Programování
s omezujícími podmínkami

6

Roman Barták
Katedra teoretické informatiky a matematické logiky

roman.bartak@mff.cuni.cz
http://ktiml.mff.cuni.cz/~bartak

Směrová hranová konzistence

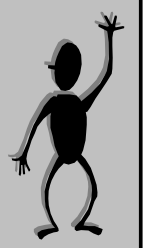
Definice:

CSP je **směrově hranově konzistentní (directional arc consistent)** při daném uspořádání proměnných, právě když každá hrana (i,j), kde i<j, je hranově konzistentní.

- Opět kontrolujeme každou hranu, tentokrát v jednom směru.
- Pokud hrany projdeme v dobrém pořadí, nemusíme revize opakovat!

Algoritmus DAC-1

```
procedure DAC-1(G)
    for j = |nodes(G)| to 1 by -1 do
        for each arc (i,j) in G such that i<j do
            REVISE((i,j))
            if Di=∅ then stop with fail
        end for
    end for
end DAC-1
```



Vztah DAC k AC

Pozorování:

CSP je hranově konzistentní, jestliže pro dané uspořádání proměnných je směrově hranově konzistentní v obou směrech.

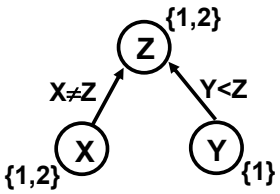
Můžeme AC dosáhnout tak, že aplikujeme DAC v obou směrech?

Obecně NE, ale…

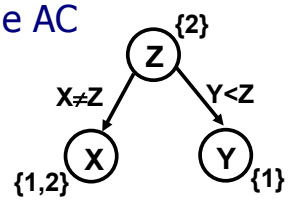
Příklad:

X in {1,2}, Y in {1}, Z in {1,2}, $X \neq Z, Y < Z$

při uspořádání X,Y,Z se domény nezmění



při uspořádání Z,Y,X se změní pouze Z, ale nedostane AC



Pokud ale jako první zvolíme uspořádání Z,Y,X, dostaneme AC.

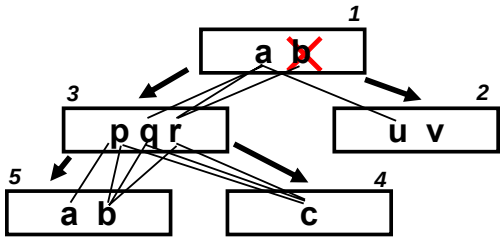
Od DAC k AC

pro stromové CSP

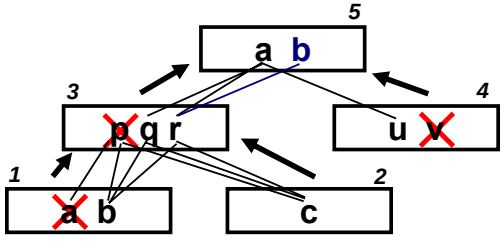
Pokud je DAC aplikováno na stromové CSP nejprve pro uspořádání vrcholů od kořene a po té v obráceném pořadí, potom získáme (plnou) hranovou konzistenci.

Důkaz:

po prvním běhu DAC zajistíme, že pro každou hodnotu rodičovského vrcholu najdeme podporu (konzistentní hodnotu) u všech potomků



pokud je **při druhém běhu DAC** (v opačném pořadí) vyřazena nějaká hodnota, potom tato hodnota nebyla podporou žádné hodnoty rodičovského uzlu (tj. hodnoty v rodičovském uzlu neztratily své podpory)



dohromady: každá hodnota má podporu ve všech potomcích (první běh) i v rodiči (druhý běh), tj. jedná se o AC

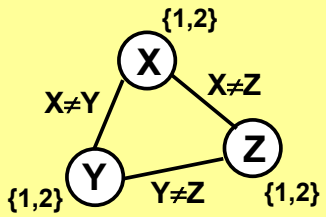
Je AC dostatečné?

Použitím AC odstraníme mnoho nekompatibilních hodnot.

- **Dostaneme potom řešení problému?**
- **Víme alespoň zda řešení existuje?**

NE a NE!

Příklad:



CSP je hranově konzistentní a přesto nemá žádné řešení

K čemu tedy AC je?

- někdy **dá řešení přímo**
  - nějaká doména se vyprázdní → řešení neexistuje
  - všechny domény jsou jednoprvkové → máme řešení
- v obecném případě alespoň **zmenší prohledávaný prostor**

Konzistence po cestě

Jak posílit konzistenci?

Budeme se zabývat několika podmínkami najednou!

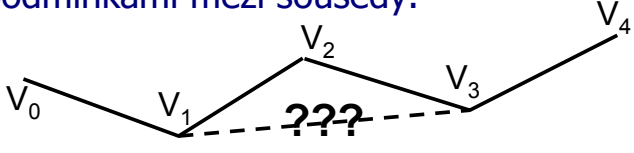
Definice:

Cesta ($V_0,V_1,\ldots, V_m$) je **konzistentní** (**path consistent**), právě když pro každou dvojici hodnot $x \in D_0$ a $y \in D_m$ splňující binární podmínky na V_0,V_m existuje ohodnocení proměnných $V_1,\ldots,V_{m-1}$ takové, že všechny binární podmínky mezi sousedy V_i,V_{i+1} jsou splněny.

CSP je **konzistentní po cestě**, právě když všechny cesty jsou konzistentní.

Pozor!

Definice PC nezaručuje, že jsou splněny všechny podmínky nad vrcholy cesty, zabývá se pouze podmínkami mezi sousedy.



PC a cesty délky 2

Zjišťovat konzistenci všech cest není moc praktické.
Naštěstí nám stačí prozkoumat pouze cesty délky 2!

Tvrzení: CSP je PC, právě 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 triviálně platí

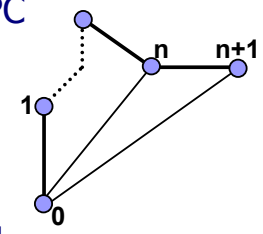
b) N+1 (za předpokladu, že N platí)

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

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

iii) podle a) najdeme hodnotu $x_n \in D_n$ tž. $P_{0,n}$ a $P_{n,n+1}$ platí

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



Vztah PC a AC

Je AC pokryto pomocí PC (je-li CSP PC, potom je i AC)?

- hrana (i, j) je konzistentní (AC), pokud je cesta (i,j,i) konzistentní (PC)
- PC tedy implikuje AC

Je PC silnější než AC (existuje CSP, které je AC a není PC)?

Příklad: X in {1,2}, Y in {1,2}, Z in {1,2}, $X \neq Z$, $X \neq Y$, $Y \neq Z$

- je AC, ale není PC (X=1, Z=2 nelze rozšířit po cestě X,Y,Z)

AC vyřazuje nekompatibilní prvky z domén proměnných.

Co bude dělat PC?

- **PC vyřazuje dvojice hodnot**
- PC dělá všechny relace implicitní (A<B,B<C $\Rightarrow$ A+1<C)
- unární podmínka = doména proměnné

Reprezentace podmínek

V PC potřebujeme vyřazovat jednotlivé dvojice hodnot

↳ je potřeba pamatovat si podmínky explicitně

Binární podmínka = {0,1}-matice

0 - hodnoty nejsou kompatibilní

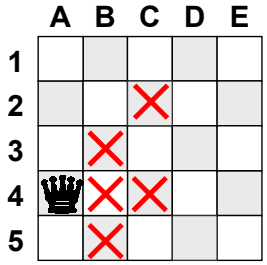
1- hodnoty jsou kompatibilní

Příklad (problém 5-ti dam)

podmínka mezi dámami i a j má tvar: $r(i) \neq r(j)$ & $|i-j| \neq |r(i)-r(j)|$

maticový záznam pro dámy A(1) a B(2)

```
0 0 1 1 1
0 0 0 1 1
1 0 0 0 1
1 1 0 0 0
1 1 1 0 0
```



maticový záznam pro dámy A(1) a C(3)

```
0 1 0 1 1
1 0 1 0 1
0 1 0 1 0
1 0 1 0 1
1 1 0 1 0
```


Operace s podmínkami

Průnik podmínek R_{ij} & R'_{ij}

AND po odpovídajících polích

A<B	&	*A≥B-1*	→	*B-1≤A<B*
011		110		010
001	&	111	=	001
000		111		000

Složení podmínek $R_{ik} * R_{kj} \rightarrow R_{ij}$

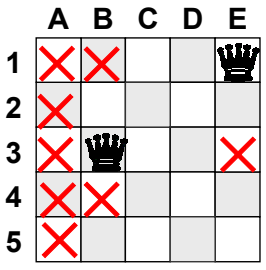
binární násobení matic

A<B	*	*B<C*	→	*A<C-1*
011		011		001
001	*	001	=	000
000		000		000

Indukovaná podmínka se spojuje s původní podmínkou

R_{ij} & (R_{ik} * R_{kj}) → R_{ij}

R_{25}	&	(R_{21}	*	R_{15})	→	R_{25}
01101		00111		01110		01101
10110		00011		10111		10110
11011	&	10001	*	11011	=	01010
01101		11000		11101		01101
10110		11100		01110		10110



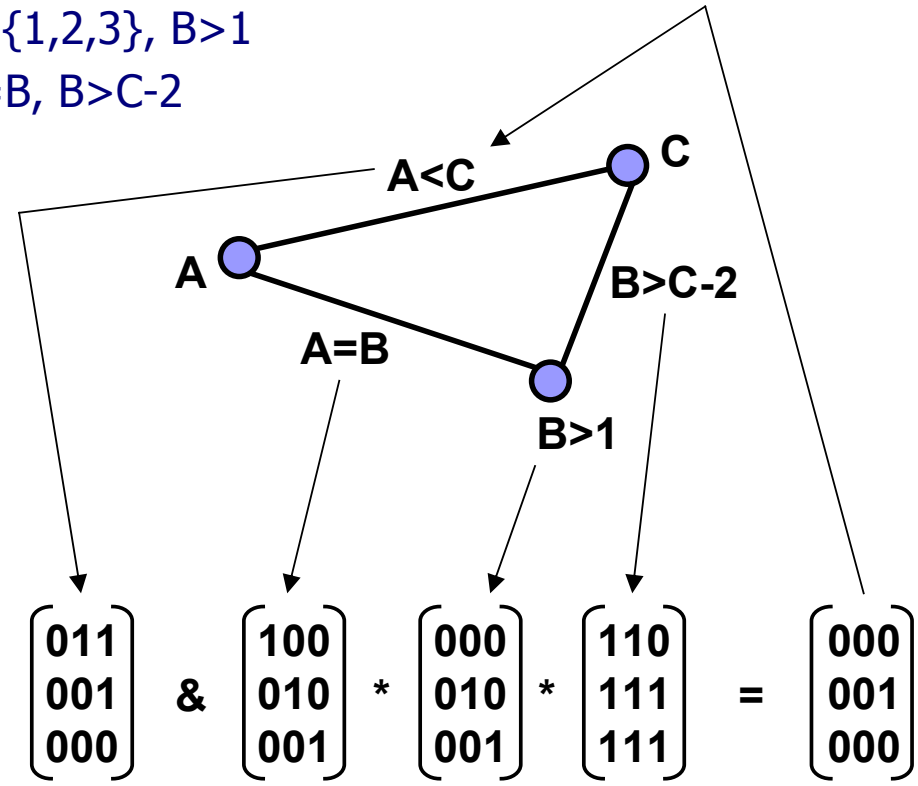
Poznámky:

$R_{ij} = R^T_{ji}$, R_{ii} je diagonální matice reprezentující doménu

REVISE((i,j)) z AC algoritmů dělá $R_{ii} \leftarrow R_{ii}$ & (R_{ij} * R_{jj} * R_{ji})

Skládáme podmínky

A,B,C in {1,2,3}, B>1
A<C, A=B, B>C-2



Algoritmus PC-1

Jak udělat cestu (i,k,j) konzistentní?

$R_{ij} \leftarrow R_{ij} \& (R_{ik} * R_{kk} * R_{kj})$

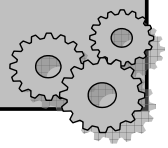
Jak udělat CSP konzistentní?

opakovat konzistenci všech cest (délky 2) dokud se mění domény

Algoritmus PC-1

```
procedure PC-1(Vars,Constraints)
    n ← |Vars|, Y^n ← Constraints
    repeat
        Y^0 ← Y^n
        for k = 1 to n do
            for i = 1 to n do
                for j = 1 to n do
                    Y^k_ij ← Y^(k-1)_ij & (Y^(k-1)_ik * Y^(k-1)_kk * Y^(k-1)_kj)
    until Y^n=Y^0
    Constraints ← Y^0
end PC-1
```

Pokud zde použijeme
$Y^k_{ii} \leftarrow Y^{k-1}_{ii} \& (Y^{k-1}_{ik} * Y^{k-1}_{kk} * Y^{k-1}_{ki})$
dostaneme AC-1



Jak zlepšit PC-1?

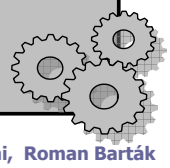
Je snad na PC-1 něco neefektivního?

- pár „drobností"
  - není potřeba držet všechny kopie Y^k stačí jedna kopie a indikátor změny
  - některé výpočty nemají žádný efekt ($Y^k_{kk} = Y^{k-1}_{kk}$)
  - polovinu výpočtů lze ušetřit ($Y_{ji} = Y^T_{ij}$)
- zásadní problém
  - při změně domény podmínky se znova revidují všechny cesty ale stačí procházet jen cesty ovlivněné revizí

Algoritmus revize cesty

```
procedure REVISE_PATH((i,k,j))
    Z ← Y_ij & (Y_ik * Y_kk * Y_kj)
    if Z=Y_ij then return false
    Y_ij ← Z
    return true
end REVISE_PATH
```

Pokud dojde ke změně budeme re-revidovat zasažené cesty



Cesty ovlivněné revizí

Protože $Y_{ji} = Y^T_{ij}$, stačí brát pouze cesty (i,k,j) pro i≤j.

Nechť při revizi cesty (i,k,j) došlo ke změně domény podmínky (i,j)

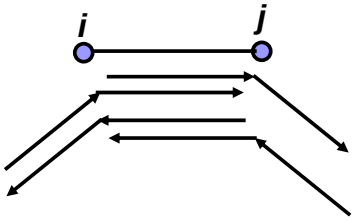
Situace a: i<j

je potřeba znova revidovat cesty obsahující (i,j) nebo (j,i)

cesty (i,j,j) a (i,i,j) není potřeba revidovat (REVISE zde nic nezmění)

$S_a(i,j) = \{(i,j,m) \mid i \le m \le n \;\&\; m \neq j\}$
$\cup \{(m,i,j) \mid 1 \le m \le j \;\&\; m \neq i\}$
$\cup \{(j,i,m) \mid j < m \le n\}$
$\cup \{(m,j,i) \mid 1 \le m < i\}$

$|S_a(i,j)| = 2n-2$



Situace b: i=j

je potřeba znova revidovat cesty obsahující i uvnitř

cesty (i,i,i) a (k,i,k) není potřeba revidovat

$S_b(i,j) = \{(p,i,m) \mid 1 \le m \le n \;\&\; 1 \le p \le m\} - \{(i,i,i),(k,i,k)\}$

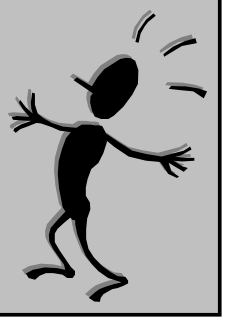
$|S_b(i,j)| = n*(n+1)/2 - 2$

Algoritmus PC-2

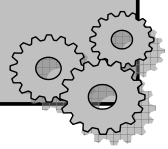
Cesty bereme jen s jednou orientací (pozor neplést s DPC)
Po revizi kontrolujeme jen zasažené cesty

Algoritmus PC-2

```
procedure PC-2(G)
    n ← |nodes(G)|
    Q ← {(i,k,j) | 1 ≤ i ≤ j ≤ n & i≠k & j≠k}
    while Q non empty do
        select and delete (i,k,j) from Q
        if REVISE_PATH((i,k,j)) then
            Q ← Q ∪ RELATED_PATHS((i,k,j))
    end while
end PC-2
```



```
procedure RELATED_PATHS((i,k,j))
    if i<j then return S_a(i,j) else return S_b(i,j)
end RELATED_PATHS
```



Další PC algoritmy

PC-3 (Mohr, Henderson - 1986)

- založen na principu počítání podpor (jako AC-4)
- **algoritmus je chybný!**

  Pokud zjistí, že dvojice (a,b) nemá na hraně (i,j) podporu u další proměnné, vyřadí a z D_i a b z D_j.

PC-4 (Han, Lee - 1988)

- opravuje PC-3 algoritmus
- založen na počítání podpor pro dvojice (b,c) hrany (i,j)

PC-5 (Singh - 1995)

- využívá myšlenky AC-6
- pamatuje si pouze jednu podporu a při její ztrátě hledá další

Směrová konzistence po cestě

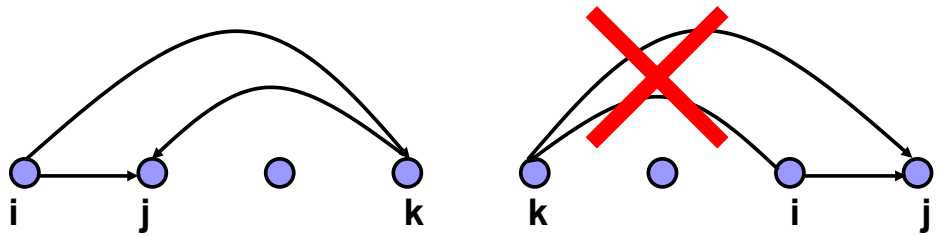
Podobně jako u AC můžeme i složitost PC algoritmů zmenšit zavedením směrovosti cesty.

Definice:

CSP je **směrově konzistentní po cestě** (**directional path consistent**) při daném uspořádání proměnných, právě když všechny cesty (i,k,j), takové, že $i \leq k$ a $j \leq k$, jsou konzistentní.

Poznámky:

- Pozor podmínky $i \leq k$ a $j \leq k$ jsou jiné než $i \leq j$ použité pro odstranění symetrie!
- Podmínku $i \leq j$ můžeme použít i u DPC.



Algoritmus DPC-1

Podobně jako u DAC-1, každou cestu procházíme právě jednou (jdeme od konce).

Navíc můžeme ušetřit díky symetrii podmínek ($i \leq j$).

Algoritmus DPC-1

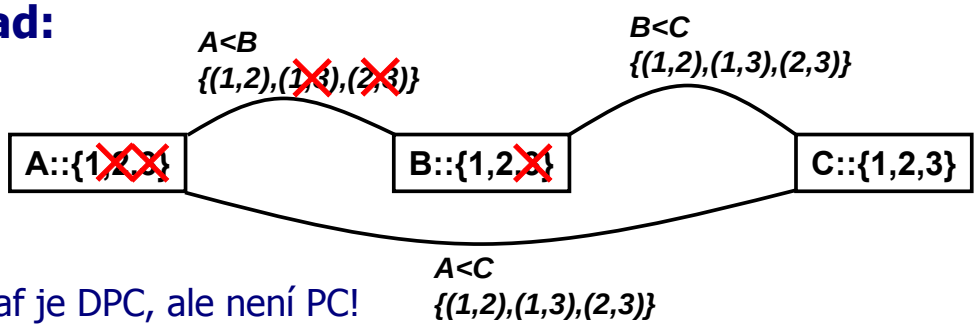
```
procedure DPC-1(Vars,Constraints)
    n ← |Vars|, E ← { (i,j) | i<j & C_{i,j}∈Constraints}
    for k = n to 1 by -1 do
        for i = 1 to k do
            for j = i to k do
                if (i,k)∈E & (j,k)∈E then
                    C_ij ← C_ij & (C_ik * C_kk * C_kj)
                    E ← E ∪ {(i,j)}
            end for
        end for
    end for
end DPC-1
```

Vztah DPC k PC a AC

Zřejmě platí, že PC implikuje DPC.

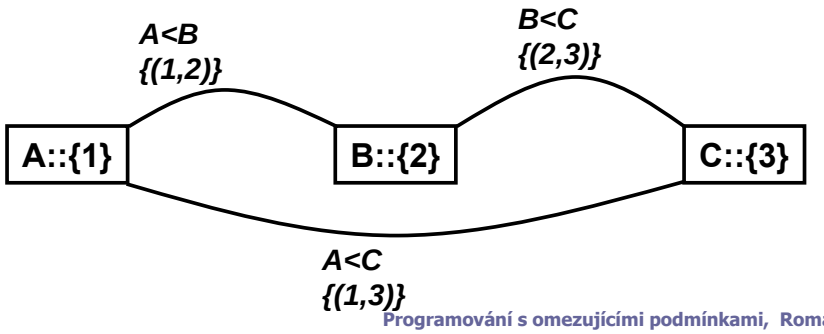
Platí to také naopak?

Příklad:



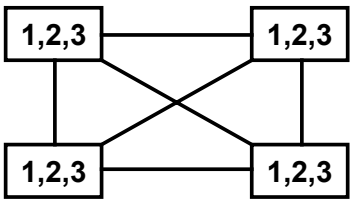
Graf je DPC, ale není PC!
Dokonce není ani AC.

PC a AC graf



Omezení PC algoritmů

- **Paměťové nároky**
  - protože PC eliminuje dvojice hodnot z podmínek, potřebuje používat extenzionální reprezentaci podmínky ({0,1}-matice)
- **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 podmínek**
  - PC přidává hrany (podmínky) 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 stejně není dostatečné**
  - A,B,C,D in {1,2,3}
    A≠B, A≠C, A≠D, B≠C, B≠D, C≠D
    je PC a přesto nemá řešení



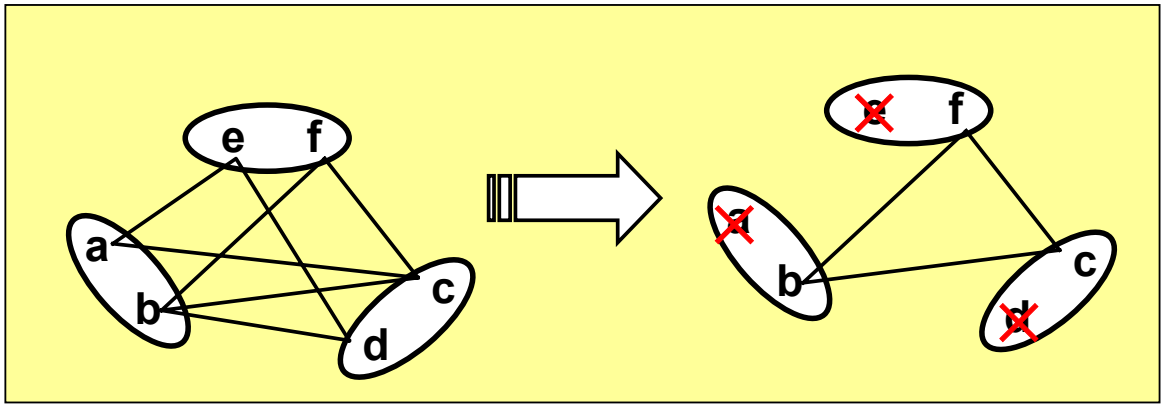
Na půli cesty od AC k PC

Jak oslabit PC, aby algoritmus:

- neměl paměťové nároky PC,
- neměnil graf podmínek,
- byl silnější než AC?

- **Testujeme PC jen v případě, když je šance, že to povede k vyřazení hodnoty z domény proměnné!**

Příklad:



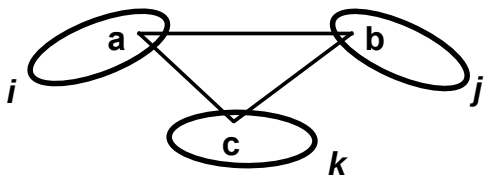
Omezená konzistence po cestě

- 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.

Definice:

Vrchol i je **omezeně konzistentní po cestě** (**restricted path consistent**) právě když:

- každá hrana vedoucí z i je hranově konzistentní (AC)
- pro každé $a \in D_i$ platí:
  je-li b jediná podpora a ve vrcholu j, potom v každém vrcholu k (spojeném s i a j) existuje hodnota c tak, že (a,c) a (b,c) jsou kompatibilní s příslušnými podmínkami (PC).

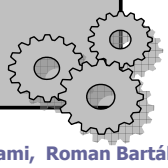


Algoritmus RPC

inicializace

Založeno na AC-4: počítání podpor + seznam cest pro PC

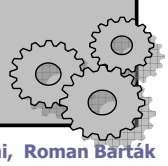
```
procedure INITIALIZE(G)
    Q_AC ← {} , Q_PC ← {} , S ← {}                    % vyprázdnění datových struktur
    for each (i,j)∈arcs(G) do
        for each a∈D_i do
            total ← 0
            for each b∈D_j do
                if (a,b) is consistent according to the constraint C_i,j then
                        total ← total + 1,  S_j,b ← S_j,b ∪ {<i,a>}
            end for
            counter[(i,j),a] ← total
            if counter[(i,j),a] = 0 then
                Q_AC ← Q_AC ∪ {<i,a>},  delete a from D_i
            else if counter[(i,j),a] = 1 then
                    for each k such that (i,k)∈arcs(G) & (k,j)∈arcs(G) do
                        Q_PC ← Q_PC ∪ {(<i,a>,j,k)}
            end if
        end for
    end for
    return (Q_AC, Q_PC)
end INITIALIZE
```



Algoritmus RPC

kontrola AC

```
procedure PRUNE(Q_AC, Q_PC)
    while Q_AC non empty do
        select and delete any pair <j,b> from Q_AC
        for each <i,a> from S_j,b do
            counter[(i,j),a] ← counter[(i,j),a] - 1
            if counter[(i,j),a] = 0 & "a" is still in D_i then
                    delete "a" from D_i
                    Q_AC ← Q_AC ∪ {<i,a>}
            else if counter[(i,j),a] = 1 then
                    for each k such that (i,k)∈arcs(G) & (k,j)∈arcs(G) do
                        Q_PC ← Q_PC ∪ {(<i,a>,j,k)}
                else
                    for each k such that (i,k)∈arcs(G) & (k,j)∈arcs(G) do
                        if counter[(i,k),a] = 1 then
                            Q_PC ← Q_PC ∪ {(<i,a>,k,j)}
            end if
        end for
    end while
    return Q_PC
end PRUNE
```



Algoritmus RPC

Nejprve uděláme AC a potom testuje vybrané PC, případně se vracíme k AC.

```
procedure RPC(G)
    (Q_AC, Q_PC) ← INITIALIZE(G)
    Q_PC ← PRUNE(Q_AC, Q_PC)                % první běh AC
    while Q_PC non empty do
        select and delete any triple (<i,a>,j,k) from Q_PC
        if a∈D_i then
            {<j,b>} ← {<j,x> ∈ S_ia | x∈D_j }   % jediná podpora pro a
            if {<k,c>∈ S_ia ∩ S_jb | c∈D_k } = ∅ then
                counter[(i,j),a] ← 0
                delete "a" from D_i
                Q_PC ← PRUNE({<i,a>}, Q_PC)   % opakujeme AC
            end if
        end if
    end while
end RPC
```