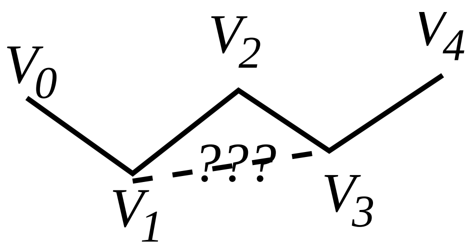
Konzistence po cestě

Konzistence po cestě (PC *path consistency*)

- Příklad: $X, Y, Z \in \{1, 2\}, \quad X \neq Y, \quad Y \neq Z, \quad Z \neq X$
- Jak posílit konzistenci? Budeme se zabývat několika podmínkami najednou.
- **Cesta ($V_0, V_1, \ldots, V_m$) je konzistentní** právě tehdy, když pro každou dvojici hodnot $x \in D_0$ a $y \in D_m$ splňující binární podmínky na hraně V_0, V_m existuje ohodnocení proměnných $V_1, \ldots V_{m-1}$ takové, že všechny binární podmínky mezi sousedy V_j, V_{j+1} jsou splněny.
- CSP je **konzistentní po cestě**, právě když jsou všechny cesty konzistentní.

Konzistence po cestě (PC *path consistency*)

- Příklad: $X, Y, Z \in \{1, 2\}, \quad X \neq Y, \quad Y \neq Z, \quad Z \neq X$
- Jak posílit konzistenci? Budeme se zabývat několika podmínkami najednou.
- **Cesta $(V_0, V_1, \ldots, V_m)$ je konzistentní** právě tehdy, když pro každou dvojici hodnot $x \in D_0$ a $y \in D_m$ splňující binární podmínky na hraně V_0, V_m existuje ohodnocení proměnných $V_1, \ldots V_{m-1}$ takové, že všechny binární podmínky mezi sousedy V_j, V_{j+1} jsou splněny.
- CSP je **konzistentní po cestě**, právě když jsou všechny cesty konzistentní.
- Definice PC nezaručuje, že jsou splněny všechny podmínky nad vrcholy cesty, zabývá se pouze podmínkami mezi sousedy



PC a cesty délky 2

- Zjišťování konzistence všech cest není efektivní
- Stačí ověřit konzistenci cest délky 2
- Věta: **CSP je PC právě tehdy, když každá cesta délky 2 je PC.**

PC a cesty délky 2

- Zjišťování konzistence všech cest není efektivní
- Stačí ověřit konzistenci cest délky 2
- Věta: **CSP je PC právě tehdy, když každá cesta délky 2 je PC.**
- Důkaz: 1) PC ⇒ cesty délky 2 jsou PC

  2) cesty délky 2 jsou PC ⇒ $\forall n$ cesty délky n jsou PC ⇒ PC   indukcí podle délky cesty

  a) $n = 2$ platí triviálně

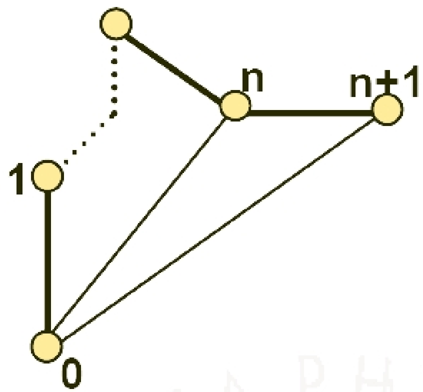
  b) $n + 1$ (za předpokladu, že platí pro n)

  i) vezmeme libovolných $n + 2$ vrcholů $V_0, V_1, \ldots, V_{n+1}$

  ii) vezmeme libov. dvě kompatibilní hodnoty $x_0 \in D_0$ a $x_{n+1} \in D_{n+1}$
  (kompatibilní = splňující všechny bin.podmínky mezi x_0 a x_{n+1})

  iii) podle a) jsou všechny cesty délky 2 PC, a tedy i V_0, V_n, V_{n+1} je PC
  najdeme tedy $x_n \in D_n$ tak, že (x_0, x_n) a (x_n, x_{n+1}) jsou konzistentní

  iv) podle indukčního kroku najdeme zbylé hodnoty na cestě $V_0, V_1, \ldots, V_n$



PC a cesty délky 2

- Zjišťování konzistence všech cest není efektivní
- Stačí ověřit konzistenci cest délky 2
- Věta: **CSP je PC právě tehdy, když každá cesta délky 2 je PC.**
- Důkaz: 1) PC $\Rightarrow$ cesty délky 2 jsou PC

  2) cesty délky 2 jsou PC $\Rightarrow$ $\forall n$ cesty délky n jsou PC $\Rightarrow$ PC   indukcí podle délky cesty

  a) $n = 2$ platí triviálně

  b) $n + 1$ (za předpokladu, že platí pro n)

  i) vezmeme libovolných $n + 2$ vrcholů $V_0, V_1, \ldots, V_{n+1}$

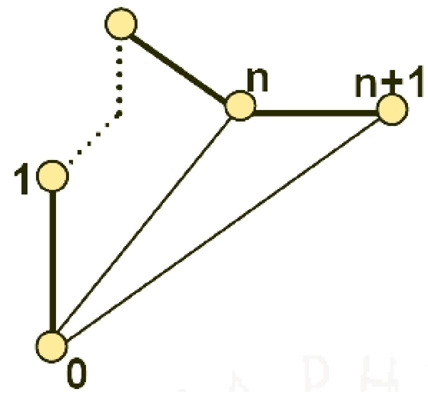
  ii) vezmeme libov. dvě kompatibilní hodnoty $x_0 \in D_0$ a $x_{n+1} \in D_{n+1}$

  (kompatibilní = splňující všechny bin.podmínky mezi x_0 a x_{n+1})

  iii) podle a) jsou všechny cesty délky 2 PC, a tedy i V_0, V_n, V_{n+1} je PC

  najdeme tedy $x_n \in D_n$ tak, že (x_0, x_n) a (x_n, x_{n+1}) jsou konzistentní

  iv) podle indukčního kroku najdeme zbylé hodnoty na cestě $V_0, V_1, \ldots, V_n$



- Definici PC lze tedy upravit tak, že vyžadujeme pouze konzistenci cest délky 2

Vztah PC a AC

- PC ⇒ AC
  - pokud je cesta (i, j, i) konzistentní (PC),
    pak je i hrana (i, j) konzistentní (AC), tj. z PC tedy plyne AC
    - PC: ke každé „dvojici hodnot” pro i, i najdu hodnotu v j

    ⇒ AC: ke každé hodnotě v i tedy najdu hodnotu v j

Vztah PC a AC

- PC $\Rightarrow$ AC
  - pokud je cesta (i, j, i) konzistentní (PC),
    pak je i hrana (i, j) konzistentní (AC), tj. z PC tedy plyne AC
    - PC: ke každé „dvojici hodnot" pro i, i najdu hodnotu v j

    $\Rightarrow$ AC: ke každé hodnotě v i tedy najdu hodnotu v j
- AC $\not\Rightarrow$ PC
  - příklad: $X, Y, Z \in \{1, 2\}$, $X \neq Y$, $Y \neq Z$, $Z \neq X$
    - je AC, ale není PC, zdůvodnění:

Vztah PC a AC

- PC $\Rightarrow$ AC
  - pokud je cesta (i, j, i) konzistentní (PC),
    pak je i hrana (i, j) konzistentní (AC), tj. z PC tedy plyne AC
    - PC: ke každé „dvojici hodnot” pro i, i najdu hodnotu v j

    $\Rightarrow$ AC: ke každé hodnotě v i tedy najdu hodnotu v j
- AC $\not\Rightarrow$ PC
  - příklad: $\mathrm{X}, \mathrm{Y}, \mathrm{Z} \in \{1, 2\}$, $\mathrm{X} \neq \mathrm{Y}$, $\mathrm{Y} \neq \mathrm{Z}$, $\mathrm{Z} \neq \mathrm{X}$
    - je AC, ale není PC, zdůvodnění: X=0, Y=1 nelze rozšířit po cestě (X,Z,Y)

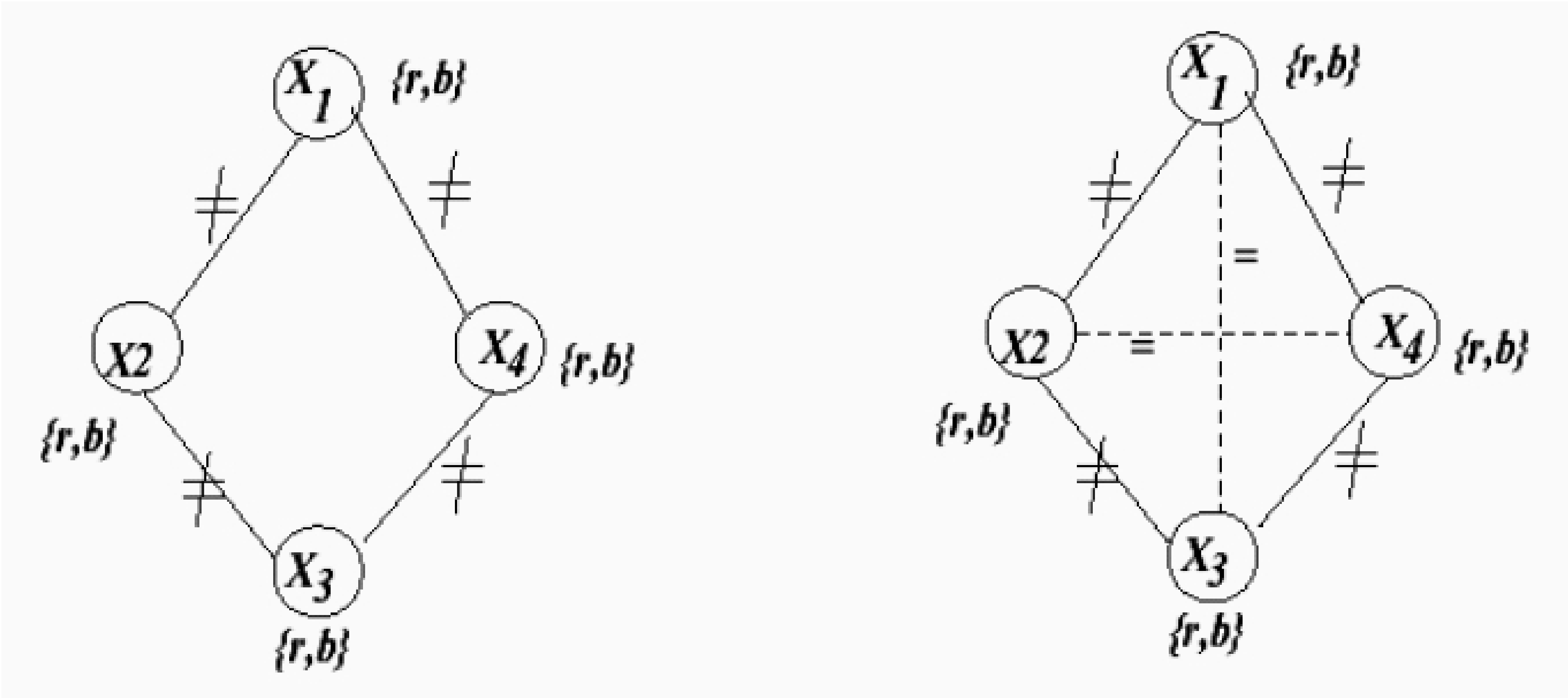
Vztah PC a AC

- PC $\Rightarrow$ AC
  - pokud je cesta (i, j, i) konzistentní (PC),
    pak je i hrana (i, j) konzistentní (AC),   tj. z PC tedy plyne AC
    - PC: ke každé „dvojici hodnot" pro i, i najdu hodnotu v j

      $\Rightarrow$ AC: ke každé hodnotě v i tedy najdu hodnotu v j
- AC $\not\Rightarrow$ PC
  - příklad: $X, Y, Z \in \{1, 2\}$, $X \neq Y$, $Y \neq Z$, $Z \neq X$
    - je AC, ale není PC, zdůvodnění: X=0, Y=1 nelze rozšířit po cestě (X,Z,Y)
- AC vyřazuje nekompatibilní prvky z domény proměnných. Co dělá PC?
  - PC vyřazuje dvojice hodnot
  - PC si pamatuje podmínky explicitně
  - PC si pamatuje, které dvojice hodnot jsou v relaci
  - PC dělá všechny relace nad dvojicemi implicitní (A<B, B<C $\Rightarrow$ A+1<C)

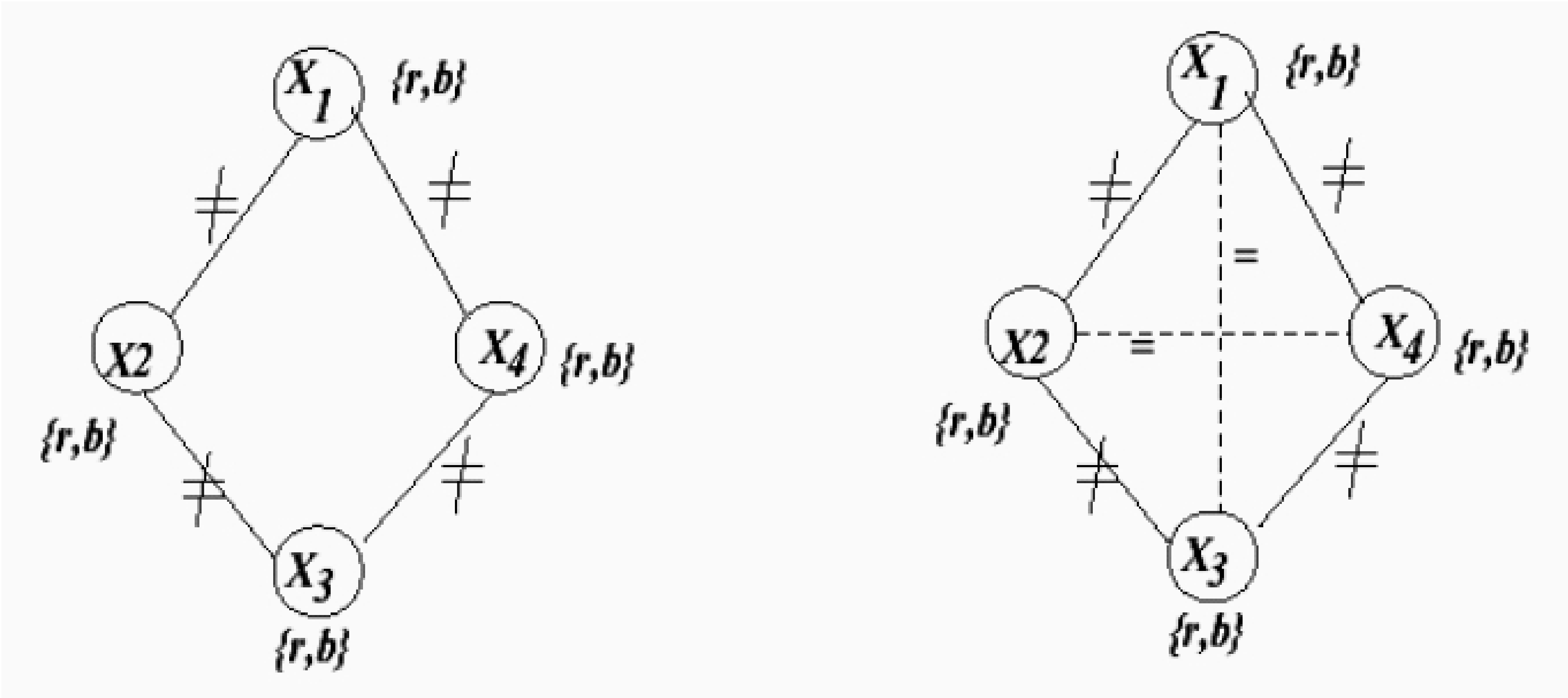
Příklad: barvení grafu

Nalezněte obarvení vrcholů grafu tak, aby sousední vrcholy neměly stejnou barvu.



Příklad: barvení grafu

Nalezněte obarvení vrcholů grafu tak, aby sousední vrcholy neměly stejnou barvu.



není PC konzistentní

lze přiřadit $X_1 = r, X_3 = b$

je PC konzistentní

Algoritmus revize cesty

- Jak udělat každou cestu z V_i do V_j přes V_m konzistentní?
  - Pro dvojici proměnných V_i a V_j aktualizuji prvky relace R_{ij}
  - Vyřadím dvojici (a, b) z R_{ij}, pokud neexistuje taková hodnota $c \in V_m$, že (a, c) je kozistentní pro R_{im} a (c, b) je kozistentní pro R_{mj}

Algoritmus revize cesty

- Jak udělat každou cestu z V_i do V_j přes V_m konzistentní?
  - Pro dvojici proměnných V_i a V_j aktualizuji prvky relace R_{ij}
  - Vyřadím dvojici (a, b) z R_{ij}, pokud neexistuje taková hodnota $c \in V_m$, že (a, c) je kozistentní pro R_{im} a (c, b) je kozistentní pro R_{mj}
- procedure revise-3(i, m, j)
  Deleted := false
  for $\forall\,(a, b) \in R_{ij}$ do
  if neexistuje $c \in D_m$ takové, že $(a, c) \in R_{im}$ a $(c, b) \in R_{mj}$
  then smaž (a, b) z R_{ij}
  Deleted := true
  return Deleted
  end revise-3

$V_1, V_2 \in \{$a,b$\}$, $V_3 \in \{$a,b,c$\}$, $\underline{V_1 \neq V_2}, \underline{V_2 \neq V_3}$, $V_1 \neq V_3$

revise-3$(1, 2, 3)$ (V_1, V_3) :

Algoritmus revize cesty

- Jak udělat každou cestu z V_i do V_j přes V_m konzistentní?
  - Pro dvojici proměnných V_i a V_j aktualizuji prvky relace R_{ij}
  - Vyřadím dvojici (a,b) z R_{ij}, pokud neexistuje taková hodnota $c \in V_m$, že (a,c) je kozistentní pro R_{im} a (c,b) je kozistentní pro R_{mj}
- ```
  procedure revise-3(i, m, j)
  Deleted := false
  for ∀ (a,b) ∈ R_ij do
      if neexistuje c ∈ D_m takové, že (a,c) ∈ R_im a (c,b) ∈ R_mj
      then smaž (a,b) z R_ij
           Deleted := true
  return Deleted
  end revise-3
  ```

$V_1, V_2 \in \{a,b\}$, $V_3 \in \{a,b,c\}$, $V_1 \neq V_2$, $V_2 \neq V_3$, $V_1 \neq V_3$

revise-3$(1,2,3)$ (V_1,V_3) : ~~aa~~ ~~ab~~ ac

~~ba~~ ~~bb~~ bc

pozor: aa bb nejsou už v relaci R_{13}
  ```

Algoritmus revize cesty

- Jak udělat každou cestu z V_i do V_j přes V_m konzistentní?
  - Pro dvojici proměnných V_i a V_j aktualizuji prvky relace R_{ij}
  - Vyřadím dvojici (a, b) z R_{ij}, pokud neexistuje taková hodnota $c \in V_m$, že (a, c) je kozistentní pro R_{im} a (c, b) je kozistentní pro R_{mj}
- procedure revise-3(i, m, j)

  Deleted := false

  for $\forall\, (a, b) \in R_{ij}$ do

      if neexistuje $c \in D_m$ takové, že $(a, c) \in R_{im}$ a $(c, b) \in R_{mj}$

      then smaž (a, b) z R_{ij}

          Deleted := true

  return Deleted

  end revise-3

  $V_1, V_2 \in \{a,b\}$, $V_3 \in \{a,b,c\}$, $\underline{V_1 \neq V_2}$, $\underline{V_2 \neq V_3}$, $V_1 \neq V_3$

  revise-3$(1,2,3)$ (V_1, V_3) : ~~aa~~ ~~ab~~ ac

  ~~ba~~ ~~bb~~ bc

  pozor: aa bb nejsou už v relaci R_{13}
- Složitost $O(k^3)$ (k maximální velikost domény) $\Longleftarrow$

  cyklus pro každou dvojici (a, b): $O(k^2)$, vnitřní cyklus přes $c \in D_j$: $O(k)$

Algoritmus PC-1

- Jak udělat CSP konzistentní po cestě?

Algoritmus PC-1

- Jak udělat CSP konzistentní po cestě? Provedeme revizi každé cesty délky 2
- Revize cesty odstraní dvojice ⇒ již zrevidované cesty opět nekonzistentní
- Revize cesty opakujeme dokud jsou nějaké dvojice smazány
- Princip je podobný jako u AC-1
- Před spuštěním algoritmu nutno **inicializovat relace R_{ij}** pomocí existujících binárních (i unárních) podmínek
- procedure PC-1(G)

```
repeat Changed := false
       for m := 1 to n
           for i := 1 to n
               for j := 1 to n
                   Changed := revise-3(i,m,j) or Changed
until not(Changed)
end PC-1
```

Složitost algoritmu PC-1

- Celková složitost $O(n^5k^5)$ odvození podobné jako pro AC-1

Složitost algoritmu PC-1

- Celková složitost $O(n^5k^5)$ odvození podobné jako pro AC-1

  $\Leftarrow$ 3 cykly pro trojice hodnot $O(n^3)$

  $\Leftarrow$ `revise-3` $O(k^3)$

  $\Leftarrow$ $O(n^2k^2)$ počet opakování `repeat` cyklu

  $\Leftarrow$ jeden cyklus smaže (v nejhorším případě) právě jednu dvojici hodnot celkem n^2k^2 hodnot (n^2 dvojic proměnných, k^2 dvojic hodnot pro každou dvojici proměnných)

- Neefektivita PC-1

Složitost algoritmu PC-1

- Celková složitost $O(n^5k^5)$ odvození podobné jako pro AC-1

  $\Leftarrow$ 3 cykly pro trojice hodnot $O(n^3)$

  $\Leftarrow$ `revise-3` $O(k^3)$

  $\Leftarrow$ $O(n^2k^2)$ počet opakování repeat cyklu

  $\Leftarrow$ jeden cyklus smaže (v nejhorším případě) právě jednu dvojici hodnot celkem n^2k^2 hodnot (n^2 dvojic proměnných, k^2 dvojic hodnot pro každou dvojici proměnných)

- Neefektivita PC-1
  - opakovaná revize všech cest, i když pro ně nedošlo k žádné změně

Složitost algoritmu PC-1

- Celková složitost $O(n^5k^5)$ odvození podobné jako pro AC-1

  $\Leftarrow$ 3 cykly pro trojice hodnot $O(n^3)$

  $\Leftarrow$ `revise-3` $O(k^3)$

  $\Leftarrow$ $O(n^2k^2)$ počet opakování repeat cyklu

  $\Leftarrow$ jeden cyklus smaže (v nejhorším případě) právě jednu dvojici hodnot celkem n^2k^2 hodnot (n^2 dvojic proměnných, k^2 dvojic hodnot pro každou dvojici proměnných)

- Neefektivita PC-1
  - opakovaná revize všech cest, i když pro ně nedošlo k žádné změně
    - při revizi stačí kontrolovat jen zasažené cesty podobně jako pro PC-1

Složitost algoritmu PC-1

- Celková složitost $O(n^5k^5)$ odvození podobné jako pro AC-1

  $\Leftarrow$ 3 cykly pro trojice hodnot $O(n^3)$

  $\Leftarrow$ revise-3 $O(k^3)$

  $\Leftarrow$ $O(n^2k^2)$ počet opakování repeat cyklu

  $\Leftarrow$ jeden cyklus smaže (v nejhorším případě) právě jednu dvojici hodnot celkem n^2k^2 hodnot (n^2 dvojic proměnných, k^2 dvojic hodnot pro každou dvojici proměnných)

- Neefektivita PC-1
  - opakovaná revize všech cest, i když pro ně nedošlo k žádné změně
    - při revizi stačí kontrolovat jen zasažené cesty podobně jako pro PC-1
  - cesty stačí brát pouze s jednou orientací ... R_{ij} je totéž co R_{ji}
    - příklad: $V_1, V_2 \in \{a, b\}, V_3 \in \{a, b, c\}, V_1 \neq V_2, V_2 \neq V_{,3} V_1 \neq V_3$

      důsledek revise-3$(1, 2, 3)$ a revise-3$(3, 2, 1)$ je totožný

      $\Longleftarrow$ ke každé dvojici hodnot z V_1, V_3 (V_3, V_1) hledám kompatibilní hodnotu z V_2

Algoritmus PC-2

- Cesty beru pouze s jednou orientací
  - aktualizuji pouze jednu z R_{ij}, R_{ji}
- Do fronty dávám pouze zasažené cesty
  - podobná modifikace jako AC-3
  - `revise-3`(i, m, j) mění R_{ij},

Algoritmus PC-2

- Cesty beru pouze s jednou orientací
  - aktualizuji pouze jednu z R_{ij}, R_{ji}
- Do fronty dávám pouze zasažené cesty
  - podobná modifikace jako AC-3
  - revise-3(i, m, j) mění R_{ij}, stačí tedy aktualizovat R_{li} přes j a R_{lj} přes i
- Před spuštěním algoritmu
  - inicializovat relace R_{ij} pomocí existujících binárních (i unárních) podmínek
- procedure PC-2(G)

```
Q := {(i,m,j) | 1 ≤ i < j ≤ n, 1 ≤ m ≤ n, m ≠ i, m ≠ j}  //seznam cest pro revizi
while Q non empty do
     vyber a smaž trojici (i,m,j) z Q
     if revise-3(i,m,j) then
        Q := Q ∪ {(l,i,j)(l,j,i) | 1 ≤ l ≤ n, l ≠ i, l ≠ j}
                //jako u AC přidáváme jen cesty, co ještě nejsou ve frontě
end while
end PC-2
```

Složitost algoritmu PC-2

- Složitost $O(n^3k^5)$

Složitost algoritmu PC-2

- Složitost $O(n^3k^5)$
  - každé (i, m, j) může být ve frontě maximálně k^2 krát $\Longrightarrow O(k^2)$

    $\Leftarrow$ když je (i, m, j) přidáno do fronty, R_{ij} bylo redukováno alespoň o jednu hodnotu
  - celkem n^3 trojic $(i, m, j) \Longrightarrow O(n^3)$
  - `revise-3` $O(k^3)$

Složitost algoritmu PC-2

- Složitost $O(n^3k^5)$
  - každé (i, m, j) může být ve frontě maximálně k^2 krát $\Longrightarrow O(k^2)$

    $\Leftarrow$ když je (i, m, j) přidáno do fronty, R_{ij} bylo redukováno alespoň o jednu hodnotu
  - celkem n^3 trojic $(i, m, j) \Longrightarrow O(n^3)$
  - `revise-3` $O(k^3)$
- Algoritmus není optimální podobně jako AC-3
  - existuje algoritmus PC-4 založen na počítání podpor
  - složitost PC-4 je $O(n^3k^3)$, což už je optimální

Složitost algoritmu PC-2

- Složitost $O(n^3k^5)$
  - každé (i, m, j) může být ve frontě maximálně k^2 krát $\Longrightarrow O(k^2)$

    $\Leftarrow$ když je (i, m, j) přidáno do fronty, R_{ij} bylo redukováno alespoň o jednu hodnotu
  - celkem n^3 trojic $(i, m, j) \Longrightarrow O(n^3)$
  - `revise-3` $O(k^3)$
- Algoritmus není optimální podobně jako AC-3
  - existuje algoritmus PC-4 založen na počítání podpor
  - složitost PC-4 je $O(n^3k^3)$, což už je optimální
- Cvičení: řešte následující problém pomocí PC-2 algoritmu

  $V1 \in \{0, 1, 2, 3\}, V2 \in \{0, 1\}, V3 \in \{1, 2\}$

  $V3 = V1 + 1, V2 \neq V3, V1 \neq V2$

Složitost algoritmu PC-2

- Složitost $O(n^3k^5)$
  - každé (i, m, j) může být ve frontě maximálně k^2 krát $\Longrightarrow O(k^2)$

    $\Leftarrow$ když je (i, m, j) přidáno do fronty, R_{ij} bylo redukováno alespoň o jednu hodnotu
  - celkem n^3 trojic $(i, m, j) \Longrightarrow O(n^3)$
  - `revise-3` $O(k^3)$
- Algoritmus není optimální podobně jako AC-3
  - existuje algoritmus PC-4 založen na počítání podpor
  - složitost PC-4 je $O(n^3k^3)$, což už je optimální
- Cvičení: řešte následující problém pomocí PC-2 algoritmu

  $V1 \in \{0, 1, 2, 3\}, V2 \in \{0, 1\}, V3 \in \{1, 2\}$

  $V3 = V1 + 1, V2 \neq V3, V1 \neq V2$
  - Nápověda: zkonstruuj iniciální relace R_{ij}

Složitost algoritmu PC-2

- Složitost $O(n^3k^5)$
  - každé (i, m, j) může být ve frontě maximálně k^2 krát $\Longrightarrow O(k^2)$

    $\Leftarrow$ když je (i, m, j) přidáno do fronty, R_{ij} bylo redukováno alespoň o jednu hodnotu
  - celkem n^3 trojic $(i, m, j) \Longrightarrow O(n^3)$
  - `revise-3` $O(k^3)$
- Algoritmus není optimální podobně jako AC-3
  - existuje algoritmus PC-4 založen na počítání podpor
  - složitost PC-4 je $O(n^3k^3)$, což už je optimální
- Cvičení: řešte následující problém pomocí PC-2 algoritmu

  $V1 \in \{0, 1, 2, 3\}, V2 \in \{0, 1\}, V3 \in \{1, 2\}$

  $V3 = V1 + 1, V2 \neq V3, V1 \neq V2$
  - Nápověda: zkonstruuj iniciální relace R_{ij}

    iniciálně přidej do fronty $(V1, V2, V3), (V1, V3, V2), (V2, V1, V3)$

Omezení PC algoritmů

- Paměťové nároky
  - protože PC eliminuje dvojice hodnot z omezení, potřebuje používat extenzionální reprezentaci omezení (pro každou dvojici hodnot si pamatuji, zda je/není v doméně)
- Poměr výkon/cena
  - PC eliminuje více (nebo stejně) nekonzistencí jako AC
  - poměr výkonu ke zjednodušení problému je ale mnohem horší než u AC
- Změny grafu omezení
  - PC přidává hrany (omezení) i tam, kde původně nebyly a mění tak konektivitu grafu
  - to vadí při dalším řešení problému, kdy se nemohou používat heuristiky odvozené od grafu (resp. dané původním problémem)
- PC stále není dostatečné

Omezení PC algoritmů

- Paměťové nároky
  - protože PC eliminuje dvojice hodnot z omezení, potřebuje používat extenzionální reprezentaci omezení (pro každou dvojici hodnot si pamatuji, zda je/není v doméně)
- Poměr výkon/cena
  - PC eliminuje více (nebo stejně) nekonzistencí jako AC
  - poměr výkonu ke zjednodušení problému je ale mnohem horší než u AC
- Změny grafu omezení
  - PC přidává hrany (omezení) i tam, kde původně nebyly a mění tak konektivitu grafu
  - to vadí při dalším řešení problému, kdy se nemohou používat heuristiky odvozené od grafu (resp. dané původním problémem)
- PC stále není dostatečné
  - $V, X, Y, Z \in \{1, 2, 3\}, \quad X \neq Y, \quad Y \neq Z, \quad Z \neq X, \quad V \neq X, \quad V \neq Y, \quad V \neq Z$
    je PC a přesto nemá řešení

Na půli cesty od AC k PC

- Jak oslabit PC, aby algoritmus:
  - neměl paměťové nároky PC
  - neměnil graf podmínek
  - byl silnější než AC?

Na půli cesty od AC k PC

- Jak oslabit PC, aby algoritmus:
  - neměl paměťové nároky PC
  - neměnil graf podmínek
  - byl silnější než AC?
- **Omezená konzistence po cestě – *Restricted Path Consistency* (RPC)**
  - testujeme PC jen v případě, když je šance, že to povede k **vyřazení hodnoty z domény proměnné**
- Příklad: $V_1, V_2 \in \{a, b\}, V_3 \in \{a, b, c\}, V_1 \neq V_2, V_2 \neq V_3, V_1 \neq V_3$

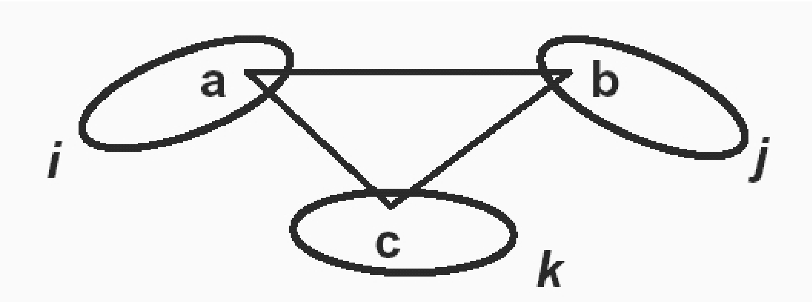
  je AC ale není PC

  RPC odstraní z domény V_3 hodnoty a, b

Omezená konzistence po cestě (RPC)

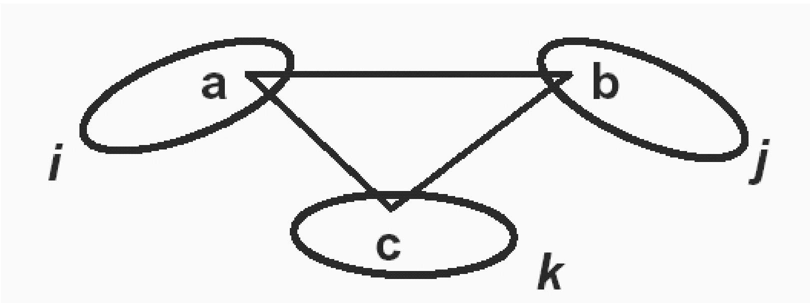
- PC hrany se testuje pouze tehdy, pokud vyřazení dvojice může vést k vyřazení některého z prvků z domény příslušné proměnné
- Jak to poznáme? Jedná se o **jedinou vzájemnou podporu**.
- Proměnná V_i je **omezeně konzistentní po cestě** ***(Restricted Path Consistent, RPC)***:
  - každá hrana vedoucí z V_i je hranově konzistentní
  - pro každé $a \in D_i$ platí:

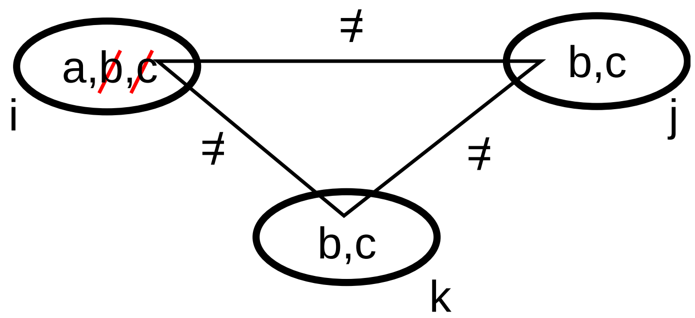
    je-li b jediná podpora a ve vrcholu j, potom v každém vrcholu k (spojenem s i a j) existuje hodnota c tak, že (a, c) a (b, c) jsou kompatibilní s příslušnými podmínkami



Omezená konzistence po cestě (RPC)

- PC hrany se testuje pouze tehdy, pokud vyřazení dvojice může vést k vyřazení některého z prvků z domény příslušné proměnné
- Jak to poznáme? Jedná se o **jedinou vzájemnou podporu**.
- Proměnná V_i je **omezeně konzistentní po cestě** ***(Restricted Path Consistent, RPC)***:
  - každá hrana vedoucí z V_i je hranově konzistentní
  - pro každé $a \in D_i$ platí:
    je-li b jediná podpora a ve vrcholu j, potom v každém vrcholu k (spojenem s i a j) existuje hodnota c tak, že (a, c) a (b, c) jsou kompatibilní s příslušnými podmínkami





- Algoritmus: založen na AC-4 + seznam cest pro PC (viz Barták přednáška)

Propagace pro nebinární omezení

Nebinární omezení

- Zobecnění principů NC, AC, a PC směrem ke **k-konzistenci**
  - zajímavé z pohledu složitosti řešení problémů
  - pracuje se s libovolnými k-ticemi proměnných
  - v praxi se vzhledem k paměťové a časové složitosti nevyužívá

Nebinární omezení

- Zobecnění principů NC, AC, a PC směrem ke **k-konzistenci**
  - zajímavé z pohledu složitosti řešení problémů
  - pracuje se s libovolnými k-ticemi proměnných
  - v praxi se vzhledem k paměťové a časové složitosti nevyužívá
- **Obecné typy konzistence** pro nebinární podmínky
  - pracuje se přímo s n-árními podmínkami
  - příklad: obecná hranová konzistence, konzistence mezí

Nebinární omezení

- Zobecnění principů NC, AC, a PC směrem ke **k-konzistenci**
  - zajímavé z pohledu složitosti řešení problémů
  - pracuje se s libovolnými k-ticemi proměnných
  - v praxi se vzhledem k paměťové a časové složitosti nevyužívá
- **Obecné typy konzistence** pro nebinární podmínky
  - pracuje se přímo s n-árními podmínkami
  - příklad: obecná hranová konzistence, konzistence mezí
- **Globální omezení:** specifické typy konzistencí
  - využívá se sémantika omezení, zaměřené opět na konkrétní omezení
  - speciální typy konzistence pro globální omezení (př. `all_different`)

Nebinární omezení

- Zobecnění principů NC, AC, a PC směrem ke **k-konzistenci**
  - zajímavé z pohledu složitosti řešení problémů
  - pracuje se s libovolnými k-ticemi proměnných
  - v praxi se vzhledem k paměťové a časové složitosti nevyužívá
- **Obecné typy konzistence** pro nebinární podmínky
  - pracuje se přímo s n-árními podmínkami
  - příklad: obecná hranová konzistence, konzistence mezí
- **Globální omezení:** specifické typy konzistencí
  - využívá se sémantika omezení, zaměřené opět na konkrétní omezení
  - speciální typy konzistence pro globální omezení (př. `all_different`)
- Pro různé podmínky lze použít různý druh konzistence
  - A<B: hranová konzistence, konzistence mezí

Obsah: propagace pro nebinární omezení

- k-konzistence
- Obecná hranová konzistence
- Konzistence mezí
  - konzistence mezí pro aritmetická omezení
- Globální podmínky
  - klasické typy podmínek a příklady jejich použití
  - propagace pro `all_different` a pro rozvrhování s unárními zdroji
- Obecný konzistenční algoritmus pro nebinární podmínky
  - v jeho rámci lze využívat výše zmíněné typy konzistence

k-konzistence

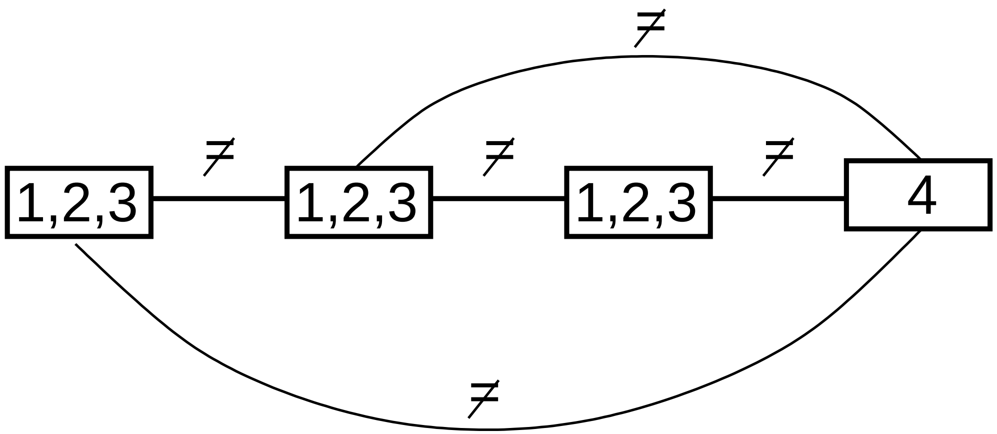
- Mají AC a PC něco společného?
  - AC: rozšiřujeme jednu hodnotu do druhé proměnné
  - PC: rozšířujeme dvojici hodnot do třetí proměnné
  - . . . můžeme pokračovat

k-konzistence

- Mají AC a PC něco společného?
  - AC: rozšiřujeme jednu hodnotu do druhé proměnné
  - PC: rozšířujeme dvojici hodnot do třetí proměnné
  - ... 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 AC a PC něco společného?
  - AC: rozšiřujeme jednu hodnotu do druhé proměnné
  - PC: rozšiřujeme dvojici hodnot do třetí proměnné
  - . . . můžeme pokračovat
- CSP je **k-konzistentní** právě tehdy, když můžeme libovolné konzistentní ohodnocení (k-1) různých proměnných rozšířit do libovolné k-té proměnné

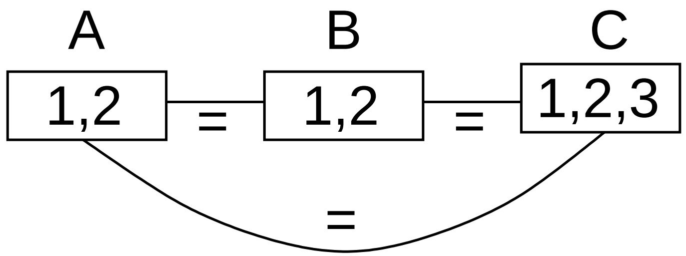


4-konzistentní graf

- Pro obecné CSP, tedy i pro **nebinární podmínky**

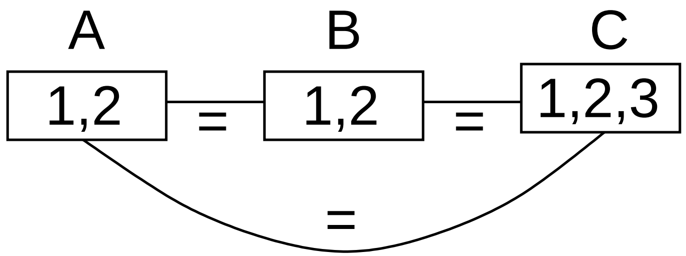
Silná k-konzistence

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 $(2,2,2)$

není 2-konzistentní

(3) nelze rozšířit

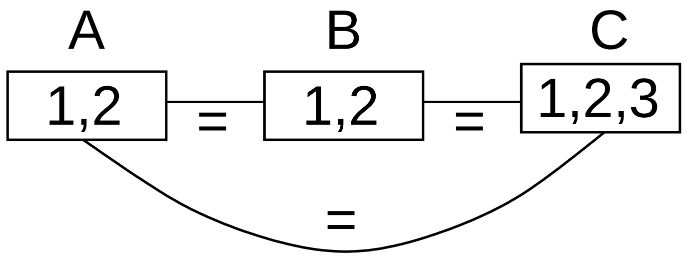
$(1,3)$ ani $(2,3)$ nejsou konzistentní dvojice (nerozšiřujeme je)

Silná k-konzistence

3-konzistentní graf

$(1,1)$ lze rozšířit na $(1,1,1)$

$(2,2)$ lze rozšířit na $(2,2,2)$



není 2-konzistentní

(3) nelze rozšířit

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

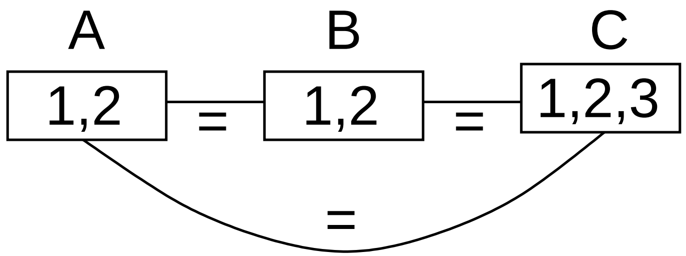
- CSP je **silně k-konzistentní** právě tehdy, když je j-konzistentní pro každé j≤k

Silná k-konzistence

3-konzistentní graf

$(1,1)$ lze rozšířit na $(1,1,1)$

$(2,2)$ lze rozšířit na $(2,2,2)$



není 2-konzistentní

(3) nelze rozšířit

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

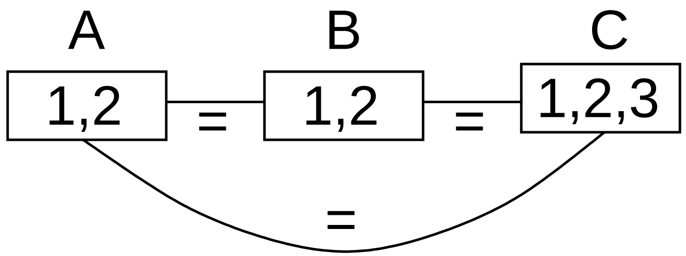
- CSP je **silně k-konzistentní** právě tehdy, když je j-konzistentní pro každé j≤k
- Silná k-konzistence $\Rightarrow$ k-konzistence
- Silná k-konzistence $\Rightarrow$ j-konzistence $\forall j \leq k$
- k-konzistence $\not\Rightarrow$ silná k-konzistence

Silná k-konzistence

3-konzistentní graf

$(1,1)$ lze rozšířit na $(1,1,1)$

$(2,2)$ lze rozšířit na $(2,2,2)$



není 2-konzistentní

(3) nelze rozšířit

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

- CSP je **silně k-konzistentní** právě tehdy, když je j-konzistentní pro každé j≤k

- Silná k-konzistence ⇒ k-konzistence

- Silná k-konzistence ⇒ j-konzistence $\forall j \leq k$

- k-konzistence $\not\Rightarrow$ silná k-konzistence

- NC = silná 1-konzistence = 1-konzistence

- AC = (silná) 2-konzistence

- PC = (silná) 3-konzistence

- **Cvičení:** uveďte příklad problému, který je 4-konzistentní, ale není 3-konzistentní

Konzistence pro nalezení řešení

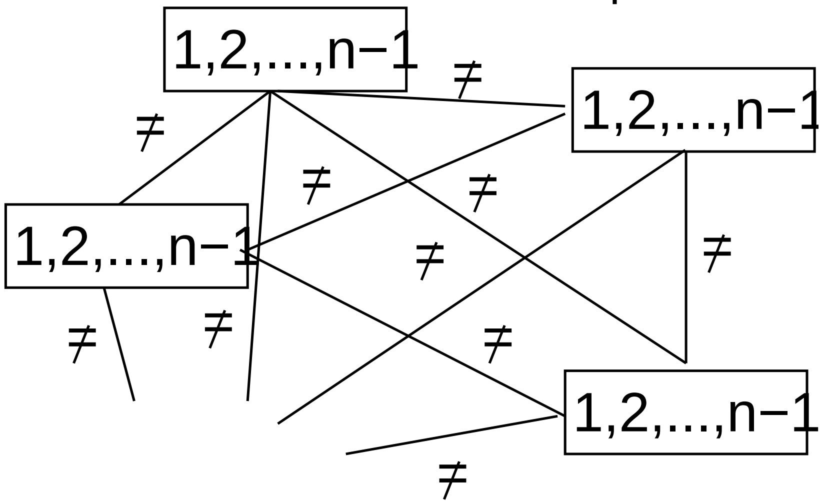
- Máme-li graf s n vrcholy, jak silnou konzistenci potřebujeme, abychom přímo našli řešení?
- **CSP vyřešíme bez navracení** vzhledem k uspořádání proměnných $(x_1, \ldots, x_n)$, jestliže pro každé $i \le n$ může být každé částečné řešení $(x_1, \ldots, x_i)$ konzistentně rozšířeno o proměnnou x_{i+1}.

Konzistence pro nalezení řešení

- Máme-li graf s n vrcholy, jak silnou konzistenci potřebujeme, abychom přímo našli řešení?
- **CSP vyřešíme bez navracení** vzhledem k uspořádání proměnných $(x_1, \ldots, x_n)$, jestliže pro každé $i \leq n$ může být každé částečné řešení $(x_1, \ldots, x_i)$ konzistentně rozšířeno o proměnnou x_{i+1}.
- Nalezení řešení bez navracení pro libovolné uspořádání proměnných?

Konzistence pro nalezení řešení

- Máme-li graf s n vrcholy, jak silnou konzistenci potřebujeme, abychom přímo našli řešení?
- **CSP vyřešíme bez navracení** vzhledem k uspořádání proměnných $(x_1, \ldots, x_n)$, jestliže pro každé $i \leq n$ může být každé částečné řešení $(x_1, \ldots, x_i)$ konzistentně rozšířeno o proměnnou x_{i+1}.
- Nalezení řešení bez navracení pro libovolné uspořádání proměnných?
  - **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

Algoritmy pro dosažení k-konzistence

- Rozšíření revize hrany a revize cesty
  - postupně odstraňujeme prvky z relace nad (k-1) proměnnými
- Aktualizujeme relace nad každou (k-1)-ticí proměnných
  - musíme si pamatovat (k-1)-tice hodnot

Algoritmy pro dosažení k-konzistence

- Rozšíření revize hrany a revize cesty
  - postupně odstraňujeme prvky z relace nad (k-1) proměnnými
- Aktualizujeme relace nad každou (k-1)-ticí proměnných
  - musíme si pamatovat (k-1)-tice hodnot
- Obecný algoritmus
  - rozšíření AC-1 a PC-1
  - opakování revizí nad (k-1)-ticemi dokud dochází ke změnám
- Velká paměťová i časová složitost
  - v praxi se pro vyšší k nepoužívá
- Algoritmy i složitost viz Dechter: Constraint Processing