí



graf s  $n$  vrcholy  
domény  $1..(n-1)$

silně  $k$ -konzistentní pro každé  $k < n$   
přesto nemá řešení