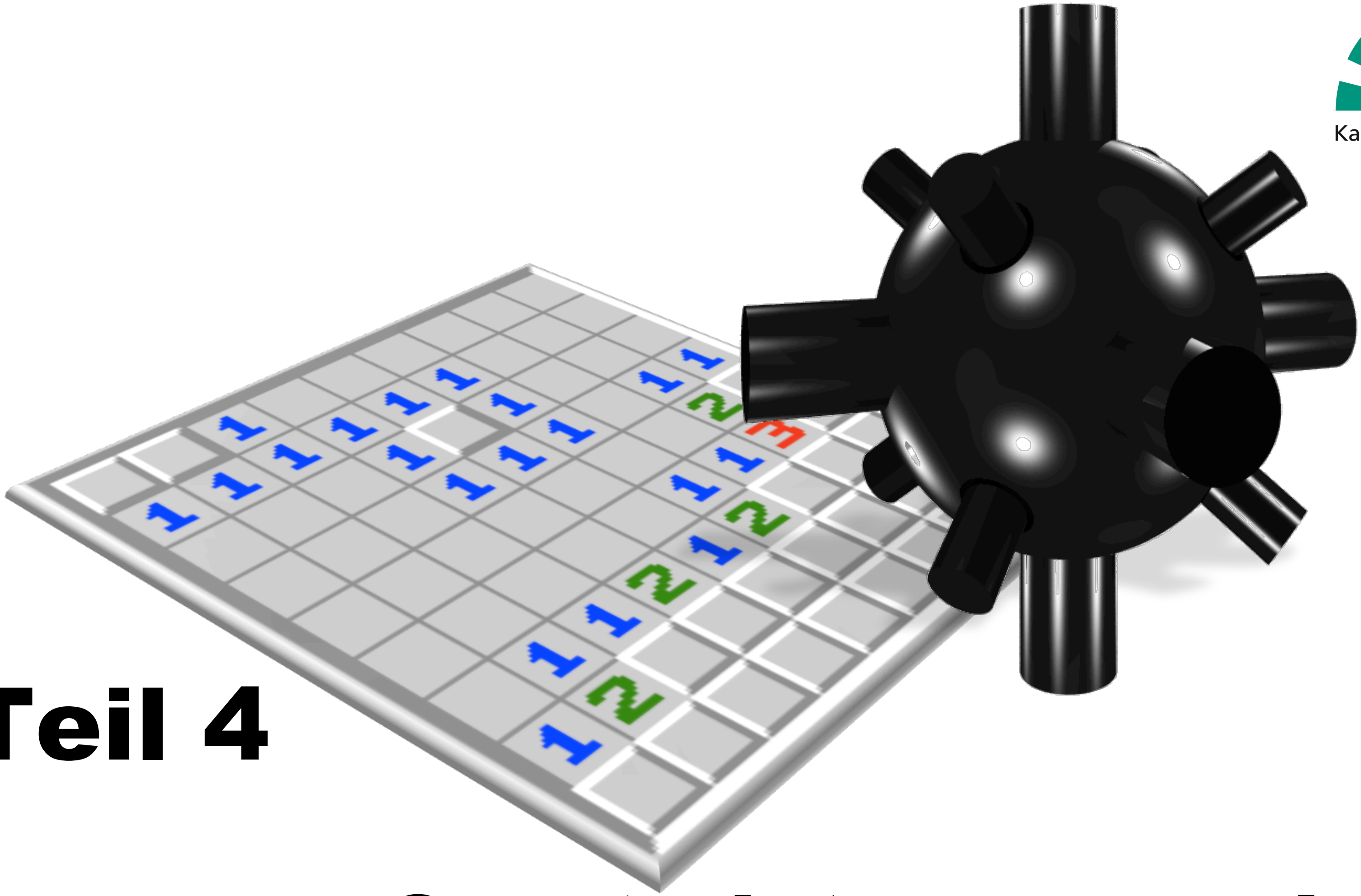
Karlsruhe Institute of Technology



Teil 4

Constraintprogrammierung

Zitat

- "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, Constraints, April 1997

Was sind Constraints?

- *constraint* (engl.) = Beschränkung, Einschränkung
- hier: Aussage darüber, welche Lösungen eines Problems (ausgedrückt durch Belegung von Variablen) verboten/erlaubt sind
- Also: constraints schränken die zulässigen Lösungen ein
- verschiedene Beschreibungsarten:
  - extensional (durch Aufzählung):
    $(X,Y,Z) \in \{(1,0,0),(0,1,0),(0,0,1)\}$
  - intensional (durch Formeln, die erfüllt sein müssen)
    $X+Y+Z = 1$

Constraint Satisfaction Problem

- CP = Ein Programm erstellt ein CSP zu einem Problem
- Ein CSP besteht aus:
  - *einer endlichen Menge von Variablen* $X=\{x_1,\ldots,x_n\}$
  - *Domänen* $D_1,\ldots,D_n$ *dieser Variablen (endliche Mengen möglicher Werte)*
  - *einer Menge von Constraints*

  Beispiel:
    - X::{1,2}, Y::{1,2}, Z::{1,2}
    - $X = Y$, $X \neq Z$, $Y > Z$
- Lösung eines CSPs:
  - *Zuweisung eines Wertes an jede Variable aus ihrer Domäne Beispiel:*
    - X=2, Y=2, Z=1

Was ist CP?

- Eine Methode, um kombinatorische Probleme zu lösen
- Beispiel: SEND+MORE=MONEY
  - ⇨ Menge von Variablen {S,E,N,D,M,O,R,Y}
  - ⇨ Domänen (Wertebereiche) für Variablen

    E,N,D,O,R,Y::0..9,   S,M::1..9
  - ⇨ Constraints (Einschränkungsbedingungen)

    schränken den Wertebereich von Variablen ein

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

- Aufgabe: Finde einen Wert für jede Variable, so dass alle Constraints erfüllt sind.

CP und der "Heilige Gral"

- Modellierung auf „natürliche“ Weise (verglichen mit „normalen“ Programmiersprachen)
- Beispiel: Send+More=Money in Prolog:

definiere Variablen und Domänen

```
send_more_money :-
   [S,E,N,D,M,O,R,Y] :: [0..9],
```

Constraints?

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

CP und der "Heilige Gral"

- Modellierung auf „natürliche“ Weise (verglichen mit „normalen“ Programmiersprachen)
- Beispiel: Send+More=Money in Prolog:

```
send_more_money :-
   [S,E,N,D,M,O,R,Y] :: [0..9],
   S #\= 0, M #\= 0,
   alldifferent([S,E,N,D,M,O,R,Y]),
   1000*S+100*E+10*N+D + 1000*M+100*O+10*R+E
      #= 10000*M+1000*O+100*N+10*E+Y,
   labeling([S,E,N,D,M,O,R,Y]).
```

definiere Variablen und Domänen

Constraints

starte Suche

- Das System sucht und findet automatisch eine Lösung
  - ⇨ Intelligenz ist eingebaut

Beispiel: Sudoku

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

Beispiel: Sudoku

- Modellierung (Problemformulierung)
  - ⇨ zu belegende Variablen bilden ein 9x9-Array: n(i,j)
  - ⇨ jede Variable kann Wert zwischen 1 und 9 annehmen: n(i,j)::{1,2,3,4,5,6,7,8,9}
  - ⇨ Constraints:
    - ✧ alle Zahlen in einer Spalte verschieden: $j \neq k \rightarrow n(i,j) \neq n(i,k)$
    - ✧ alle Zahlen in einer Zeile verschieden: $j \neq k \rightarrow n(j,i) \neq n(k,i)$
    - ✧ alle Zahlen in einem Teilquadrat verschieden: h div 3 = i div 3, j div 3 = k div 3, $(h \neq i$ oder $j \neq k) \rightarrow n(h,j) \neq n(i,k)$

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

Kurzer Überblick über CP

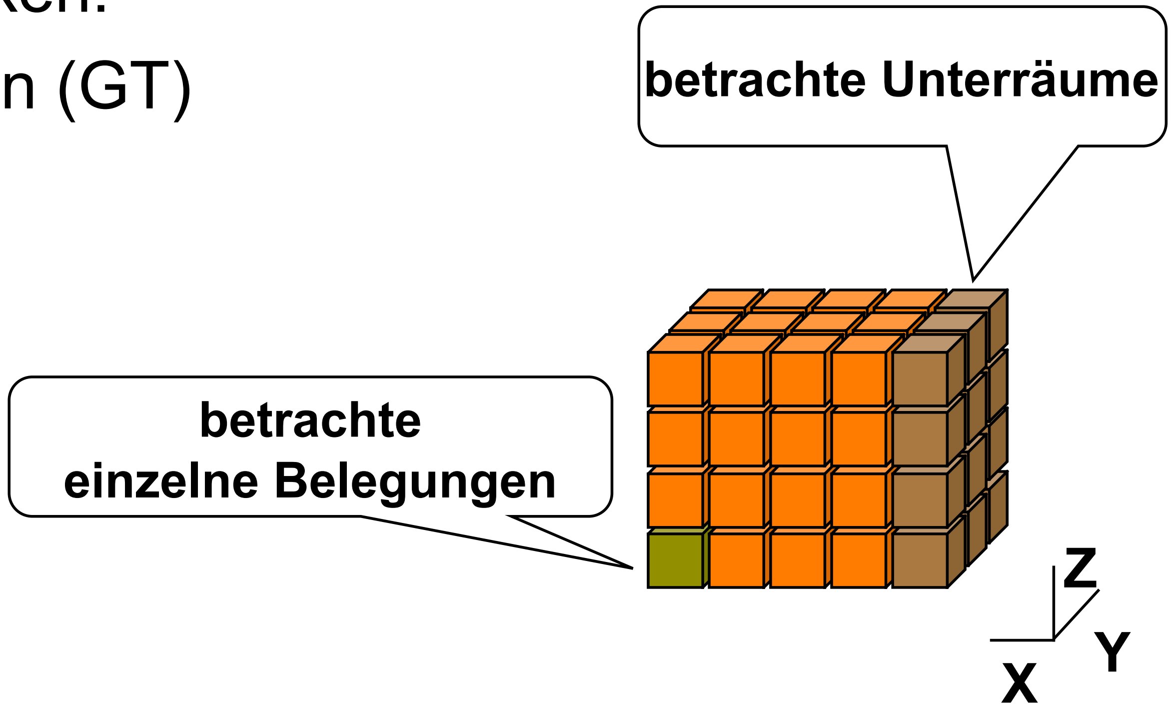
- Modellierung
  - Ein Problem der realen Welt wird mittels Constraintprogrammierung beschrieben
- Suche einer Instanz (labeling)
  - mittels der bekannten Techniken der KI
- Propagierung (domain filtering)
  - Inkonsistenzen werden im Voraus entfernt

Systematische Suche

- Durchsuchung des gesamten Lösungsraumes
- vollständig und korrekt
- Effizienzprobleme (sehr aufwendig)

Grundlegende Techniken:

- Generieren & Testen (GT)
- Backtracking (BT)



Erschöpfende Suche

- Generiere und Teste (GT)
  - ⇨ Erzeuge systematisch einen Lösungskandidaten
  - ⇨ Teste, ob alle Constraints erfüllt sind
  - ⇨ Beispiel:
    - ✧ X::{1,2}, Y::{1,2}, Z::{1,2}
    - ✧ $X = Y,\ X \neq Z,\ Y > Z$

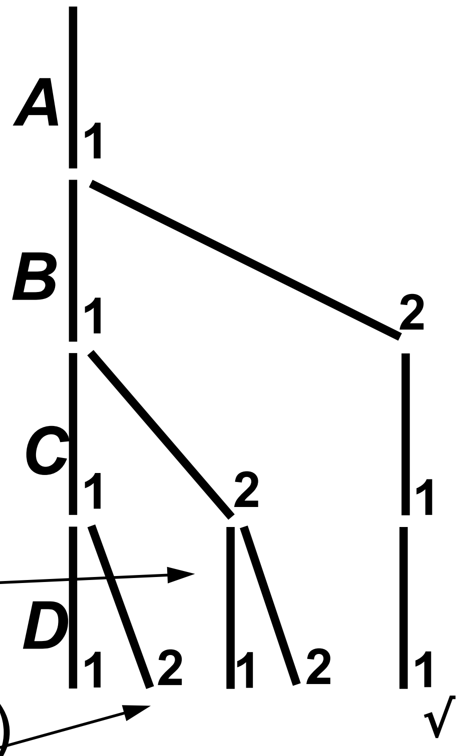
X	1	1	1	1	2	2	2
Y	1	1	2	2	1	1	2
Z	1	2	1	2	1	2	1

Probleme:

- der Generator ist „blind“
  - ⇨ generiere Lösungskandidaten intelligent
- Probleme (Inkonsistenzen) werden zu spät erkannt
  - ⇨ teste Constraints bei Instanziierung der beteiligten Variablen

Backtracking (BT)

- Erweitert eine Teillösung inkrementell zu einer vollständigen Lösung
- Algorithmus:
  - weise einer Variablen einen Wert zu
  - teste auf Konsistenz
  - wiederhole, bis alle Variablen belegt sind
- Tiefensuche (komplexe Rücksetzung)
- Nachteile:
  - ⇨ thrashing
  - ⇨ unnötige Wiederholungen (Redundanz)
  - ⇨ Konflikte werden spät erkannt



A = D, B ≠ D, A+C < 4

GT & BT - Beispiel

- Problem:

  X::{1,2}, Y::{1,2}, Z::{1,2}

  $X = Y,\ X \neq Z,\ Y > Z$

Generiere & Teste

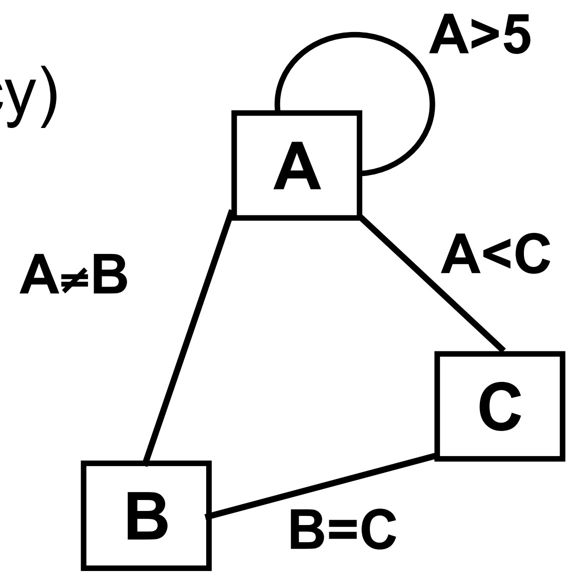
X	Y	Z	Test
1	1	1	nein
1	1	2	nein
1	2	1	nein
1	2	2	nein
2	1	1	nein
2	1	2	nein
2	2	1	ja

Backtracking

X	Y	Z	Test
1	1	1	nein
		2	nein
	2		nein
2	1		nein
	2	1	ja

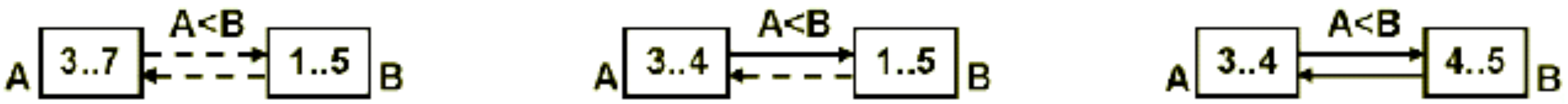
Konsistenztechniken

- entfernen inkonsistenter Werte aus den Domänen der Variablen
- Repräsentierung eines CSPs als Graph
  - ⇨ nur binäre und unäre Constraints (n-äre Constraints sind kein Problem!)
  - ⇨ Knoten = Variablen
  - ⇨ Kanten = Constraints
- Knotenkonsistenz (NC) (node consistency)
- Kantenkonsistenz (AC) (arc consistency)
- Pfadkonsistenz (PC)
- (starke) k-Konsistenz



Kantenkonsistenz AC

- Constraint = Kante im Constraint-Graphen
- Definition:
  - ⇨ Kante(i,j) ist kantenkonsistent gdw. es für jeden Wert in D_i einen kompatiblen Wert in D_j gibt. (gerichtet!)
  - ⇨ Ein CSP ist kantenkonsistent gdw. es für alle Kanten kantenkonsistent ist (in beiden Richtungen).



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

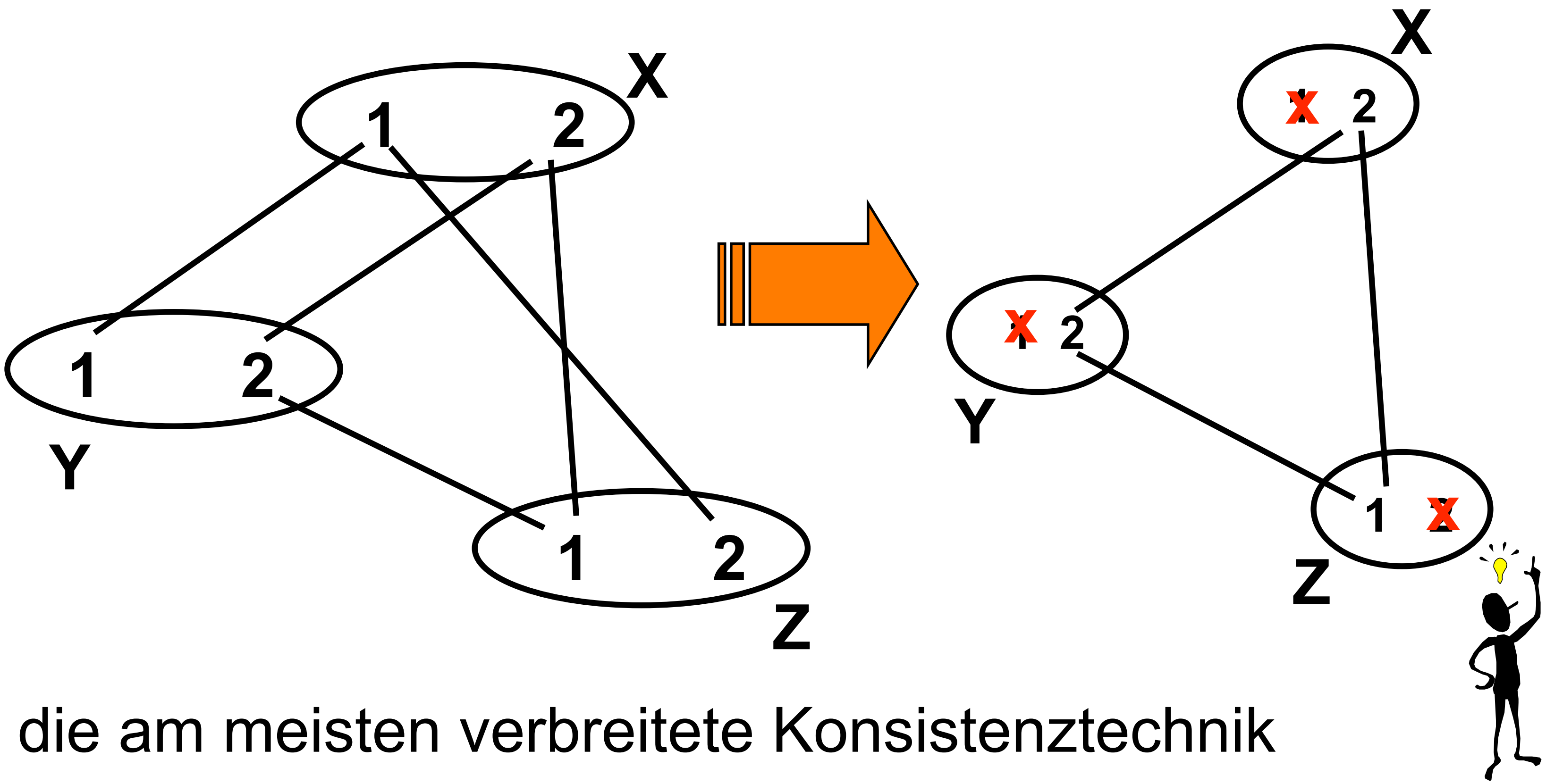
- Erreichen der Kantenkonsistenz:
  - ⇨ Einschränkungen solange wiederholen bis sich keine Domänen mehr ändern

AC - Beispiel II

- Problem:

  X::{1,2}, Y::{1,2}, Z::{1,2}

  X = Y, X ≠ Z, Y > Z



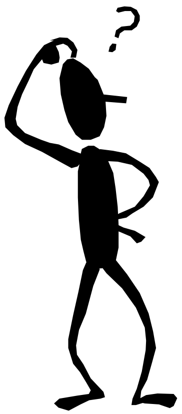
- AC ist die am meisten verbreitete Konsistenztechnik
- viele Algorithmen mit unterschiedlichen Leistungsmerkmalen

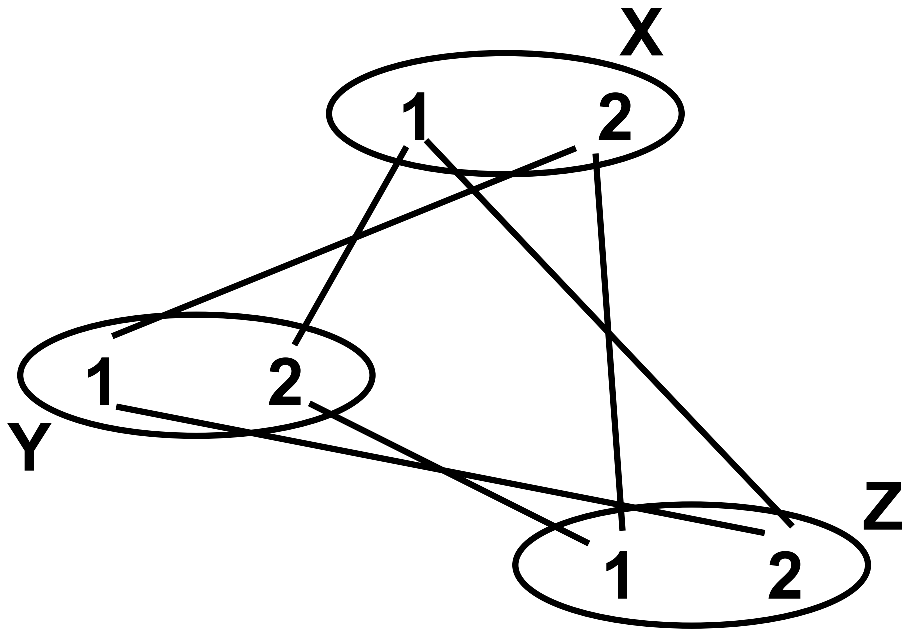
Ist AC ausreichend?

- leere Domäne → Inkonsistenz gefunden, keine Lösung
- Kardinalität aller Domänen ist 1 → Lösung gefunden
- Problem:

  X::{1,2}, Y::{1,2}, Z::{1,2}

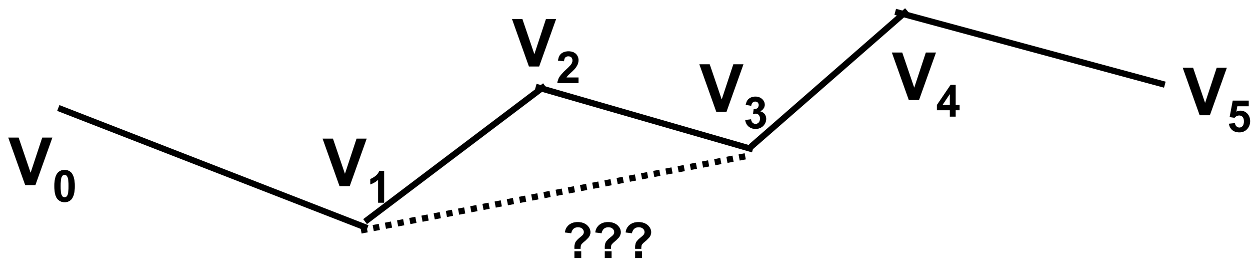
  $X \neq Y$, $X \neq Z$, $Y \neq Z$





Pfadkonsistenz (PC)

- nur Konsistenz entlang von Pfaden



- es genügt, Pfade der Länge 2 zu untersuchen
- Vor- und Nachteile
  + findet mehr Inkonsistenzen als AC
  - extensionale Repräsentation der constraints
  - verändert die Konnektivität des Graphen
- gerichtete PC, eingeschränkte PC

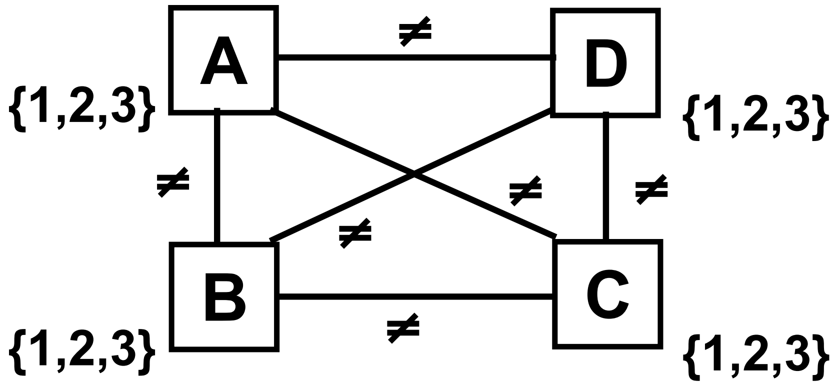
k-Konsistenz

- k-Konsistenz
  - ⇨ konsistente Belegung von (k-1) Variablen kann auf die k-te Variable erweitert werden
- starke k-Konsistenz
  - ≡ j-Konsistenz für jedes j≤k
- NC ≡ starke 1-Konsistenz
- AC ≡ starke 2-Konsistenz
- PC ≡ starke 3-Konsistenz

Vollständige Konsistenz

- starke n-Konsistenz eines Constraint Graphen mit n Knoten → Lösung
- starke k-Konsistenz eines Constraint Graphen mit n Knoten (k<n) → ???

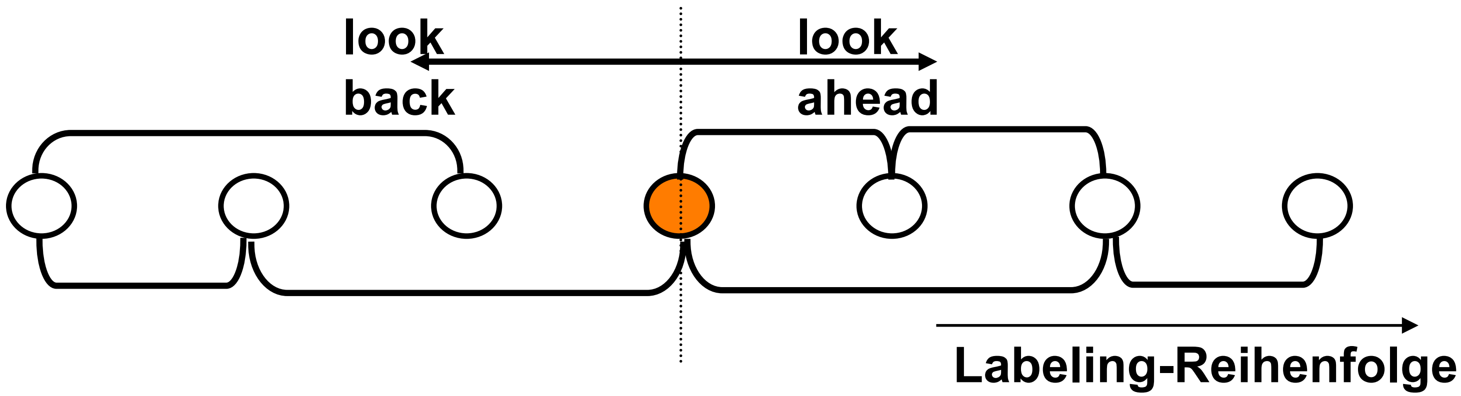
Pfadkonsistent
aber keine Lösung



- spezielle Graphen
  - ⇨ baumartiger Graph → (D)AC ist ausreichend
  - ⇨ eliminiere Knoten (cycle cutset)
  - ⇨ fasse Knoten zusammen

Constraintpropagierung

- nur systematische Suche → ineffizient
- nur Sicherstellung der Konsistenz → unvollständig
- Also: Kombination von Suche (Backtracking) mit Konsistenztechniken
- Methoden:
  - ⇨ look back (Wiederherstellen nach Konflikten)
  - ⇨ look ahead (Verhindern von Konflikten)

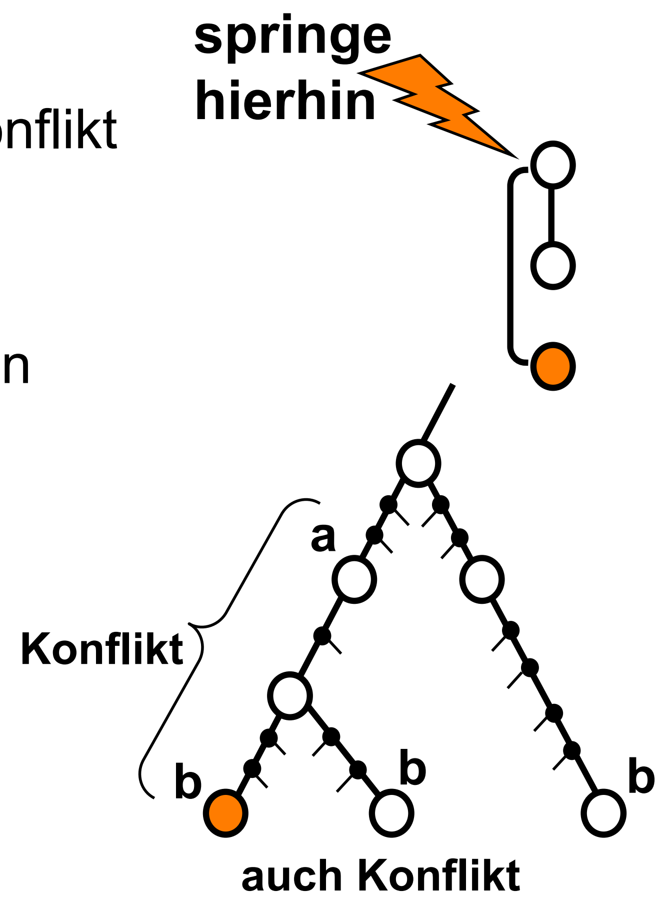


Look Back Methoden

- intelligentes Backtracking
- Konsistenzprüfung der instanziierten Variablen

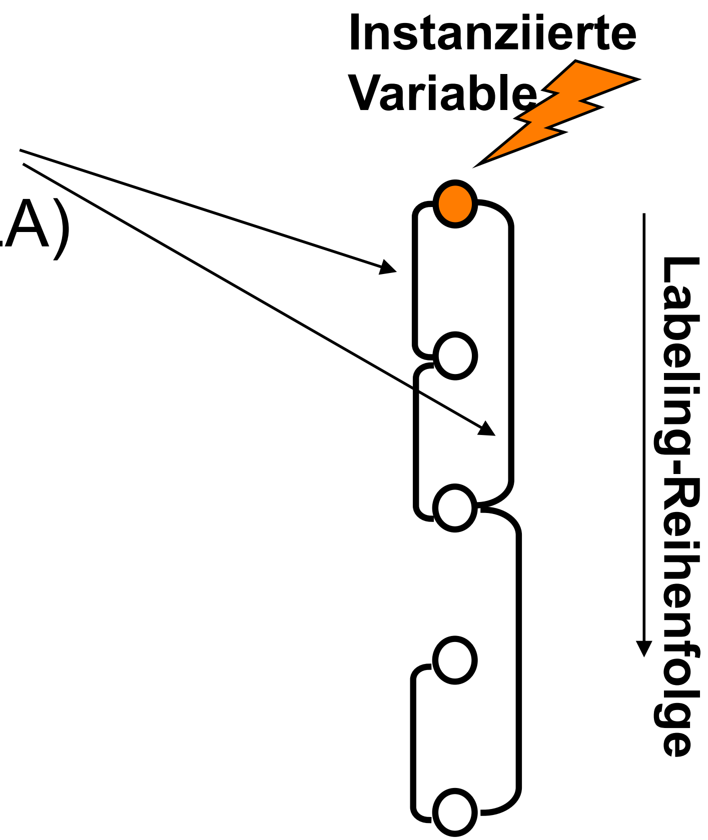
- *backjumping*
  - ⇨ Backtracking zu der Variable mit dem Konflikt

- *backchecking* und *backmarking*
  - ⇨ vermeidet redundante Überprüfungen von Constraints durch das Speichern von erfolgreichen und fehlgeschlagenen Wertzuweisungen



Look Ahead Methoden

- vermeide zukünftige Konflikte durch Konsistenz-überprüfung mit noch nicht instanziierten Variablen
- forward checking (FC)
  - AC to mit dem direkten Nachbarn
- teilweiser (partial) look ahead (PLA)
  - DAC
- (vollständiger) look ahead (LA)
  - Kantenkonsistenz
  - Pfadkonsistenz



Look Ahead - Beispiel

- Problem:

  X::{1,2}, Y::{1,2}, Z::{1,2}

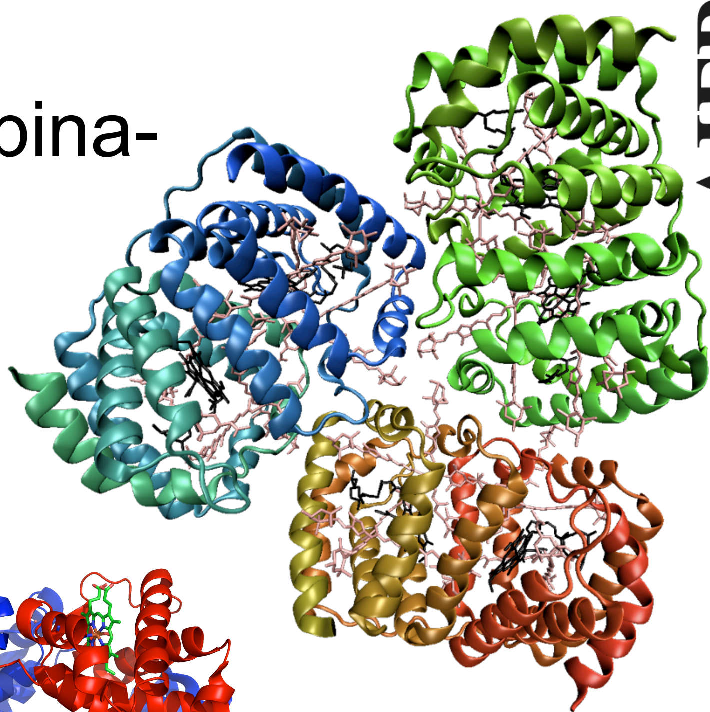
  $X = Y, X \neq Z, Y > Z$

X	Y	Z	Aktion	Ergebnis
1			Labeling	
	{1}	{}	Propagierung	Fehlschlag
2			Labeling	
	{2}	{1}	Propagierung	Lösung

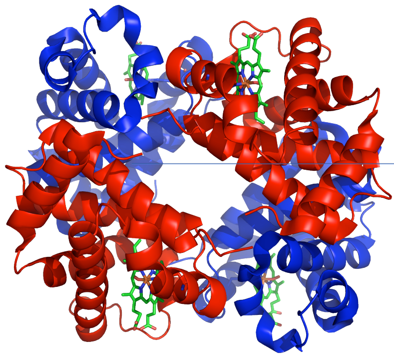
Generieren & Testen - 7 Schritte
mit Backtracking - 5 Schritte
mit Propagierung - 2 Schritte

Anwendungsfelder I

Alle Arten von schwierigen kombinatorischen Problemen



- ➢ Molekularbiologie
  - ⇨ Sequenzierung von DNA
  - ⇨ Protein-Strukturanalyse



- ➢ Interaktive Grafik
  - ⇨ Web-Layout
  - ⇨ Multimodale Ausgabe, z.B. im DynAMITE-Projekt
- ➢ Network-Management und -Konfiguration

Job Shop Scheduling

- Aufgabe bei Scheduling:
  - ⇨ Zuweisung von Aktivitäten an Ressourcen zu einer Zeit
- CP erlaubt die Modellierung auf „natürliche" Weise
  - ⇨ Nebenbedingungen werden abgedeckt!
- Aktivitäten werden mittels Variablen beschrieben:
  - ⇨ Anfangszeit
  - ⇨ Ressourcen (wenn nötig)
  - ⇨ Dauer (falls variabel)
- Constraints beschreiben Beziehungen
  - ⇨ zwischen einzelnen Aktivitäten (Vorrang, Reihenfolge, ...)
  - ⇨ zwischen Aktivitäten und Ressourcen (Kapazität, ...)
- Lösungstechniken: standard Constraint-Verfahren

Temporale Constraints

- Termin-Constraints (due times)
  - ⇨ Fertigstellung vor einem festgelegtem Zeitpunkt
  - ⇨ start(A) + dauer(A) ≤ ende(A)
- Vorrang-Constraints (precedence constraints)
  - ⇨ Aktivität A muß einer andere Aktivität B vorangehen (aus technologischen Gründen)
  - ⇨ start(A) + dauer(A) ≤ start(B)
- Zeitfenster (time windows)
  - ⇨ Aktivität muß während eines Zeitfensters stattfinden
  - ⇨ start(A) in MinStart...MaxStart
- Einricht-Zeiten (set-up times), ...

Ressourcen Constraints

- Disjunktive Constraints
  - ⇨ Ressource ist auf eine gleichzeitige Aktivität beschränkt
  - ⇨ start(A)+dauer(A)≤start(B) oder start(B)+dauer(B)≤start(A)
  - ⇨ globale Constraints
    - ✧ serialized([start(Ai)], [dauer(Ai])
- Kapazitätsconstraints
  - ⇨ Ressource kann Aktivitäten mit einem gegebenen Limit parallel bearbeiten
  - ⇨ deckt auch Disjunktive Constraints ab (Kapazität = 1)
  - ⇨ cumulative([start(A_i)], [dauer(A_i)], [verbrauch(A_i)], limit)
    - ✧ Trick: Ist die Ressource an der Aktivität nicht beteiligt, dann ist ihr Verbrauch gleich Null.

Was wurde verschwiegen?

- Techniken der lokalen Suche

  nicht-systematisches Generieren und Testen

  ⇨ hill-climbing, min-conflicts, random-walk, tabu search
- Probleme mit zu vielen Constraints (over-constraint)

  Was ist zu tun, wenn die Constraintmenge unerfüllbar ist?

  ⇨ Abschwächen des Problems (weakening, PCSP)

  ⇨ Verwenden von Präferenzen (Hierarchien von Constraints)
- Optimierung

  Suche nach optimalen Lösungen, gemäß einer Zielfunktion

  ⇨ Branch und Bound
- Realzahlen

  Methoden der numerischen Mathematik (Newton, Taylor)

Was wurde gesagt?

- Constraint Programmierung
  - kommt dem Heiligen Gral der Informatik nahe
  - ist relevant bei der Lösung interessanter Probleme
- Modellierung und systematische Suche
- Erschöpfende Suche ist zu aufwendig
- Verbesserungen:
  - Konsistenztechniken (look ahed)
    - verhindere Konflikte in der Zukunft
  - Techniken zur Vermeidung von Redundanzen (look back)
    - verwendet Informationen über vorangegangene Schritte

Quellen und Literatur

- Roman Barták. On-Line Guide to Constraint Programming, http://kti.mff.cuni.cz/~bartak/constraints. (englisch)
- Russell und Norvig: Artificial Intelligence: A Modern Approach, Prentice Hall, 1995, 2te Auflage, Kapitel 5 (online!)
- Roman Barták. Vortrag CPDC99. (englisch)
- N-Queens v1.0, ©1996 CyLog Software, http://www.cylog.org/games_6.asp