Selbst-organisierende, adaptive Systeme

Vorlesung 12: Constraint Programming

Einordnung

Institute for Software & Systems Engineering

Bisher:

- Mechanismen für eigennützige Agenten
- Optimierungsprobleme bei Auszahlung
- Systematische Ansätze für SOAS
- RIA: Zustände über Constraints beschrieben

In dieser Vorlesung:

- Prinzipien Diskreter Optimierung
- Constraint Programming
- Propagationen
- Suchstrategien + Heuristiken

Constraint Programming: Einordnung

Institute for Software & Systems Engineering

- Generischer Ansatz zur Lösung von Erfüllbarkeitsproblemen
- Ausnützen von **Struktur** von logischen Bedingungen (Tsang, 1993)
- Konzentration auf *endliche* Wertebereiche und Erfüllbarkeit (Russell and Norvig, 2010, Kap. 5)

Constraint Programming: Einordnung

Institute for Software & Systems Engineering

- Generischer Ansatz zur Lösung von Erfüllbarkeitsproblemen
- Ausnützen von **Struktur** von logischen Bedingungen (Tsang, 1993)
- Konzentration auf *endliche* Wertebereiche und Erfüllbarkeit (Russell and Norvig, 2010, Kap. 5)
  - Im Gegensatz z.B. zu *linearer Programmierung*, *konvexe Optimierung*
  - Verallgemeinert Boolesche Erfüllbarkeitsprobleme (SAT)
  - Scheduling-Probleme
  - Reorganisationen
  - Zuweisungsprobleme (z.B. Frequenzen an Sender, Energie and Produzenten, ...)

Constraint Programming: Einordnung

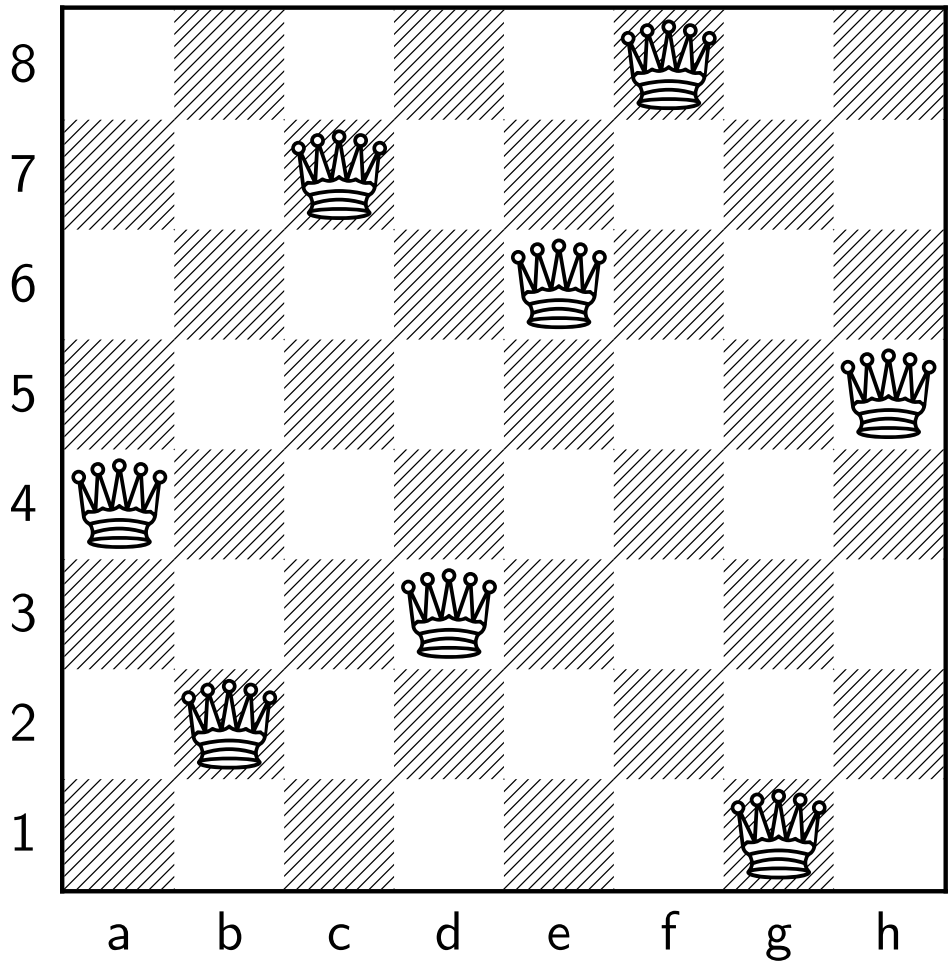
Institute for Software & Systems Engineering

- Generischer Ansatz zur Lösung von Erfüllbarkeitsproblemen
- Ausnützen von **Struktur** von logischen Bedingungen (Tsang, 1993)
- Konzentration auf *endliche* Wertebereiche und Erfüllbarkeit (Russell and Norvig, 2010, Kap. 5)
  - Im Gegensatz z.B. zu *linearer Programmierung*, *konvexe Optimierung*
  - Verallgemeinert Boolesche Erfüllbarkeitsprobleme (SAT)
  - Scheduling-Probleme
  - Reorganisationen
  - Zuweisungsprobleme (z.B. Frequenzen an Sender, Energie and Produzenten, ...)
- Deklarativ, aber Constraints sind an Algorithmen geknüpft (Rossi et al., 2006)!

Ein prototypisches CSP

Und noch eins

Institute for
Software & Systems
Engineering



Basics

Institute for Software & Systems Engineering

Definition (Constraint-Problem)

Ein Constraint-Problem (X, D, C) ist beschrieben durch

- *Variablen* X, *Domänen* $D = (D_x)_{x \in X}$ (endlich), *Constraints* C

Basics

Institute for Software & Systems Engineering

Definition (Constraint-Problem)

Ein Constraint-Problem (X, D, C) ist beschrieben durch

- *Variablen* X, *Domänen* $D = (D_x)_{x \in X}$ (endlich), *Constraints* C

Technisch ist ein Constraint $c \in C$ mit *Scope* $\mathrm{sc}(c) \subseteq X$ eine Einschränkung auf gültige Tupel des kartesischen Produktes der Domänen des Scopes: $c \subseteq \prod_{x \in \mathrm{sc}(c)} D_x$.

Institute for
Software & Systems
Engineering

Definition (Constraint-Problem)

Ein Constraint-Problem (X, D, C) ist beschrieben durch

- *Variablen* X, *Domänen* $D = (D_x)_{x \in X}$ (endlich), *Constraints* C

Technisch ist ein Constraint $c \in C$ mit *Scope* $\mathrm{sc}(c) \subseteq X$ eine Einschränkung auf gültige Tupel des kartesischen Produktes der Domänen des Scopes: $c \subseteq \prod_{x \in \mathrm{sc}(c)} D_x$.

Constraints sind entweder:

- **Extensional**; direkte Auflistung der Menge (z.B. $\{1, 2\}, \{2, 3\}$
- **Intensional**; Syntaktische Vorschrift gegeben (z.B. $x + 1 = y$)

Basics

Institute for Software & Systems Engineering

Definition (Constraint-Problem)

Ein Constraint-Problem (X, D, C) ist beschrieben durch

- *Variablen* X, *Domänen* $D = (D_x)_{x \in X}$ (endlich), *Constraints* C

Technisch ist ein Constraint $c \in C$ mit *Scope* $\mathrm{sc}(c) \subseteq X$ eine Einschränkung auf gültige Tupel des kartesischen Produktes der Domänen des Scopes: $c \subseteq \prod_{x \in \mathrm{sc}(c)} D_x$.

Constraints sind entweder:

- **Extensional**; direkte Auflistung der Menge (z.B. $\{1, 2\}, \{2, 3\}$
- **Intensional**; Syntaktische Vorschrift gegeben (z.B. $x + 1 = y$)

Wir verwenden beide in *funktionaler* Form: $c : [X \to D] \to \mathbb{B} = \{\mathrm{tt}, \mathrm{ff}\}$, wobei $[X \to D]$ die Menge aller Abbildungen von X nach D ist.

Das eigentliche Erfüllbarkeitsproblem

Institute for
Software & Systems
Engineering

Eine (partielle) Zuweisung ist eine (partielle) Abbildung $\theta : X \to D$ mit Scope sc als definierter Wertebereich.

Definition (Lösung)

Eine Lösung für ein Constraint-Problem (X, D, C) ist eine Zuweisung θ mit $\mathrm{sc}(\theta) = X$, sodass θ alle Constraints $c \in C$ erfüllt, also $\forall c \in C : c(\theta) = \mathrm{tt}$.

Ein Constraint-Satisfaction-Problem (CSP) besteht darin, eine Lösung θ aus dem Suchraum $[X \to D]$ zu finden (Rossi et al., 2006).

Beispiel

Institute for Software & Systems Engineering

Problem

CSP (X, D, C) mit

- $X = \{x, y, z\}$
- $D_x = D_y = \{0, 1, 2\}$, $D_z = \{0, 1\}$
- C
  - $c_1 : x \neq y,\ y \neq z,\ x \neq z$
  - $c_2 : x + 1 = y$

Beispiel

Institute for Software & Systems Engineering

Problem

CSP (X, D, C) mit

- $X = \{x, y, z\}$
- $D_x = D_y = \{0, 1, 2\}$, $D_z = \{0, 1\}$
- C
  - $c_1 : x \neq y,\ y \neq z,\ x \neq z$
  - $c_2 : x + 1 = y$

Beispiel

Institute for Software & Systems Engineering

Problem

CSP (X, D, C) mit

- $X = \{x, y, z\}$
- $D_x = D_y = \{0, 1, 2\}$, $D_z = \{0, 1\}$
- C
  - $c_1 : x \neq y,\ y \neq z,\ x \neq z$
  - $c_2 : x + 1 = y$

```
var 0..2: x;
var 0..2: y;
var 0..1: z;

% c1
constraint x != y /\ y != z /\ x != z;
% c2
constraint x + 1 = y;

solve satisfy;
```

Welche Zuweisung ist eine Lösung dieses Problems?

- $\Theta = \{(x \to 1, y \to 2, z \to ?), (x \to 0, y \to 1, z \to ?)\}$ erfüllen c_2;

Beispiel

Institute for Software & Systems Engineering

Problem

CSP (X, D, C) mit

- $X = \{x, y, z\}$
- $D_x = D_y = \{0, 1, 2\}$, $D_z = \{0, 1\}$
- C
  - $c_1 : x \neq y,\ y \neq z,\ x \neq z$
  - $c_2 : x + 1 = y$

```
var 0..2: x;
var 0..2: y;
var 0..1: z;

% c1
constraint x != y /\ y != z /\ x != z;
% c2
constraint x + 1 = y;

solve satisfy;
```

Welche Zuweisung ist eine Lösung dieses Problems?

- $\Theta = \{(x \to 1, y \to 2, z \to ?), (x \to 0, y \to 1, z \to ?)\}$ erfüllen c_2;
- $(x \to 0, y \to 1, z \to ?)$ lässt sich aber zu keiner Lösung erweitern, da z entweder 0 oder 1 sein muss und somit garantiert c_1 verletzt

Beispiel

Institute for Software & Systems Engineering

Problem

CSP (X, D, C) mit

- $X = \{x, y, z\}$
- $D_x = D_y = \{0, 1, 2\}$, $D_z = \{0, 1\}$
- C
  - $c_1 : x \neq y,\ y \neq z,\ x \neq z$
  - $c_2 : x + 1 = y$

```
var 0..2: x;
var 0..2: y;
var 0..1: z;

% c1
constraint x != y /\ y != z /\ x != z;
% c2
constraint x + 1 = y;

solve satisfy;
```

Welche Zuweisung ist eine Lösung dieses Problems?

- $\Theta = \{(x \to 1, y \to 2, z \to ?), (x \to 0, y \to 1, z \to ?)\}$ erfüllen c_2;
- $(x \to 0, y \to 1, z \to ?)$ lässt sich aber zu keiner Lösung erweitern, da z entweder 0 oder 1 sein muss und somit garantiert c_1 verletzt
- Also ist die einzige Lösung $(x \to 1, y \to 2, z \to 0)$

Komplexität?

Sind Constraint-Probleme schwierig?

Theorem

Das zu einem CSP gehörende Entscheidungsproblem ist NP-vollständig.

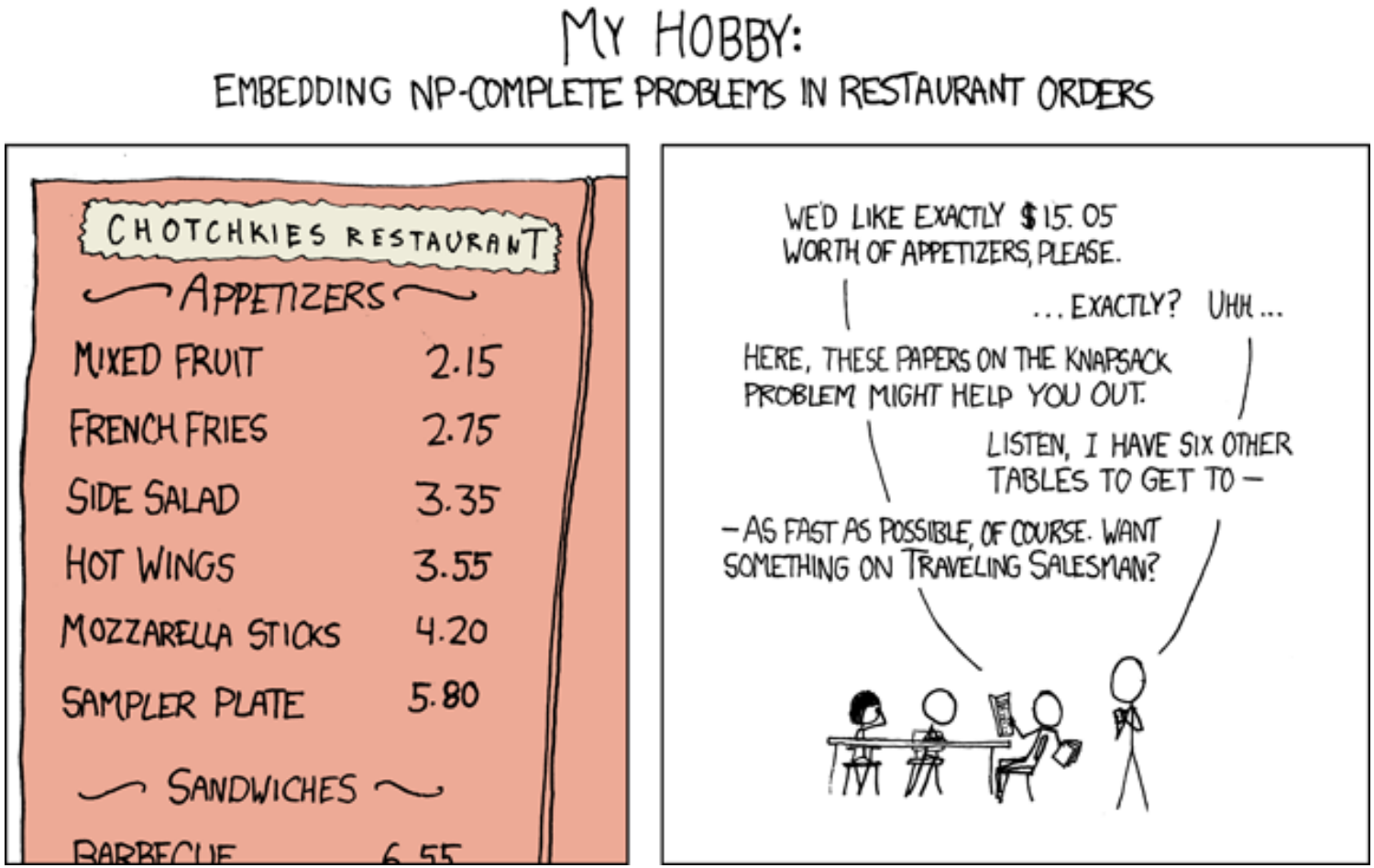
Bew. Sei $A = \{a_1, \ldots, a_n\} \subseteq \mathbb{Z}$ und $s \in \mathbb{Z}$; Gibt es eine Teilmenge $A' \subseteq A$ sodass $\sum_{a \in A'} a = s$? (Teilsummenproblem, *subset sum* $\rightarrow$ NP-vollständig)

Definiere als CSP (X, D, C):

- $X = \{x_1, \ldots, x_n\}$ mit $D_{x_i} = \{0, 1\}$
- $C \equiv \{[\![s = \sum_{i=1}^{n} x_i \cdot a_i]\!]\}$

```
int: n = 5; int: s = 4;
array[1..n] of int: a = [-7, -3, -2, 5, 8];

array[1..n] of var 0..1: x;

constraint sum(i in 1..n) (x[i]*a[i]) = s;
solve satisfy;
```

NP-Vollständigkeit



https://xkcd.com/287/

Constraint-Algorithmen

Institute for Software & Systems Engineering

Wesentliche Kernrichtungen für Algorithmen:

- Systematische (vollständige) Suche ("Try")
  - Backtracking
  - Branch & Bound

Constraint-Algorithmen

Institute for Software & Systems Engineering

Wesentliche Kernrichtungen für Algorithmen:

- Systematische (vollständige) Suche ("Try")
  - Backtracking
  - Branch & Bound
- Constraint-Propagation, Inferenz
  - Einfache, lokale Konsistenzchecks (*Logische Schlüsse*)
  - Reduktion der *Domänen*

Constraint-Algorithmen

Institute for Software & Systems Engineering

Wesentliche Kernrichtungen für Algorithmen:

- Systematische (vollständige) Suche ("Try")
  - Backtracking
  - Branch & Bound
- Constraint-Propagation, Inferenz
  - Einfache, lokale Konsistenzchecks (*Logische Schlüsse*)
  - Reduktion der *Domänen*
- Relaxierung
  - Löse einfachere Teilprobleme
  - Nehme Ergebnis als *Schranken*

Constraint-Algorithmen

Institute for Software & Systems Engineering

Wesentliche Kernrichtungen für Algorithmen:

- Systematische (vollständige) Suche ("Try")
  - Backtracking
  - Branch & Bound
- Constraint-Propagation, Inferenz
  - Einfache, lokale Konsistenzchecks (*Logische Schlüsse*)
  - Reduktion der *Domänen*
- Relaxierung
  - Löse einfachere Teilprobleme
  - Nehme Ergebnis als *Schranken*
- Lokale (*heuristische*) Suche
  - *Min-Conflicts*-Heuristik
  - Large-neighborhood Search
  - Tabu-Suche / Simulated Annealing

Systematische Suche

Institute for Software & Systems Engineering

Partielle Zuweisungen schrittweise um ein Variablen-Wert-Paar erweitert.

Ausnützen der Konjunktivität: Wenn eine partielle Zuweisung bereits einen Constraint verletzt, wird die letzte Zuweisung rückgängig gemacht (*backtracking*) und neuer Wert versucht.

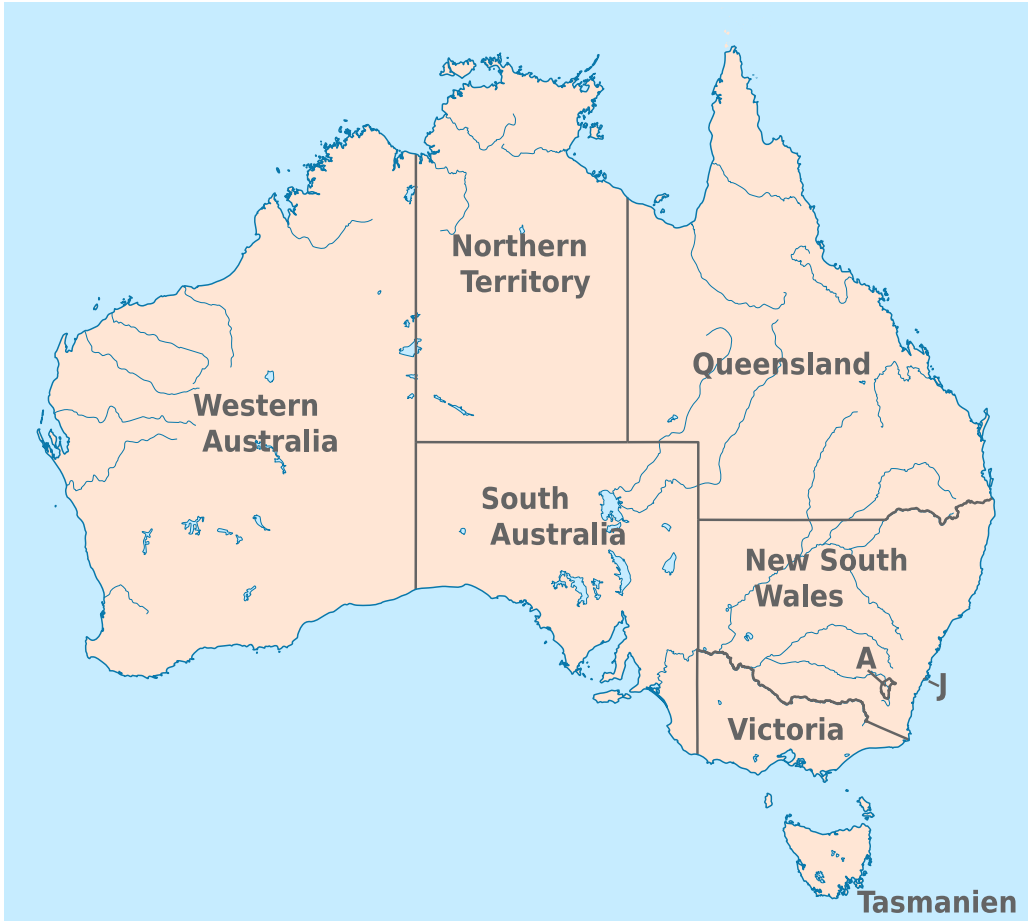
Im schlimmsten Fall exponentielle Exploration aller vollständigen Zuweisungen $O(|D|^{|X|})$.

$\rightarrow$ in der Praxis:

- Einschränkung der Lösungsraums durch Propagation
- Frühzeitiges Abschneiden von "Sackgassen"
- Frühzeitiges Probieren von vielversprechenden Kandidaten

Suche am Beispiel Map Coloring

Institute for
Software & Systems
Engineering

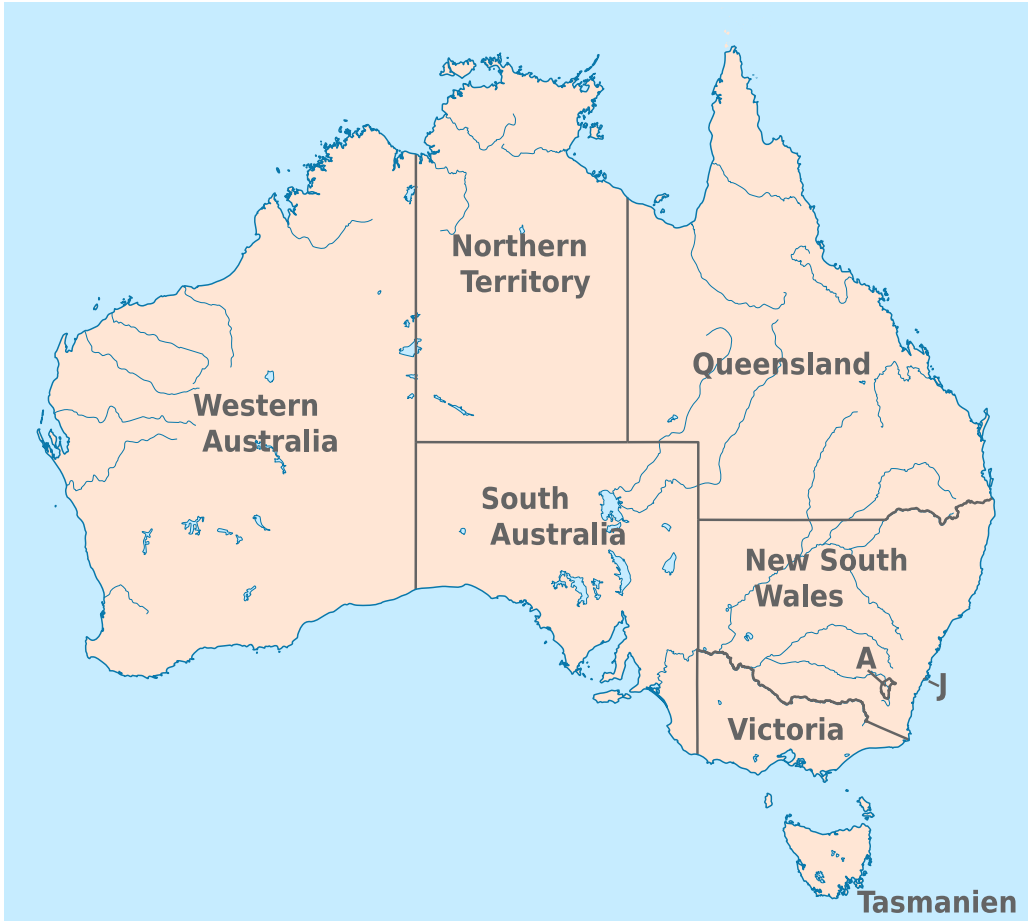


Modell?

$X =$

Suche am Beispiel Map Coloring

Institute for
Software & Systems
Engineering

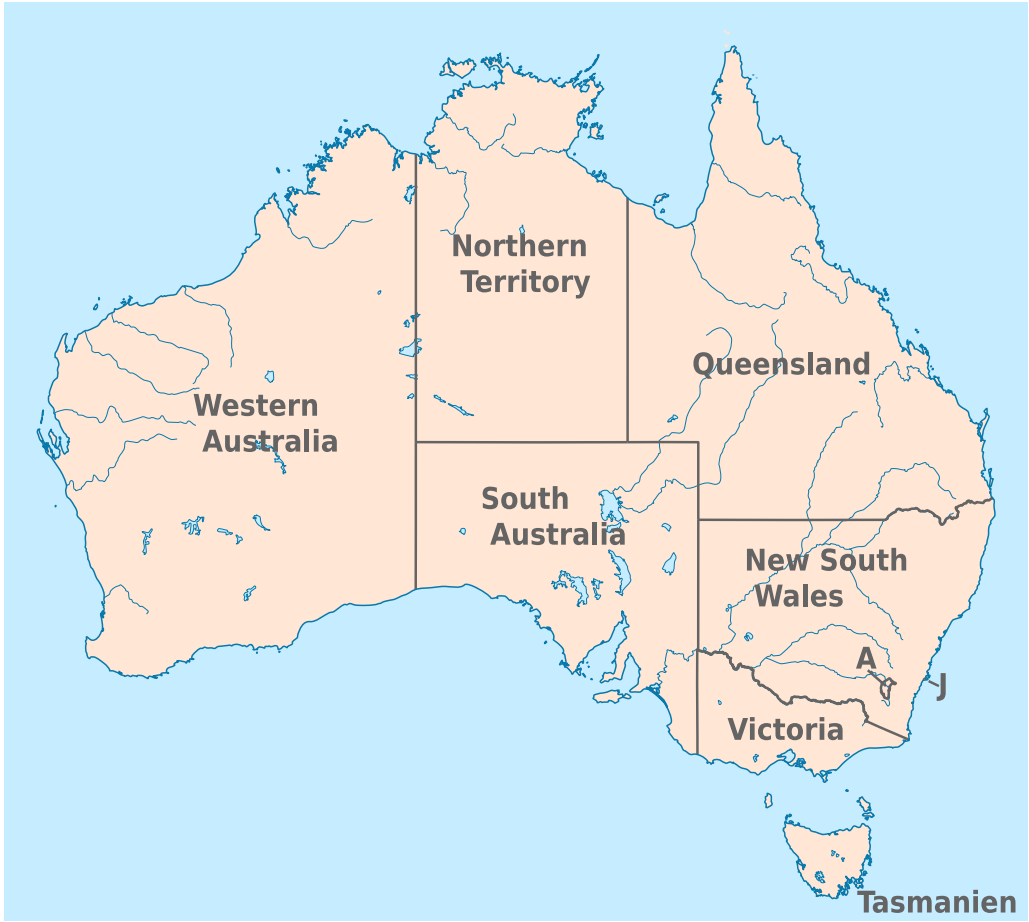


Modell?

$X = \{\mathrm{WA}, \mathrm{NT}, \ldots \mathrm{V}\}$, $D_x =$

Suche am Beispiel Map Coloring

Institute for
Software & Systems
Engineering

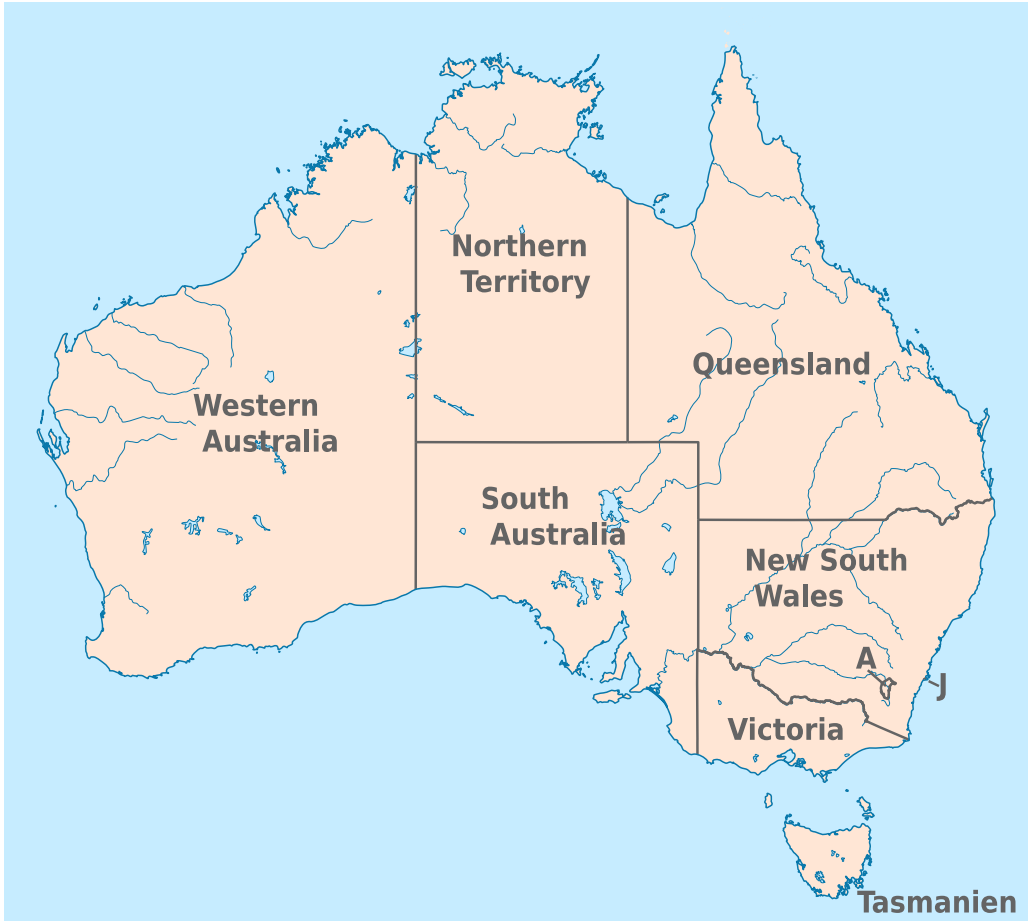


Modell?

$X = \{\mathrm{WA}, \mathrm{NT}, \ldots \mathrm{V}\}$, $D_x = \{\mathrm{r}, \mathrm{g}, \mathrm{b}\}$, $C =$

Suche am Beispiel Map Coloring

Institute for
Software & Systems
Engineering

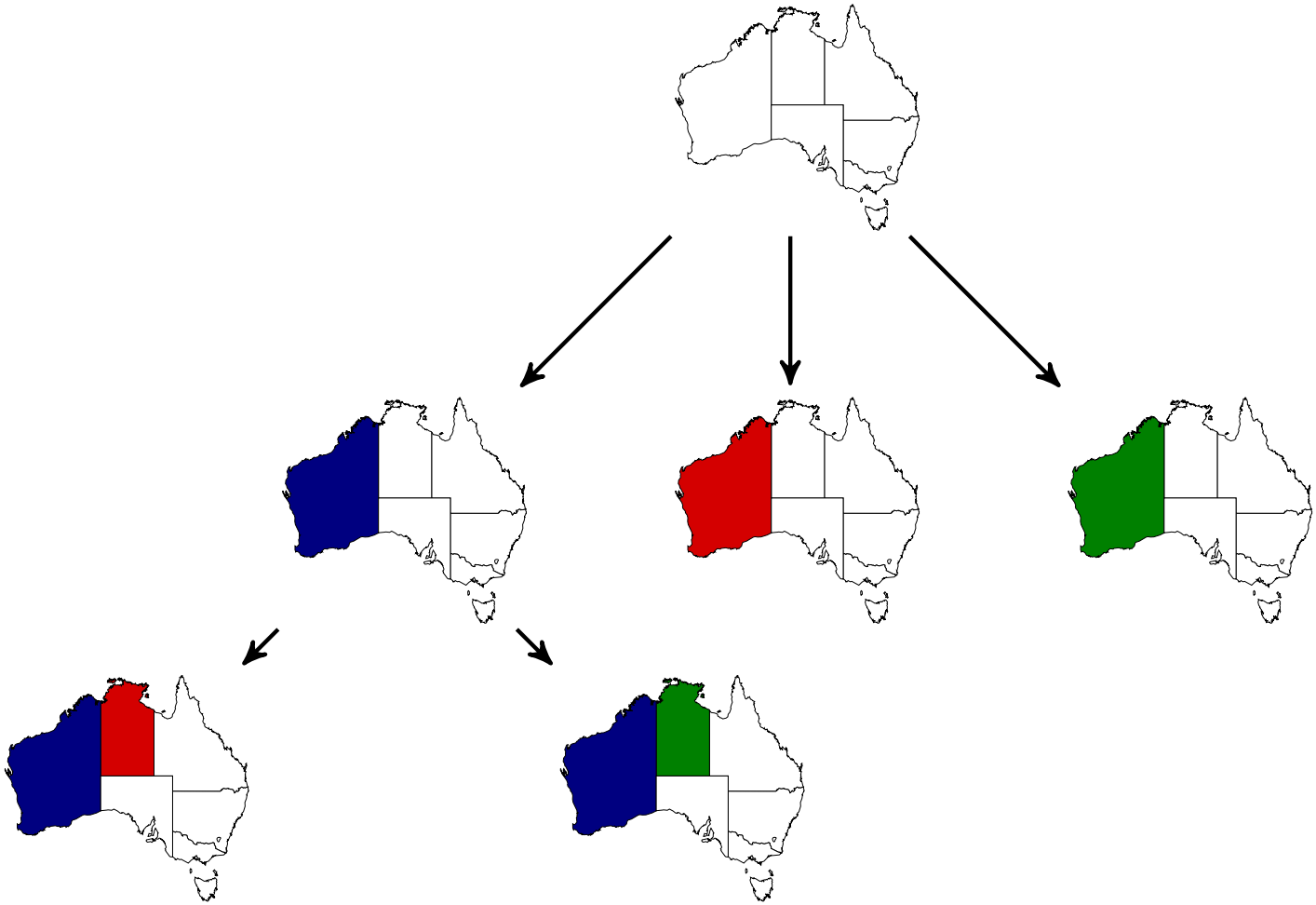


Modell?

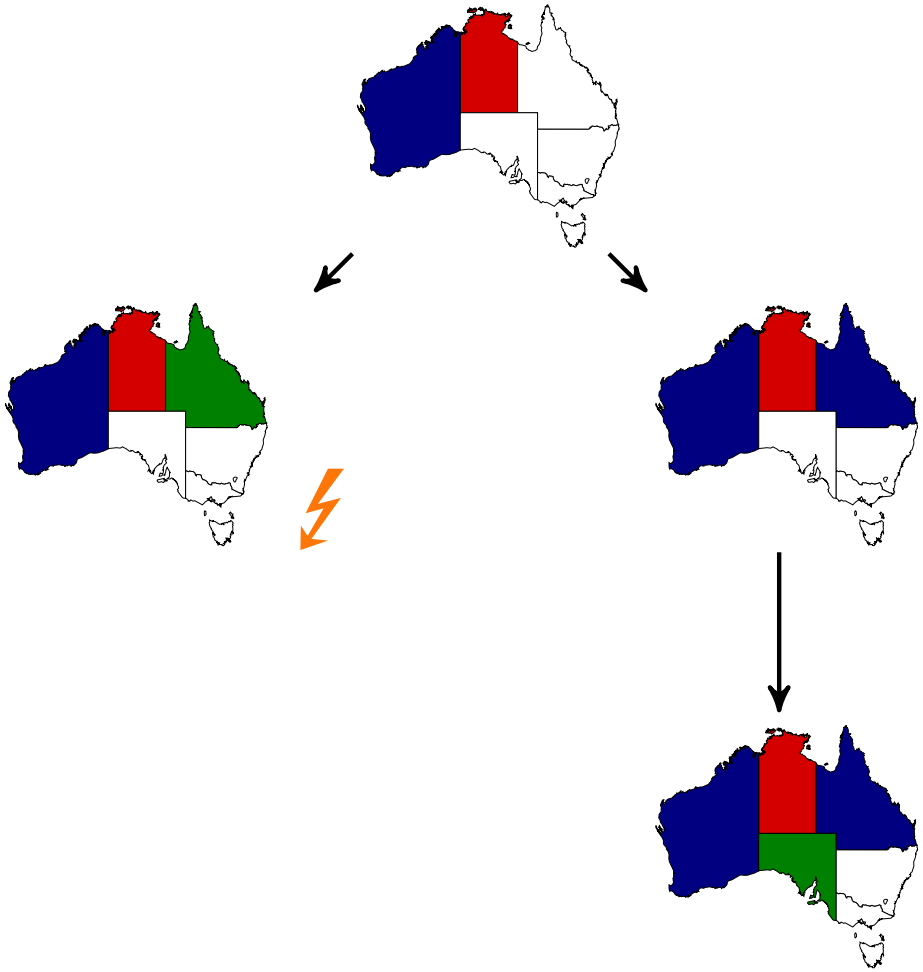
$X = \{\mathrm{WA}, \mathrm{NT}, \ldots \mathrm{V}\},\ D_x = \{\mathrm{r}, \mathrm{g}, \mathrm{b}\},\ C = \{\mathrm{WA} \neq \mathrm{NT}, \mathrm{NT} \neq \mathrm{SA}, \ldots\}$

Systematische Suche I

Institute for
Software & Systems
Engineering



Systematische Suche II



Systematische Suche

Institute for Software & Systems Engineering

function SEARCH((X, D, C))
  $\theta \leftarrow \emptyset$
  return SEARCHREC($\theta, (X, D, C)$)

function SEARCHREC(θ, (X, D, C))
  if $|\theta| = |X|$ **then return** θ
  else
    $x \leftarrow$ SELECTUNASSIGNEDVARIABLE(θ, (X, D, C))
    for all $d \in D_x$ **do**
      $\theta' \leftarrow \theta \cup (x, d)$
      if $\forall c \in C : c(\theta') \neq \text{ff}$ **then** ▷ Feasibility Check
        $D' \leftarrow$ PROPAGATECONSTRAINTS(θ', D) ▷ Shrinks D
        $\theta'' \leftarrow$ SEARCHREC($\theta', (X, D', C)$)
        if $\theta'' \neq \perp$ **then return** θ''
    return $\perp$

Stellschrauben

Institute for Software & Systems Engineering

Heuristiken entwickelt, um "geschickte" Auswahl der Reihenfolgen von Variablen $x \in X$ und Werten $d \in D_x$ zu treffen.

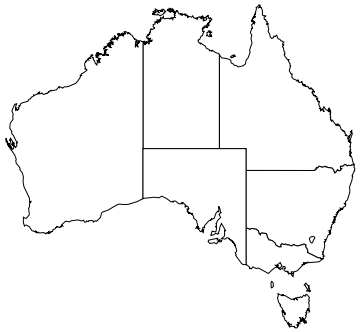
- *Minimum-Remaining-Values* (MRV): Wähle die Variable mit der kleinsten verbleibenden Domäne als nächste
- *Most-constrained* (MC): Wähle die Variable aus, die an den meisten Constraints beteiligt ist.
- *Minimum reduction* (MR): Wähle den nächsten zu testenden Wert so aus, dass minimal viele Domänenreduktionen erfolgen.

Dadurch frühe Sackgassenerkennung und sinnvolle Wahl, um Suchbaum zu begrenzen.

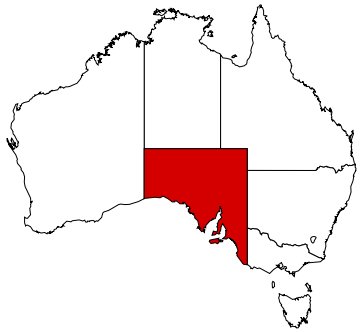
Variablenordnung

Institute for Software & Systems Engineering

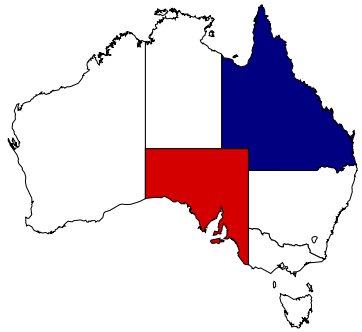
Beispiel für MRV+MC (MC bricht Unentschieden nach MRV):



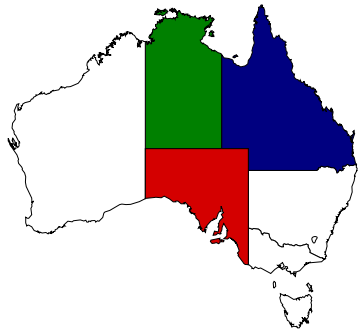
Alle unbelegt



SA in 5 Constraints



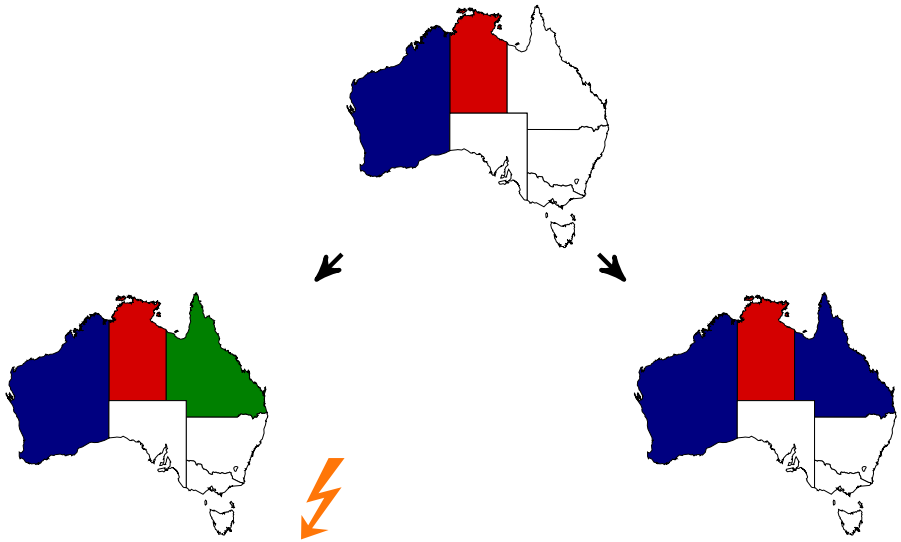
$|D_{\mathrm{QL}}|$ nur 2



$|D_{\mathrm{NT}}|$ nur 1

Wertordnung

Wähle die Belegung für die selektierte Variable, die die wenigsten Domäneneinschränkungen zur Folge haben.

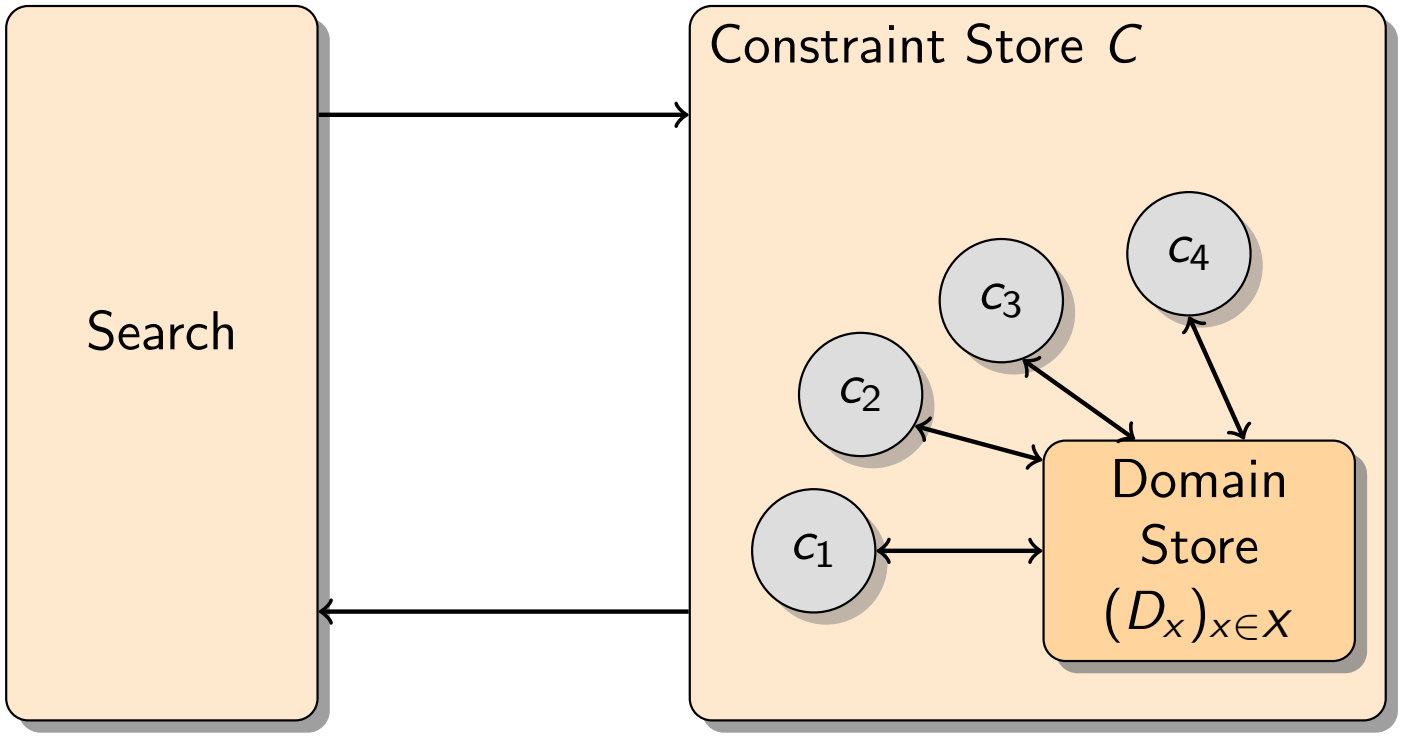


Es bleibt kein Wert für SA.

Es bleibt ein Wert für SA.

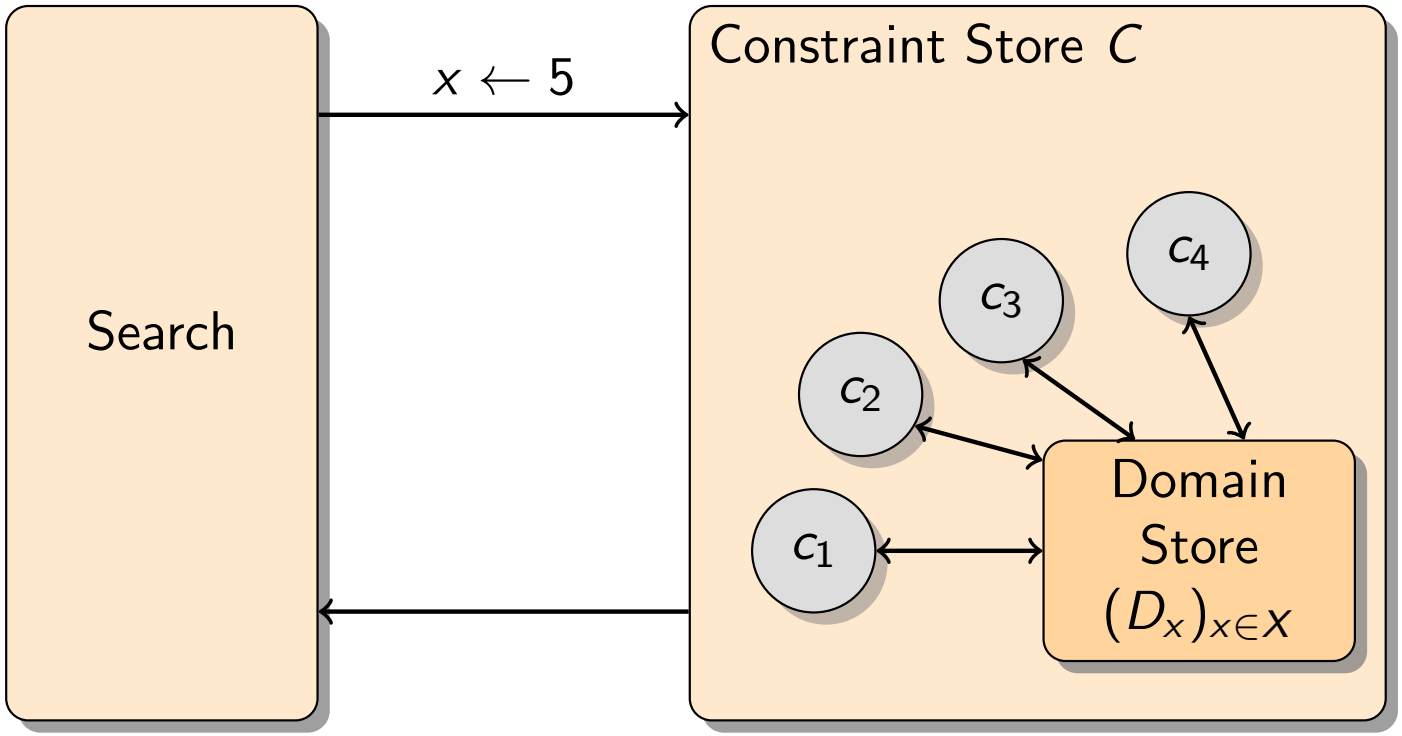
Architektur von Constraint-Lösern

Institute for
Software & Systems
Engineering



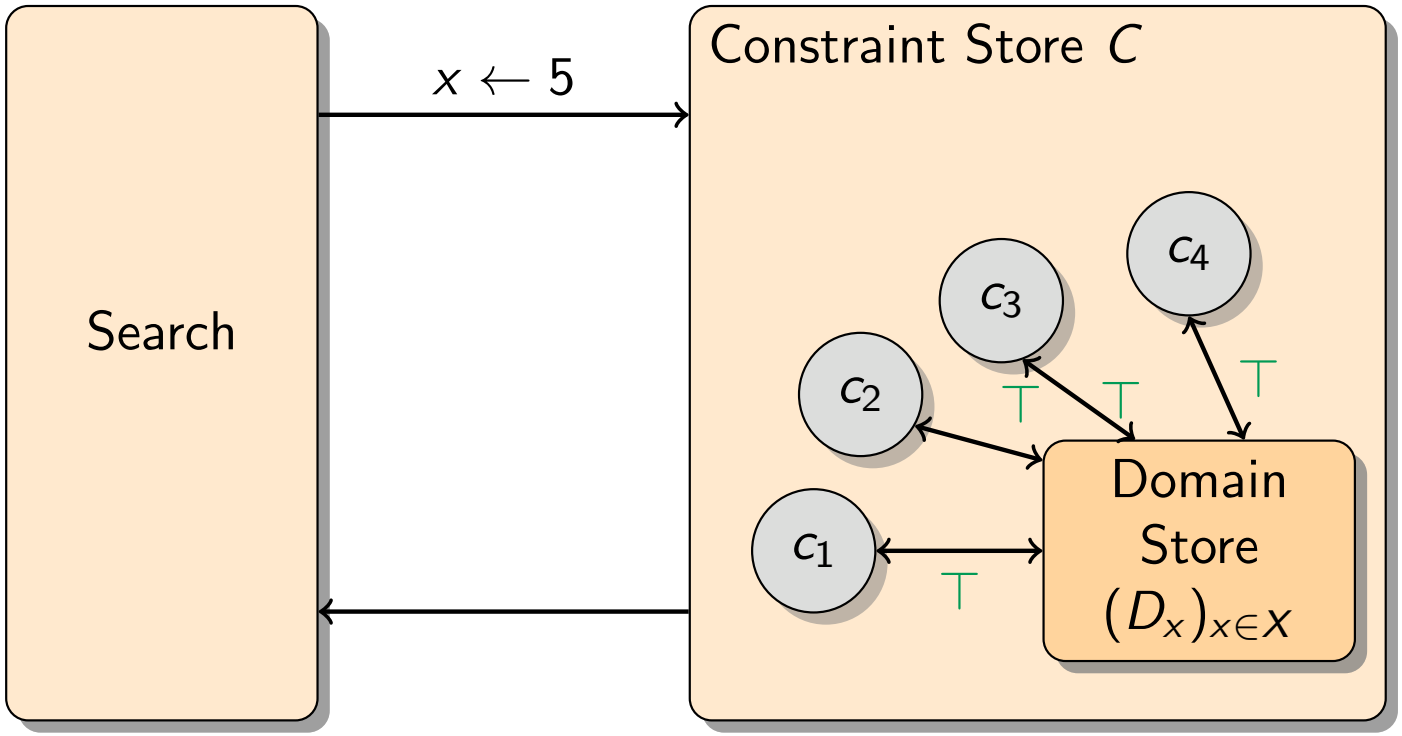
Architektur von Constraint-Lösern

Institute for
Software & Systems
Engineering



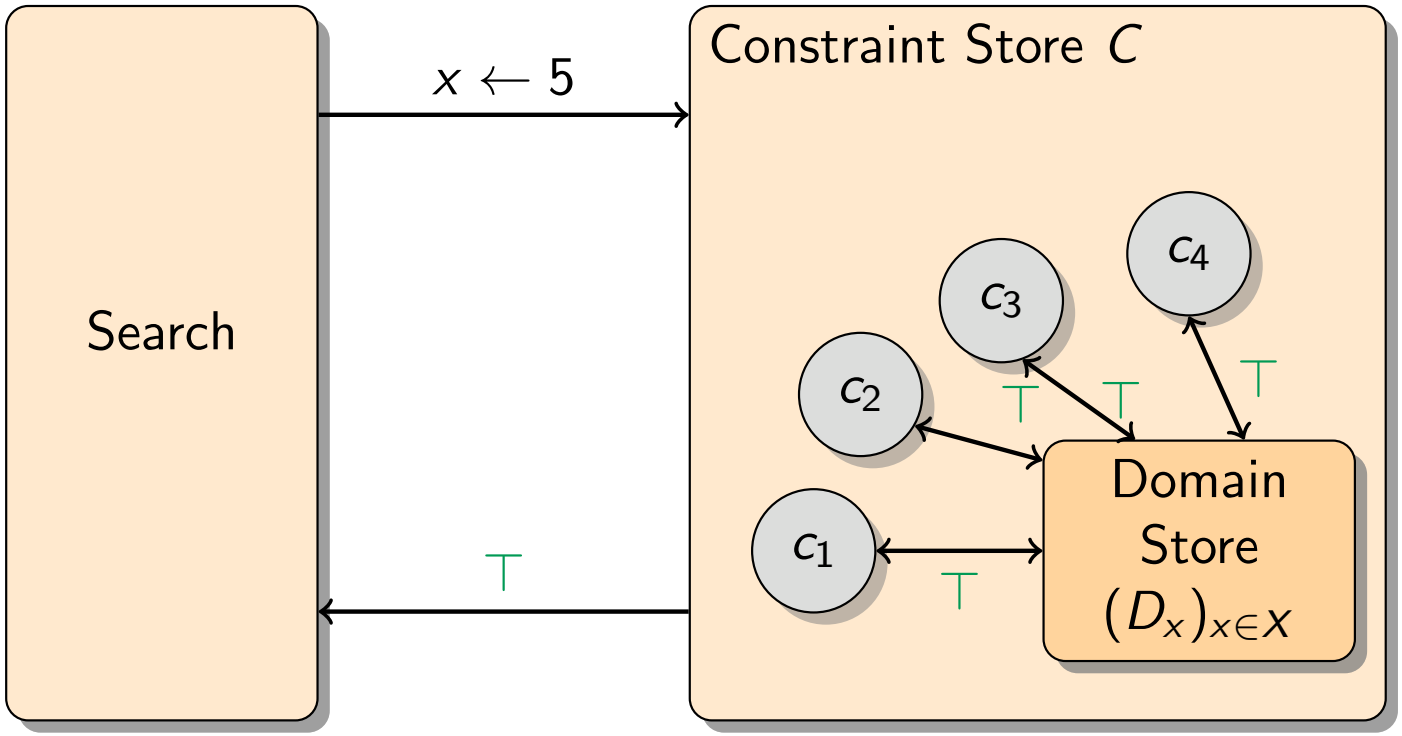
Architektur von Constraint-Lösern

Institute for
Software & Systems
Engineering



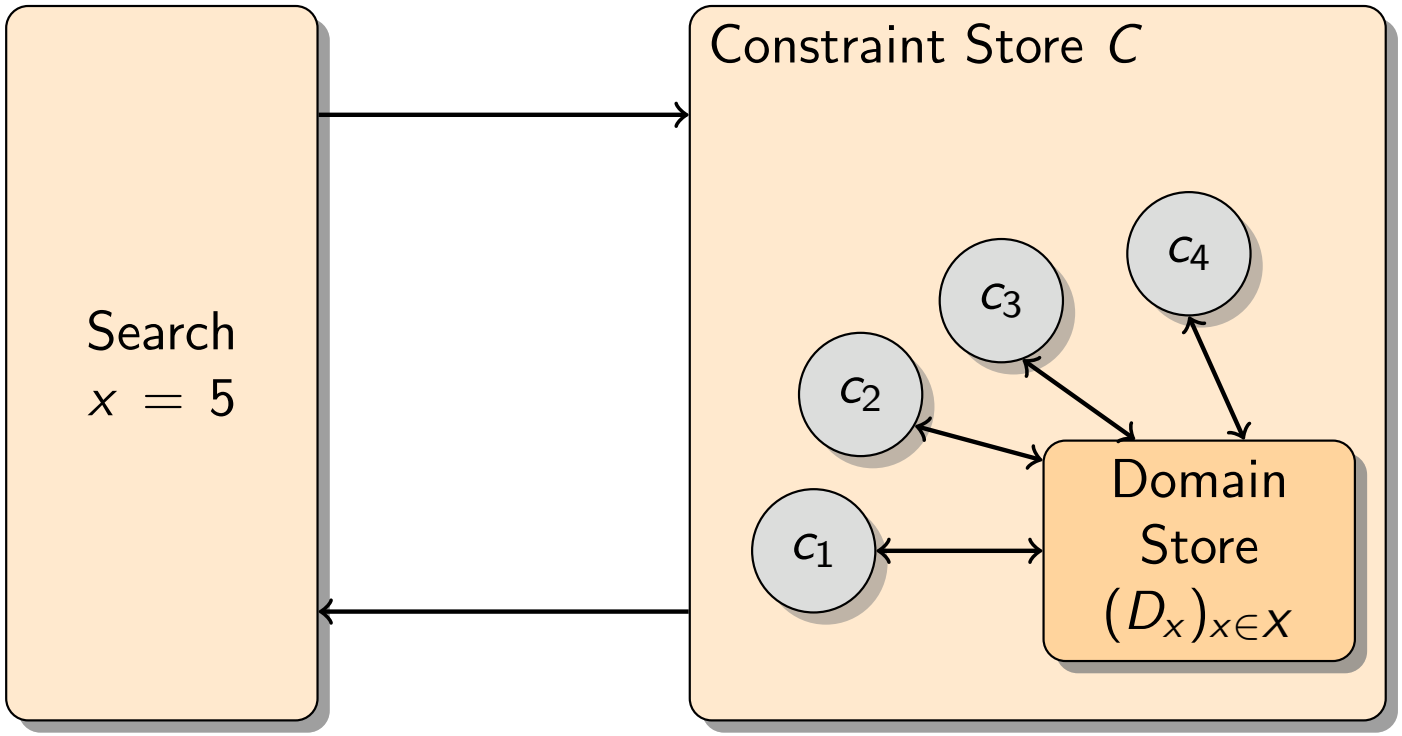
Architektur von Constraint-Lösern

Institute for
Software & Systems
Engineering

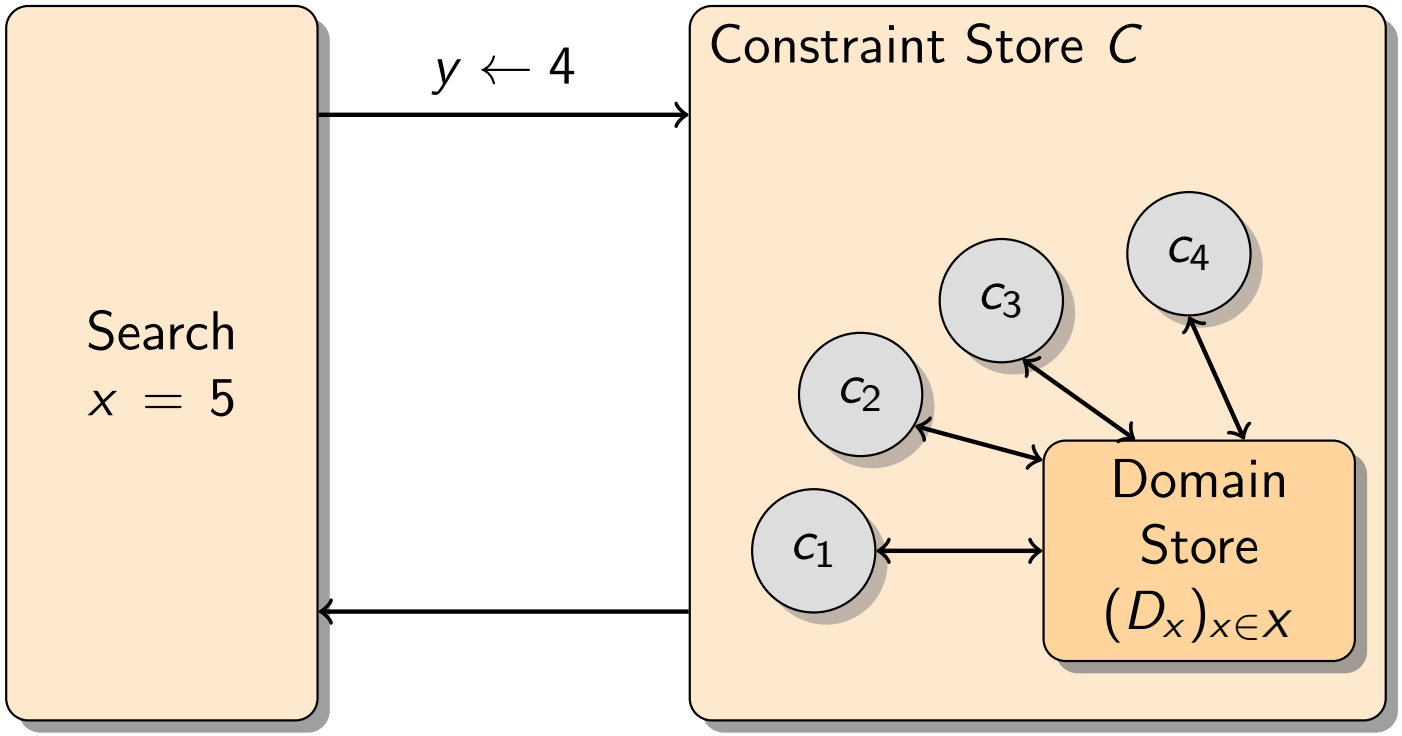


Architektur von Constraint-Lösern

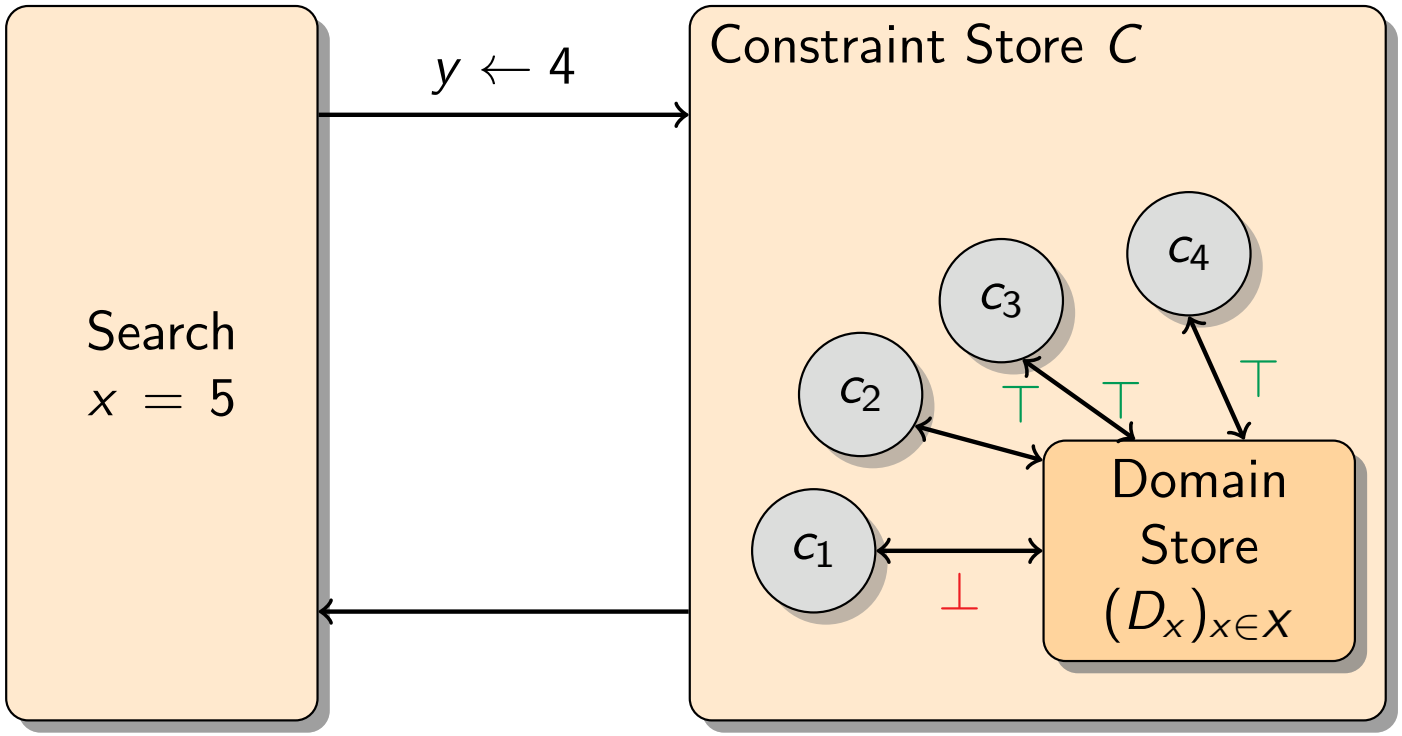
Institute for
Software & Systems
Engineering



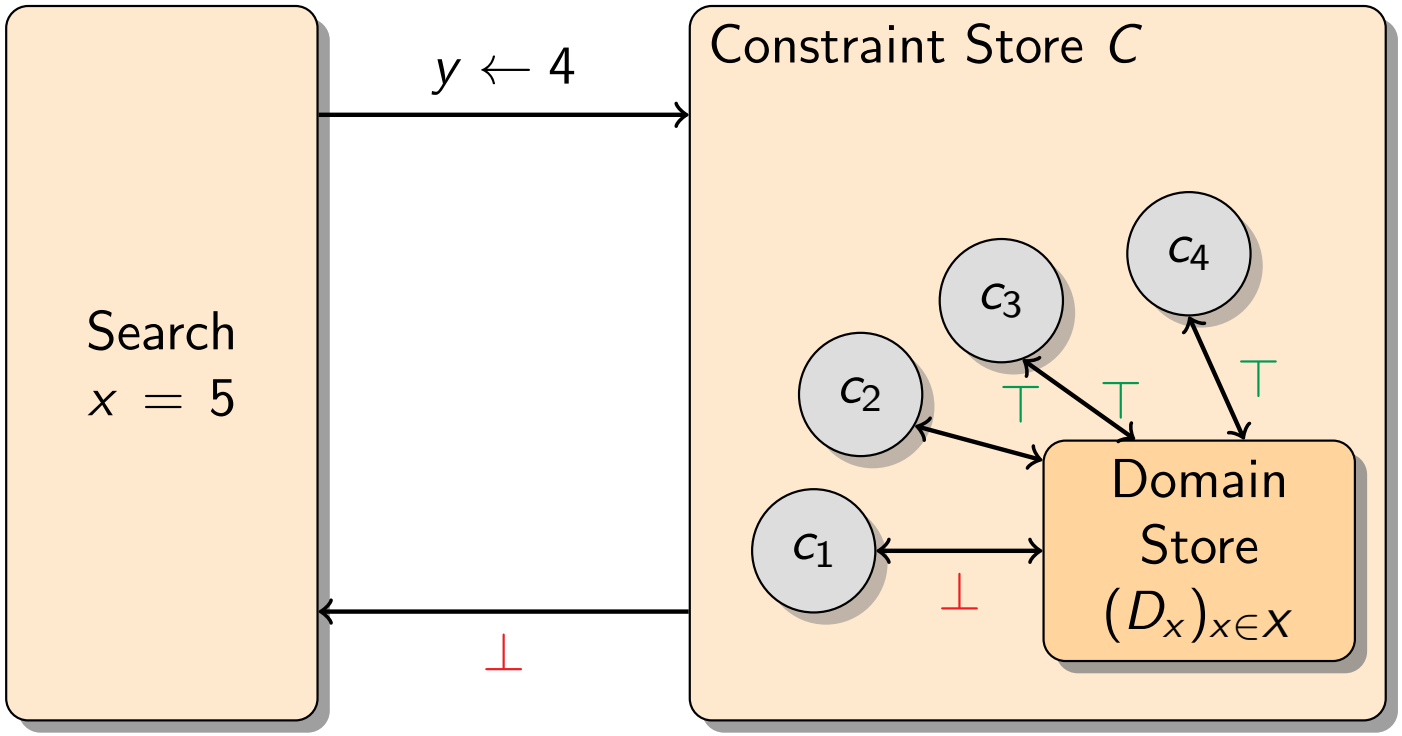
Architektur von Constraint-Lösern



Architektur von Constraint-Lösern



Architektur von Constraint-Lösern



Constraint-Propagation

Institute for Software & Systems Engineering

- Nutze Constraints, um Suchraum einzugrenzen
- Idee: Entferne alle Werte aus Domänen, die in keiner Lösung vorkommen können (*Domain Store*)

Constraint-Propagation

Institute for
Software & Systems
Engineering

- Nutze Constraints, um Suchraum einzugrenzen
- Idee: Entferne alle Werte aus Domänen, die in keiner Lösung vorkommen können (*Domain Store*)
- $|x_1 - x_2| > 5$ für $x_1, x_2 \in X$ und $D_{x_i} = \{1, \ldots, 10\}$
- Welche Werte sind nicht möglich?
- jeweils 5 und 6

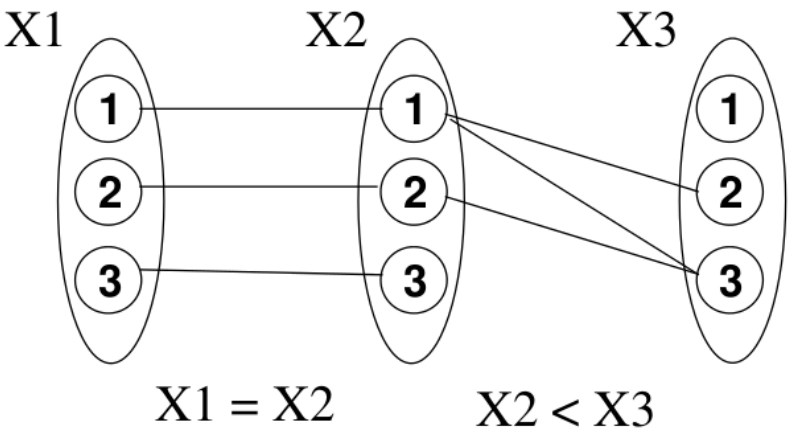
Constraint-Propagation

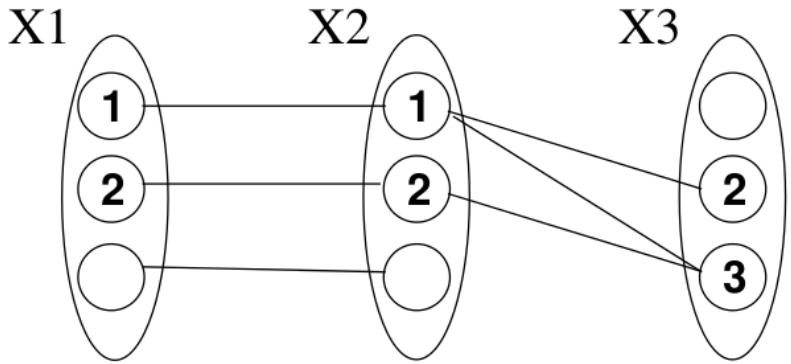
Institute for Software & Systems Engineering

- Nutze Constraints, um Suchraum einzugrenzen
- Idee: Entferne alle Werte aus Domänen, die in keiner Lösung vorkommen können (*Domain Store*)
- $|x_1 - x_2| > 5$ für $x_1, x_2 \in X$ und $D_{x_i} = \{1, \ldots, 10\}$
- Welche Werte sind nicht möglich?
- jeweils 5 und 6
- Propagierungsschritte beeinflussen einander
  - "Kettenreaktion"
  - Fixpunktalgorithmus $\rightarrow$ Keine Propagierung möglich
  - Wenn Domäne nur mehr einen Wert enthält, *muss* dieser zugewiesen werden.

Constraint-Propagation: Beispiel

Institute for
Software & Systems
Engineering





Entfernen von Werten, die zu keiner Lösung führen können.

Send More Money

Institute for Software & Systems Engineering

Problem

```
  S E N D
+ M O R E
---------
M O N E Y
```

Send More Money

Institute for Software & Systems Engineering

Problem

```
  S E N D
+ M O R E
---------
M O N E Y
```

```
C4 C3 C2 C1
    S  E  N  D
+   M  O  R  E
---------
 M  O  N  E  Y
```

Send More Money

Institute for Software & Systems Engineering

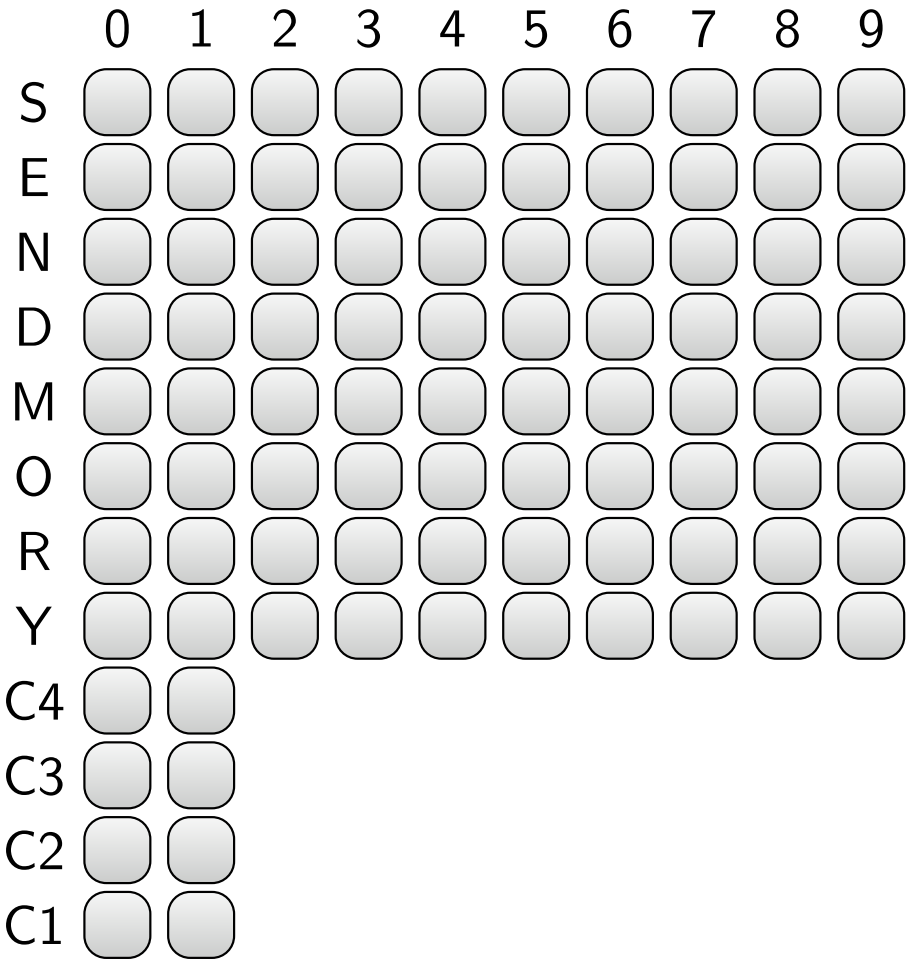
Problem

```
  S E N D
+ M O R E
---------
M O N E Y

C4 C3 C2 C1
    S  E  N  D
+   M  O  R  E
---------
 M  O  N  E  Y
```

```
% send more money
include "alldifferent.mzn";

int: s = 1; int: e = 2; int: n = 3;
int: d = 4; int: m = 5; int: o = 6;
int: r = 7; int: y = 8;

set of int: Letters = {s,e,n,d,m,o,r,y};
set of int: Digits = 0..9;
array[Letters] of var Digits: value;
array[1..4] of var 0..1: carry;

constraint alldifferent(value);
constraint value[s] != 0 /\ value[m] != 0;
constraint carry[4] = value[m];
constraint carry[3] + value[s] + value[m] =
           value[o] + 10*carry[4];
constraint carry[2] + value[e] + value[o] =
           value[n] + 10*carry[3];
constraint carry[1] + value[n] + value[r] =
           value[e] + 10*carry[2];
constraint value[d] + value[e] =
           value[y] + 10*carry[1];
solve satisfy;
```

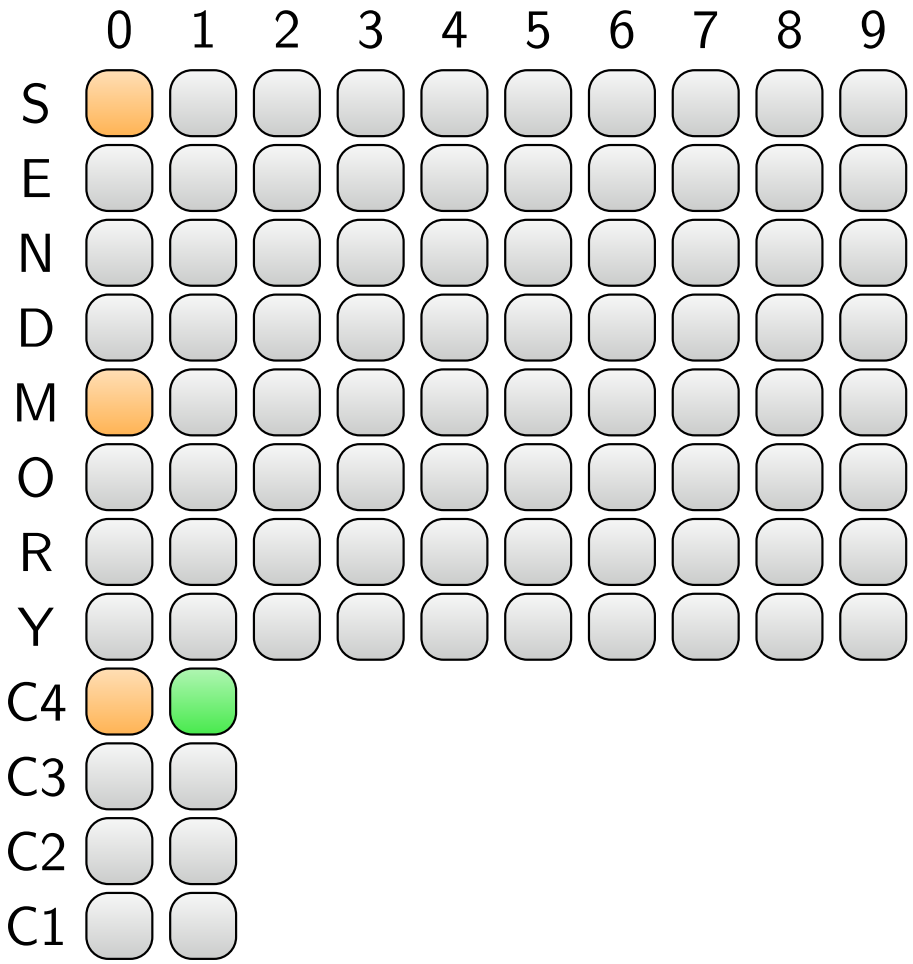
Propagation in Action

Institute for
Software & Systems
Engineering



Propagation in Action

Institute for
Software & Systems
Engineering



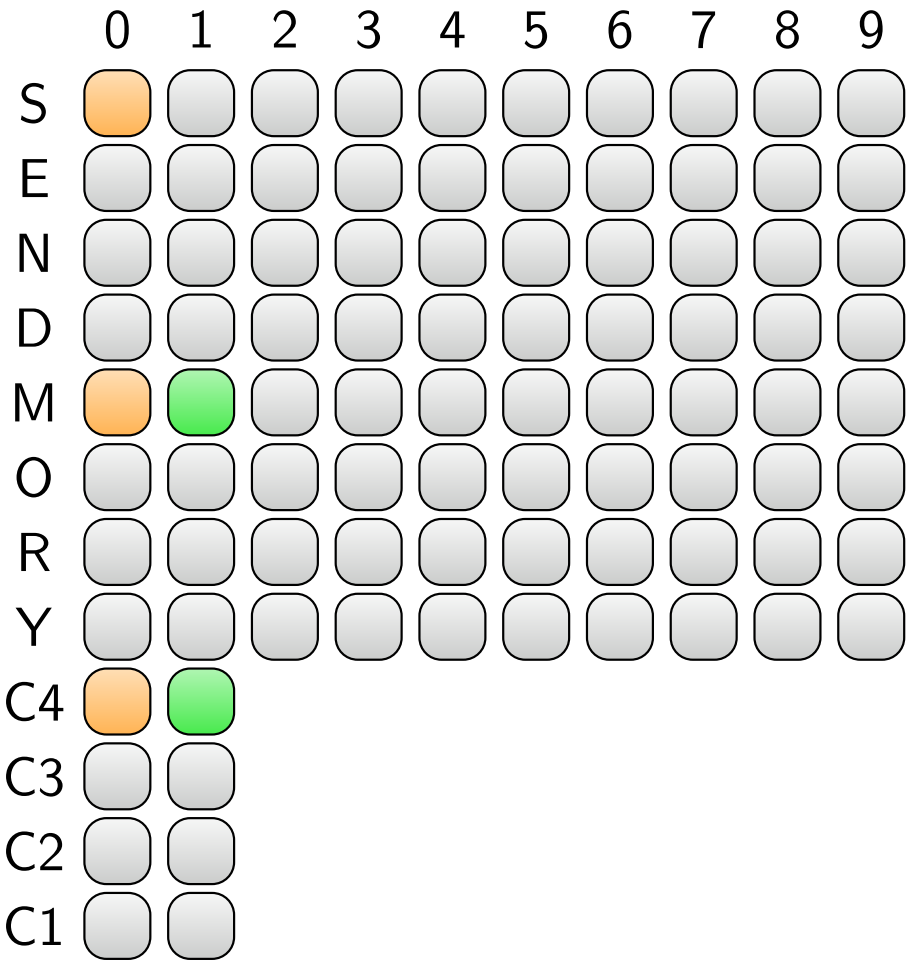
$$\texttt{value[S]} \neq 0$$

$$\texttt{value[M]} \neq 0$$

$$\texttt{c[4]} = \texttt{value[M]}$$

Propagation in Action

Institute for
Software & Systems
Engineering



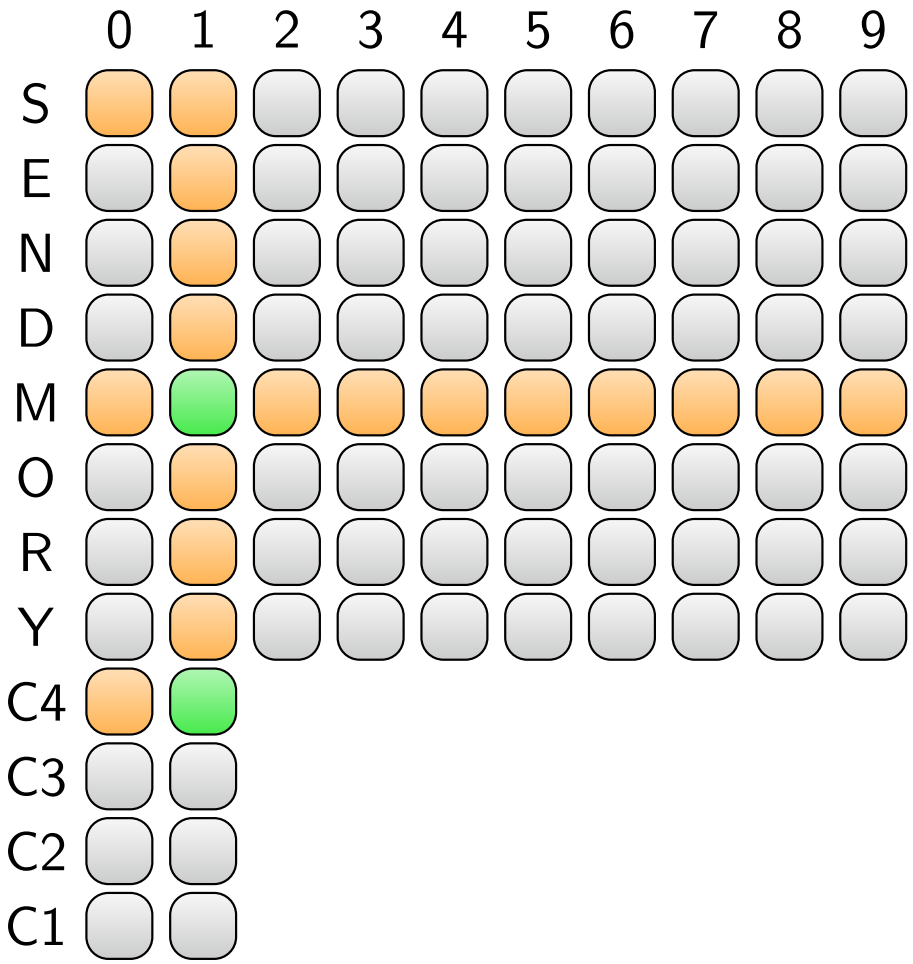
$$\texttt{value[S]} \neq 0$$

$$\texttt{value[M]} \neq 0$$

$$\texttt{c[4]} = \texttt{value[M]}$$

Propagation in Action

Institute for
Software & Systems
Engineering



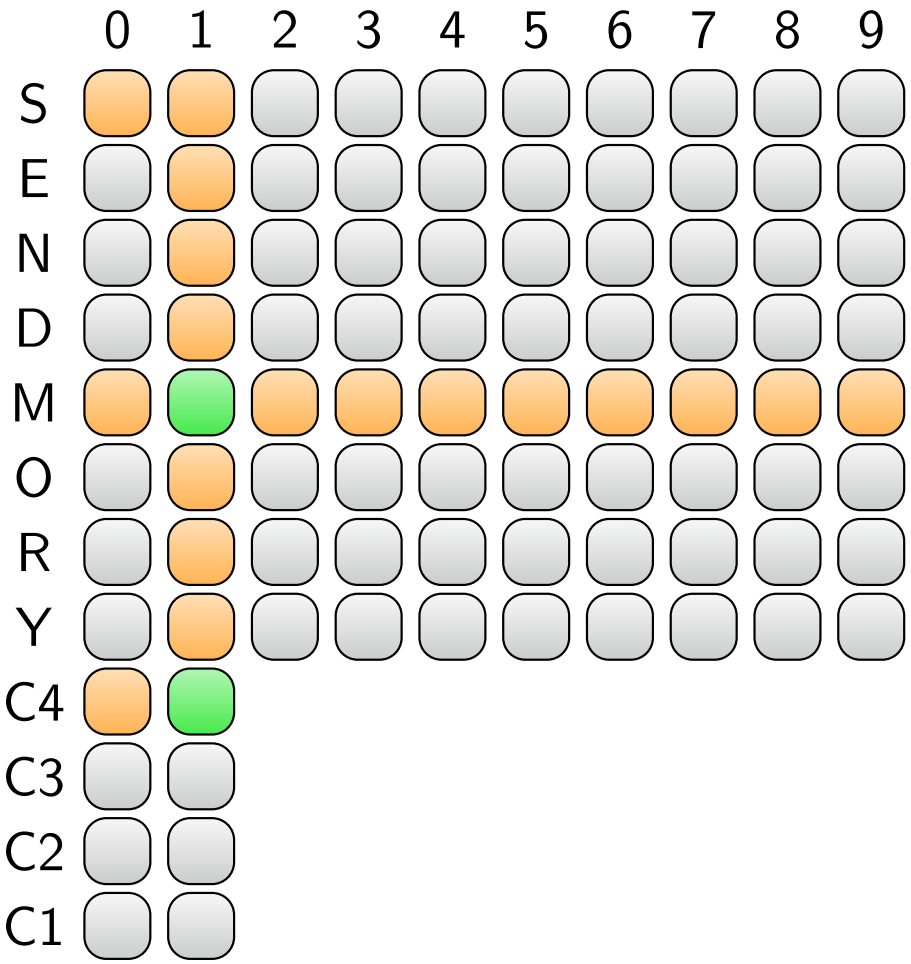
$$\texttt{value[S]} \neq 0$$

$$\texttt{value[M]} \neq 0$$

$$\texttt{c[4]} = \texttt{value[M]}$$

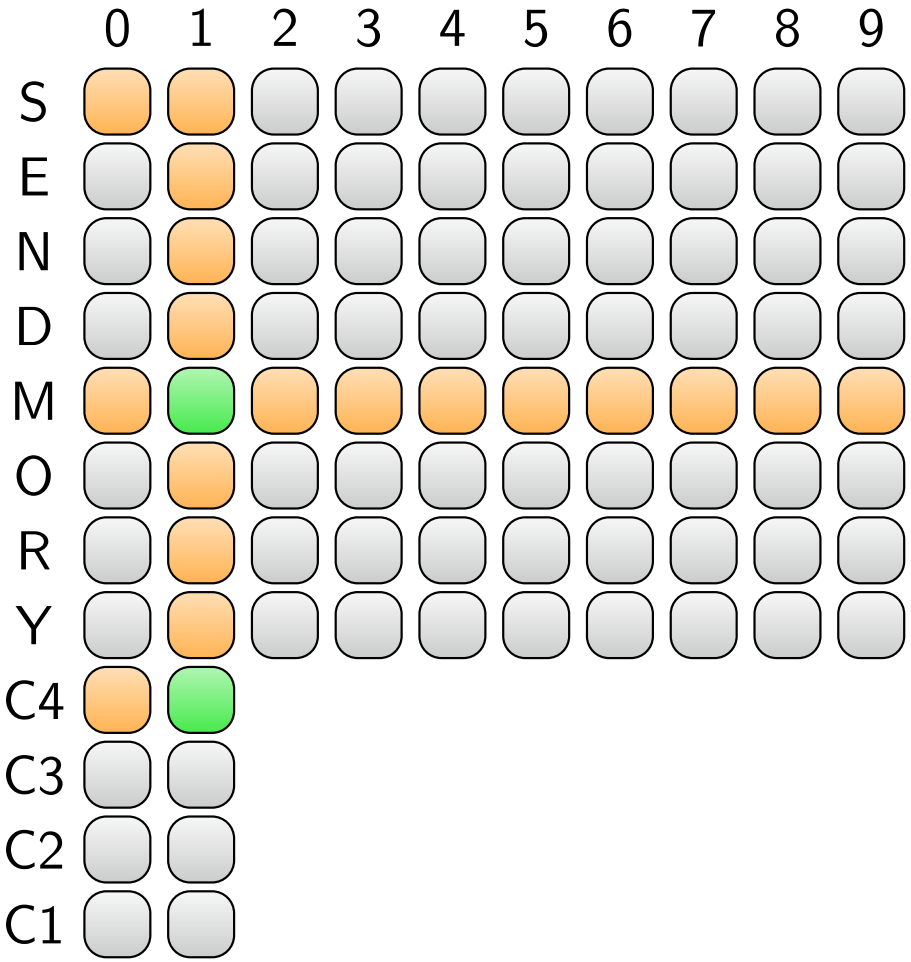
Propagation in Action

Institute for
Software & Systems
Engineering



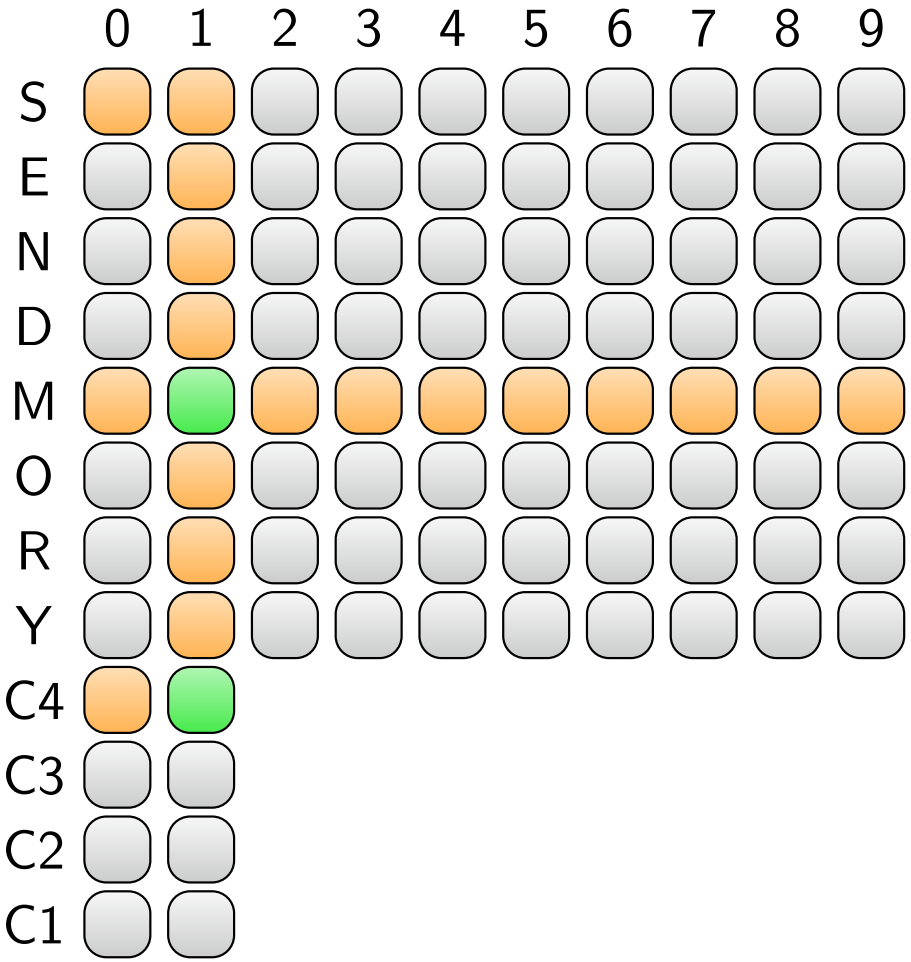
$$\texttt{carry}[3] + \texttt{value}[\texttt{S}] + \texttt{value}[\texttt{M}] = \texttt{value}[\texttt{O}] + 10 * \texttt{carry}[4];$$

Propagation in Action



$$\overbrace{\underbrace{\texttt{carry}[3] + \texttt{value}[\texttt{S}] + 1}_{\in[3..11]} = \texttt{value}[\texttt{0}] + 10;}^{\in[10..19]}$$

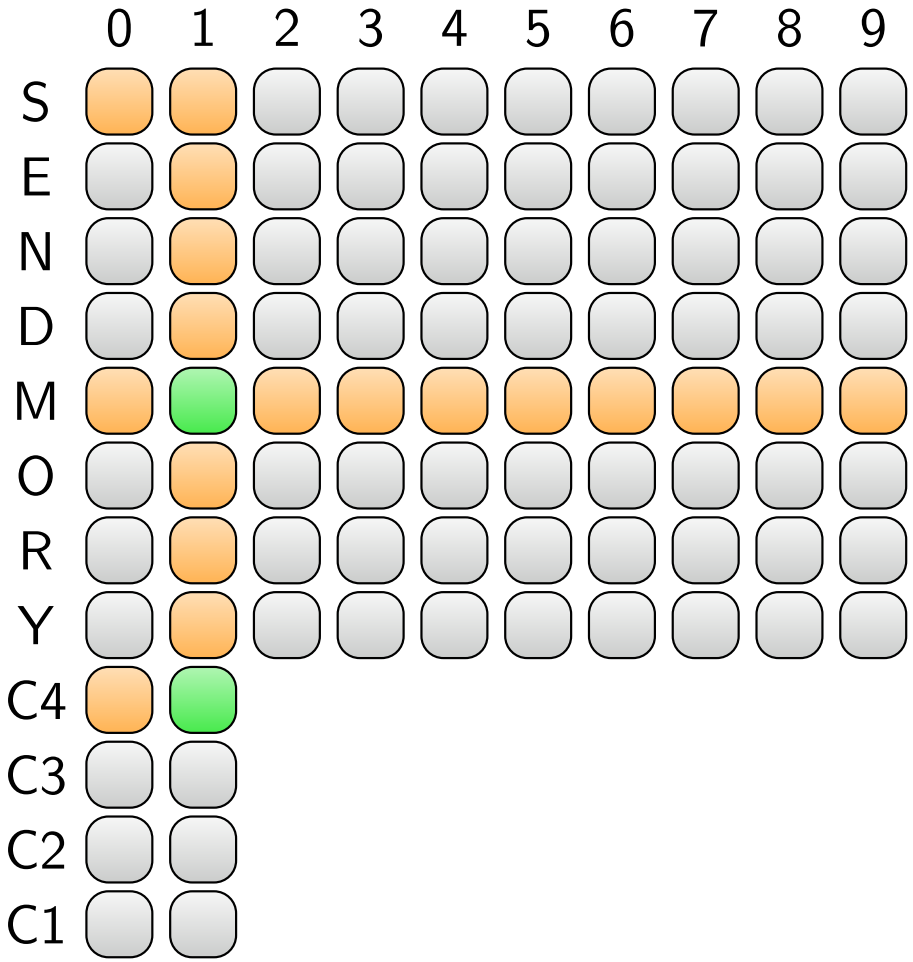
Propagation in Action



$$\overbrace{\underbrace{\texttt{carry}[3] + \texttt{value}[\texttt{S}] + 1}_{\in[3..11]} = \texttt{value}[\texttt{0}] + 10;}^{\in[10..19]}$$

$$[3..11] \cap [10..19] = [10..11]$$

Propagation in Action



$$\overbrace{\underbrace{\texttt{carry}[3] + \texttt{value}[S] + 1}_{\in[3..11]} = \texttt{value}[0] + 10;}^{\in[10..19]}$$

$$[3..11] \cap [10..19] = [10..11]$$

$$10 \leq \texttt{carry}[3] + \texttt{value}[S] + 1 \leq 11$$

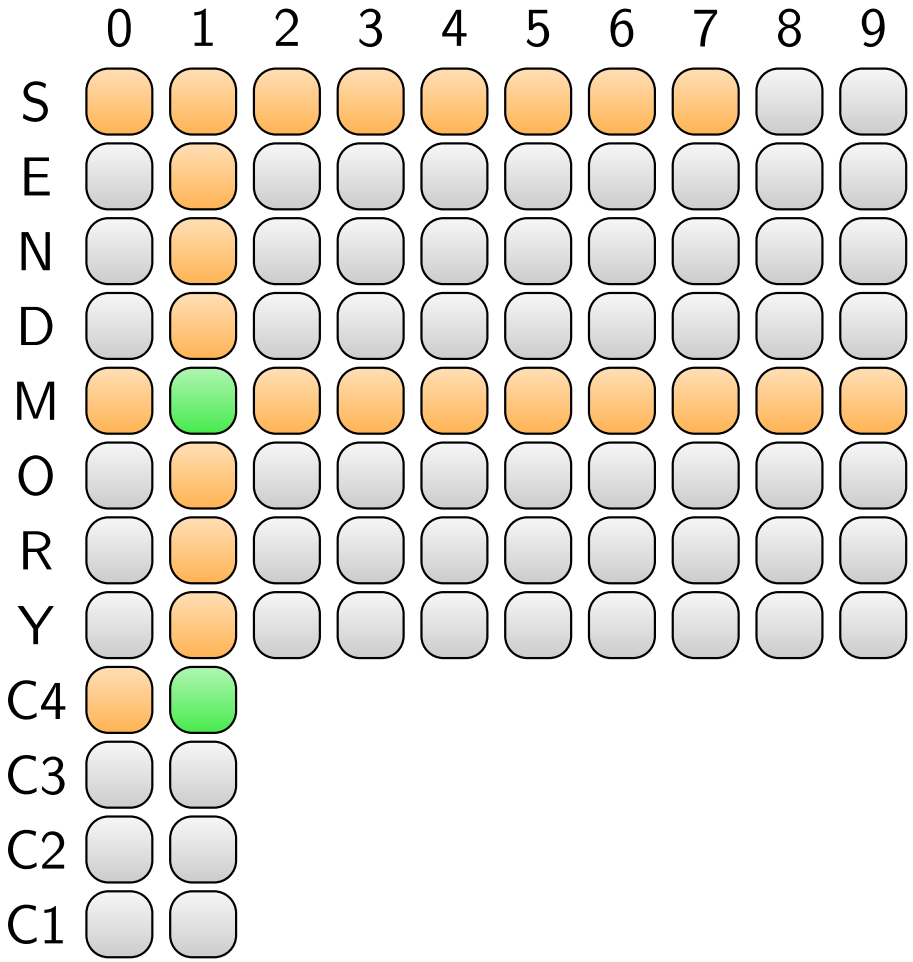
$$9 \leq \texttt{carry}[3] + \texttt{value}[S] \leq 10$$

$$9 - \texttt{carry}[3] \leq \texttt{value}[S]$$

$$\leq 10 - \texttt{carry}[3]$$

$$8 \leq \texttt{value}[S] \leq 10$$

Propagation in Action



$$\overbrace{\underbrace{\texttt{carry}[3] + \texttt{value}[S] + 1}_{\in[3..11]} = \texttt{value}[0] + 10;}^{\in[10..19]}$$

$$[3..11] \cap [10..19] = [10..11]$$

$$10 \leq \texttt{carry}[3] + \texttt{value}[S] + 1 \leq 11$$

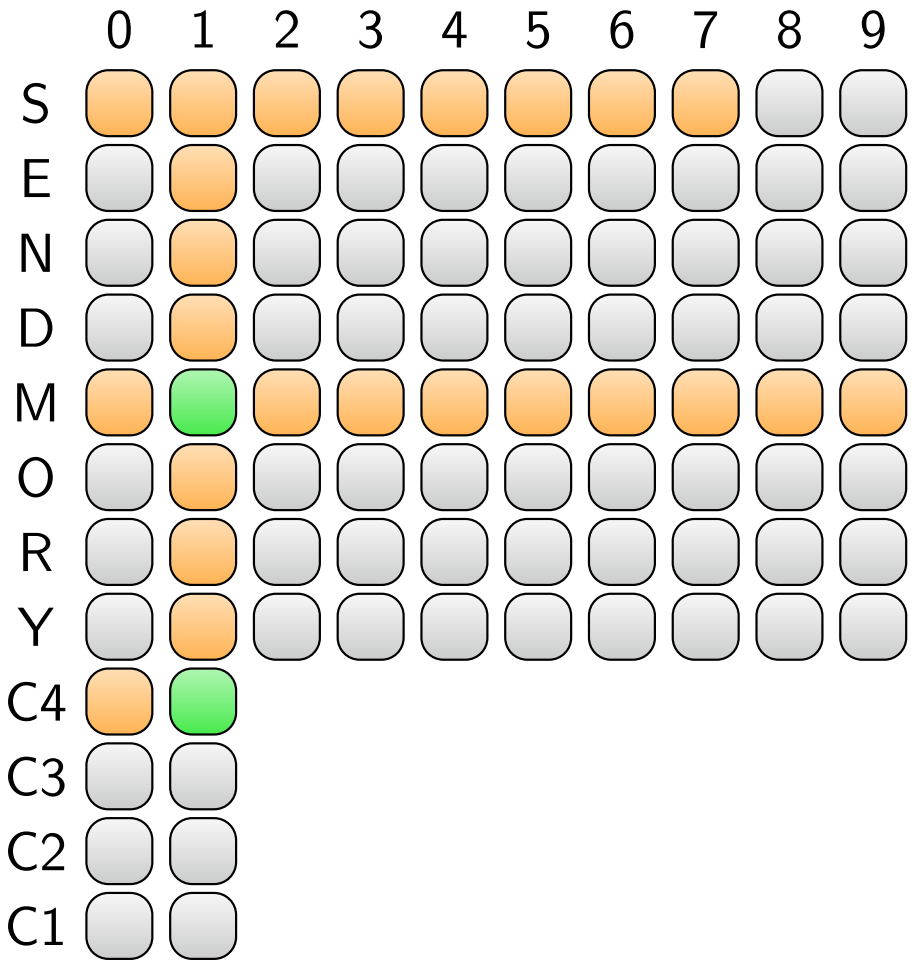
$$9 \leq \texttt{carry}[3] + \texttt{value}[S] \leq 10$$

$$9 - \texttt{carry}[3] \leq \texttt{value}[S]$$

$$\leq 10 - \texttt{carry}[3]$$

$$8 \leq \texttt{value}[S] \leq 10$$

Propagation in Action

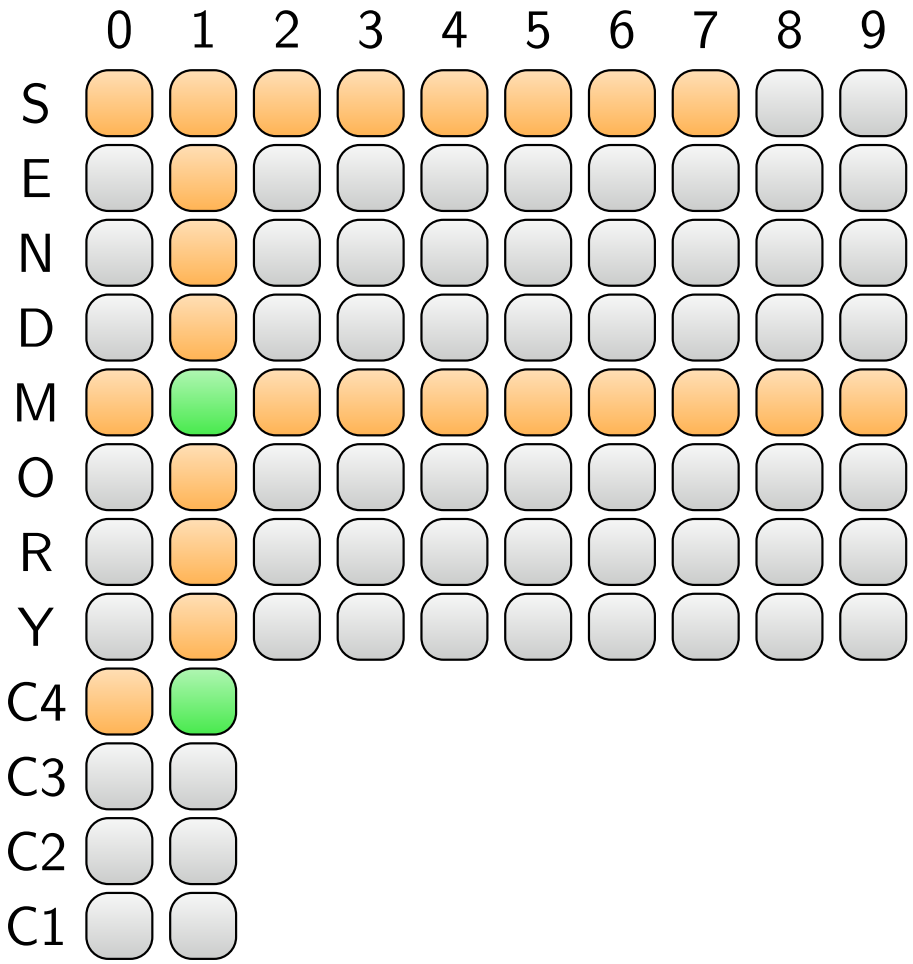


$$\overbrace{\underbrace{\texttt{carry}[3] + \texttt{value}[\texttt{S}] + 1}_{\in[10..11]} = \texttt{value}[\texttt{O}] + 10;}^{\in[10..11]}$$

$$10 \leq \texttt{value}[\texttt{O}] + 10 \leq 11$$

Propagation in Action

Institute for
Software & Systems
Engineering



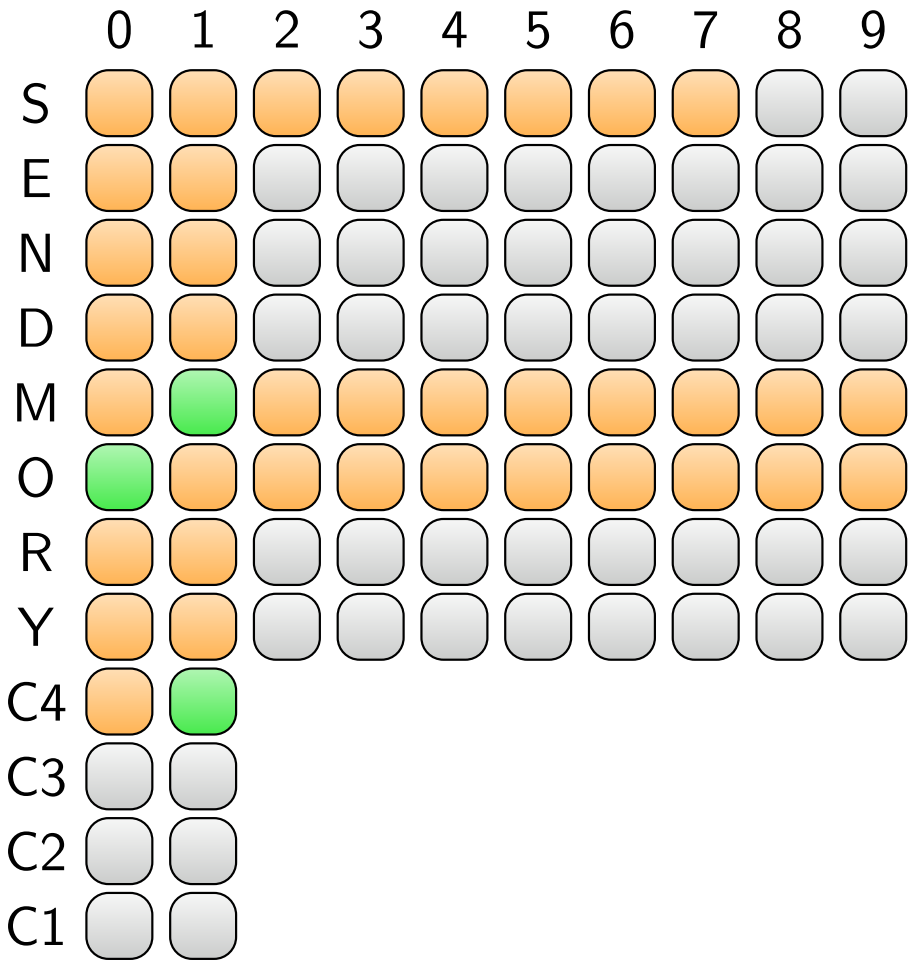
$$\overbrace{\underbrace{\texttt{carry}[3] + \texttt{value}[\texttt{S}] + 1}_{\in[10..11]} = \texttt{value}[\texttt{O}] + 10;}^{\in[10..11]}$$

$$10 \leq \texttt{value}[\texttt{O}] + 10 \leq 11$$

$$0 \leq \texttt{value}[\texttt{O}] \leq 1$$

Propagation in Action

Institute for
Software & Systems
Engineering



$$\overbrace{\underbrace{\texttt{carry}[3] + \texttt{value}[\texttt{S}] + 1}_{\in[10..11]} = \texttt{value}[\texttt{O}] + 10;}^{\in[10..11]}$$

$$10 \leq \texttt{value}[\texttt{O}] + 10 \leq 11$$

$$0 \leq \texttt{value}[\texttt{O}] \leq 1$$

Lösung

Institute for Software & Systems Engineering

Problem

```
  C4 C3 C2 C1
     S  E  N  D
+    M  O  R  E
---------------
 M  O  N  E  Y
```

Lösung

Institute for Software & Systems Engineering

Problem

```
  C4 C3 C2 C1
     S  E  N  D
+    M  O  R  E
----------
 M  O  N  E  Y
```

Output:

```
s = 9, e = 5, n = 6, d = 7
m = 1, o = 0, r = 8, y = 2
carry = [1, 1, 0, 1]
----------
```

Lösung

Institute for Software & Systems Engineering

Problem

```
  C4 C3 C2 C1
     S  E  N  D
+    M  O  R  E
---------------
  M  O  N  E  Y
```

Output:

```
s = 9, e = 5, n = 6, d = 7
m = 1, o = 0, r = 8, y = 2
carry = [1, 1, 0, 1]
----------
```

```
   1  0  1  1
      9  5  6  7
+     1  0  8  5
---------------
   1  0  6  5  2
```

Globale Constraints

Institute for
Software & Systems
Engineering

- Betrachten wir folgendes einfaches Problem
  - $X = \{x_1, x_2, x_3\}$
  - $(D_x)_{x \in X} = \{1, 2\}$
  - $C : x_1 \neq x_2,\ x_2 \neq x_3,\ x_1 \neq x_3$
- Ist dieses Problem nach Constraint-Propagation mit binären Constraints lösbar?

Globale Constraints

Institute for Software & Systems Engineering

- Betrachten wir folgendes einfaches Problem
  - $X = \{x_1, x_2, x_3\}$
  - $(D_x)_{x \in X} = \{1, 2\}$
  - $C : x_1 \neq x_2,\ x_2 \neq x_3,\ x_1 \neq x_3$
- Ist dieses Problem nach Constraint-Propagation mit binären Constraints lösbar?
- Ja, für jedes $d \in D_x$ gibt es einen Partner

Globale Constraints

Institute for
Software & Systems
Engineering

- Betrachten wir folgendes einfaches Problem
  - $X = \{x_1, x_2, x_3\}$
  - $(D_x)_{x \in X} = \{1, 2\}$
  - $C : x_1 \neq x_2,\ x_2 \neq x_3,\ x_1 \neq x_3$
- Ist dieses Problem nach Constraint-Propagation mit binären Constraints lösbar?
- Ja, für jedes $d \in D_x$ gibt es einen Partner
- Insgesamt allerdings nicht, da mindestens 3 unterschiedliche Werte nötig
- $\rightarrow$ daher globale Constraints, die eine größere Menge von Variablen im Auge betrachten können
- Und *spezialisierte* Propagationsalgorithmen haben!
- $\texttt{alldifferent}(\texttt{x}_1, \texttt{x}_2, \texttt{x}_3)$

Globale Constraints: Beispiele

Institute for Software & Systems Engineering

Häufig auftretende Constraints samt Propagierungsalgorithmen. Gesammelt im *Global Constraints Catalogue*[1]

`alldifferent(VARIABLES)` Erfordert, dass alle Variablen unterschiedliche Werte haben, z.B: $[1, 2, 3]$

[1] `http://www.emn.fr/z-info/sdemasse/gccat/sec5.html`

Globale Constraints: Beispiele

Institute for Software & Systems Engineering

Häufig auftretende Constraints samt Propagierungsalgorithmen. Gesammelt im *Global Constraints Catalogue*[1]

`alldifferent(VARIABLES)` Erfordert, dass alle Variablen unterschiedliche Werte haben, z.B: $[1, 2, 3]$

`allequal(VARIABLES)` Erfordert, dass alle Variablen den gleichen Wert haben, z.B: $[5, 5, 5]$

[1] `http://www.emn.fr/z-info/sdemasse/gccat/sec5.html`

Globale Constraints: Beispiele

Institute for Software & Systems Engineering

Häufig auftretende Constraints samt Propagierungsalgorithmen. Gesammelt im *Global Constraints Catalogue*[1]

`alldifferent(VARIABLES)` Erfordert, dass alle Variablen unterschiedliche Werte haben, z.B: $[1, 2, 3]$

`allequal(VARIABLES)` Erfordert, dass alle Variablen den gleichen Wert haben, z.B: $[5, 5, 5]$

`nvalue(VARIABLES, N)` Erfordert, dass genau N verschiedene Werte vorkommen, z.B: $([1, 2, 1], 2)$

[1] `http://www.emn.fr/z-info/sdemasse/gccat/sec5.html`

Globale Constraints: Beispiele

Institute for Software & Systems Engineering

Häufig auftretende Constraints samt Propagierungsalgorithmen. Gesammelt im *Global Constraints Catalogue*[1]

`alldifferent(VARIABLES)` Erfordert, dass alle Variablen unterschiedliche Werte haben, z.B: $[1, 2, 3]$

`allequal(VARIABLES)` Erfordert, dass alle Variablen den gleichen Wert haben, z.B: $[5, 5, 5]$

`nvalue(VARIABLES, N)` Erfordert, dass genau N verschiedene Werte vorkommen, z.B: $([1, 2, 1], 2)$

`at_most(N, VARIABLES, V)` Erfordert, dass höchstens N Variablen den Wert V annehmen z.B: $(2, [2, 2, 4], 2)$

[1] `http://www.emn.fr/z-info/sdemasse/gccat/sec5.html`

Diskrete Optimierungsprobleme

Institute for Software & Systems Engineering

Reine Erfüllbarkeitsprobleme reichen mitunter nicht aus

- Lösungen können nach Zielfunktion *geordnet* sein
- Alle Constraints sind nicht gleichzeitig erfüllbar – es existiert keine Lösung

Daher Erweiterungen des klassischen Constraint-Frameworks

- Zielfunktion $f : [X \to D] \to \mathbb{R}$
- Macht aus Constraint-Satisfaction-Problem (X, D, C) ein Constraint-Satisfaction-Optimization-Problem (CSOP)

Ziel: Bestimme diejenige Lösung θ, die f maximiert (bzw. minimiert)

Branch and Bound

Institute for
Software & Systems
Engineering

- Sehr weit anwendbare Lösungstechnik
- Systematische Suche mit Zielfunktion f
  - "Branching" $\rightarrow$ Auffächern von möglichen Lösungen
  - Best-first, Depth-first
- Zusätzlich "Bounding" mit *best-case*-Abschätzungen
  - Wenn "best-case" eines Knotens im Suchbaum ($\hat{f}(\hat{\theta})$ bereits schlechter als beste bisher gefundene Lösung, schneide Teilbaum ab

CSOP: Beispiel Knapsack

Institute for Software & Systems Engineering

Betrachten wir das eindimensionale Knapsack-Problem mit Gegenständen $i \in I$ samt Gewichten $(w_i)_{i \in I}$ und Werten $(v_i)_{i \in I}$, die in den Rucksack mit Kapazität K aufgenommen werden.

Knapsack

```
int: n; set of int: ITEMS = 1..n;
int: K; % capacity
array[ITEMS] of int: v;
array[ITEMS] of int: w;

array[ITEMS] of var bool: x;

constraint sum(i in ITEMS) (x[i] * w[i]) <= K;
solve maximize sum(i in ITEMS) (x[i] * v[i]);
```

Geeignete Best-Case-Abschätzung an Knoten?

CSOP: Beispiel Knapsack

Institute for Software & Systems Engineering

Betrachten wir das eindimensionale Knapsack-Problem mit Gegenständen $i \in I$ samt Gewichten $(w_i)_{i \in I}$ und Werten $(v_i)_{i \in I}$, die in den Rucksack mit Kapazität K aufgenommen werden.

Knapsack

```
int: n; set of int: ITEMS = 1..n;
int: K; % capacity
array[ITEMS] of int: v;
array[ITEMS] of int: w;

array[ITEMS] of var bool: x;

constraint sum(i in ITEMS) (x[i] * w[i]) <= K;
solve maximize sum(i in ITEMS) (x[i] * v[i]);
```

Geeignete Best-Case-Abschätzung an Knoten? Nehme an, alle verbleibenden Gegenstände passen in den Rucksack

MiniZinc

MiniZinc-IDE zum Download unter `http://www.minizinc.org/` (neueste Version)

Alle Beispiele aus dieser Vorlesung im Digicampus!

Zusammenfassung

Institute for Software & Systems Engineering

- Constraint Programming bietet generische Algorithmen für Erfüllbarkeitsprobleme
- Effizienz hängt von Propagationsstärke der verwendeten Constraints ab
- Heuristiken beschleunigen die Suche
- Erweiterung auf Optimierungsprobleme mit klassischen Techniken
  - Branch and Bound

Quellen I

Rossi, F., Beek, P. v., and Walsh, T. (2006).
Handbook of Constraint Programming (Foundations of Artificial Intelligence).
Elsevier Science Inc., New York, NY, USA.

Russell, S. and Norvig, P. (2010).
Artificial Intelligence: A Modern Approach.
Prentice Hall series in artificial intelligence. Prentice Hall.

Tsang, E. (1993).
Foundations of constraint satisfaction, volume 289.
Academic press London.