

Constraint Solving

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

A. Conan Doyle

Einführungsvortrag MAMS-Projekt

1.Dez 2006

Katrin Lang

user.cs.tu-berlin.de/~langk

Inhaltsübersicht

- Begriffsklärung
- Einsatzgebiete
- TUCS- ein Finite Domain Constraint-Solver
 - Lösungsstrategien
 - Datenstrukturen
 - Algorithmen
 - Implementierung

Begriffe I

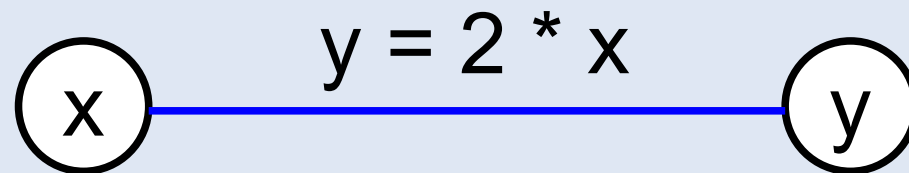
- *Constraint Satisfaction Problem (CSP)*
 - eine Menge von *Variablen*,
 - *Wertebereiche (Domains)* der Variablen,
 - *Relationen* zwischen den Variablen
 - Bilden *Constraint-Graph* mit Variablen
- *Constraints* beschreiben gültige Variablenbelegungen
 - Werden verundet
 - Kriterien für gültige Lösungen
 - N-stellige Constraints: Beziehung zwischen n Variablen

Begriffe II

- *Constraint Solver: Verfahren/Algorithmen* zur Einschränkung der Wertebereiche, zur Bestimmung der Erfüllbarkeit von Constraints und zur Berechnung von Lösungen.
- Basisalgorithmus: Aufbauen eines Lösungsbaums mit Backtracking
- *Unäre* (einstellige) Constraints müssen nur einmal angewandt werden

Begriffe III

- *Lokale Konsistenz*: Für jeden Wert aus der Domain einer Variable x gibt es einen Wert aus den Domains der durch das Constraint C mit x verbundenen Variablen, so daß C erfüllt ist.



$$x = \{1, 3, 4\}$$

$$y = \{0, \dots, 7\}$$

\Rightarrow

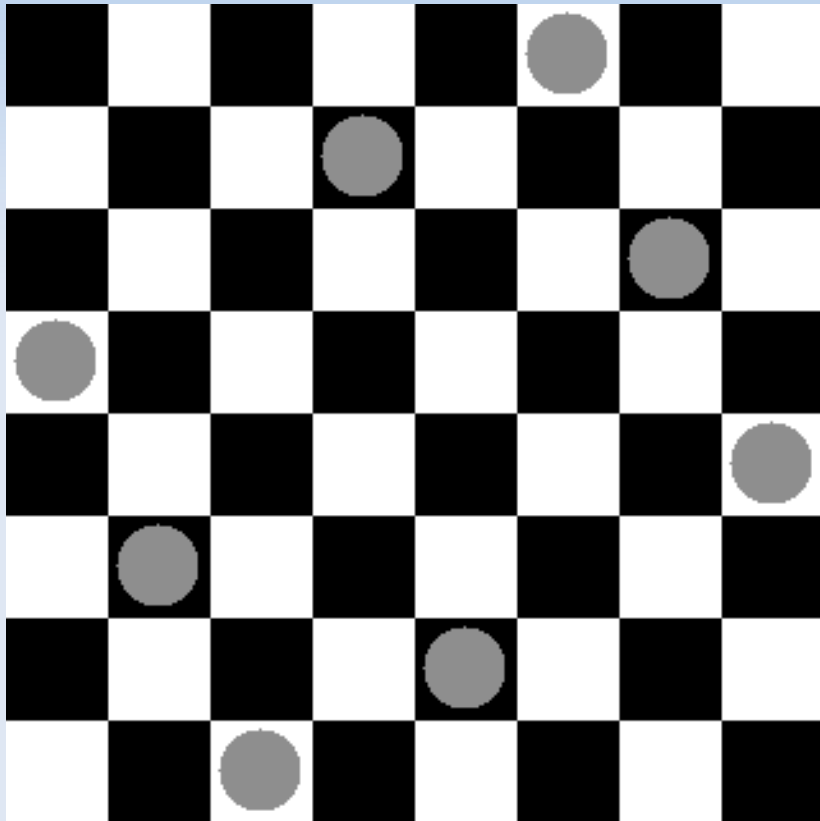
$$x = \{1, 3\}$$

$$y = \{2, 6\}$$

Einsatzgebiete

- NP-harte Probleme!

N-Queens



- Variablen = Spaltenindizes
- Domains: $\{1, \dots, \text{Zeilenanzahl}\}$
- 1 All-Different-Constraint für die Spalten
- 2 All-Different-Offset-Constraints für die Diagonalen

Send-More-Money

$$\begin{array}{r} 1000 S + 100 E + 10 N + D \\ + \quad 1000 M + 100 O + 10 R + E \\ \hline = 10000 M + 1000 O + 100 N + 10 E + Y \end{array}$$

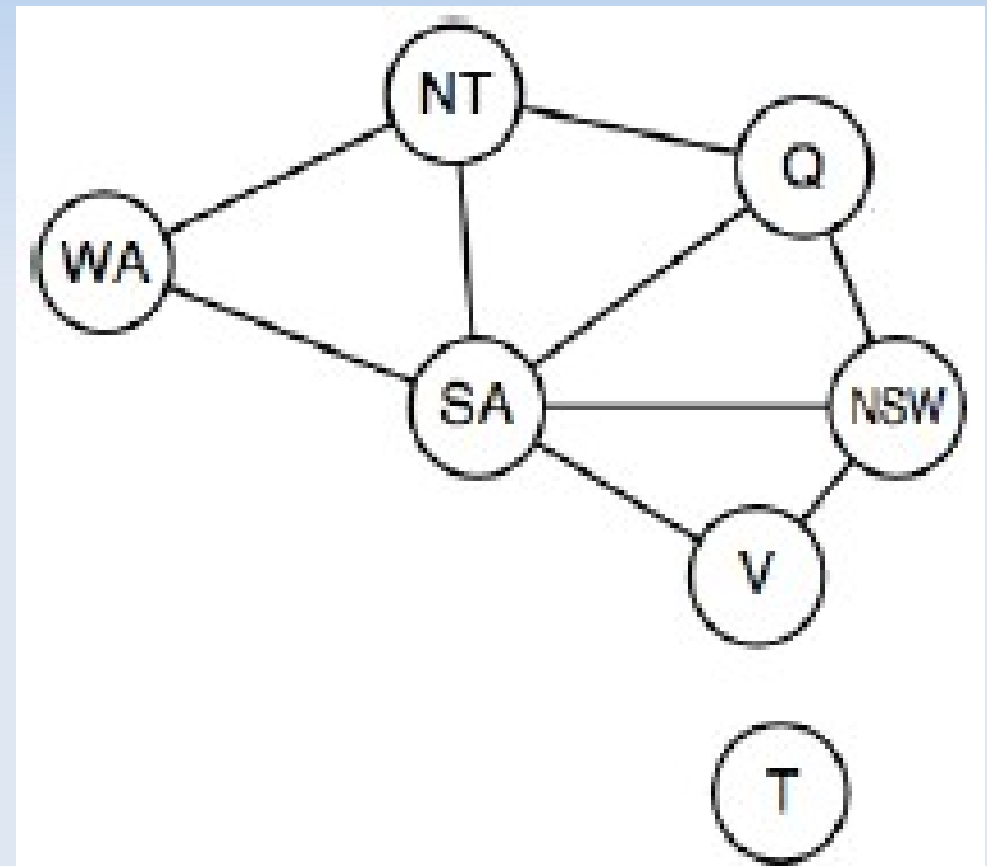
- Variablen S, E, N, D, M, O, R, Y
- Domains jew. $\{0, \dots, 9\}$
- zwei unäre Constraints, die für S bzw M den Wert 0 ausschließen,
- ein All-Different-Constraint
- ein lineares Constraint

Sudoku

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

- 81 Variablen
- Domains: $\{1, \dots, 9\}$
- 27 All-Different-Constraints
 - 9 für die Spalten
 - 9 für die Zeilen
 - 9 für die Subquadrate
- Plus unäre Constraints für die vorgegebenen Werte

Färbbarkeit von Graphen



Scheduling-Probleme

- **Jobs** - Liste, die fuer jeden Job eine Liste der benötigten Betriebsmittel (BM) angibt
- **Resources** - Anzahl der Ressourcen fuer jedes BM
- **Total-time** - Anzahl der gesamten Zeitschritte
- **Duration** - Benötigte Zeit (Zeitschritte) fuer jeden Job
Default: 1 fuer alle Jobs
- **Begin** - Bereitstellungszeit für jeden Job
Default: 0 für alle Jobs
- **Before** - Liste mit zeitlichen Abhängigkeiten, wobei gilt: Job i muss vor Job j ausgeführt werden gdw. (i j) in before enthalten
Default : keine Abhängigkeiten
- **Dead-lines** - Liste von Deadlines fuer jeden job
Default: Total-Time

- FD-Constraint-Solver in *Common Lisp*
- Beliebige finite Domains
 - Effiziente Algorithmen teilweise nur für Integer-Domains
- Beliebige Polynome als Constraints
- Strategien zur Laufzeit wählbar

Common Lisp

- Dynamisch getypt, Typdeklarationen optional
- Generische Listen
 - Abgebildet auf beliebige finite Domains
- Funktional + Seiteneffekte
- Objektorientiert
 - Multiple Vererbung
 - Generische Funktionen
 - Methodenkombination
- Makros, Code als Daten

Ziele

- Hybrides Framework für CSPs

Strategien

- Strategien = Klassen
- Vorteile:
 - Verwaltung eigener Datenstrukturen
 - Leichte Erweiterbarkeit und Kombinierbarkeit

Ziele

- Hybrides Framework für CSPs

Strategien

```
graph TD; A[Strategien] --> B[Look-Back];
```

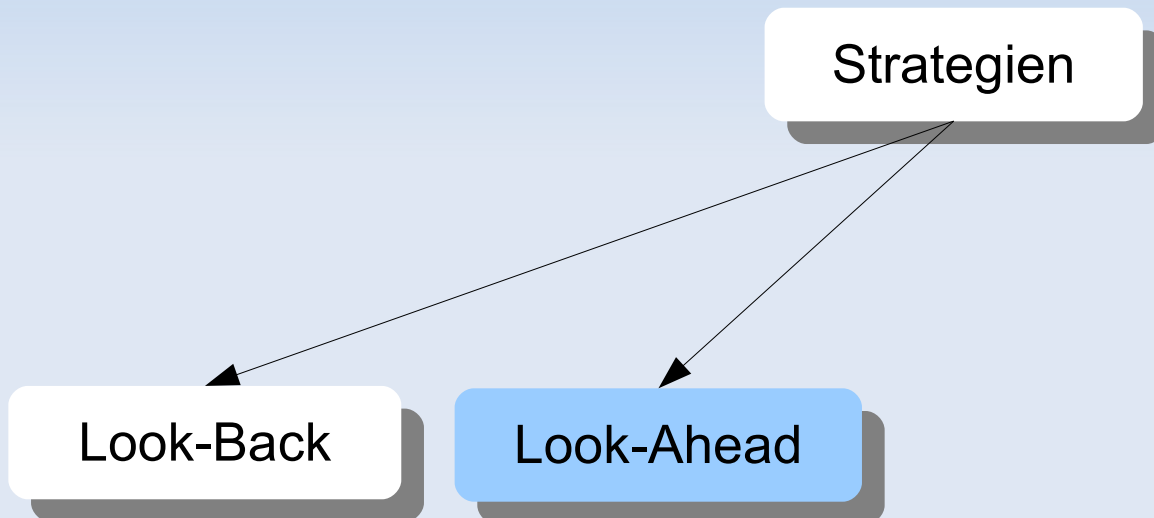
The diagram consists of two rounded rectangular boxes. The top box is white with a dark grey drop shadow and contains the text 'Strategien'. The bottom box is light blue with a dark grey drop shadow and contains the text 'Look-Back'. A thin black arrow points from the bottom-left corner of the 'Strategien' box to the top-left corner of the 'Look-Back' box.

Look-Back

- Bestimmt die Traversierung im Suchbaum
→ Einfachste Variante: *Backtracking*

Ziele

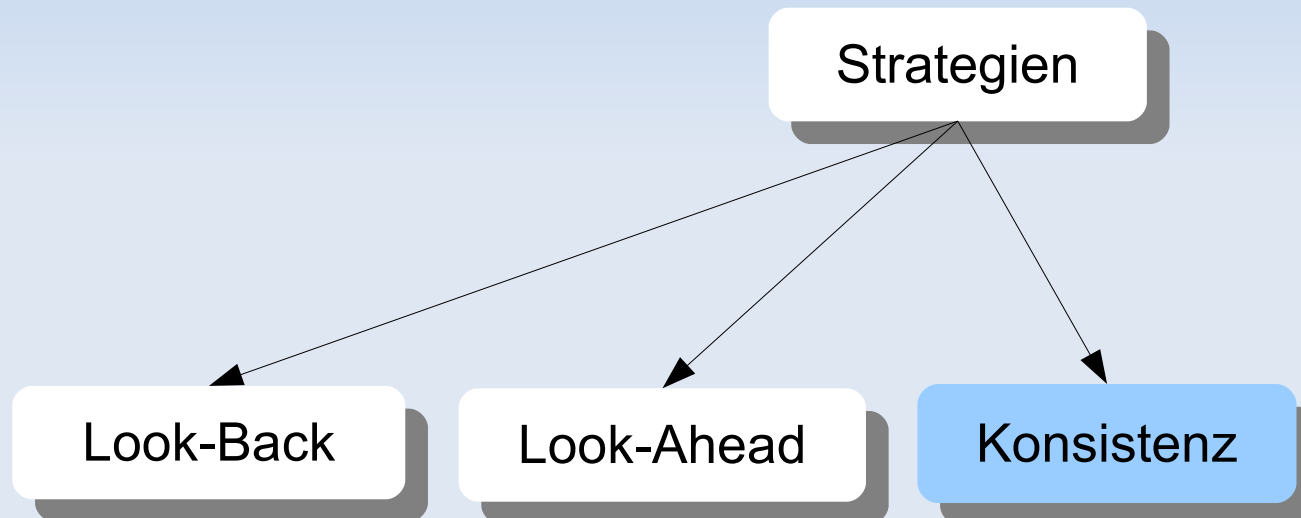
- Hybrides Framework für CSPs



- Vorausschauendes Einschränken von Domains
- Propagieren der (teilweisen) lokalen Konsistenz durch den Constraint-Graphen

Ziele

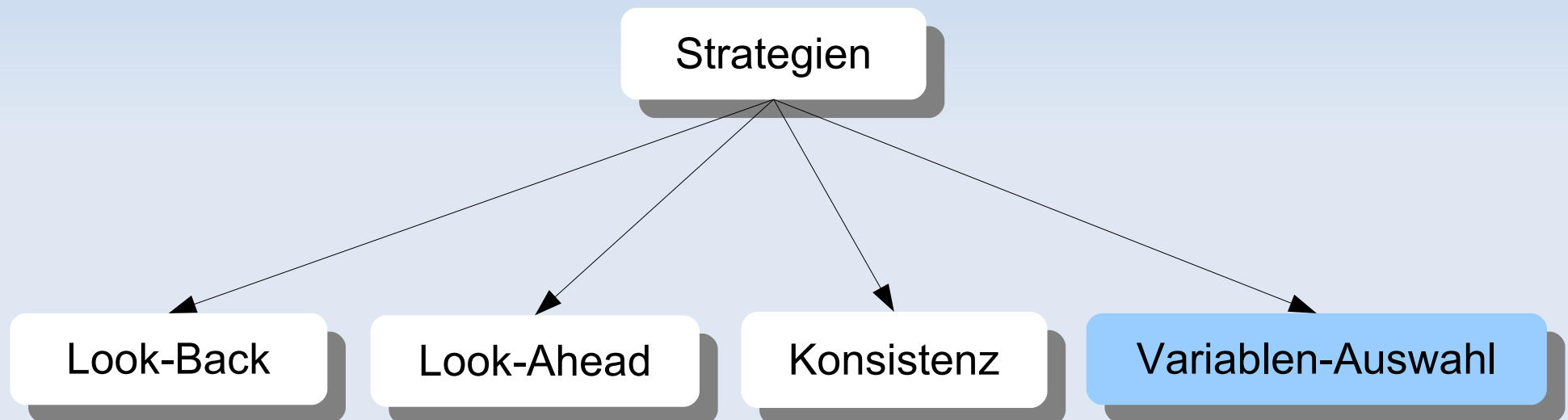
- Hybrides Framework für CSPs



- (teilweise) Herstellung der lokalen Konsistenz (Domain-Konsistenz / Grenzkonsistenz)

Ziele

- Hybrides Framework für CSPs



- Heuristiken zur Auswahl der zu instanziiierenden Variablen
 - nach *Anzahl* beteiligter *Constraints* (Fail-First)
 - nach *Kardinalität* der *Domains* (Success-First)

Die Modellierung eines CSP

CSP

init-domain-store

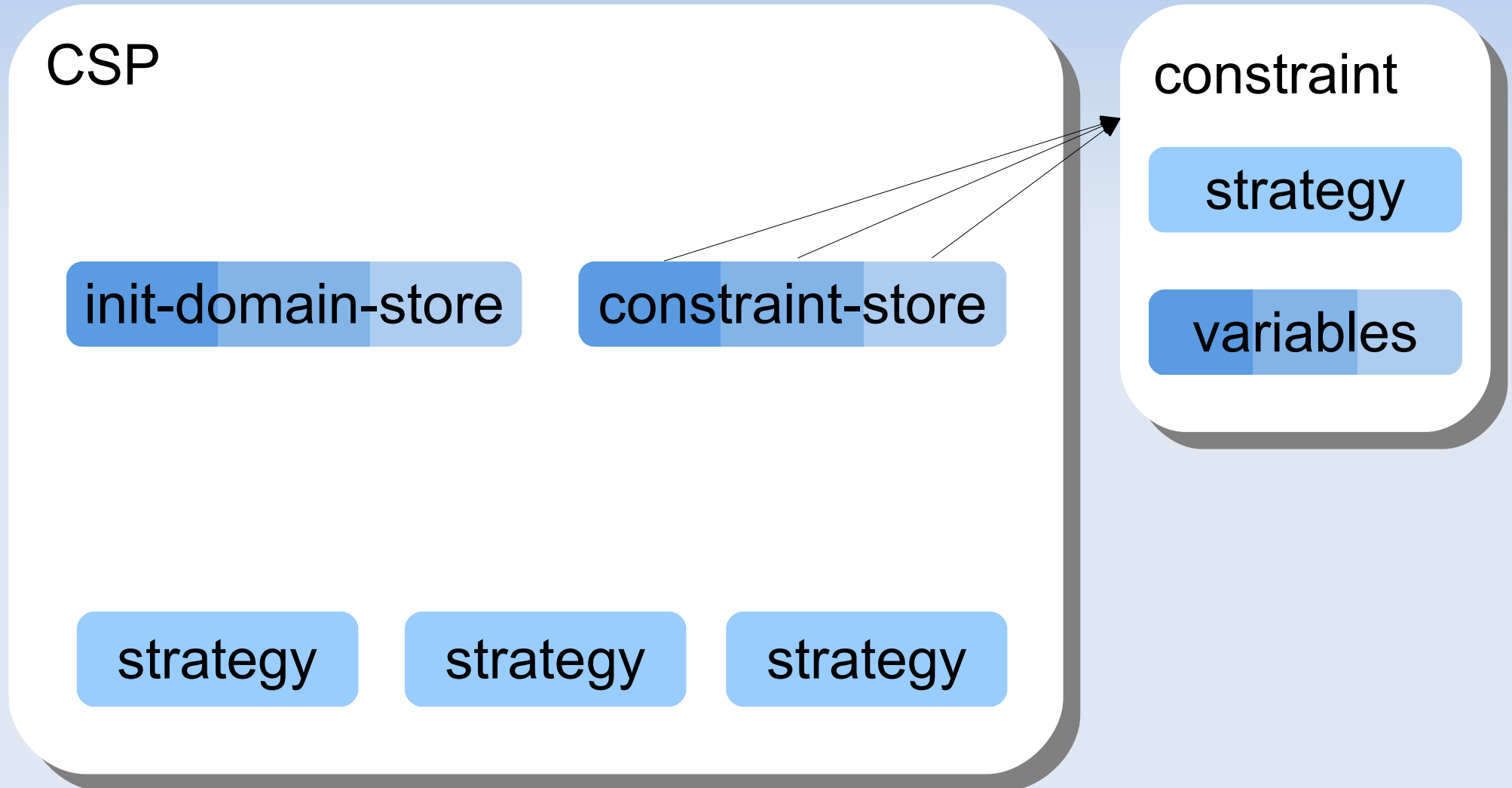
constraint-store

strategy

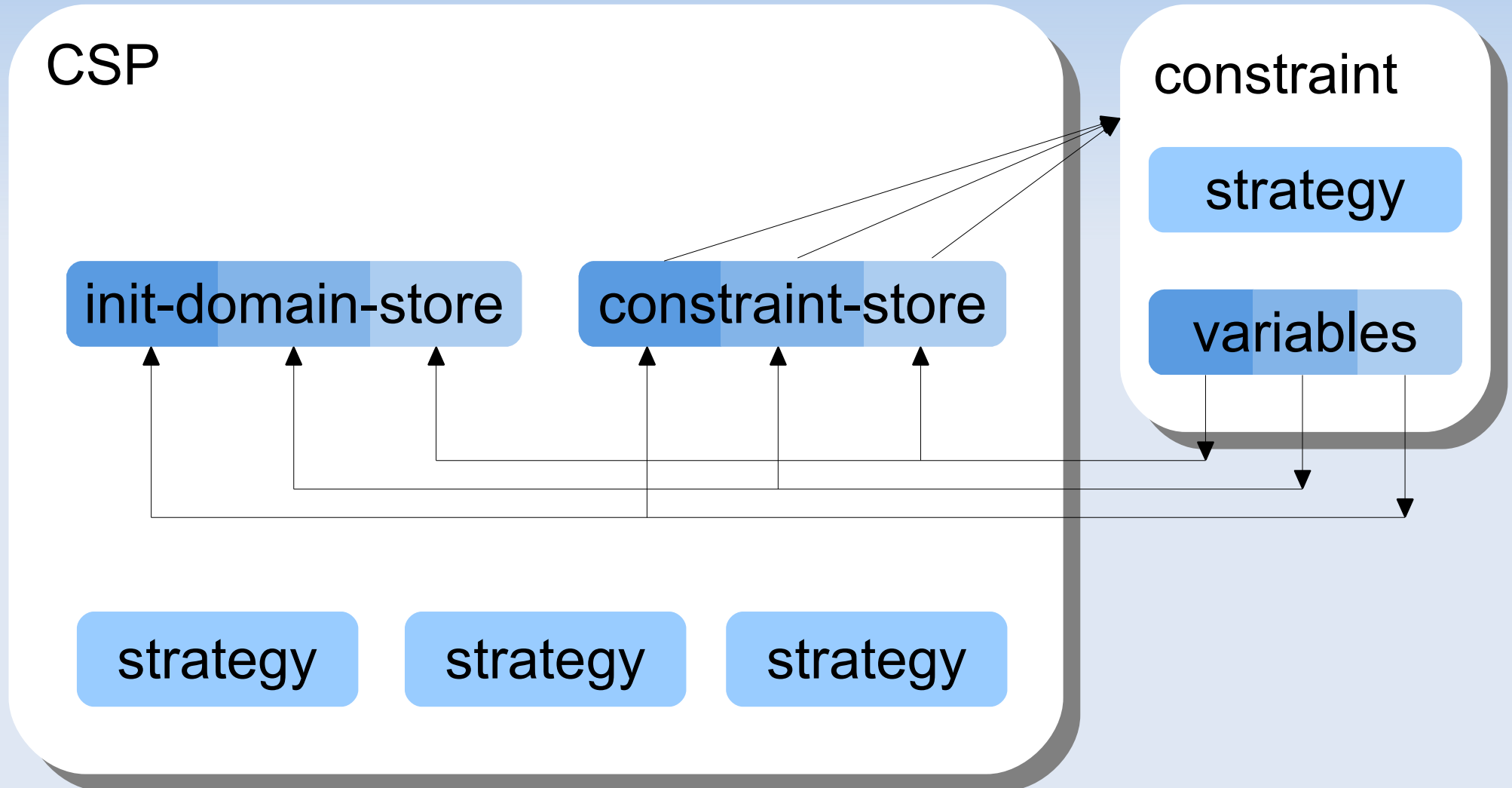
strategy

strategy

Die Modellierung eines CSP



Die Modellierung eines CSP



Die Modellierung eines CSP

CSP

indirekte Indizierung (nur Tiefensuche)

init-domain-store

constraint-store

constraint

strategy

variables

strategy

strategy

strategy

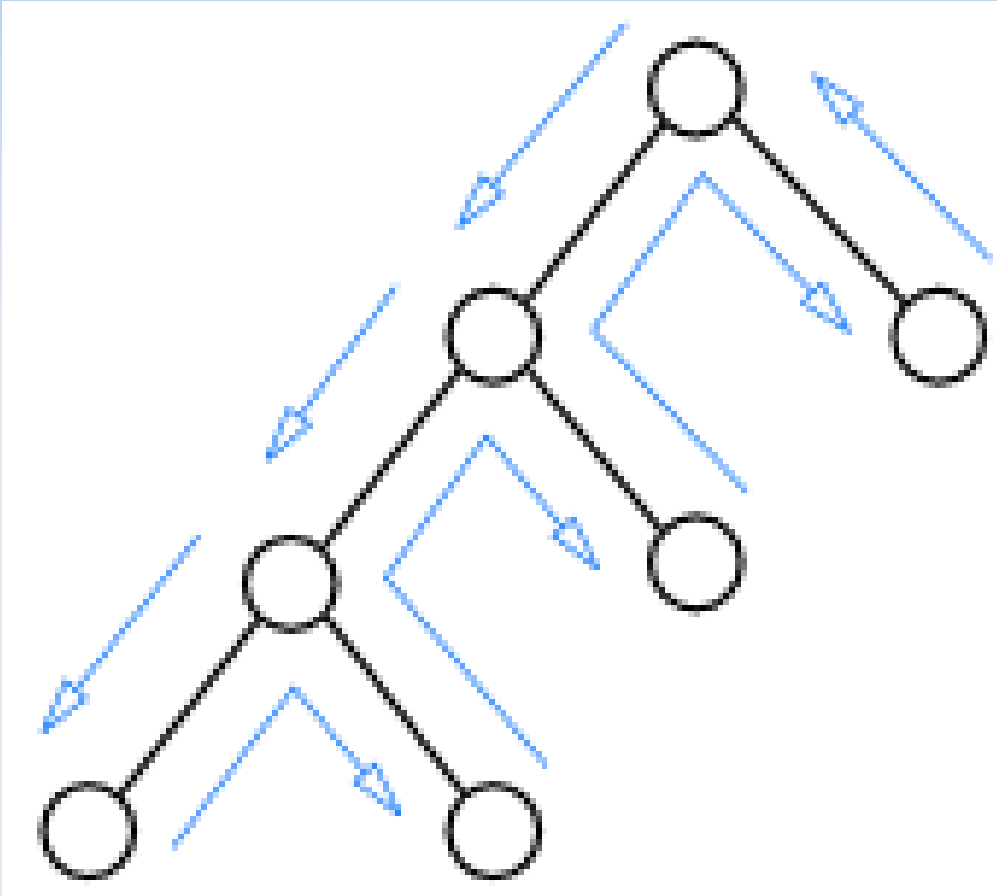
Strategien

- Heuristiken: Sortierung der Variablen
 - nach *Anzahl* beteiligter *Constraints*
 - einmal zu Beginn
 - Fail-First-Heuristik
 - nach *Kardinalität* der *Domains*
 - in jeder Rekursion der Tiefensuche
 - Success-First-Heuristik
 - Kombinierte Strategie

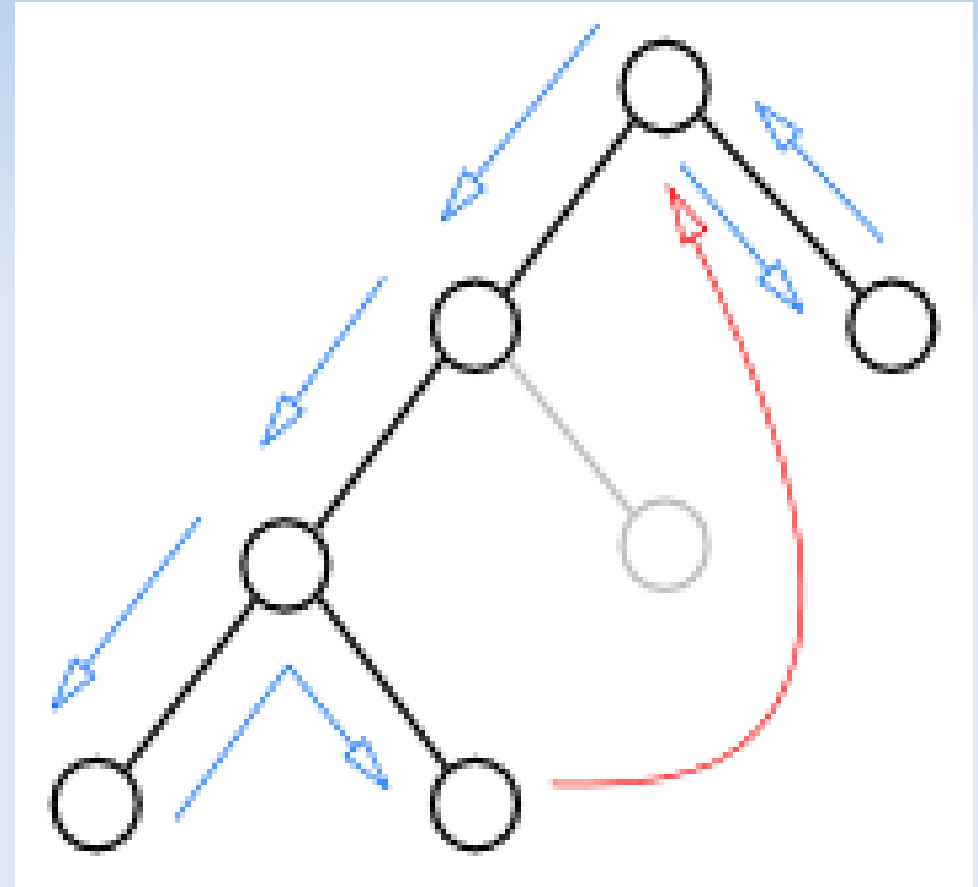
Strategien II

- Look-back-Strategien
 - Vermeiden wiederholtes Testen (Backmarking)
 - und/oder Thrashing (Backjumping)

Backjumping

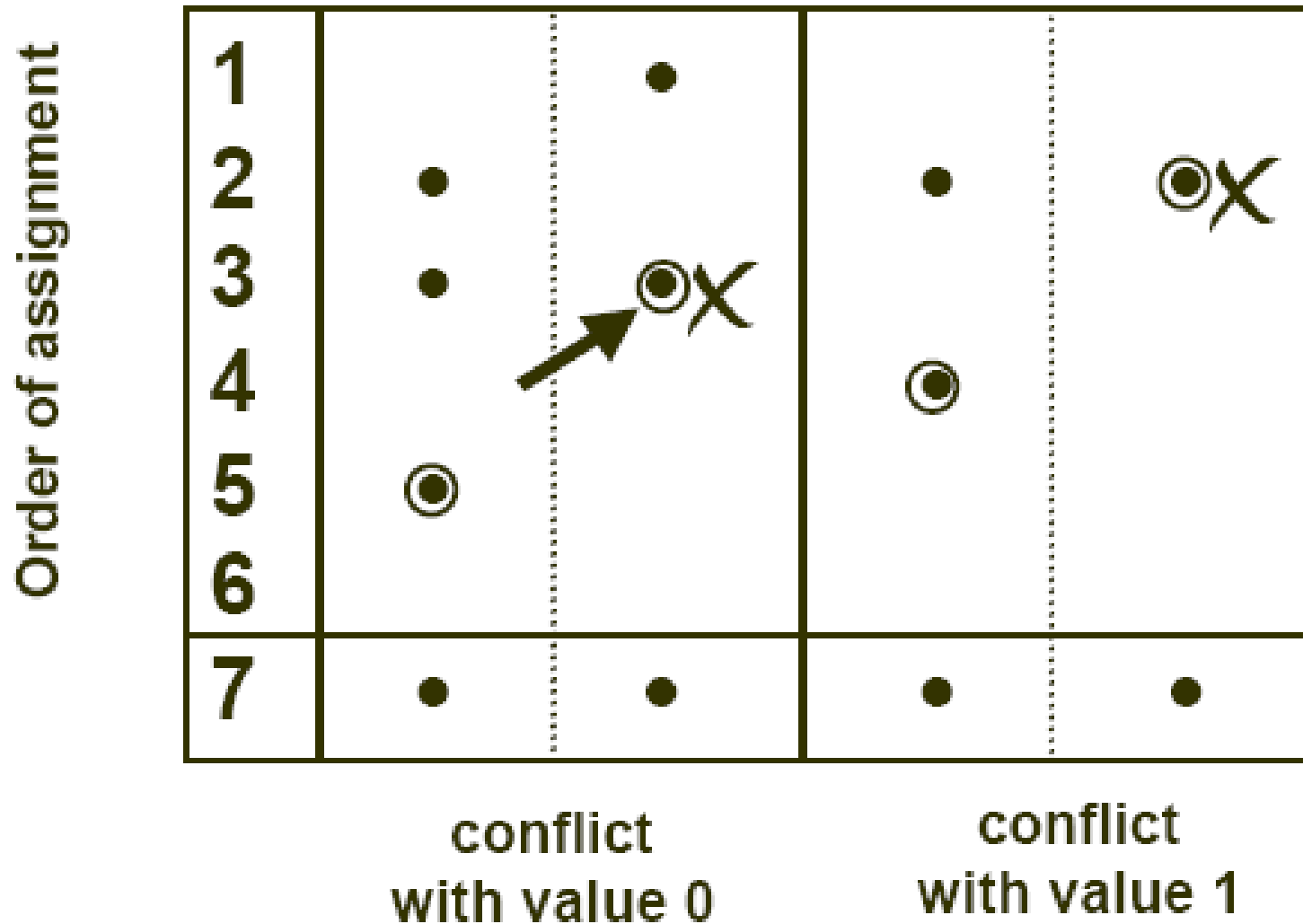


Backtracking



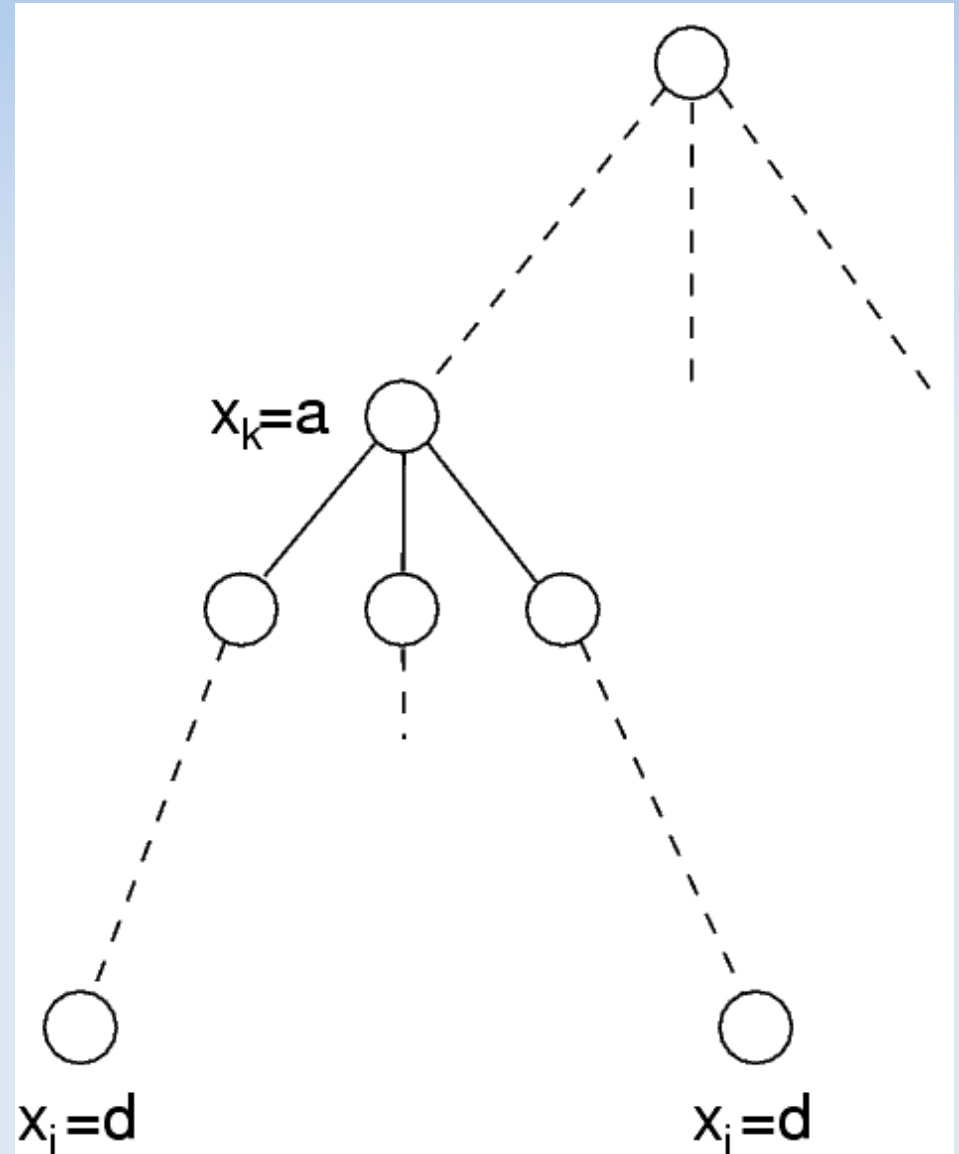
Backjumping

Sicherer Rücksprung

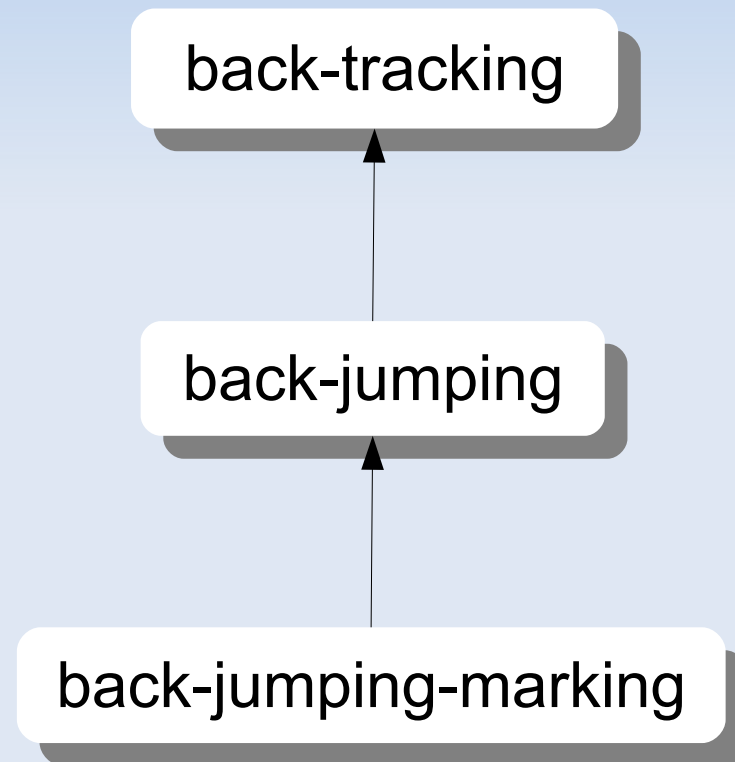


Backmarking

- vermeidet unnötiges Testen
- eigene Datenstrukturen
- funktioniert nicht mit dynamischer Variablensortierung



Look-Back-Strategien



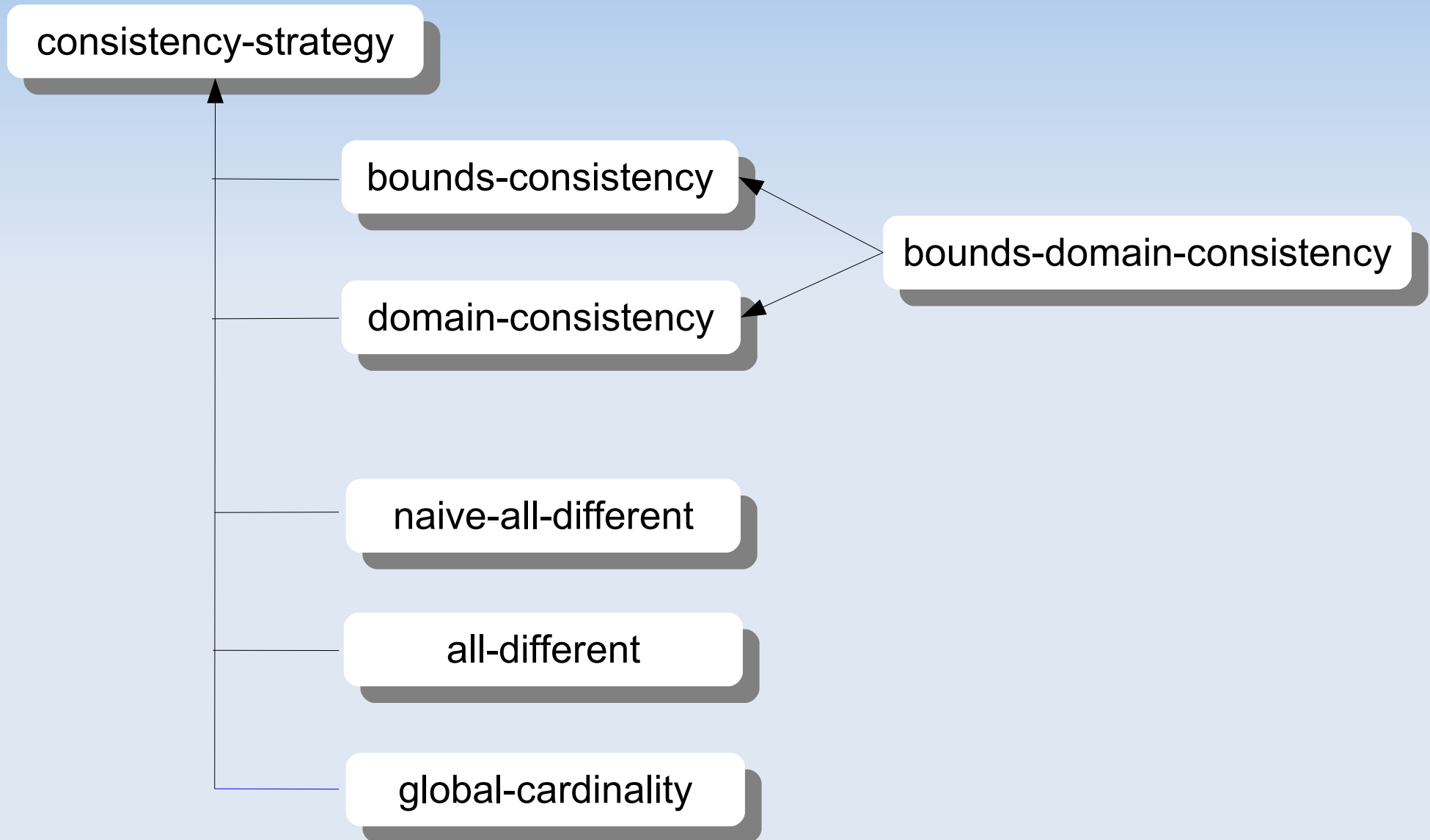
Strategien III

- Look-Ahead-Strategien
 - Bei jeder Instanziierung einer Variablen
 - Forward Checking
 - Herstellung der (teilweisen) lokalen Konsistenz nur für die Constraints, an denen die instanziierte Variable direkt beteiligt ist
 - AC3
 - Propagation der lokalen Konsistenz bis zum Fixpunkt
 - Verwaltung der zu testenden Constraints in Queue

Strategien III

- Consistency-Strategien
 - Stellt lokale Konsistenz her
 - Attribut eines Constraints
 - Zusätzlich Default/Fallback-Strategie
 - › Zur Laufzeit änderbar

Consistency-Strategien



Domain-Konsistenz

- Input: Constraint mit assoziierten Domains
- Lösungsbaum für jeden Wert jeder Domain
- Abbruch bei erster gültiger Lösung und Belassen des Wertes
- Andernfalls Entfernen des Wertes aus Domain
- Bei leerer Domain Backtracking nötig

Grenzenkonsistenz

- Schränkt nur die Grenzen von Domains ein
- Implementiert für lineare / multiplikative Constraints
- Koeffizienten werden als Liste, Konstanten als Variable k in Slots zusammengefasst
- $\text{koeff}_1 x_1 + \dots + \text{koeff}_n x_n = k$
- $x_1 = (k - \text{koeff}_2 x_2 - \dots - \text{koeff}_n x_n) / \text{koeff}_1$
- Untere Schranke: negativer Koeffizient -> kleinsten Wert aus Domain einsetzen

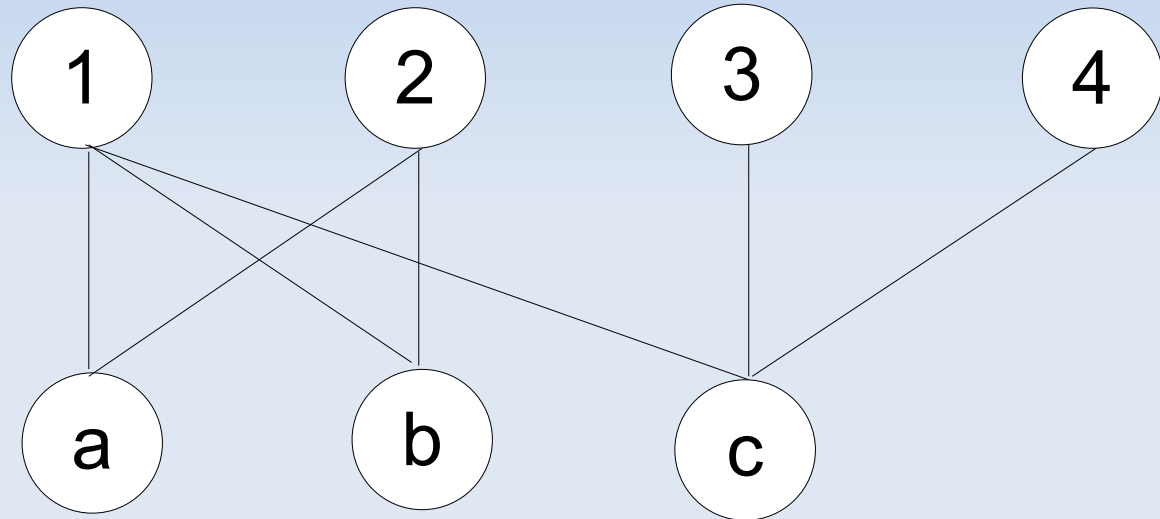
Naive All-Different

- Entfernen des neu instanziierten Wertes aus den Domains aller anderen am Constraint beteiligten Variablen

Graph-All-Different

- Algorithmus nach Hopcroft-Carp

Werte aus den
Domains



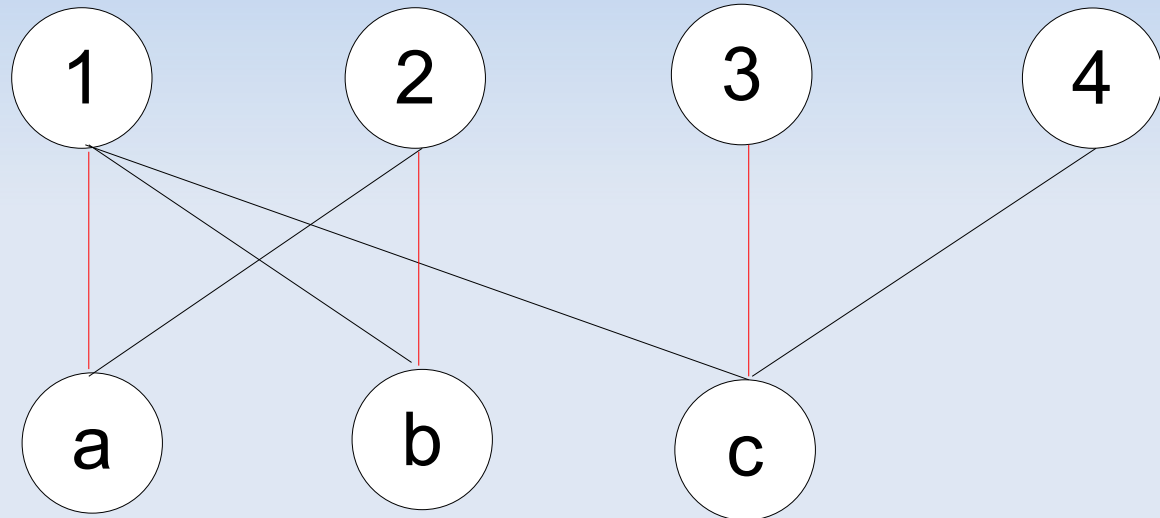
Variablen

Bipartiter Graph!

Graph-All-Different

- Algorithmus nach Hopcroft-Carp

Werte aus den
Domains



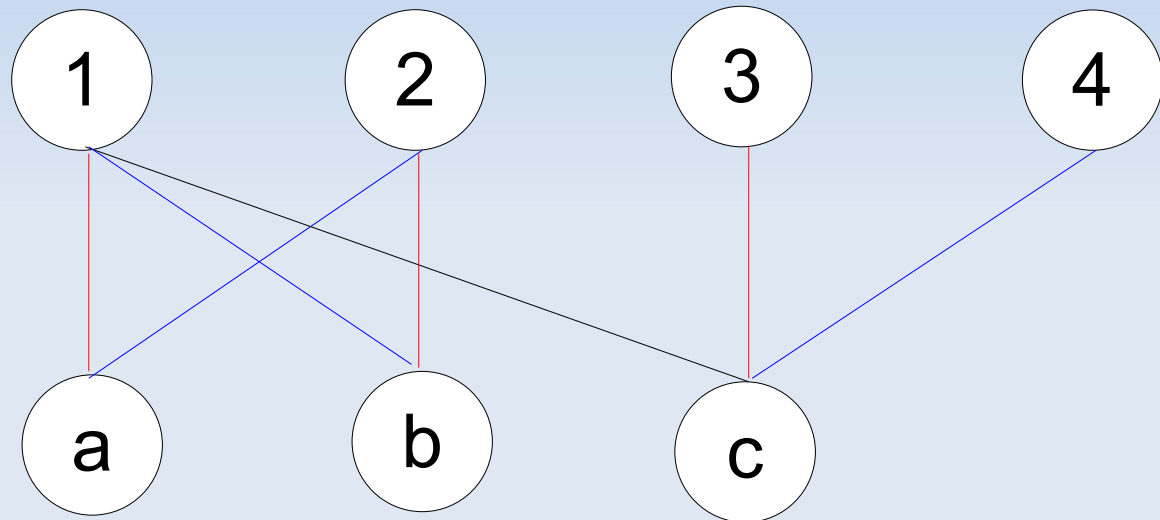
Variablen

Matching: Untergraph, bei dem ein Knoten höchstens zu einer Kante gehört
maximales Matching: kein anderes Matching enthält mehr Variablen

Graph-All-Different

- Algorithmus nach Hopcroft-Carp

Werte aus den
Domains



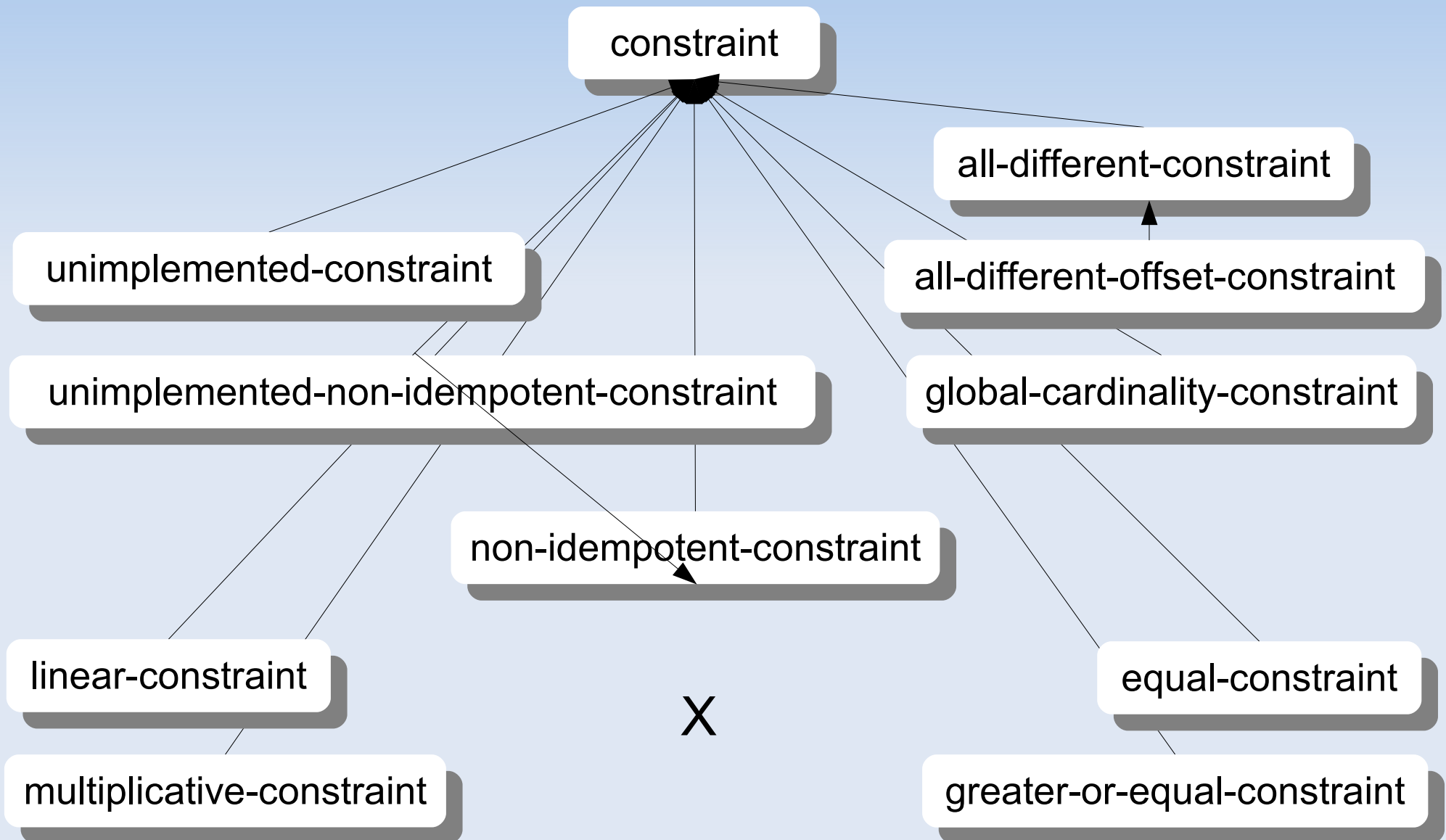
Variablen

Maximale Matchings können durch alternierende Pfade
ineinander überführt werden
Ungematchte Kanten werden entfernt

Global-Cardinality

- „ein bestimmter Wert darf höchstens n -mal in der Lösung vorkommen“
- Zwei Graphen für minimale und maximale Kardinalität
- Werteknoten werden entsprechend der Kardinalität dupliziert

Die Constraint-Klasse



Makros

- Parsen eines *Lambda-Ausdrucks*
 - Unterscheidung zwischen *linear* / *multiplikativ*
 - keines von beiden: Constraint mit Domain-Consistency als Consistency-Strategie
 - Idempotenz muss spezifiziert werden
 - Koeffizientenberechnung für lineare Constraints
 - $a * x \geq (b * y) * (c * z)$ wird umgewandelt in $(-1 * b * y) * (c * z) \geq -1 * a * x$
 - \leq / $<$ - Constraints werden in \geq / $>$ umgewandelt
- Codegenerierung für Closures
 - *All-different* / *Global-Cardinality* -Constraints

Binarisierung

- Nur für *globale Constraints* implementiert
- Es werden $\langle \text{Variablenanzahl} \rangle$ über 2 neue Constraints generiert
- Enorme Geschwindigkeitsvorteile, wenn keine effiziente Consistency-Strategie vorhanden und man von Look-back-Strategien profitieren will

Quellen

- <http://en.wikipedia.org/wiki/Backjumping>
- <http://en.wikipedia.org/wiki/Backmarking>
- Roman Barták: **On-Line Guide To Constraint Programming** Charles Universät Prag
<http://ktiml.mff.cuni.cz/~bartak/constraints>
- Wolfgang Runte: **Ein hybrides Framework für Constraint-Solver**. Diplomarbeit Informatik an der Universität Bremen. 27. Januar 2006.
<http://www.tzi.de/~woru/pub/diplom/html/Diplom.html>