Intelligente Systeme
- Constraints -

Hochschule für Angewandte Wissenschaften Hamburg
Department Informatik

Dr.-Ing. Sabine Schumann

Roadmap

1. Einleitung
2. Logik: Aussagenlogik, Prädikatenlogik erster Stufe
3. Prolog

(4. DCG)

5. Problemlösen
6. Constraints
7. Soft Computing
8. Neuronale Netze
9. Semantische Netze & Frames

Einleitung

Eliminate all other factors,
and the one which remains,
must be the truth.

Sir Arthur Conan Doyle

Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming:
the user states the problem, the computer solves it.

Eugene C. Freuder, 1997

Einleitung (1)

Constraints

- sind „Zwangsbedingungen“,
- in vielen Programmiersprachen zu finden,
- Wert einer Variable muss der/den Bedingungen genügen, um in das System übernommen zu werden,
- Kombination von logischer und Constraint-Programmierung führt meist zu erheblicher Steigerung der Ausdrucksstärke, Flexibilität und Effektivität,
- sind ein Wissensrepräsentationsformat
    - für das es spezielle Suchverfahren gibt,
    - das zur Vorbehandlung von Suchbereichen für allgemeine Suchverfahren verwendet werden kann.

Einleitung (2)

Constraint Typen:

Extensionale Beschreibung

- Tupel-Constraints,
- DB-Tabellen

Intensionale Beschreibung

- Funktionen,
- Prädikate

Einleitung (3)

Beispiele

DBS:

Zeichenkette max. 7 Zeichen

Pflichtfeld

nur die Werte Dozent, Labor, Seminar erlaubt

```
CREATE TABLE raum (
funktion VARCHAR(7) NOT NULL
      CHECK(funktion IN ('Dozent', 'Labor', 'Seminar)),
plaetze  NUMBER(3)  CHECK (plaetze>0));
```

Werte zwischen 0 und 999 erlaubt

Blocksworld:

block1 ist oberhalb von block2.
block2 ist oberhalb von block3.
block3 ist oberhalb von block1.

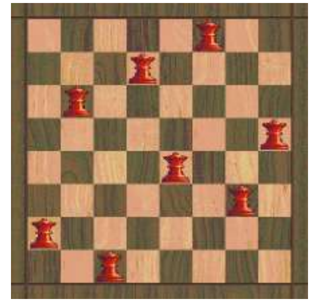
Diese Constraintmenge ist nicht erfüllbar.

Einleitung (3)

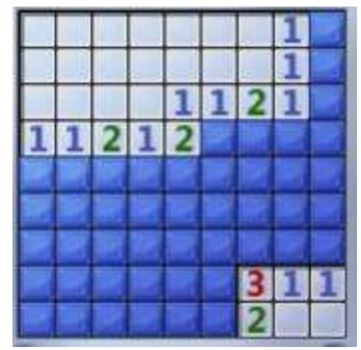
Anwendung:

Spiele und Denksport:

- Vierfarbenproblem
- 4 bzw. 8-Dameproblem
- Sudoku
- …



```
   SEND
+  MORE
-------
  MONEY
```



Konfiguration technischer Systeme

- Fahrerlose Transportsysteme

Planungsaufgaben

- Produktionsplanung
- Stundenplan-Erstellung

Kombinatorische Optimierung

- Transport-Optimierung

Terminologie (1)

Constraint Variable (CV):

Eine Variable, deren Wert aus einem (endlichen oder unendlichen) Wertebereich (WB, domain) stammt.

Instantiierte Variable:

Eine Constraint Variable, der ein Wert zugewiesen wurde, sonst uninstantiiert.

Constraint:

- legt (deklarativ!) Bedingungen fest,
- ein logischer Ausdruck, der auf eine oder mehrere instantiierte Constraint Variablen angewendet werden kann.

Constraintnetz (CN):

Ein Graph, der aus Constraint Variablen und Constraints besteht.

Terminologie (2)

Variablen: {X,Y,Z}

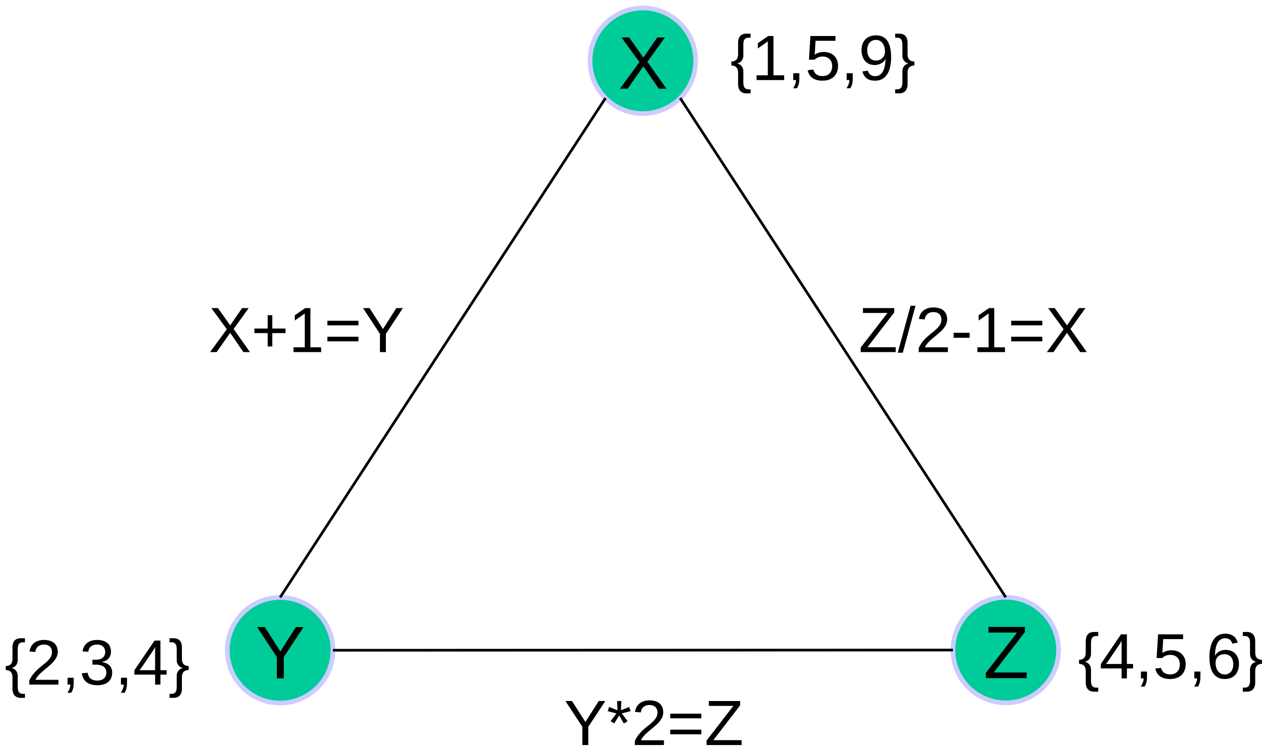
Wertebereiche:

- X $\leftarrow$ {1,5,9}
- Y $\leftarrow$ {2,3,4}
- Z $\leftarrow$ {4,5,6}

Constraints: {X+1=Y, Y*2=Z, Z/2-1=X}

Lösung: {X=1, Y=2, Z=4}

Constraintnetz



Terminologie (3)

Belegung:

Eine Zuordnung von CV in einem Constraint Netz und Werten, die aus dem Wertebereich der jeweiligen CV stammen.

Vollständig:

Eine Belegung, die *jeder* Constraint Variablen des Constraint Netzes einen entsprechenden Wert zuweist.

Konsistent:

Eine Belegung, die alle Constraints des Netzes erfüllt.

Constraint-Satisfaction-Problem (CSP):

Finden einer (aller) konsistenten vollständigen Belegung(en) eines CN durch **Propagation**, bzw. Meldung, dass es soetwas nicht gibt.

Constraints: Ziele

- Darstellung und Verarbeitung komplexer Abhängigkeiten,
- Konsistenzüberprüfung für Wertebelegungen,
- Einschränkungen von Werten durch Propagation,
- Bestimmen aller Lösungen durch Constraint-Satisfaction-Techniken.

Beispiel: SEND+MORE=MONEY (1)

```
  SEND
+ MORE
------
 MONEY
```

Wie löst man dieses Problem in Standard Prolog?

		1000*S	+100*E	+10*N	+D
+		1000*M	+100*O	+10*R	+E
=	10000*M	+1000*O	+100*N	+10*E	+Y

Was ist am Prolog Programm nachteilig?

Wie löst der Mensch dieses Problem?

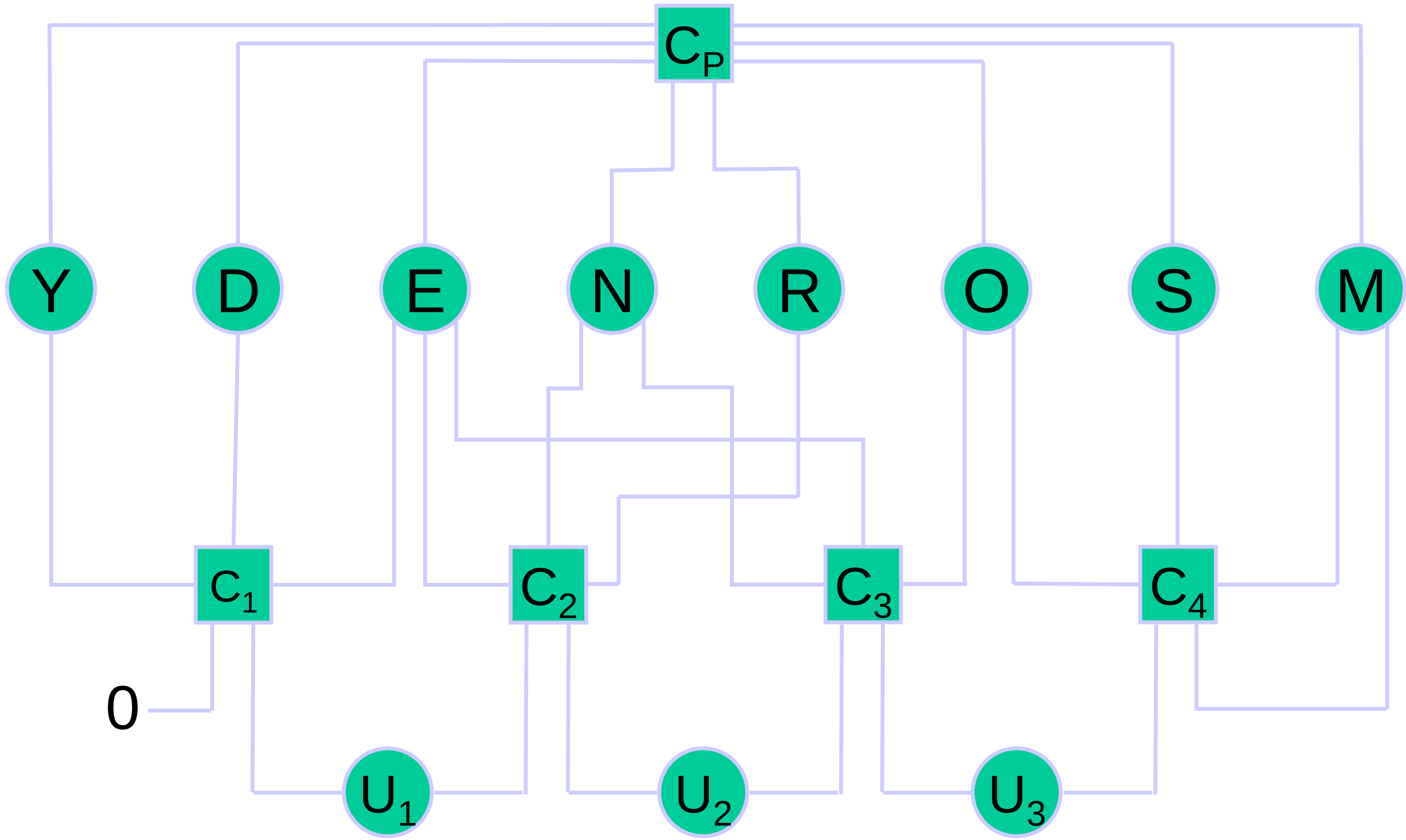
Beispiel: SEND+MORE=MONEY (2)

```
  SEND
+ MORE
------
 MONEY
```

- Belegung aller Variablen mit verschiedenen Werten finden, so dass die Gleichung aufgeht.
- Formulierung als Constraintnetz unter Verwendung von Überträgen (zusätzliche Variablen).
- Wertebereiche der Variablen:
  D,E,N,O,R,Y $\in \{0,1,2,3,4,5,6,7,8,9\}$
  S $\in \{1,2,3,4,5,6,7,8,9\}$
  M $= 1$
  U_1,U_2,U_3 $\in \{0,1\}$
- Constraint als Tripel:
  (Name, Variablenmenge, Relation)
  $(C_1, \{D,E,Y, U_1\}, D+E=Y+10*U_1)$

Beispiel: SEND+MORE=MONEY (3)

Constraintnetz, bestehend aus 4 Constraints



Constraint Logic Programming (CLP)

Verbindung aus zwei deklarativen Paradigmen:

- Constraint Solving,
- Logic Programming

Vorteile, ein Constraint System innerhalb einer Programmier-sprache zur Verfügung zu stellen:

- Volle Flexibilität einer allgemeinen Programmiersprache ,
- Integration nicht-constraint-spezifischer Algorithmen,
- z. Bsp. Ein-/Ausgabe-Funktionen

Zwei Möglichkeiten der Integration

1. Bibliothek
   - Variablen, Relationen, etc. sind Bestandteile der Bibliothek
   - Unterscheiden sich von Variablen, Methoden der Basis-Sprache
2. Erweiterung der Ausdrucksmöglichkeiten der Sprache

CLP in PROLOG

Für PROLOG bietet sich der 2. Weg an!

- Große Nähe zur „Constraint-Sicht“

Notwendige Erweiterungen

- Constraint Solver,
- Ausdrucksmöglichkeiten für Constraints,
- Kommunikationsschnittstelle mit Solver
  - Übergabe von Constraints
  - Abfrage von Werten

SWI-PROLOG-Erweiterung (1)

Laden von Modul *modul* mittels

```
:- use_module(library(modul)).
```

Module beschrieben im SWI Online Manual.

swi-prolog.org/pldoc/refman/

CLP-Erweiterungsmodul für endliche Wertebereiche (FiniteDomains)

```
:- use_module(library(clpfd)).
```

Syntax Constraint Logic Programming: SWI Online Manual

A.7 library(clpfd): Constraint Logic Programming over Finite Domains

SWI-PROLOG-Erweiterung (2)

Syntax Auszug:

Var in Range (Range: L..U)

Vars ins Range (Range: L..U)

- z.Bsp.: `Ziffer in 0..9`
- z.Bsp.: `[Z1,Z2,Z3] ins 0..9`

Expr #> Expr

- #<, #=, #\=, #=<, #>=, #\/, … analog
- z.Bsp.: `A #<B+C`
- `A #\/ B` (A oder B)

all_different(Vars)

- z.Bsp.: `all_different([Z1, Z2, Z3])`

label(Vars)

- z. Bsp.: `label(Ziffer)`
  `label([Z1, Z2, Z3])`

…

Beispiel: SEND+MORE=MONEY (4)

es wird **die** Lösung gefunden

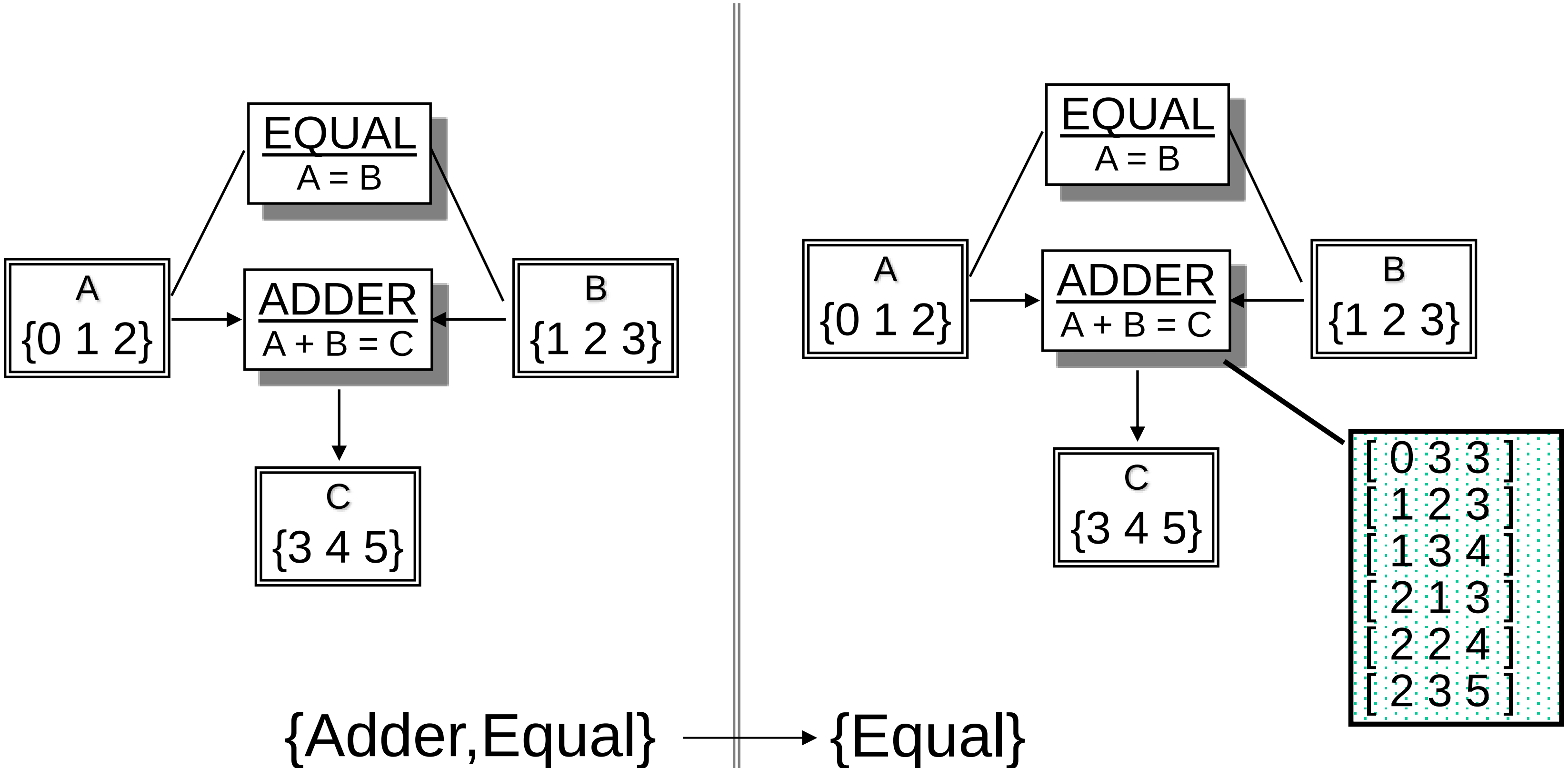
SWI Prolog Version 6.2.6

	smm.pl	smm_ sortiert.pl	smm_ permutation.pl	smm_ constraints.pl	smm_ clp.pl
WIE	konventionell, generate & test	konventionell, besser sortiert	konv. mit Permutation	mit Constraints	mit CLP
Finden 1 Lösung nach ca.	49 s	3,5 s	6 s	4,5 s	0,000 s
Inferenzen ca.	287.622.549	12.857.017	15.579.410	17.595.371	13.610
Finden „aller“ Lö-sungen nach ca.	3 min	13 s	14 s	7,5 s	0,02 s
Inferenzen ca.	1.025.967.580	45.891.198	36.850.046	31.939.604	17.189

Die Lösung:
9567+1085=10652

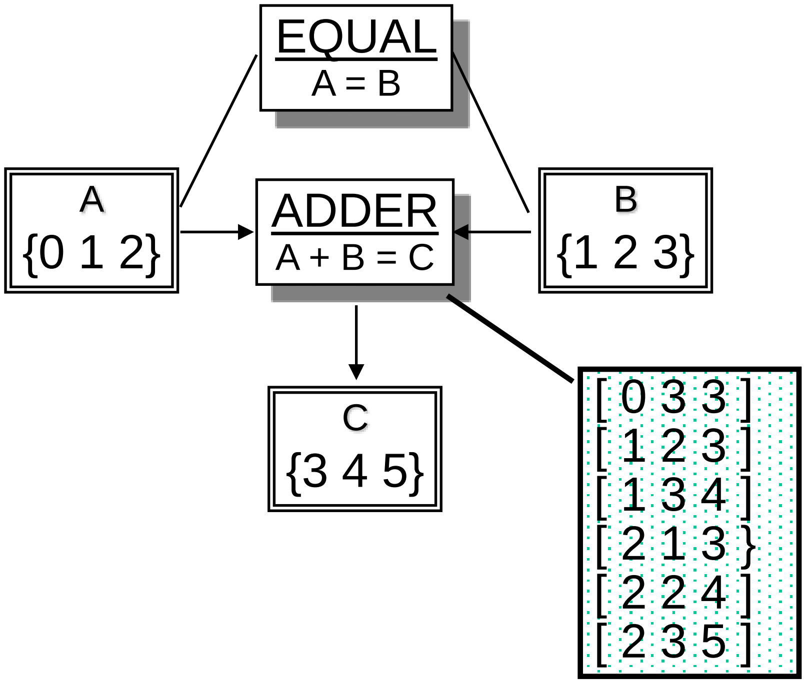
Propagation: Lokale Konsistenz (1)

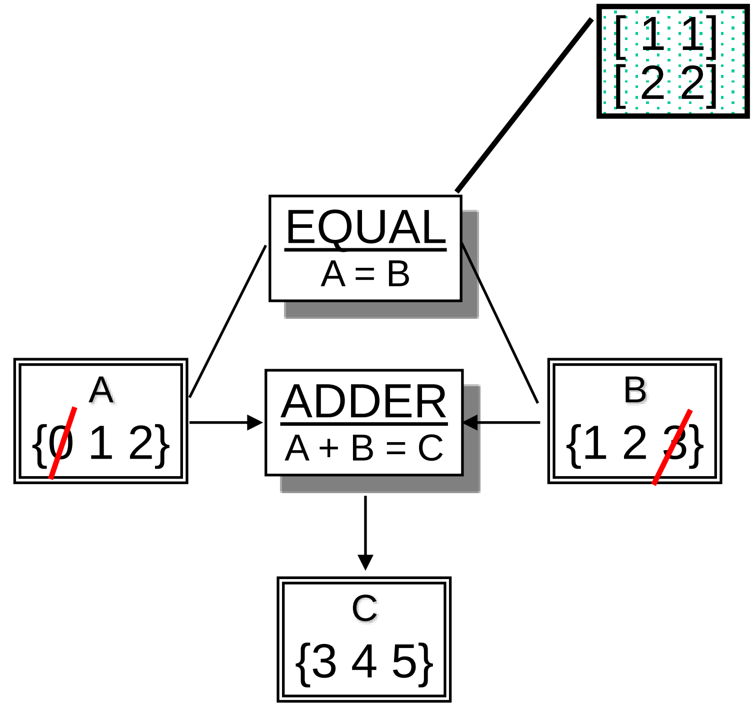
(aus *Lothar Hotz (Hitec): Constraints und ihre Verwendung (Vorlesungsfolien)*)



Constraints werden unabhängig voneinander ausgewertet.

Propagation: Lokale Konsistenz (2)

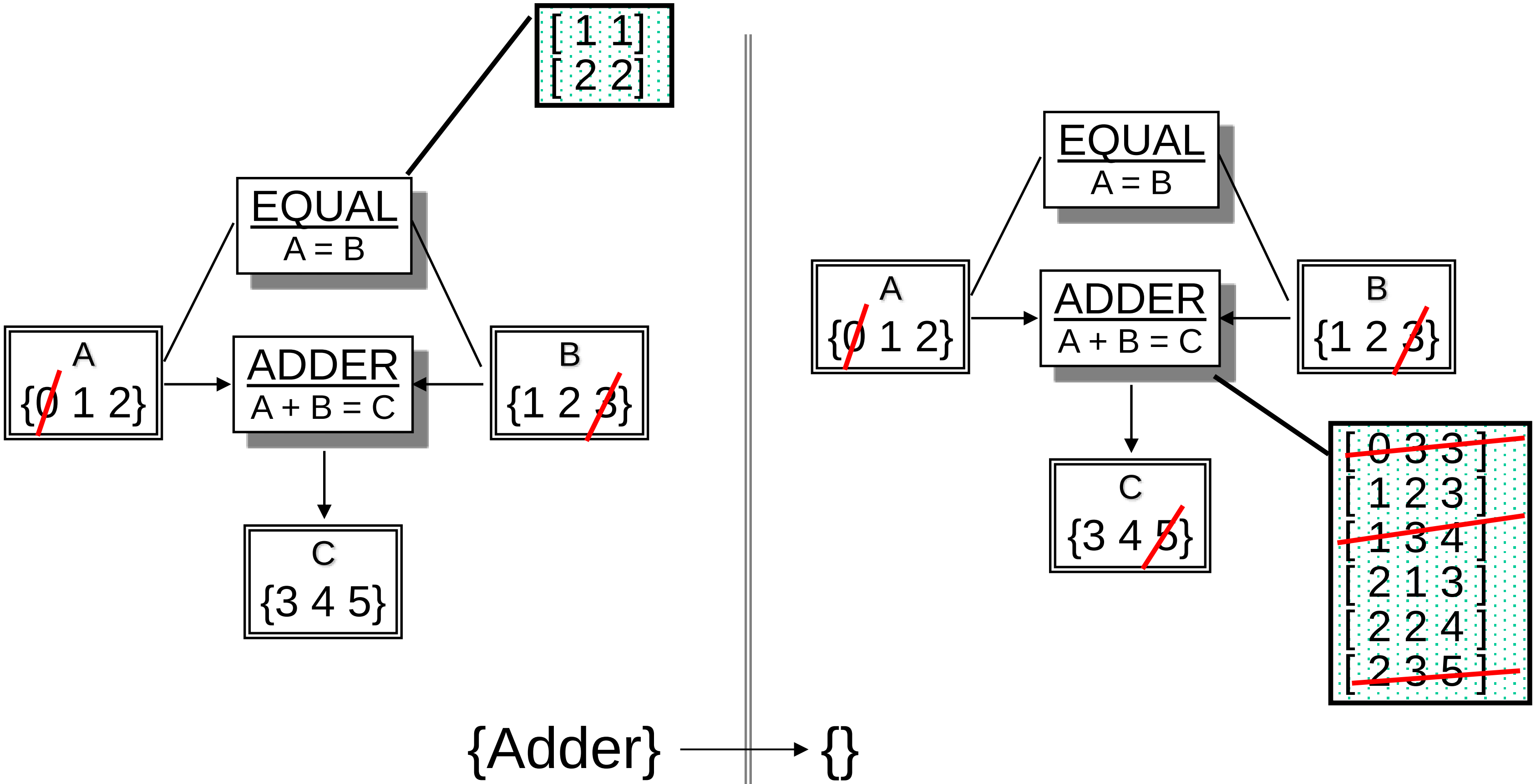




{Equal} → {} → <u>{Adder}</u>

Constraints werden unabhängig voneinander ausgewertet.

Propagation: Lokale Konsistenz (3)



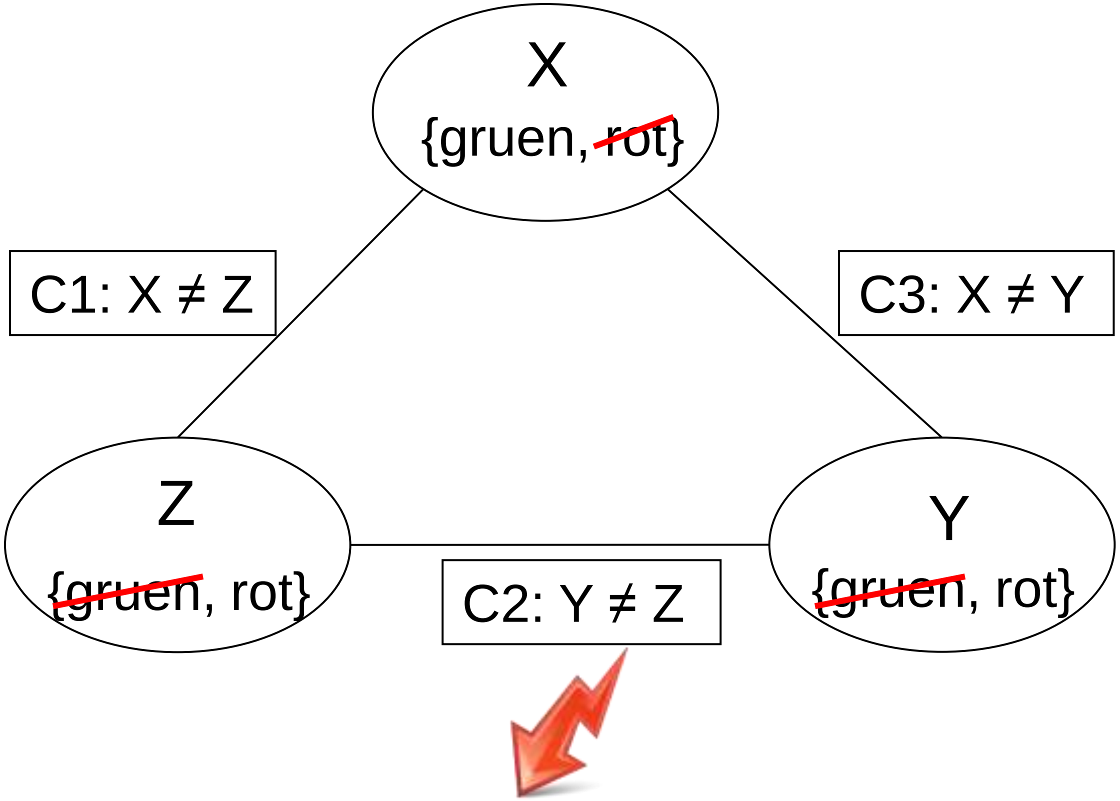
Constraints werden unabhängig voneinander ausgewertet.

Propagation: Lokale Konsistenz (4)

- Idee:
  Lokale Entscheidungen beeinflussen globale Ergebnisse.

- Betrachte nur ein Constraint und seine Variablen zur Zeit.

Grenzen der lokalen Propagation (1)



Das Netz ist lokal konsistent,
hat aber keine Lösung!

K-Konsistenz von CSP (1)

Ein Constraint-Netz mit n Variablen heißt **k-konsistent** gdw. für jede Variablendomäne aus V_{j1}, V_{j2}, ..., V_{jk} und für jede Wertewahl aus den ersten (k-1) Domänen ein Wert aus V_{jk} existiert, so dass das entstehende k-Tupel kein Constraint verletzt, d.h. konsistent ist.

Strenge k-Konsistenz :
wenn zusätzlich für alle 0<i<k gilt, i-konsistent

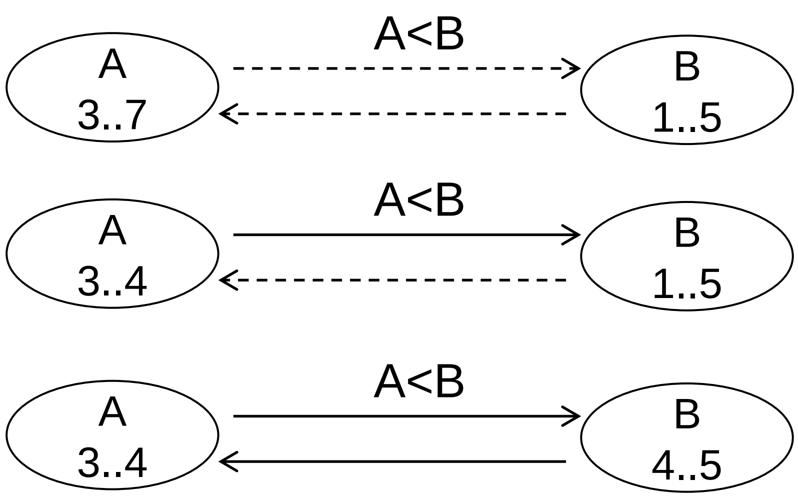
k=1 strenge Konsistenz :
Knoten Konsistenz (*node consistency, NC*)

k=2 strenge Konsistenz :
Kanten Konsistenz (*arc consistency, AC*)

k=n strenge Konsistenz :
<u>Globale Konsistenz</u>

Kanten-Konsistenz von CSP (2)

- Kante(i,j) ist kantenkonsistent gdw. es für jeden Wert in V_i einen kompatiblen Wert in V_j gibt. (gerichtet!)
- Ein CSP ist kantenkonsistent gdw. es für alle Kanten kantenkonsistent ist (in beiden Richtungen).



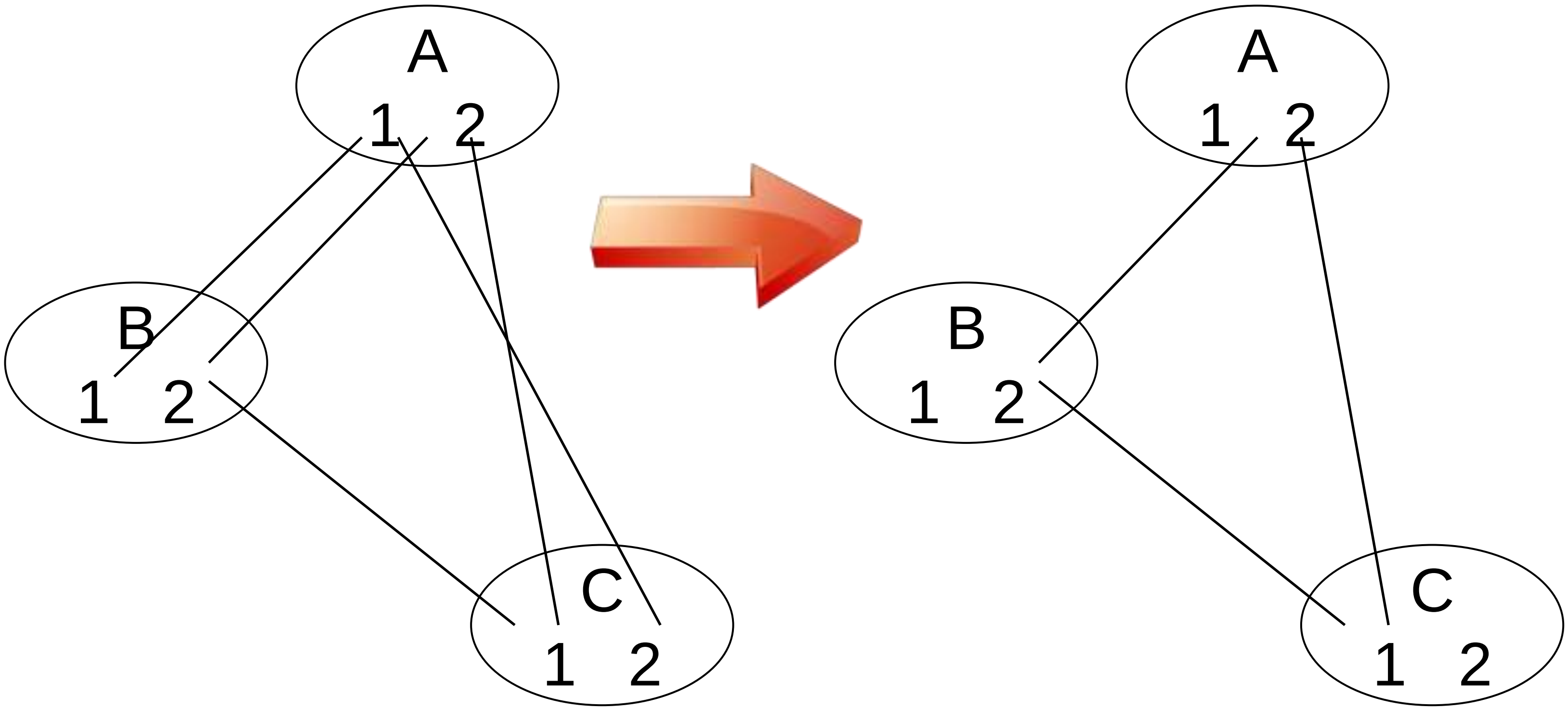
nicht kantenkonsistent

(A,B) konsistent

kantenkonsistent
(A,B) und (B,A) konsistent

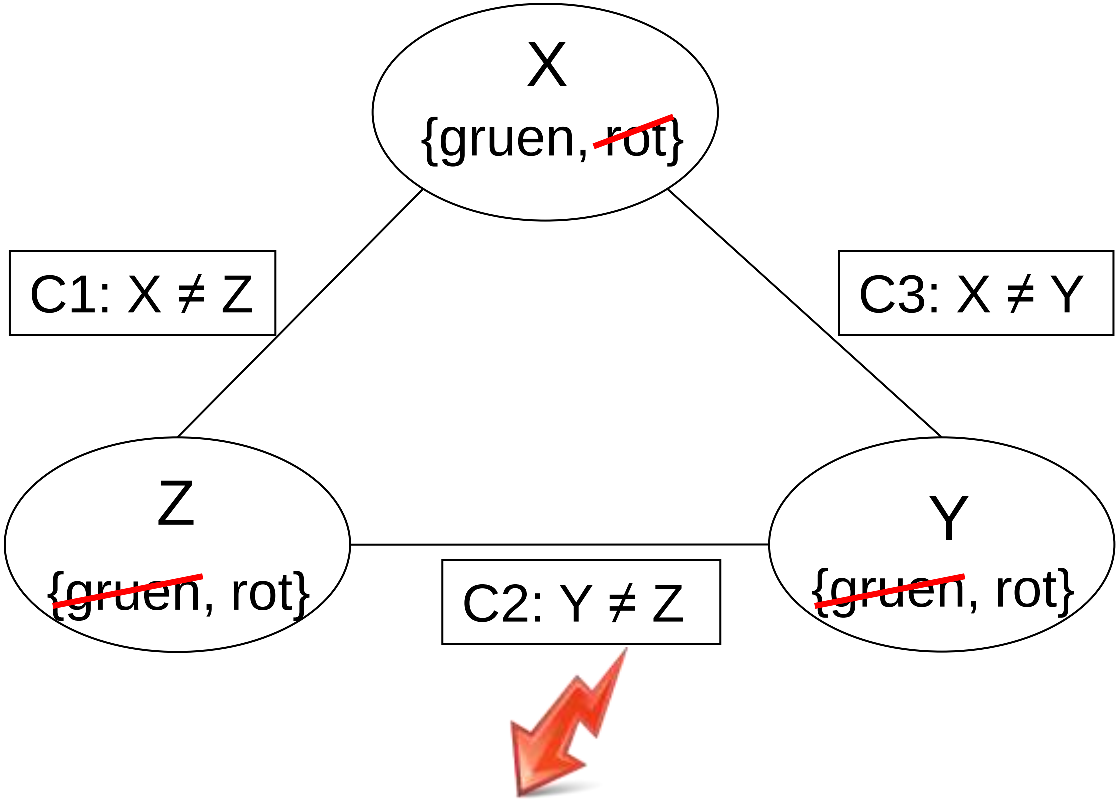
Kanten-Konsistenz von CSP (3)

Constraints: $A = B$, $A \neq C$, $B > C$



AC ist die am meisten verbreitete Konsistenztechnik

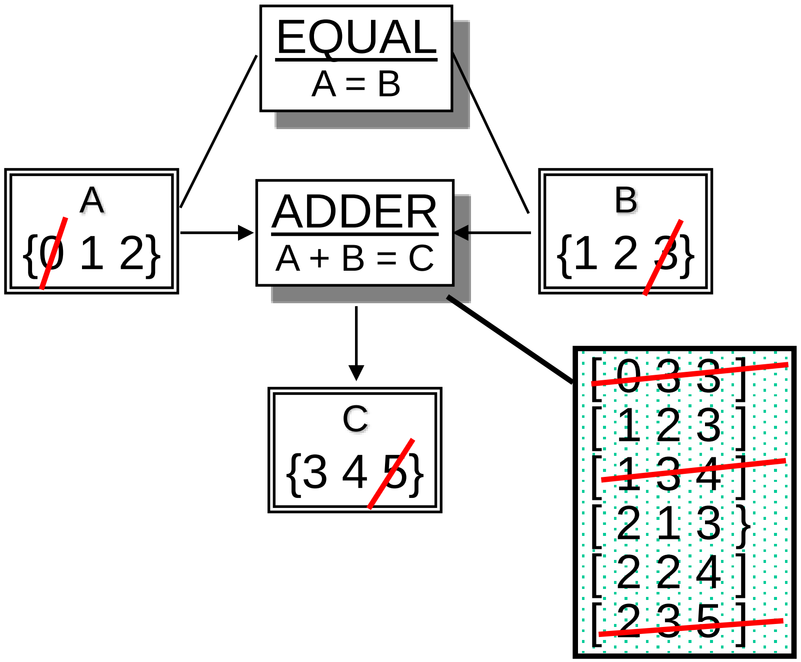
Grenzen der lokalen Propagation (2)



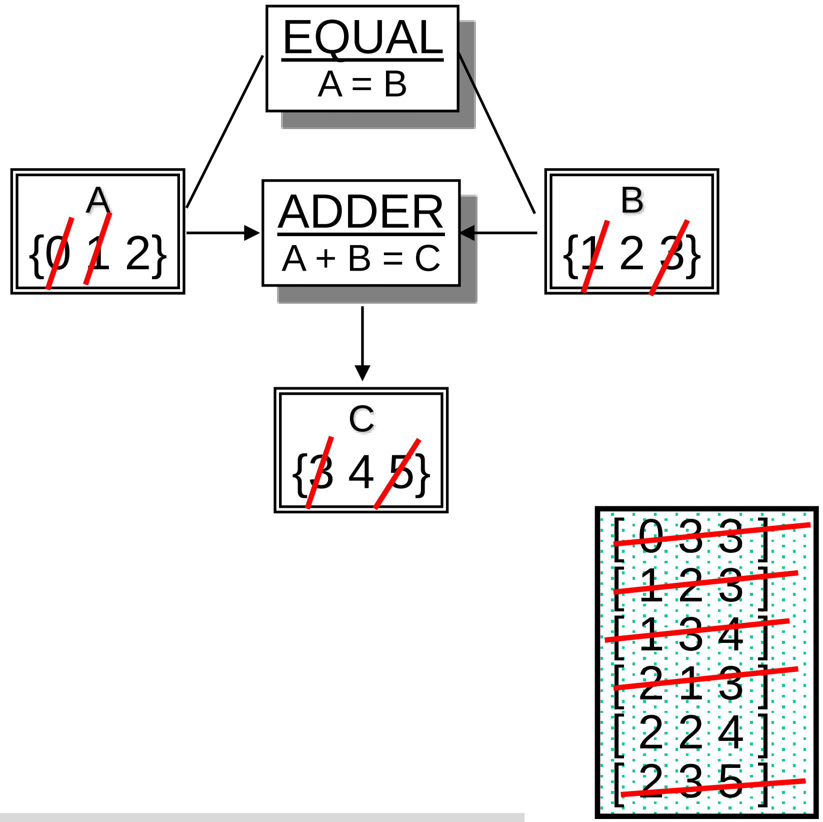
Das Netz ist lokal konsistent,
hat aber keine Lösung!

1-konsistent
2-konsistent
aber nicht 3-konsistent!

Propagation: Globale Konsistenz



Abhängige Berechnung



Globale Lösung des Netzes:
Ein Wert für alle Variablen pro Lösung.
Mehrere Lösungen möglich.

Sudoku (1)

6			7			5		
	2	8						
			6	4		3		
7	4						2	
		1				8		
	5						3	7
		3		7	6			
						1	9	
		4			5			8

Trage Ziffern von 1 bis 9 ein.

Pro Zeile unterschiedliche Ziffern.

Pro Spalte unterschiedliche Ziffern.

Pro Quadrat unterschiedliche Ziffern.

Sudoku (2)

6			7			5		
	2	8						
			6	4		3		
7	4						2	
		1				8		
	5						3	7
		3		7	6			
						1	9	
		4			5			8

Trage Ziffern von 1 bis 9 ein.

Pro Zeile unterschiedliche Ziffern.

Pro Spalte unterschiedliche Ziffern.

Pro Quadrat unterschiedliche Ziffern.

Sudoku (3)

6			7			5		
	2	8						
			6	4		3		
7	4						2	
		1				8		
	5						3	7
		3		7	6			
						1	9	
		4			5			8

Trage Ziffern von 1 bis 9 ein.

Pro Zeile unterschiedliche Ziffern.

Pro Spalte unterschiedliche Ziffern.

Pro Quadrat unterschiedliche Ziffern.

Sudoku (4)

6			7			5		
	2	8						
			6	4		3		
7	4						2	
		1				8		
	5						3	7
		3		7	6			
						1	9	
		4			5			8

Trage Ziffern von 1 bis 9 ein.

Pro Zeile unterschiedliche Ziffern.

Pro Spalte unterschiedliche Ziffern.

Pro Quadrat unterschiedliche Ziffern.

Sudoku (5)

Variablen und Domänen:

- Jedes Feld (F_{xy}) eine Variable (R=Row)

```
R1 = [F11,F12,F13,F14,F15,F16,F17,F18,F19],
R2 = [F21,F22,F23,F24,F25,F26,F27,F28,F29],
% …
R1 ins 1..9,
R2 ins 1..9,
% …
```

- Initialbelegung

```
board([R1,R2,R3,R4,R5,R6,R7,R8,R9]),
% …
board([[6,_,_,7,_,_,5,_,_],
       [_,2,8,_,_,_,_,_,_],
       [_,_,_,6,4,_,3,_,_],
       [7,4,_,_,_,_,_,2,_],
       [_,_,1,_,_,_,8,_,_],
       [_,5,_,_,_,_,_,3,7],
       [_,_,3,_,7,6,_,_,_],
       [_,_,_,_,_,_,1,9,_],
       [_,_,4,_,_,5,_,_,8]]).
```

Sudoku (6)

oder eleganter:
Variablen und Domänen:

- Initialbelegung

```
board([[6,_,_,7,_,_,5,_,_],
       [_,2,8,_,_,_,_,_,_],
       [_,_,_,6,4,_,3,_,_],
       [7,4,_,_,_,_,_,2,_],
       [_,_,1,_,_,_,8,_,_],
       [_,5,_,_,_,_,_,3,7],
       [_,_,3,_,7,6,_,_,_],
       [_,_,_,_,_,_,1,9,_],
       [_,_,4,_,_,5,_,_,8]]).
```

- Spielfeld abgebildet auf die Zeilen (Row 1-9) und Domänen

```
board([R1,R2,R3,R4,R5,R6,R7,R8,R9]),
R1 ins 1..9,
R2 ins 1..9,
% …
```

Sudoku (7)

Constraints:

- Ungleichheit von Feldern einer Zeile

```
all_different(R1),
all_different(R2),
all_different(R3),
% …
```

eleganter:

```
maplist(all_different,[R1,R2,R3,R4,R5,R6,R7,R8,R9]),
```

- Ungleichheit von Feldern einer Spalte

```
all_different([F11,F21,F31,F41,F51,F61,F71,F81,F91]),
all_different([F12,F22,F32,F42,F52,F62,F72,F82,F92]),
all_different([F13,F23,F33,F43,F53,F63,F73,F83,F93]),
% …
```

eleganter:

```
% Matrix um 90 Grad drehen, C für Column
  transpose([R1,R2,R3,R4,R5,R6,R7,R8,R9], [C1,C2,C3,C4,C5,C6,C7,C8,C9]),
% Felder in allen Spalten verschieden
  maplist(all_different,[C1,C2,C3,C4,C5,C6,C7,C8,C9]),
```

Sudoku (8)

Constraints (Fortsetzung):

- Ungleichheit von Feldern eines Quadrates

```
all_different([F11,F12,F13,F21,F22,F23,F31,F32,F33]),
all_different([F14,F15,F16,F24,F25,F26,F34,F35,F36]),
all_different([F17,F18,F19,F27,F28,F29,F37,F38,F39]),
% …
```

eleganter

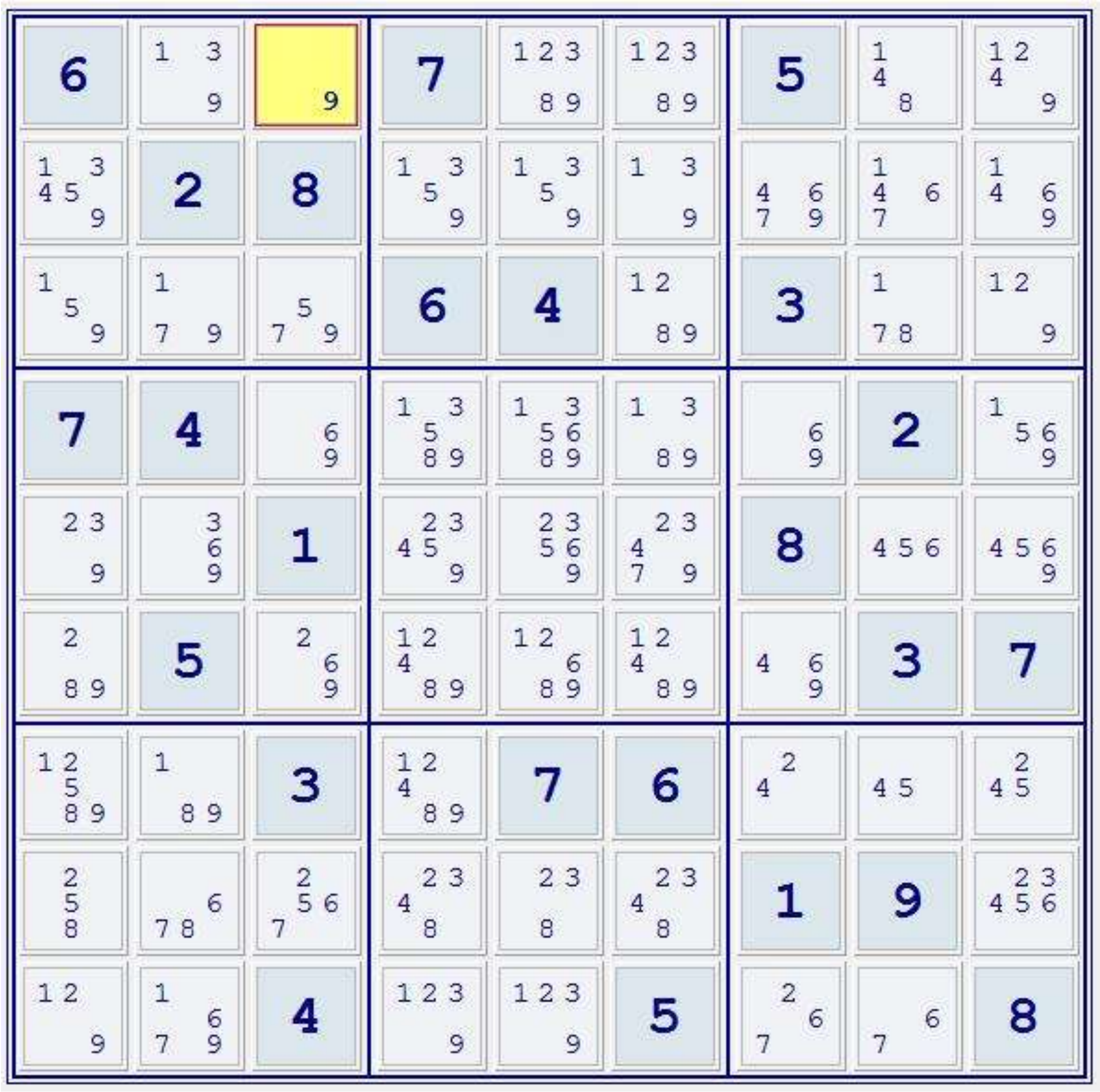
```
% Felder in allen Quadranten verschieden
  squares(R1,R2,R3),
  squares(R4,R5,R6),
  squares(R7,R8,R9),

squares([], [], []).
squares([A,B,C|RestR1], [D,E,F|RestR2], [G,H,I|RestR3]):-
  all_different([A,B,C,D,E,F,G,H,I]),
  squares(RestR1, RestR2, RestR3).
```

Sudoku

6			7			5		
	2	8						
			6	4		3		
7	4						2	
		1				8		
	5						3	7
		3		7	6			
						1	9	
		4			5			8

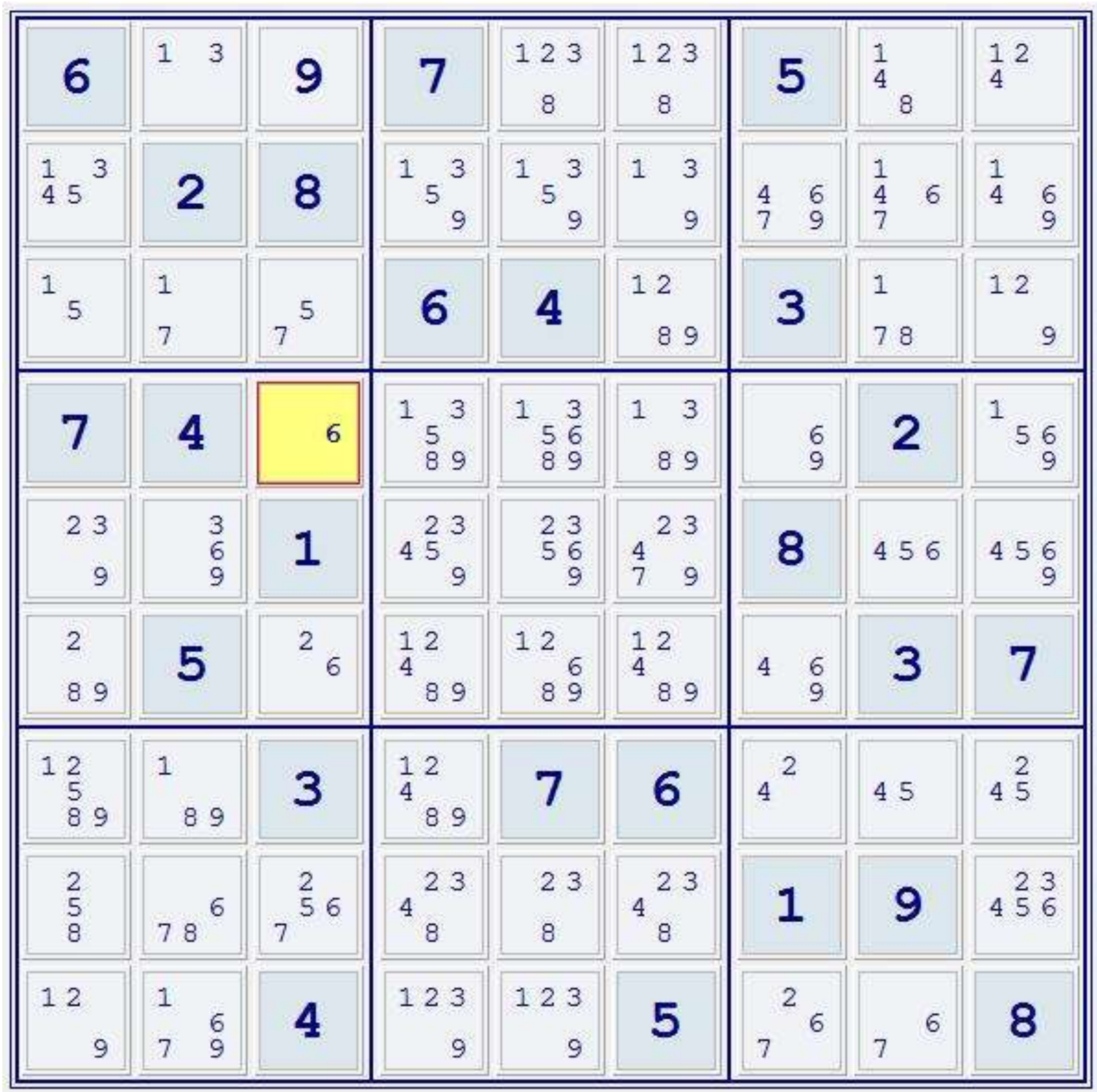
Sudoku – Lokale Lösung (1)



Sudoku – Lokale Lösung (2)

6	1 3 ~~9~~	9	7	1 2 3 8 ~~9~~	1 2 3 8 ~~9~~	5	1 4 8	1 2 4 ~~9~~
1 3 4 5 ~~9~~	2	8	1 3 5 9	1 3 5 9	1 3 9	4 6 7 9	1 4 6 7	1 4 6 9
1 5 ~~9~~	1 7 ~~9~~	5 7 ~~9~~	6	4	1 2 8 9	3	1 7 8	1 2 9
7	4	6 ~~9~~	1 3 5 8 9	1 3 5 6 8 9	1 3 8 9	6 9	2	1 5 6 9
2 3 9	3 6 9	1	2 3 4 5 9	2 3 5 6 9	2 3 4 7 9	8	4 5 6	4 5 6 9
2 8 9	5	2 6 ~~9~~	1 2 4 8 9	1 2 6 8 9	1 2 4 8 9	4 6 9	3	7
1 2 5 8 9	1 8 9	3	1 2 4 8 9	7	6	2 4	4 5	2 4 5
2 5 8	6 7 8	2 5 6 7	2 3 4 8	2 3 8	2 3 4 8	1	9	2 3 4 5 6
1 2 9	1 6 7 9	4	1 2 3 9	1 2 3 9	5	2 6 7	6 7	8

Sudoku – Lokale Lösung (3)



Sudoku – Globale Lösung

6	3	9	7	1	8	5	4	2
4	2	8	5	3	9	7	1	6
1	7	5	6	4	2	3	8	9
7	4	6	8	5	3	9	2	1
3	9	1	4	2	7	8	6	5
8	5	2	9	6	1	4	3	7
9	8	3	1	7	6	2	5	4
5	6	7	2	8	4	1	9	3
2	1	4	3	9	5	6	7	8

Dynamische CSP (1)

Dynamische CSP

- werden benötigt, wenn das Constraintnetz nicht konstant bleibt, d.h. Variablen bzw. Constraints hinzukommen, verändert und/oder gelöscht werden.
- Constraintnetz muss ab der Änderung neu berechnet werden.
- werden betrachtet als eine Folge von statischen CSPs und jedes ist eine Transformation des Vorhergehenden, wobei Variablen bzw. Constraints hinzugefügt, verändert und/oder entfernt werden können.
- Informationen der Vorgänger-CSP werden genutzt, um das aktuelle CSP (schneller) zu lösen. Dazu gibt es verschiedene Techniken:

Dynamische CSP (2)

Techniken, um Informationen der Vorgänger CSP zu nutzen:

- Oracles:
  Constraintnetz wird neu berechnet,
  vorhergehende Lösungen dienen als Heuristik.

- Local repair:
  Constraintnetz wird ausgehend von der Teillösung des Vorgänger-CSP weiter berechnet, wobei die Änderungen durch lokale Suche repariert werden.

- Constraint recording:
  Im Laufe einer CSP Lösung gewonnene Erkenntnisse über Inkonsistenzen mancher Variablenbelegungen werden als neue Constraints aufgezeichnet und dem Constraintnetz hinzugefügt. Diese Erkenntnisse stehen nachfolgenden CSPs zur Verfügung und werden weitergegeben.

Flexible CSP - Constraint Relaxierung (1)

Auch bei rein statischen CSPs lassen sich diverse Probleme jedoch nicht vollständig lösen, da unter Einbeziehung aller Constraints keine Lösung gefunden werden kann.

Ansatz: Flexible CSP:

- Constraints werden „aufgeweicht" (relaxiert).
- Lösung muss nicht alle Constraints erfüllen.

Beispiele:

Stundenplanerstellung,
Konfigurationsprobleme,
Computer Vision,

…

Flexible CSP - Constraint Relaxierung (2)

Idee:

Constraints klassifizieren in unbedingt notwendige und nicht unbedingt notwendige (aber erstrebenswerte) Bedingungen.

- harte Constraints:
  müssen erfüllt sein, um eine Lösung zu generieren.
- weiche Constraints (soft Constraints):
  - sind in eine Prioritätenhierarchie eingeteilt
  - Ziel:
    möglichst viele weiche Constraints (unter Berücksichtigung ihrer Position in der Prioritätenhierarchie) erfüllen.

Flexible CSP (3)

- erstrebenswerte Nebenbedingungen können sich teilweise gegenseitig ausschliessen,
- keine Garantie einer Lösung, in der alle erstrebenswerten Nebenbedingungen erfüllt sind

erstrebenswerte Nebenbedingungen können nicht in Form von harten Constraints implementiert werden.

Für die Verarbeitung von weichen Constraints ist es notwendig, zusätzliche Informationen zu speichern, bspw. eine Priorität oder den Hinweis, dass das Constraint nicht unbedingt erfüllt sein muss. (Bspw. Speicherung als Fakt in Prolog)

Beispiel Stundenplan:

Dozenten dürfen Raum-, Wochentags- und Zeitwünsche für ihre Veranstaltung einreichen.

```
veranstaltungzeit(VeranstaltungsID,
                  GewünschterWochentag,
                  GewünschteZeit,
                  Priorität).
veranstaltungort( VeranstaltungsID,
                  GewünschterRaum,
                  Priorität).
```

Flexible CSP - Constraint Relaxierung (5)

MAX-CSP:

- Anzahl der Constraints, die verletzt werden dürfen, wird festgelegt.
- Anzahl darf unterschritten, jedoch nicht überschritten werden.
- Qualität einer Lösung wird daran gemessen, wieviele (weiche) Constraints erfüllt sind.

gewichtete CSP:

- ein MAX-CSP, bei dem die Constraints je nach ihrer Wichtigkeit mit unterschiedlichen Gewichten versehen werden (bspw. Priorität)
- Erfüllung von Constraints mit höheren Gewichten bevorzugt.

Flexible CSP - Constraint Relaxierung (6)

Fuzzy Constraints:

- Constraints werden als Fuzzy Relationen modelliert.
- Propagation in einem Fuzzy-CSP.
- Variable wird kein Wert „vorgeschrieben“
- Grad der Erfüllbarkeit (inwieweit weicht die Belegung einer Variable vom Sollwert ab),
  Grad hat einen Wert zwischen 0 und 1.

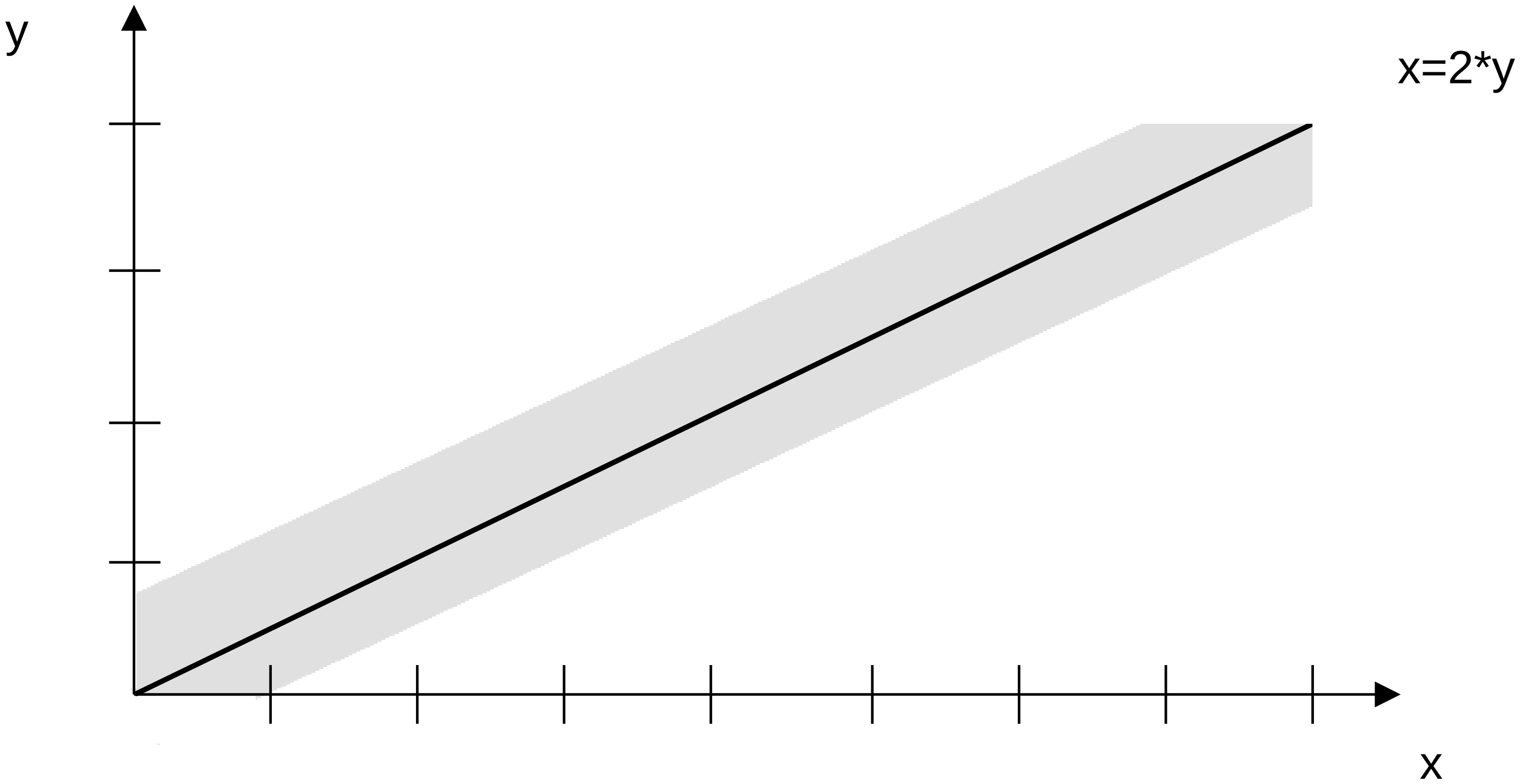
Constraint C über den Variablen $V_1, \ldots, V_k$ mit der konkreten Belegung $(d_1, \ldots, d_k) \in D_1 \times \ldots D_k$
erhält einen Wert zwischen 0 und 1.

$C(d_1, \ldots, d_k)=1$, Constraint ist mit dieser Belegung zu 100% erfüllt.

$C(d_1, \ldots, d_k)=0$, Constraint ist mit dieser Belegung vollständig verletzt.

Beispiel: Fuzzy Funktions-Constraint

Projektion einer Fuzzy Menge (Fuzzy Relation $\tilde{R}$) auf eine scharfe Menge (Relation R_α)



Beispiel: Fuzzy Tupel Constraint

Pseudo-Code:

```
C=(gleiche_farbe,
       {Farbe1,Farbe2},
       { ((rot,rot), 1), ((orange,orange), 1), ((gelb,gelb), 1),
         ((rot,orange) , 0.8), ((orange,rot), 0.8),
         ((gelb,orange), 0.5), ((orange,gelb), 0.5) }  )
```

- Im klassischen Fall „muss vollständig erfüllt sein“ :
  (rot,rot) (orange,orange) (gelb,gelb)
- Schwellwert-Herabsetzung von 1 auf 0.8:
  (rot,rot) (orange,orange) (gelb,gelb) (rot,orange) (orange,rot)
- Schwellwert-Herabsetzung von 1 auf 0.5:
  (rot,rot) (orange,orange) (gelb,gelb) (rot,orange) (orange,rot) (gelb,orange) (orange,gelb)

Flexible CSP - Constraint Relaxierung (7)

Bei einem Fuzzy-CSP ist man also nicht an einer beliebigen Wertebelegung (unter Umständen mit einer großen Abweichung) interessiert.

Ziel ist die Maximierung des Erfüllungsgrades für das gesamte Problem.

Ein Fuzzy-CSP ist somit auch ein Optimierungs-problem.

… Eigenschaften und Arbeitsweise von Constraints, Constraint Satisfaction Problemen und Propagierung kennen,

… Probleme mittels Constraints formulieren können,

… k-Konsistenz in einem Constraintnetz ermitteln können.