CSP
(6. přednáška)

Kdo chová zebru?

1. There are five houses.
2. The Englishman lives in the red house.
3. The Spaniard owns the dog.
4. Coffee is drunk in the green house.
5. The Ukrainian drinks tea.
6. The green house is immediately to the right of the ivory house.
7. The Old Gold smoker owns snails.
8. Kools are smoked in the yellow house.
9. Milk is drunk in the middle house.
10. The Norwegian lives in the first house.
11. The man who smokes Chesterfields lives in the house next to the man with the fox.
12. Kools are smoked in a house next to the house where the horse is kept.
13. The Lucky Strike smoker drinks orange juice.
14. The Japanese smokes Parliaments.
15. The Norwegian lives next to the blue house.

Kdo chová zebru?

1. There are five houses.
2. The Englishman lives in the red house.
3. The Spaniard owns the dog.
4. Coffee is drunk in the green house.
5. The Ukrainian drinks tea.
6. The green house is immediately to the right of the ivory house.
7. The Old Gold smoker owns snails.
8. Kools are smoked in the yellow house.
9. Milk is drunk in the middle house.
10. The Norwegian lives in the first house.
11. The man who smokes Chesterfields lives in the house next to the man with the fox.
12. Kools are smoked in a house next to the house where the horse is kept.
13. The Lucky Strike smoker drinks orange juice.
14. The Japanese smokes Parliaments.
15. The Norwegian lives next to the blue house.

Now, who drinks water? Who owns the zebra? In the interest of clarity, it must be added that each of the five houses is painted a different color, and their inhabitants are of different national extractions, own different pets, drink different beverages and smoke different brands of American cigarets [sic]. One other thing: in statement 6, right means your right.

— *Life International, December 17, 1962*

CSP — Constraint Satisfaction Problem

$X_0, \ldots X_n$ proměnné

CSP — Constraint Satisfaction Problem

$X_0, \ldots X_n$ proměnné

$D_0, \ldots, D_n$ obory hodnot jednotlivých proměnných

CSP — Constraint Satisfaction Problem

$X_0, \ldots X_n$ proměnné

$D_0, \ldots, D_n$ obory hodnot jednotlivých proměnných

$C_0, \ldots, C_k$ omezující podmínky,

CSP — Constraint Satisfaction Problem

$X_0, \ldots X_n$	proměnné
$D_0, \ldots, D_n$	obory hodnot jednotlivých proměnných
$C_0, \ldots, C_k$	omezující podmínky, $C_i = (A_i, R_i)$

CSP — Constraint Satisfaction Problem

$X_0, \ldots X_n$ proměnné

$D_0, \ldots, D_n$ obory hodnot jednotlivých proměnných

$C_0, \ldots, C_k$ omezující podmínky, $C_i = (A_i, R_i)$ kde

$A_i = \{j_0, \ldots, j_{l_i}\} \subseteq n + 1,$

CSP — Constraint Satisfaction Problem

$X_0, \ldots X_n$ proměnné

$D_0, \ldots, D_n$ obory hodnot jednotlivých proměnných

$C_0, \ldots, C_k$ omezující podmínky, $C_i = (A_i, R_i)$ kde

A_i $= \{j_0, \ldots, j_{l_i}\} \subseteq n + 1$, seznam proměnných vstupujících do této podmínky

CSP — Constraint Satisfaction Problem

$X_0, \ldots X_n$ proměnné

$D_0, \ldots, D_n$ obory hodnot jednotlivých proměnných

$C_0, \ldots, C_k$ omezující podmínky, $C_i = (A_i, R_i)$ kde

A_i $= \{j_0, \ldots, j_{l_i}\} \subseteq n + 1$, seznam proměnných vstupujících do této podmínky

R_i $\subseteq D_{j_0} \times \cdots \times D_{j_{l_i}}$

CSP — Constraint Satisfaction Problem

$X_0, \ldots X_n$ proměnné

$D_0, \ldots, D_n$ obory hodnot jednotlivých proměnných

$C_0, \ldots, C_k$ omezující podmínky, $C_i = (A_i, R_i)$ kde

A_i $= \{j_0, \ldots, j_{l_i}\} \subseteq n + 1$, seznam proměnných vstupujících do této podmínky

R_i $\subseteq D_{j_0} \times \cdots \times D_{j_{l_i}}$ povolené hodnoty proměnných

CSP — Constraint Satisfaction Problem

$X_0, \ldots X_n$ proměnné

$D_0, \ldots, D_n$ obory hodnot jednotlivých proměnných

$C_0, \ldots, C_k$ omezující podmínky, $C_i = (A_i, R_i)$ kde

A_i $= \{j_0, \ldots, j_{l_i}\} \subseteq n + 1$, seznam proměnných vstupujících do této podmínky

R_i $\subseteq D_{j_0} \times \cdots \times D_{j_{l_i}}$ povolené hodnoty proměnných

CSP

CSP — Constraint Satisfaction Problem

$X_0, \ldots X_n$ proměnné

$D_0, \ldots, D_n$ obory hodnot jednotlivých proměnných

$C_0, \ldots, C_k$ omezující podmínky, $C_i = (A_i, R_i)$ kde

A_i $= \{j_0, \ldots, j_{l_i}\} \subseteq n + 1$, seznam proměnných vstupujících do této podmínky

R_i $\subseteq D_{j_0} \times \cdots \times D_{j_{l_i}}$ povolené hodnoty proměnných

CSP

Nalezněte ohodnocení proměnných $x_0, \ldots, x_n$

CSP — Constraint Satisfaction Problem

$X_0, \ldots X_n$ proměnné

$D_0, \ldots, D_n$ obory hodnot jednotlivých proměnných

$C_0, \ldots, C_k$ omezující podmínky, $C_i = (A_i, R_i)$ kde

A_i $= \{j_0, \ldots, j_{l_i}\} \subseteq n + 1$, seznam proměnných vstupujících do této podmínky

R_i $\subseteq D_{j_0} \times \cdots \times D_{j_{l_i}}$ povolené hodnoty proměnných

CSP

Nalezněte ohodnocení proměnných $x_0, \ldots, x_n$ hodnotami $v_0, \ldots, v_n$ z odpovídajích oborů hodnot

CSP — Constraint Satisfaction Problem

$X_0, \ldots X_n$ proměnné

$D_0, \ldots, D_n$ obory hodnot jednotlivých proměnných

$C_0, \ldots, C_k$ omezující podmínky, $C_i = (A_i, R_i)$ kde

A_i $= \{j_0, \ldots, j_{l_i}\} \subseteq n + 1$, seznam proměnných vstupujících do této podmínky

R_i $\subseteq D_{j_0} \times \cdots \times D_{j_{l_i}}$ povolené hodnoty proměnných

CSP

Nalezněte ohodnocení proměnných $x_0, \ldots, x_n$ hodnotami $v_0, \ldots, v_n$ z odpovídajích oborů hodnot splňující všechny omezující podmínky,

CSP — Constraint Satisfaction Problem

$X_0, \ldots X_n$ proměnné

$D_0, \ldots, D_n$ obory hodnot jednotlivých proměnných

$C_0, \ldots, C_k$ omezující podmínky, $C_i = (A_i, R_i)$ kde

A_i $= \{j_0, \ldots, j_{l_i}\} \subseteq n+1$, seznam proměnných vstupujících do této podmínky

R_i $\subseteq D_{j_0} \times \cdots \times D_{j_{l_i}}$ povolené hodnoty proměnných

CSP

Nalezněte ohodnocení proměnných $x_0, \ldots, x_n$ hodnotami $v_0, \ldots, v_n$ z odpovídajích oborů hodnot splňující všechny omezující podmínky, t.j. $(v_{j_0}, \ldots, v_{j_{l_i}}) \in R_i$ pro každé $i = 0, \ldots, k$.

Zebra formulovaná jako CSP

Proměnné

Zebra formulovaná jako CSP

Proměnné

yellow, blue, red, ivory, green, norwegian, ukrainian, englishman, spaniard, japanese, water, tea, milk, orange juice, coffee, kools, chesterfield, old gold, lucky strike, parliament fox, horse, snails, dog, zebra

Zebra formulovaná jako CSP

Proměnné

yellow, blue, red, ivory, green, norwegian, ukrainian, englishman, spaniard, japanese, water, tea, milk, orange juice, coffee, kools, chesterfield, old gold, lucky strike, parliament fox, horse, snails, dog, zebra

Obory hodnot

Zebra formulovaná jako CSP

Proměnné

yellow, blue, red, ivory, green, norwegian, ukrainian, englishman, spaniard, japanese, water, tea, milk, orange juice, coffee, kools, chesterfield, old gold, lucky strike, parliament fox, horse, snails, dog, zebra

Obory hodnot

$$\{1, 2, 3, 4, 5\}$$

(odpovídají jednotlivým domům)

Zebra formulovaná jako CSP

Proměnné

yellow, blue, red, ivory, green, norwegian, ukrainian, englishman, spaniard, japanese, water, tea, milk, orange juice, coffee, kools, chesterfield, old gold, lucky strike, parliament fox, horse, snails, dog, zebra

Obory hodnot

$$\{1, 2, 3, 4, 5\}$$

(odpovídají jednotlivým domům)

Omezující podmínky

Zebra formulovaná jako CSP

Proměnné

yellow, blue, red, ivory, green, norwegian, ukrainian, englishman, spaniard, japanese, water, tea, milk, orange juice, coffee, kools, chesterfield, old gold, lucky strike, parliament fox, horse, snails, dog, zebra

Obory hodnot

$$\{1, 2, 3, 4, 5\}$$

(odpovídají jednotlivým domům)

Omezující podmínky

2. The Englishman lives in the red house.

Zebra formulovaná jako CSP

Proměnné

yellow, blue, red, ivory, green, norwegian, ukrainian, englishman, spaniard, japanese, water, tea, milk, orange juice, coffee, kools, chesterfield, old gold, lucky strike, parliament fox, horse, snails, dog, zebra

Obory hodnot

$$\{1, 2, 3, 4, 5\}$$

(odpovídají jednotlivým domům)

Omezující podmínky

2. The Englishman lives in the red house.

2. englishman = red

Zebra formulovaná jako CSP

Proměnné

yellow, blue, red, ivory, green, norwegian, ukrainian, englishman, spaniard, japanese, water, tea, milk, orange juice, coffee, kools, chesterfield, old gold, lucky strike, parliament fox, horse, snails, dog, zebra

Obory hodnot

$$\{1, 2, 3, 4, 5\}$$

(odpovídají jednotlivým domům)

Omezující podmínky

2. The Englishman lives in the red house.
3. The Spaniard owns the dog.

2. englishman = red

Zebra formulovaná jako CSP

Proměnné

yellow, blue, red, ivory, green, norwegian, ukrainian, englishman, spaniard, japanese, water, tea, milk, orange juice, coffee, kools, chesterfield, old gold, lucky strike, parliament fox, horse, snails, dog, zebra

Obory hodnot

$$\{1, 2, 3, 4, 5\}$$

(odpovídají jednotlivým domům)

Omezující podmínky

2. The Englishman lives in the red house.
3. The Spaniard owns the dog.

2. englishman $=$ red
3. spaniard $=$ dog

Zebra formulovaná jako CSP

Proměnné

yellow, blue, red, ivory, green, norwegian, ukrainian, englishman, spaniard, japanese, water, tea, milk, orange juice, coffee, kools, chesterfield, old gold, lucky strike, parliament fox, horse, snails, dog, zebra

Obory hodnot

$$\{1, 2, 3, 4, 5\}$$

(odpovídají jednotlivým domům)

Omezující podmínky

2. The Englishman lives in the red house.
3. The Spaniard owns the dog.

...

2. englishman $=$ red
3. spaniard $=$ dog

...

Zebra formulovaná jako CSP

Proměnné

yellow, blue, red, ivory, green, norwegian, ukrainian, englishman, spaniard, japanese, water, tea, milk, orange juice, coffee, kools, chesterfield, old gold, lucky strike, parliament fox, horse, snails, dog, zebra

Obory hodnot

$$\{1, 2, 3, 4, 5\}$$

(odpovídají jednotlivým domům)

Omezující podmínky

2. The Englishman lives in the red house.
3. The Spaniard owns the dog.

. . .

15. The Norwegian lives next to the blue house.

2. englishman = red
3. spaniard = dog

. . .

Zebra formulovaná jako CSP

Proměnné

yellow, blue, red, ivory, green, norwegian, ukrainian, englishman, spaniard, japanese, water, tea, milk, orange juice, coffee, kools, chesterfield, old gold, lucky strike, parliament fox, horse, snails, dog, zebra

Obory hodnot

$$\{1, 2, 3, 4, 5\}$$

(odpovídají jednotlivým domům)

Omezující podmínky

2.	The Englishman lives in the red house.	2. englishman = red
3.	The Spaniard owns the dog.	3. spaniard = dog
...		...
15.	The Norwegian lives next to the blue house.	15. norwegian = blue + 1 or norwegian = blue - 1

Zebra formulovaná jako CSP

Proměnné

yellow, blue, red, ivory, green, norwegian, ukrainian, englishman, spaniard, japanese, water, tea, milk, orange juice, coffee, kools, chesterfield, old gold, lucky strike, parliament fox, horse, snails, dog, zebra

Obory hodnot

$$\{1, 2, 3, 4, 5\}$$

(odpovídají jednotlivým domům)

Omezující podmínky

2.	The Englishman lives in the red house.	2. englishman = red
3.	The Spaniard owns the dog.	3. spaniard = dog
...		...
15.	The Norwegian lives next to the blue house.	15. norwegian = blue + 1 or norwegian = blue - 1
A	it must be added that each of the five houses is painted a different color	

Zebra formulovaná jako CSP

Proměnné

yellow, blue, red, ivory, green, norwegian, ukrainian, englishman, spaniard, japanese, water, tea, milk, orange juice, coffee, kools, chesterfield, old gold, lucky strike, parliament fox, horse, snails, dog, zebra

Obory hodnot

$$\{1, 2, 3, 4, 5\}$$

(odpovídají jednotlivým domům)

Omezující podmínky

2.	The Englishman lives in the red house.	2.	englishman = red
3.	The Spaniard owns the dog.	3.	spaniard = dog
...		...	
15.	The Norwegian lives next to the blue house.	15.	norwegian = blue + 1 or norwegian = blue - 1
A	it must be added that each of the five houses is painted a different color	A	proměnné yellow, blue, red, ivory, green mají každá jinou hodnotu

Zebra formulovaná jako CSP

Proměnné

yellow, blue, red, ivory, green, norwegian, ukrainian, englishman, spaniard, japanese, water, tea, milk, orange juice, coffee, kools, chesterfield, old gold, lucky strike, parliament fox, horse, snails, dog, zebra

Obory hodnot

$$\{1, 2, 3, 4, 5\}$$

(odpovídají jednotlivým domům)

Omezující podmínky

2.	The Englishman lives in the red house.
3.	The Spaniard owns the dog.
...	
15.	The Norwegian lives next to the blue house.
A	it must be added that each of the five houses is painted a different color
...	

2.	englishman = red
3.	spaniard = dog
...	
15.	norwegian = blue + 1 or norwegian = blue - 1
A	proměnné yellow, blue, red, ivory, green mají každá jinou hodnotu
...	

Další CSP problémy

Sudoku

Další CSP problémy

Sudoku

obarvení grafu

Další CSP problémy

Sudoku

obarvení grafu

umístění dam na šachovnici

Další CSP problémy

Sudoku

obarvení grafu

umístění dam na šachovnici

křížovky

Další CSP problémy

Sudoku

obarvení grafu

umístění dam na šachovnici

křížovky

SAT

Další CSP problémy

Sudoku

obarvení grafu

umístění dam na šachovnici

křížovky

SAT

BID (building identification problem)

Další CSP problémy

Sudoku

obarvení grafu

umístění dam na šachovnici

křížovky

SAT

BID (building identification problem)

rozvrhování, plánování ...

CSP Solver: Backtracking

```
def CSPBackTrackSolver(problem, partial_assignment):
  var = selectUnassignedVar(problem, partial_assignment)

  # pokud jsou vsechny promenne prirazene, uspeli jsme
  if var is None:
    return partial_assignment

  for val in problem.domain(var):
      partial_assignment[var] = val

      # pokud jsme stale konzistentni, rekurzivne hledame dal
      if problem.isConsistent(partial_assignment):
        ret = CSPBackTrackSolver(problem, partial_assignment)

        # hura, nasli jsme reseni
        if ret:
          return ret
  return None
```

CSP Solver: Zlepšení

- vhodně volit pořadí přiřazování proměnných (`selectUnassignedVar`)

CSP Solver: Zlepšení

- vhodně volit pořadí přiřazování proměnných (`selectUnassignedVar`)
- vhodně volit pořadí hodnot

CSP Solver: Zlepšení

- vhodně volit pořadí přiřazování proměnných (`selectUnassignedVar`)
- vhodně volit pořadí hodnot
- neprocházet neperspektivní hodnoty (**constraint propagation**)

CSP Solver: Zlepšení

- vhodně volit pořadí přiřazování proměnných (`selectUnassignedVar`)
- vhodně volit pořadí hodnot
- neprocházet neperspektivní hodnoty (**constraint propagation**)
- vynechávat podstromy, o kterých víme, že jsou neperspektivní (**backjumping**)

CSP Solver: Zlepšení

- vhodně volit pořadí přiřazování proměnných (`selectUnassignedVar`)
- vhodně volit pořadí hodnot
- neprocházet neperspektivní hodnoty (**constraint propagation**)
- vynechávat podstromy, o kterých víme, že jsou neperspektivní (**backjumping**)

Pořadí proměnných — `selectUnassignedVar`

Zvolme proměnnou, která má nejméně možných přiřazení konzistentních s aktuálním přiřazením.

Pořadí proměnných — `selectUnassignedVar`

Zvolme proměnnou, která má nejméně možných přiřazení konzistentních s aktuálním přiřazením.

- spolu s v přiřazení si budeme pro proměnné pamatovat konzistentní hodnoty: `pa.restricted` — seznam dvojic (proměnná, množina legálních hodnot)

Pořadí proměnných — `selectUnassignedVar`

Zvolme proměnnou, která má nejméně možných přiřazení konzistentních s aktuálním přiřazením.

- spolu s v přiřazení si budeme pro proměnné pamatovat konzistentní hodnoty: `pa.restricted` — seznam dvojic (proměnná, množina legálních hodnot)
- při každém přiřazení je třeba aktualizovat `pa.restricted`

Pořadí proměnných — `selectUnassignedVar`

Zvolme proměnnou, která má nejméně možných přiřazení konzistentních s aktuálním přiřazením.

- spolu s v přiřazení si budeme pro proměnné pamatovat konzistentní hodnoty: `pa.restricted` — seznam dvojic (proměnná, množina legálních hodnot)
- při každém přiřazení je třeba aktualizovat `pa.restricted`

```
def selectUnassignedVar(problem, partial_assignment):
  if len(partial_assignment.restricted()) > 0:
    var, dom = min(partial_assignment.restricted,
                   key = lambda (var,dom): len(dom))
    return var
  else:
    for var in problem.vars:
      if var not in partial_assignment:
        return var
  return None
```

Constraint propagation

- založeno na pojmu tzv. 2-konzistence

Constraint propagation

- založeno na pojmu tzv. 2-konzistence
- pro jednoduchost předpokládejme, že všechny omezující podmínky jsou binární

Constraint propagation

- založeno na pojmu tzv. 2-konzistence
- pro jednoduchost předpokládejme, že všechny omezující podmínky jsou binární
- obecný případ lze převést na binární podmínky,

Constraint propagation

- založeno na pojmu tzv. 2-konzistence
- pro jednoduchost předpokládejme, že všechny omezující podmínky jsou binární
- obecný případ lze převést na binární podmínky, za cenu zvýšení počtu proměnných

Constraint propagation

- založeno na pojmu tzv. 2-konzistence
- pro jednoduchost předpokládejme, že všechny omezující podmínky jsou binární
- obecný případ lze převést na binární podmínky, za cenu zvýšení počtu proměnných (cvičení)

Constraint propagation

- založeno na pojmu tzv. 2-konzistence
- pro jednoduchost předpokládejme, že všechny omezující podmínky jsou binární
- obecný případ lze převést na binární podmínky, za cenu zvýšení počtu proměnných (cvičení)

Definujme tzv. **constraint graph**

vrcholy — jednotlivé proměnné

Constraint propagation

- založeno na pojmu tzv. 2-konzistence
- pro jednoduchost předpokládejme, že všechny omezující podmínky jsou binární
- obecný případ lze převést na binární podmínky, za cenu zvýšení počtu proměnných (cvičení)

Definujme tzv. **constraint graph**

vrcholy — jednotlivé proměnné

hrany — jednotlivé omezení (dvě proměnné jsou spojeny hranou, pokud figurují v nějaké omezující podmínce)

Constraint propagation

- založeno na pojmu tzv. 2-konzistence
- pro jednoduchost předpokládejme, že všechny omezující podmínky jsou binární
- obecný případ lze převést na binární podmínky, za cenu zvýšení počtu proměnných (cvičení)

Definujme tzv. **constraint graph**

vrcholy — jednotlivé proměnné

hrany — jednotlivé omezení (dvě proměnné jsou spojeny hranou, pokud figurují v nějaké omezující podmínce)

Omezené obory hodnot proměnných jsou **2-konzistentní**, pokud je každé částečné přiřazení definované na dvou proměnných

Constraint propagation

- založeno na pojmu tzv. 2-konzistence
- pro jednoduchost předpokládejme, že všechny omezující podmínky jsou binární
- obecný případ lze převést na binární podmínky, za cenu zvýšení počtu proměnných (cvičení)

Definujme tzv. **constraint graph**

vrcholy — jednotlivé proměnné

hrany — jednotlivé omezení (dvě proměnné jsou spojeny hranou, pokud figurují v nějaké omezující podmínce)

Omezené obory hodnot proměnných jsou **2-konzistentní**, pokud je každé částečné přiřazení definované na dvou proměnných (a mající hodnoty v těchto omezených oborech)

Constraint propagation

- založeno na pojmu tzv. 2-konzistence
- pro jednoduchost předpokládejme, že všechny omezující podmínky jsou binární
- obecný případ lze převést na binární podmínky, za cenu zvýšení počtu proměnných (cvičení)

Definujme tzv. **constraint graph**

vrcholy — jednotlivé proměnné

hrany — jednotlivé omezení (dvě proměnné jsou spojeny hranou, pokud figurují v nějaké omezující podmínce)

Omezené obory hodnot proměnných jsou **2-konzistentní**, pokud je každé částečné přiřazení definované na dvou proměnných (a mající hodnoty v těchto omezených oborech) konzistentní.

Constraint propagation — implementace

```
def assignAndReduce(problem, partial_assignment, var,val):
""" Rozsiri partial_assignment o var=val a aktualizuje
    partial_assignment.restricted """
 import copy
 ret_assignment = copy.deepcopy(partial_assignment)
 ret_assignment[var]=val
 work = []
 for cvar in problem.constrainedby(var):
   work.append((var,cvar))

 while len(work) > 0:
   (vA,vB) = work.pop()
   if vA in ret_assignment:
     restricted, consistent = restrict(problem, ret_assignment, vA, vB)
     if not consistent:
       return None
     if restricted:
       for cvar in problem.constrainedby(vB):
       if not cvar == vA:
         work.append((vB,cvar))
 return ret_assignment
```

Constraint propagation — implementace (pokračování)

```
def restrict(problem, ret_assignment, vA, vB):
""" Aktualizuje partial_assignment.restricted pro promennou vB
    na zaklade omezeneho oboru hodnot promenne vA."""
```

Constraint propagation — implementace (pokračování)

```
def restrict(problem, ret_assignment, vA, vB):
""" Aktualizuje partial_assignment.restricted pro promennou vB
    na zaklade omezeneho oboru hodnot promenne vA."""

 if varB not in assignment.restricted:
   dom = assignment.restricted[varB] = set(problem.domain(varB))
 else:
   dom = assignment.restricted[varB]
```

Constraint propagation — implementace (pokračování)

```
def restrict(problem, ret_assignment, vA, vB):
""" Aktualizuje partial_assignment.restricted pro promennou vB
    na zaklade omezeneho oboru hodnot promenne vA."""

 if varB not in assignment.restricted:
   dom = assignment.restricted[varB] = set(problem.domain(varB))
 else:
   dom = assignment.restricted[varB]

 # Hodnoty, ktere budeme zahazovat
 removed = set([])
 for valB in dom:
   # Pokud neexistuje zadne legalni prirazeni promenn  A tak,
   # aby nejaka omezujici podminka na A a B nezakazovala hodnotu valB
   # tak hodnotu valB vyhodime
   if not isLegal(problem, assignment, varA, varB, valB):
     removed.add(valB)
```

Constraint propagation — implementace (pokračování)

```
def restrict(problem, ret_assignment, vA, vB):
""" Aktualizuje partial_assignment.restricted pro promennou vB
    na zaklade omezeneho oboru hodnot promenne vA."""

 if varB not in assignment.restricted:
   dom = assignment.restricted[varB] = set(problem.domain(varB))
 else:
   dom = assignment.restricted[varB]

 # Hodnoty, ktere budeme zahazovat
 removed = set([])
 for valB in dom:
   # Pokud neexistuje zadne legalni prirazeni promenn  A tak,
   # aby nejaka omezujici podminka na A a B nezakazovala hodnotu valB
   # tak hodnotu valB vyhodime
   if not isLegal(problem, assignment, varA, varB, valB):
     removed.add(valB)

 # Obor hodnot nebyl nijak omezen, nenasli jsme nekonzistenci
 if len(removed) == 0:
   restricted=False

 # Aktualizuj assignment.restricted pro prom nnou varB
 else:
   restricte = True
   dom.difference_update(removed)
```

Constraint propagation — implementace (pokračování)

```
def restrict(problem, ret_assignment, vA, vB):
""" Aktualizuje partial_assignment.restricted pro promennou vB
    na zaklade omezeneho oboru hodnot promenne vA."""

 if varB not in assignment.restricted:
   dom = assignment.restricted[varB] = set(problem.domain(varB))
 else:
   dom = assignment.restricted[varB]

 # Hodnoty, ktere budeme zahazovat
 removed = set([])
 for valB in dom:
   # Pokud neexistuje zadne legalni prirazeni promenn  A tak,
   # aby nejaka omezujici podminka na A a B nezakazovala hodnotu valB
   # tak hodnotu valB vyhodime
   if not isLegal(problem, assignment, varA, varB, valB):
     removed.add(valB)

 # Obor hodnot nebyl nijak omezen, nenasli jsme nekonzistenci
 if len(removed) == 0:
   restricted=False

 # Aktualizuj assignment.restricted pro prom nnou varB
 else:
   restricte = True
   dom.difference_update(removed)

 # Pokud nezbyly zadne hodnoty, jsme nekonzistentni
 if len(dom) == 0:
   consistent = False
   raise cspError.EmptyDomainInAssignment
 else:
   consistent = True
```

CSP — složitost

- mají-li proměnné spojitý obor hodnot a podmínky jsou lineární

CSP — složitost

- mají-li proměnné spojitý obor hodnot a podmínky jsou lineární (resp. kvadratické)

CSP — složitost

- mají-li proměnné spojitý obor hodnot a podmínky jsou lineární (resp. kvadratické), existuje polynomiální algoritmus

CSP — složitost

- mají-li proměnné spojitý obor hodnot a podmínky jsou lineární (resp. kvadratické), existuje polynomiální algoritmus
- pokud má constraint graf speciální strukturu

CSP — složitost

- mají-li proměnné spojitý obor hodnot a podmínky jsou lineární (resp. kvadratické), existuje polynomiální algoritmus
- pokud má constraint graf speciální strukturu (strom)

CSP — složitost

- mají-li proměnné spojitý obor hodnot a podmínky jsou lineární (resp. kvadratické), existuje polynomiální algoritmus
- pokud má constraint graf speciální strukturu (strom) existuje polynomiální řešení

CSP — složitost

- mají-li proměnné spojitý obor hodnot a podmínky jsou lineární (resp. kvadratické), existuje polynomiální algoritmus
- pokud má constraint graf speciální strukturu (strom) existuje polynomiální řešení
- obecně NP-úplný problém

CSP — složitost

- mají-li proměnné spojitý obor hodnot a podmínky jsou lineární (resp. kvadratické), existuje polynomiální algoritmus
- pokud má constraint graf speciální strukturu (strom) existuje polynomiální řešení
- obecně NP-úplný problém
- dokonce i když omezíme hodnoty proměnných na 0,1

CSP — složitost

- mají-li proměnné spojitý obor hodnot a podmínky jsou lineární (resp. kvadratické), existuje polynomiální algoritmus
- pokud má constraint graf speciální strukturu (strom) existuje polynomiální řešení
- obecně NP-úplný problém
- dokonce i když omezíme hodnoty proměnných na 0,1 (**SAT**)

CSP — složitost

- mají-li proměnné spojitý obor hodnot a podmínky jsou lineární (resp. kvadratické), existuje polynomiální algoritmus
- pokud má constraint graf speciální strukturu (strom) existuje polynomiální řešení
- obecně NP-úplný problém
- dokonce i když omezíme hodnoty proměnných na 0,1 (**SAT**), je problém NP-úplný.

SAT — Boolean satisfiability problem

- 3SAT — splnitelnost formule v CNF (conjunctive normal form), kde každá klauzule má nejvýše tři literály

SAT — Boolean satisfiability problem

- 3SAT — splnitelnost formule v CNF (conjunctive normal form), kde každá klauzule má nejvýše tři literály
- HORNSAT, 3SAT, XORSAT, DNF-SAT — polynomiální čas (DNF SAT dokonce lineární)

SAT — Boolean satisfiability problem

- 3SAT — splnitelnost formule v CNF (conjunctive normal form), kde každá klauzule má nejvýše tři literály
- HORNSAT, 3SAT, XORSAT, DNF-SAT — polynomiální čas (DNF SAT dokonce lineární)