Algoritmy pro CSP

(pokračování)

Řešení nebinárních podmínek

- k-konzistence má exponenciální složitost, v reálu se nepoužívá
- 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

Řešení nebinárních podmínek

- k-konzistence má exponenciální složitost, v reálu se nepoužívá
- 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((V,D,C))
Q := V
while Q non empty do
      vyber a smaž V_j ∈ Q
      for ∀ C takové, že V_j ∈ scope(C) do
          W := revise(V_j,C)
             // W je množina proměnných jejichž, doména se změnila
          if ∃ V_i ∈ W taková, že D_i = ∅ then return fail
          Q := Q ∪ {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((V,D,C))
Q := V
while Q non empty do
      vyber a smaž V_j ∈ Q
      for ∀ C takové, že V_j ∈ scope(C) do
          W := revise(V_j,C)
             // W je množina proměnných jejichž, doména se změnila
          if ∃ V_i ∈ W taková, že D_i = ∅ then return fail
          Q := Q ∪ {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

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` => `min(A) = min(B)+1, max(B) = max(A)-1`
  - příklad: `A in 4..10, B in 6..18, A #> B`

    `min(A) = 6+1` ⇒ `A in 7..10`

    `max(B) = 10-1` ⇒ `B 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` => `min(A) = min(B)+1, max(B) = max(A)-1`
  - příklad: `A in 4..10, B in 6..18, A #> B`
    `min(A) = 6+1` ⇒ `A in 7..10`
    `max(B) = 10-1` ⇒ `B in 6..9`
  - podobně: `A #< B`, `A #>= B`, `A #=< B`

Konzistence mezí a aritmetická omezení

- `A #= B + C => 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 mezí a aritmetická omezení

- `A #= B + C => 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 mezí a aritmetická omezení

- `A #= B + C` => `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 in 1..10, B in 1..10, A #= B + 2, A #> 5, A #\= 8`

  `A #= B + 2` ⇒

Konzistence mezí a aritmetická omezení

- `A #= B + C` => `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 in 1..10, B in 1..10, A #= B + 2, A #> 5, A #\= 8`

  `A #= B + 2` ⇒ `min(A)=1+2, max(A)=10+2` ⇒ `A in 3..10`
  ⇒ `min(B)=1-2, max(B)=10-2` ⇒ `B in 1..8`

Konzistence mezí a aritmetická omezení

- `A #= B + C => 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 in 1..10, B in 1..10, A #= B + 2, A #> 5, A #\= 8`

  `A #= B + 2` ⇒ `min(A)=1+2, max(A)=10+2` ⇒ `A in 3..10`

  ⇒ `min(B)=1-2, max(B)=10-2` ⇒ `B in 1..8`

  `A #> 5` ⇒ `min(A)=6` ⇒ `A in 6..10`

  ⇒ `min(B)=6-2` ⇒ `B in 4..8` (nové vyvolání `A #= B + 2`)

Konzistence mezí a aritmetická omezení

- `A #= B + C` => `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 in 1..10, B in 1..10, A #= B + 2, A #> 5, A #\= 8`

  `A #= B + 2` ⇒ `min(A)=1+2, max(A)=10+2` ⇒ `A in 3..10`

  ⇒ `min(B)=1-2, max(B)=10-2` ⇒ `B in 1..8`

  `A #> 5` ⇒ `min(A)=6` ⇒ `A in 6..10`

  ⇒ `min(B)=6-2` ⇒ `B in 4..8` (nové vyvolání `A #= B + 2`)

  `A #\= 8` ⇒ `A in (6..7) \/ (9..10)` (meze stejné, k propagaci `A #= B + 2` nedojde)

Konzistence mezí a aritmetická omezení

- `A #= B + C => 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 in 1..10, B in 1..10, A #= B + 2, A #> 5, A #\= 8`

  `A #= B + 2` ⇒ `min(A)=1+2, max(A)=10+2` ⇒ `A in 3..10`
  ⇒ `min(B)=1-2, max(B)=10-2` ⇒ `B in 1..8`

  `A #> 5` ⇒ `min(A)=6` ⇒ `A in 6..10`
  ⇒ `min(B)=6-2` ⇒ `B in 4..8` (nové vyvolání `A #= B + 2`)

  `A #\= 8` ⇒ `A in (6..7) \/ (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_distinct` omezení: hodnoty všech proměnných různé
  - `serialized` omezení:
    rozvržení úloh zadaných startovním časem a dobou trvání tak, aby se nepřekrývaly

Propagace pro `all_distinct`

- U = {X2, X4, X5}, dom(U) = {2, 3, 4}:

  `{2,3,4}` nelze pro `X1, X3, X6`

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}, dom(U) = {2, 3, 4}**:

  `{2,3,4}` nelze pro `X1, X3, X6`

  `X1 in 5..6, X3 = 5, X6 in {1} \/ (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 = {X2, X4, X5}, dom(U) = {2, 3, 4}**:

  `{2,3,4}` nelze pro `X1, X3, X6`

  `X1 in 5..6, X3 = 5, X6 in {1} \/ (5..6)`

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

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 = {X2, X4, X5}, dom(U) = {2, 3, 4}**:

  `{2,3,4}` nelze pro `X1, X3, X6`

  `X1 in 5..6, X3 = 5, X6 in {1} \/ (5..6)`

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

  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 = {X2, X4, X5}, dom(U) = {2, 3, 4}**:

  `{2,3,4}` nelze pro `X1, X3, X6`

  `X1 in 5..6, X3 = 5, X6 in {1} \/ (5..6)`

učitel	min	max
Jan	3	6
Petr	3	4
Anna	2	5
Ota	2	4
Eva	3	4
Marie	1	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**
  - $U = \{X_1, \dots, X_k\}, dom(U) = \{D_1 \cup \dots \cup D_k\}$
  - $card(U) = card(dom(U)) \Rightarrow \forall v \in dom(U), \forall X \in (V - U), X \neq v$

Propagace pro `all_distinct`

- **U = {X2, X4, X5}, dom(U) = {2, 3, 4}**:

  `{2,3,4}` nelze pro `X1, X3, X6`

  `X1 in 5..6, X3 = 5, X6 in {1} \/ (5..6)`

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

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

- **Inferenční pravidlo**
  - $U = \{X_1, \ldots, X_k\}, dom(U) = \{D_1 \cup \cdots \cup D_k\}$
  - $card(U) = card(dom(U)) \Rightarrow \forall v \in dom(U), \forall X \in (V - U), X \neq v$
  - 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 = {X2, X4, X5}, dom(U) = {2, 3, 4}**:

  `{2,3,4}` nelze pro `X1, X3, X6`

  `X1 in 5..6, X3 = 5, X6 in {1} \/ (5..6)`

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

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

  stačí hledat **Hallův interval** I: velikost intervalu I je rovna

  počtu proměnných, jejichž doména je v I

- **Inferenční pravidlo**
  - $U = \{X_1,\ldots,X_k\}, dom(U) = \{D_1 \cup \cdots \cup D_k\}$
  - $card(U) = card(dom(U)) \Rightarrow \forall v \in dom(U), \forall X \in (V - U), X \neq v$
  - hodnoty v Hallově intervalu jsou pro ostatní proměnné nedostupné

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

Propagace pro `all_distinct`

- **U = {X2, X4, X5}, dom(U) = {2, 3, 4}**:

  `{2,3,4}` nelze pro `X1, X3, X6`

  `X1 in 5..6, X3 = 5, X6 in {1} \/ (5..6)`

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

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

  stačí hledat **Hallův interval** I: velikost intervalu I je rovna

  počtu proměnných, jejichž doména je v I

- **Inferenční pravidlo**
  - $U = \{X_1, \ldots, X_k\}$, $dom(U) = \{D_1 \cup \cdots \cup D_k\}$
  - $card(U) = card(dom(U)) \Rightarrow \forall v \in dom(U), \forall X \in (V - U), X \neq v$
  - 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)

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 do hloubky

- Základní prohledávací algoritmus pro problémy splňování podmínek
- **Prohledávání stavového prostoru do hloubky** ***(depth first search)***
- Dvě fáze prohledávání s navracením
  - **dopředná fáze**: proměnné jsou postupně vybírány, rozšiřuje se částečné řešení přiřazením konzistení 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 do hloubky

- Základní prohledávací algoritmus pro problémy splňování podmínek
- **Prohledávání stavového prostoru do hloubky *(depth first search)***
- Dvě fáze prohledávání s navracením
  - **dopředná fáze**: proměnné jsou postupně vybírány, rozšiřuje se částečné řešení přiřazením konzistení 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

Základní algoritmus prohledávání do hloubky

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

```
procedure labeling(G,a)
if a > |uzly(G)| then return uzly(G)
for ∀ x ∈ D_a do
    if consistent(G,a) then   % consistent(G,a) je nahrazeno FC(G,a), LA(G,a), ...
        R := labeling(G,a+1)
        if R ≠ fail then return R
return fail
end labeling
```

Po přiřazení všech proměnných vrátíme jejich ohodnocení

- Procedury `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:=$\{(V_i, V_a) \in$ hrany(G), $i < a\}$ % hrany vedoucí z minulých proměnných do aktuální

  Consistent := true

  while Q není prázdná $\wedge$ Consistent do

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

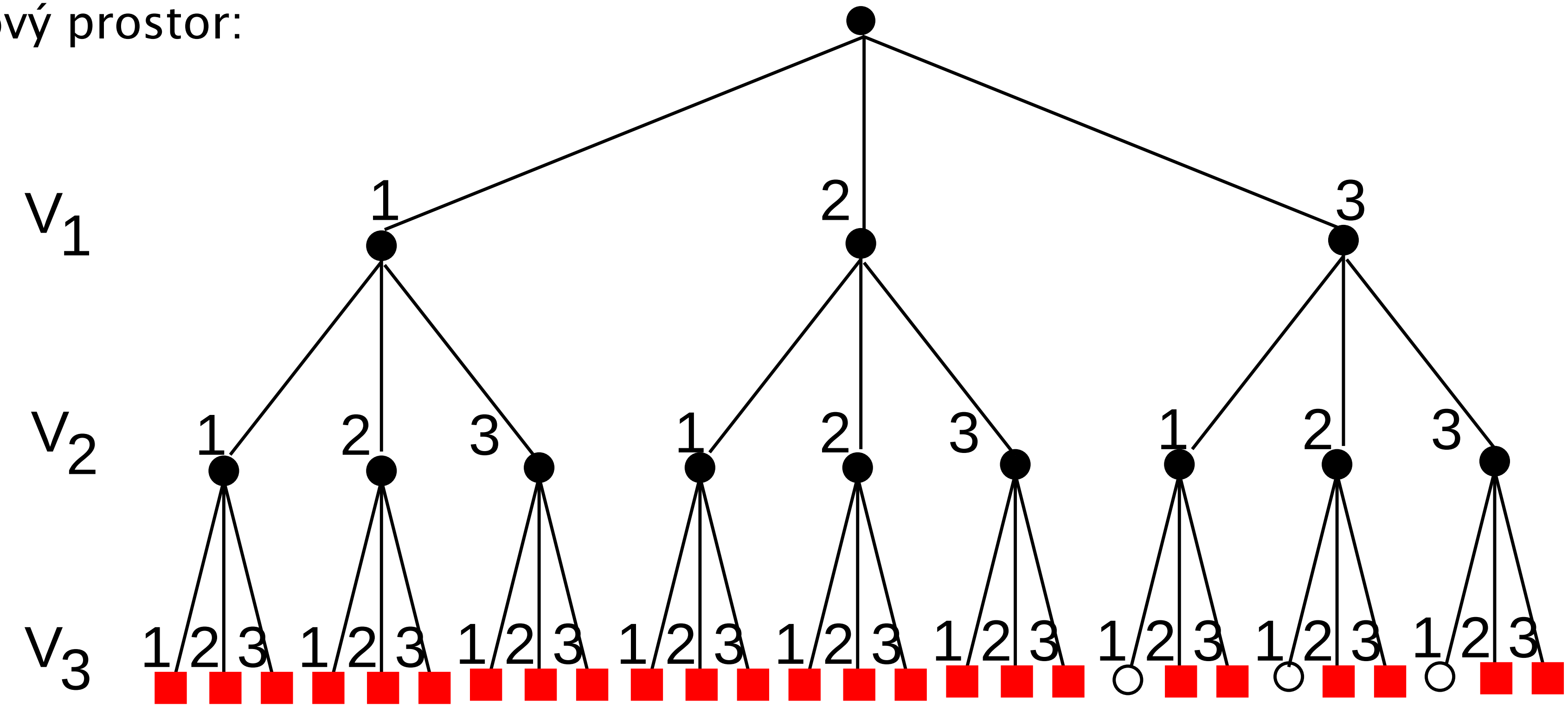
    Consistent := not revise(V_k, V_m) % pokud vyřadíme prvek, bude doména prázdná

  return Consistent

  end BT

Příklad: backtracking

- Omezení: V_1, V_2, V_3 in $1 \ldots 3$, $\quad V_1\# = 3 \times V_3$
- 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:=$\{(V_i, V_a) \in$ hrany(G), $i > a\}$ % přidání hran z budoucích do aktuální proměnné

  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

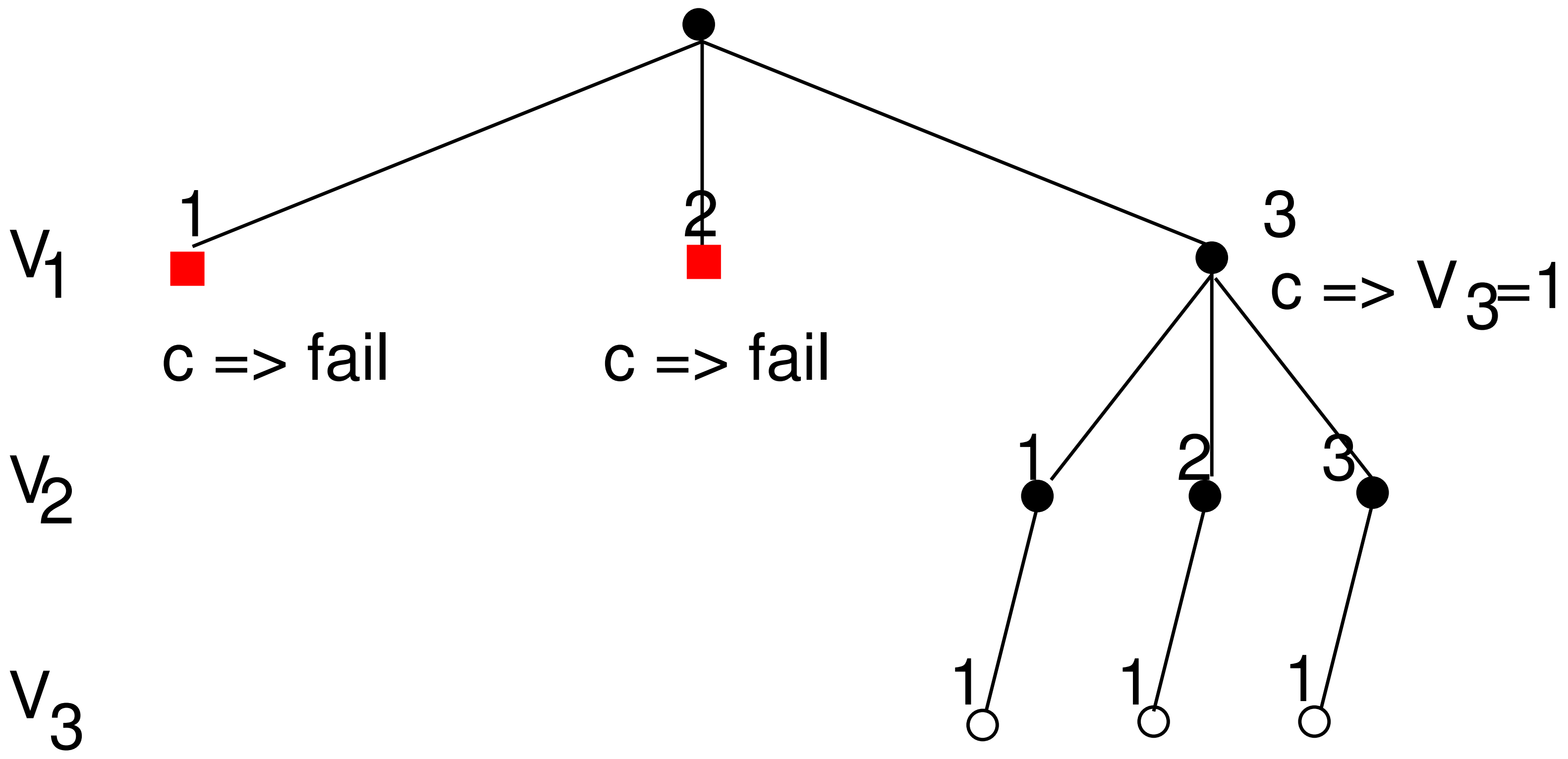
        Consistent := $(|D_k| > 0)$ % vyprázdnění domény znamená nekonzistenci

  return Consistent

  end FC
- Hrany z minulých proměnných do aktuální proměnné není nutno testovat

Příklad: kontrola dopředu

- Omezení: V_1, V_2, V_3 in $1 \ldots 3$, $\quad c : V_1 \# = 3 \times V_3$
- Stavový prostor:



Pohled dopředu *(LA – looking ahead)*

- LA je rozšíření FC, navíc ověřuje konzistenci hran mezi budoucími proměnnými
- procedure LA(G, a)

  Q := $\{(V_i, V_a) \in$ 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 $\cup$ $\{(V_i, V_k) | (V_i, V_k) \in$ 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, navíc ověřuje konzistenci hran mezi budoucími proměnnými
- procedure LA(G,a)

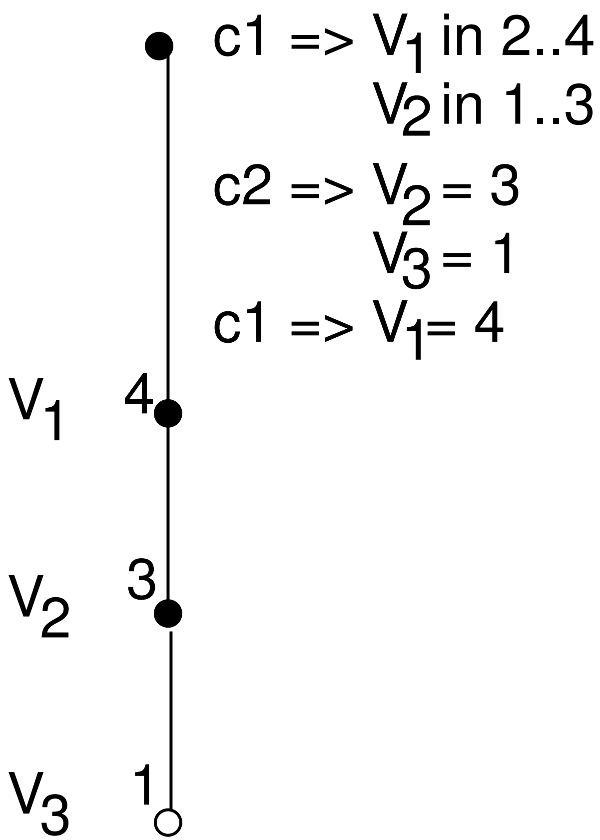
```
Q := {(V_i,V_a) ∈ hrany(G), i > a}        % začínáme s hranami do a
Consistent := true
while Q není prázdná ∧ Consistent do
      vyber a smaž libovolnou hranu (V_k,V_m) z Q
      if revise((V_k,V_m)) then
         Q := Q ∪ {(V_i,V_k)|(V_i,V_k) ∈ hrany(G), i ≠ k, i ≠ m, i > a}
         Consistent := (|D_k| > 0)
return Consistent
end LA
```

  - Hrany z minulých proměnných do aktuální proměnné opět netestujeme
  - Tato LA procedura je založena na AC-3, lze použít i jiné AC algoritmy
- **LA udržuje hranovou konzistenci:** protože ale LA(G,a) používá AC-3, musíme **zajistit iniciální konzistenci** pomocí AC-3 ještě před startem prohledávání

Příklad: pohled dopředu (pomocí AC-3)

- Omezení: V_1, V_2, V_3 in $1 \ldots 4$, $\quad c1 : V_1 \# > V_2$, $\quad c2 : V_2 \# = 3 \times V_3$
- Stavový prostor

  (spouští se iniciální konzistence se před startem prohledávání)

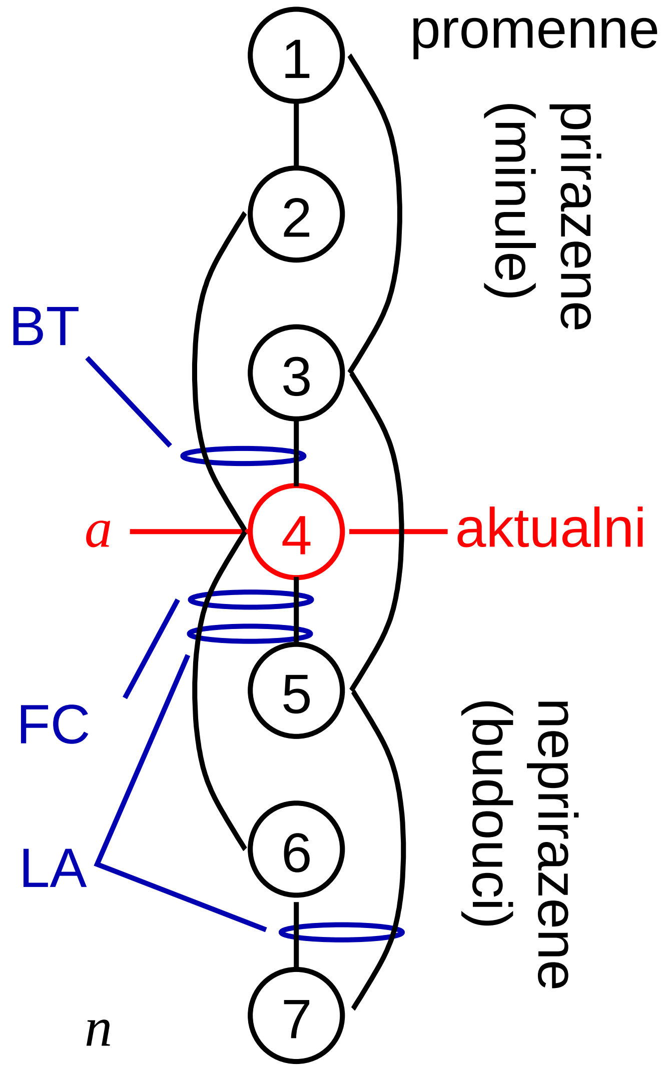


Přehled algoritmů

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

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

  z minulých proměnných do aktuální proměnné



Přehled algoritmů

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

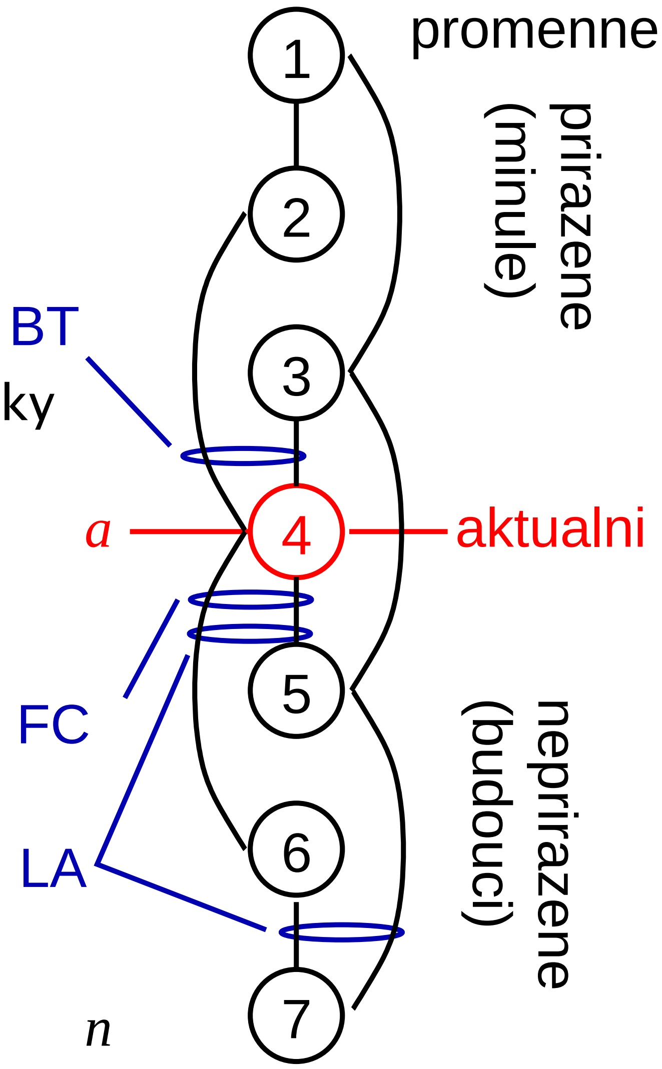
  $c(V_1, V_a), \ldots, c(V_{a-1}, V_a)$

  z minulých proměnných do aktuální proměnné

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

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

  z budoucích proměnných do aktuální proměnné



Přehled algoritmů

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

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

  z minulých proměnných do aktuální proměnné

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

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

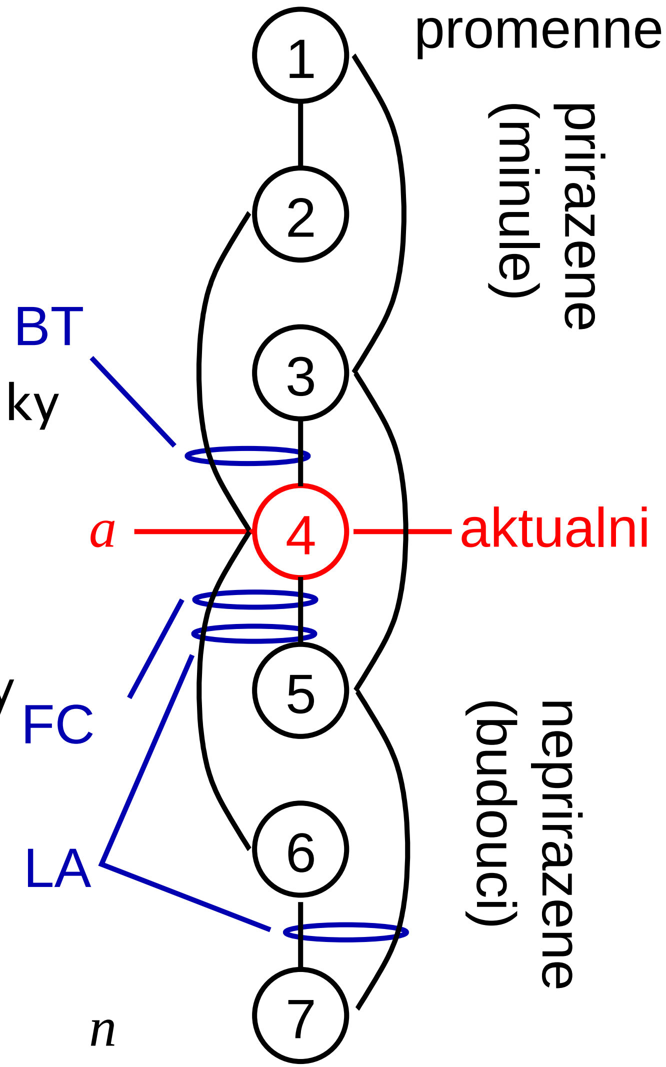
  z budoucích proměnných do aktuální proměnné

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

  $\forall l(a \le l \le n), \forall k(a \le k \le n), k \ne l : c(V_k, V_l)$

  z budoucích proměnných do aktuální proměnné

  a mezi budoucími proměnnými



Cvičení

1. Jak vypadá stavový prostor řešení pro následující omezení

   A in 1..4, B in 3..4, C in 3..4, B #< C, A #= C

   při použití kontroly dopředu a uspořádání proměnných A,B,C? Popište, jaký typ propagace proběhne v jednotlivých uzlech.

2. Jak vypadá stavový prostor řešení pro následující omezení

   A in 1..4, B in 3..4, C in 3..4, B #< C, A #= C

   při použití pohledu dopředu a uspořádání proměnných A,B,C? Popište, jaký typ propagace proběhne v jednotlivých uzlech.

3. Jak vypadá stavový prostor řešení pro následující omezení

   domain([A,B,C],0,1), A #= B-1, C #= A*A

   při použití backtrackingu a pohledu dopředu a uspořádání proměnných A,B,C? Popište, jaký typ propagace proběhne v jednotlivých uzlech.

Cvičení

1. Jaká jsou pravidla pro konzistenci mezí u omezení X#= Y+5? Jaké typy propagací pak proběhnou v následujícím příkladě při použití konzistence mezí?

   X in 1..20, Y in 1..20, X #= Y + 5, Y #> 10.

2. Ukažte, jak je dosaženo hranové konzistence v následujícím příkladu:

   domain([X,Y,Z],1,5]), X #< Y, Z#= Y+1 .