



# YACS: Ein hybrides Framework für Constraint-Solver zur Unterstützung wissensbasierter Konfigurierung

*DA-III - Wolfgang Runte*

Fachbereich Mathematik / Informatik  
Universität Bremen

5. April 2006

# Übersicht

- ▶ Motivation
- ▶ Problemstellung
- ▶ Lösungsansatz
- ▶ Evaluierung
- ▶ Zusammenfassung

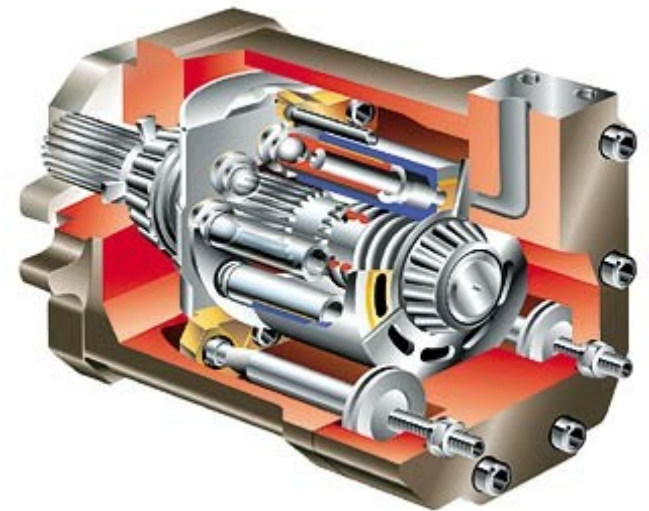
Motivation

# Konfigurierung (1)

- ▶ Beherrschung komplexer Systeme
- ▶ Fehlerminimierung durch konsistente Lösungen
- ▶ Beschleunigung der Angebotserstellung
- ▶ höhere Qualität der Lösungen

Beispiele:

- Antriebssysteme
- Gebäude
- Küchen
- PCs
- ...



Motivation

# Konfigurierung (2)

Konfigurieren ist das Zusammenfügen von Einzelkomponenten in einer Sequenz von einzelnen Konfigurierungsschritten zu einer Gesamtlösung (*Konfiguration*).

Gekennzeichnet durch:

- ▶ großer Lösungsraum bei variantenreichen Produkten
- ▶ Rücknahme von Entscheidungen
- ▶ Behandlung von komplexen Abhängigkeiten

Motivation

# Konfigurierung (2)

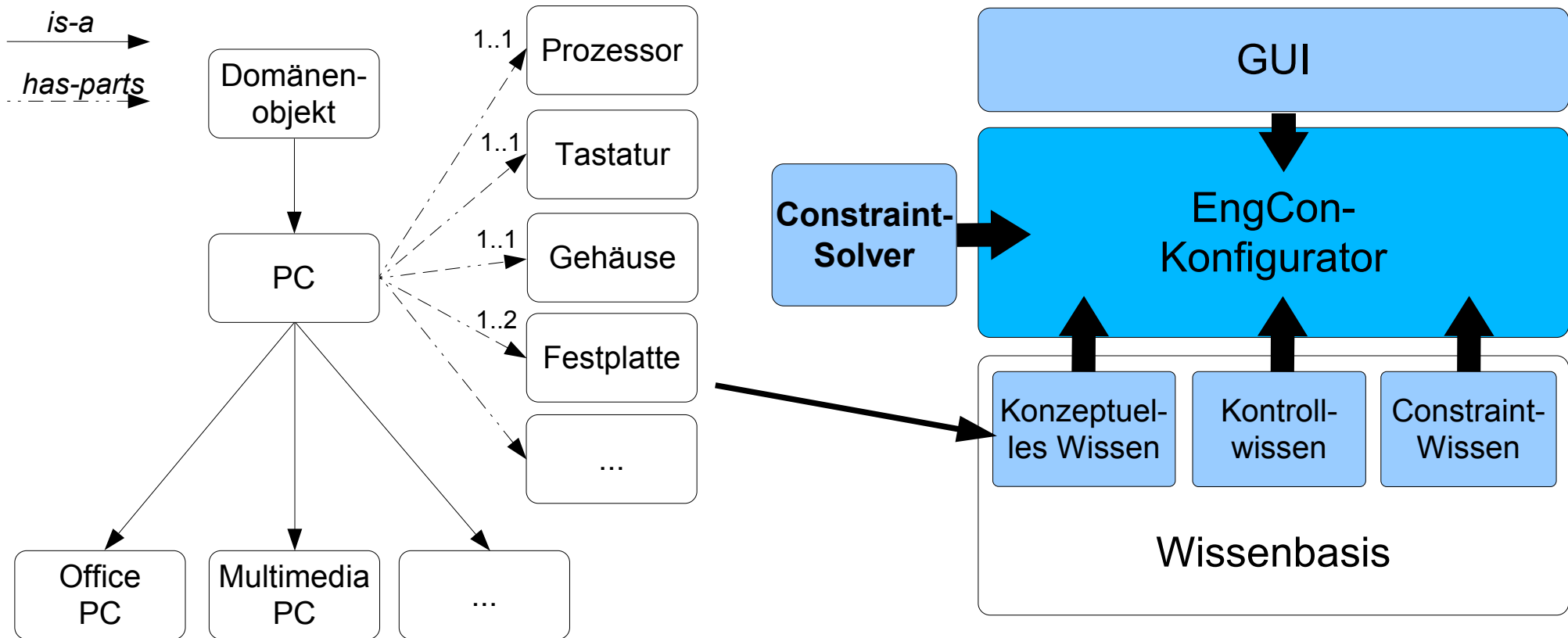
Konfigurieren ist das Zusammenfügen von Einzelkomponenten in einer Sequenz von einzelnen Konfigurierungsschritten zu einer Gesamtlösung (*Konfiguration*).

Gekennzeichnet durch:

- ▶ großer Lösungsraum bei variantenreichen Produkten
- ▶ Rücknahme von Entscheidungen
- ▶ **Behandlung von komplexen Abhängigkeiten**

Motivation

# Konfigurierung in EngCon



Motivation

# Constraint-Wissen

- ▶ repräsentiert Konfigurierungsrestriktionen zwischen Konzepten und Parametern der Begriffshierarchie
- ▶ Sicherstellung der Konsistenz der Konfiguration
- ▶ Propagation von Änderungen in einem Constraint-Netz
- ▶ 3-stufiges Constraint-Modell:
  - Konzeptuelle Constraints
  - Constraint-Relationen
  - Constraint-Netz

# Übersicht

- ▶ Motivation
- ▶ **Problemstellung**
- ▶ Lösungsansatz
- ▶ Evaluierung
- ▶ Zusammenfassung



Problemstellung  
**Problem**

- ▶ Algebraische Constraints (bzw. Funktions-Constraints) werden in EngCon über einen externen Constraint-Solver propagiert (*tolerance propagation*).
- ▶ Eine Eigenlösung mit hoher Modularität hat den Vorteil der besseren Erweiterbarkeit, z.B.:
  - Constraint-Hierarchien
  - Constraint-Relaxierung
  - ...
- ▶ Die Effizienz von Constraint-Lösungsverfahren ist abhängig von der Problemstellung (Topologie des Constraint-Netzes).

Problemstellung

# Anforderungen an die Problemlösung (1)

Funktionale Anforderungen:

- ▶ Berücksichtigung bestehender Schnittstellen und Beibehaltung der Trennung von Kontrolle und Constraint-System (*Black Box*)
- ▶ Propagation eines inkrementell anwachsenden Constraint-Netzes
- ▶ Verarbeitung von algebraischen Constraints mit finiten und unendlichen Domänen (inkl. Intervallarithmetik mit hohem Präzisionsgrad)
- ▶ Lösungsverfahren für möglichst „beliebige“ algebraische Constraint-Ausdrücke (numerisch, symbolisch, n-är, nichtlinear, zyklisch)
- ▶ Möglichkeit für Kompromiss zwischen Präzision und Effizienz

Problemstellung

# Anforderungen an die Problemlösung (2)

Nichtfunktionale Anforderungen:

- ▶ akzeptables Antwortverhalten
- ▶ Schnittstelle möglichst in Java (alternativ C/C++)
- ▶ Verfügbarkeit für MS Windows

Problemstellung

# Zielsetzung

- ▶ Entwicklung eines **objektorientierten Frameworks** zur flexiblen Anbindung unterschiedlicher Constraint-Solver resp. Constraint-Lösungsverfahren.
- ▶ Berücksichtigung von Constraints über **finite** und **infinite** Domänen.
- ▶ Vor dem Hintergrund der Konfigurierung ist jeweils abzuwägen, **welcher Solver** für **welche Constraints** zum Einsatz kommt.
- ▶ Ablaufsteuerung, Lösungsverfahren und Schnittstellen sehen **inkrementelle** Constraint-Verarbeitung vor.

# Übersicht

- ▶ Motivation
- ▶ Problemstellung
- ▶ **Lösungsansatz**
  - Analyse bestehender Verfahren
  - Konzeption eines *Frameworks*
- ▶ Evaluierung
- ▶ Zusammenfassung

## Lösungsansatz (Analyse)

# Constraint-Lösungsverfahren (1)

### Klassische Constraint Satisfaction Probleme (CSP)

- Konsistenzverfahren:
  - *node consistency* (NC), *arc consistency* (AC), *path consistency* (PC), *k-consistency* (vgl. Montanari '74; Walz '75; Mackworth '77; Freuder '78; ...)
  - *(i,j)-consistency*, *inverse consistency*, *path inverse consistency*, *neighborhood inverse consistency* (vgl. Freuder '85, Freuder u. Elfe '96, Debruyne '00)
  - *lazy arc consistency* (LAC) (vgl. Schiex et al. '96)
  - *directional arc consistency* (DAC), *directional path consistency* (DPC), *adaptive consistency* (vgl. Dechter u. Pearl '87)
  - *restricted path consistency* (RPC) (vgl. Berlandier '95; Debruyne u. Bessière '97a; Debruyne '98)
  - *singleton consistency* (vgl. Debruyne u. Bessière '97b; Prosser et al. '00)
- Suchverfahren:
  - allgemeine Suchstrategien: *generate & test* (GT), chronologisches *backtracking* (BT) (vgl. Bittner u. Reingold '75; ...)
  - *look-back*-Strategien: *backjumping* (BJ), *graph-based backjumping* (GBJ), *conflict-directed backjumping* (CBJ), *backchecking* (BC), *backmarking* (BM) (vgl. Gaschnig '79; Haralick u. Elliot '80; Dechter '90a; Prosser '93b; Dechter u. Frost '02; ...)
  - *look-ahead*-Strategien: *forward checking* (FC), *partial look-ahead* (PLA), *full look-ahead* (FLA), *maintaining arc consistency* (MAC) (vgl. Haralick u. Elliot '80; Sabin u. Freuder '94a,b; Grant u. Smith '96; Frost u. Dechter '96a,b; Bessière u. Régim '96; Sabin u. Freuder '97; Gent u. Prosser '00; ...)
  - hybride Verfahren: BMJ, BM-CBJ, FC-BM, FC-CBJ, FC-BM-CBJ, MAC-CBJ (vgl. Bessière u. Régim '96; Chen u. van Beek '01; Grant u. Smith '96; Prosser 93a,b; Prosser 95a,b, ...)
- Ordnungsheuristiken:
  - Variablenordnungsheuristiken: *fail first*, *minimal width ordering*, *maximum cardinality ordering*, *maximum degree ordering*, *minimal bandwidth ordering* (vgl. Haralick u. Elliot '80; Dechter u. Meiri '94; Frost u. Dechter '94, '95; ...)
  - Werteordnungsheuristiken: *min-conflicts*, *max-domain-size*, *weighted-max-domain-size*, *point-domain-size* (vgl. Minton et al. '92; Frost u. Dechter '95; ...)

Lösungsansatz (Analyse)

# Constraint-Lösungsverfahren (2)

## Intervall Constraint Satisfaction Probleme (ICSP)

- *interval splitting* (vgl. Cleary '87)
- *label inference* (vgl. Davis '87)
- *tolerance propagation* (vgl. Hyvönen '89; Hyvönen '91; Hyvönen '92)
- *2B-, 3B-, kB-consistency / hull consistency* (vgl. Lhomme '93; Lhomme et al. '96, '98; Bordeaux et al. '01; Lebbah u. Lhomme '98, '02; ...)
- *box consistency* (vgl. Benhamou et al. '94; Van Hentenryck et al. '95, '97; Puget u. Van Hentenryck '96,'98; Collavizza et al. '98, '99; Benhamou et al. '99a; Granvilliers et al. '99; ...)
- *2k-trees* (vgl. Haroud u. Faltings '94; Sam-Haroud '95, Sam-Haroud u. Faltings '96a,b; ...)

## Lösungsansatz (Analyse)

# Constraint-Systeme

	FD	IN	RW	IS	RE	HO	NL	IK	KO	QC	SP	MS
Cassowary	-	-	X	X	X	X	-	X	-	X	Java	X
ILOG Solver	X	X	(X)	X	X	X	X	-	X	-	C++	X
ILOG JSolver	X	-	-	X	X	-	-	X	X	-	Java	X
UniCalc	-	X	-	X	X	X	X	-	X	-	C	X
ALIAS	-	X	-	X	X	X	X	-	-	-	C++	-
RealPaver	-	X	-	X	X	X	X	-	-	X	C++	X
C-Lib	X	-	-	-	X	-	-	-	-	X	C	X
JCL	X	-	-	X	X	-	-	-	-	X	Java	X
DJ	X	-	-	X	X	X	-	-	-	-	Java	X
KCS	X	-	-	X	-	X	-	-	X	-	Java	X
J.CP	X	-	-	X	-	X	-	X	-	-	Java	X
IASolver	-	X	-	X	X	X	X	-	-	X	Java	X

FD : finite Domänen  
 IN : reellwertige Intervalldomänen  
 RW : reellwertige Domänen  
 IS : intensionale Constraints  
 RE : beliebige Relationen möglich  
 HO :  $n$ -äre Constraints

NL : nichtlineare Constraints  
 IK : Inkrementalität  
 KO : kommerzielle Bibliothek  
 QC : Quellcode verfügbar  
 SP : Sprache der Schnittstelle  
 MS : für Microsoft Windows verfügbar



Lösungsansatz (Konzeption)

# Flexibilität

► Flexibilität durch strategiebasierte Constraint-Verarbeitung.

► Einteilung des Lösungsprozesses in Phasen:

1. Preprozessing
2. Konsistenzherstellung
3. Lösungssuche

1	Preprozessing
2	Konsistenzherstellung
3	Lösungssuche

► Beispiele für Constraint-Lösungsstrategien:

1	-
2	Kantenkonsistenz
3	Forward Checking

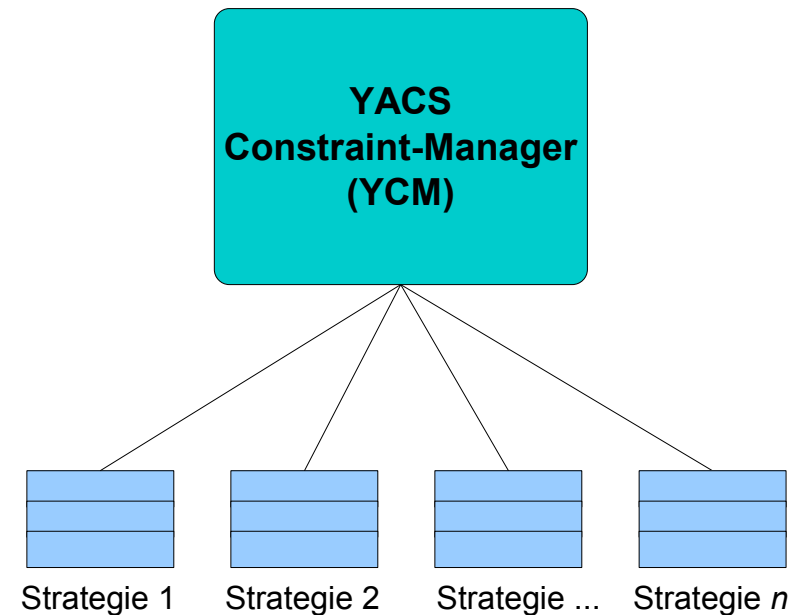
1	Binärisierung
2	(1) Knotenkonsistenz (2) Kantenkonsistenz
3	konfliktbasiertes Backjumping

1	Zerlegung in primitive Constraints
2	Hull-Konsistenz
3	-

Lösungsansatz (Konzeption)

# Abstraktion durch Strategien (1)

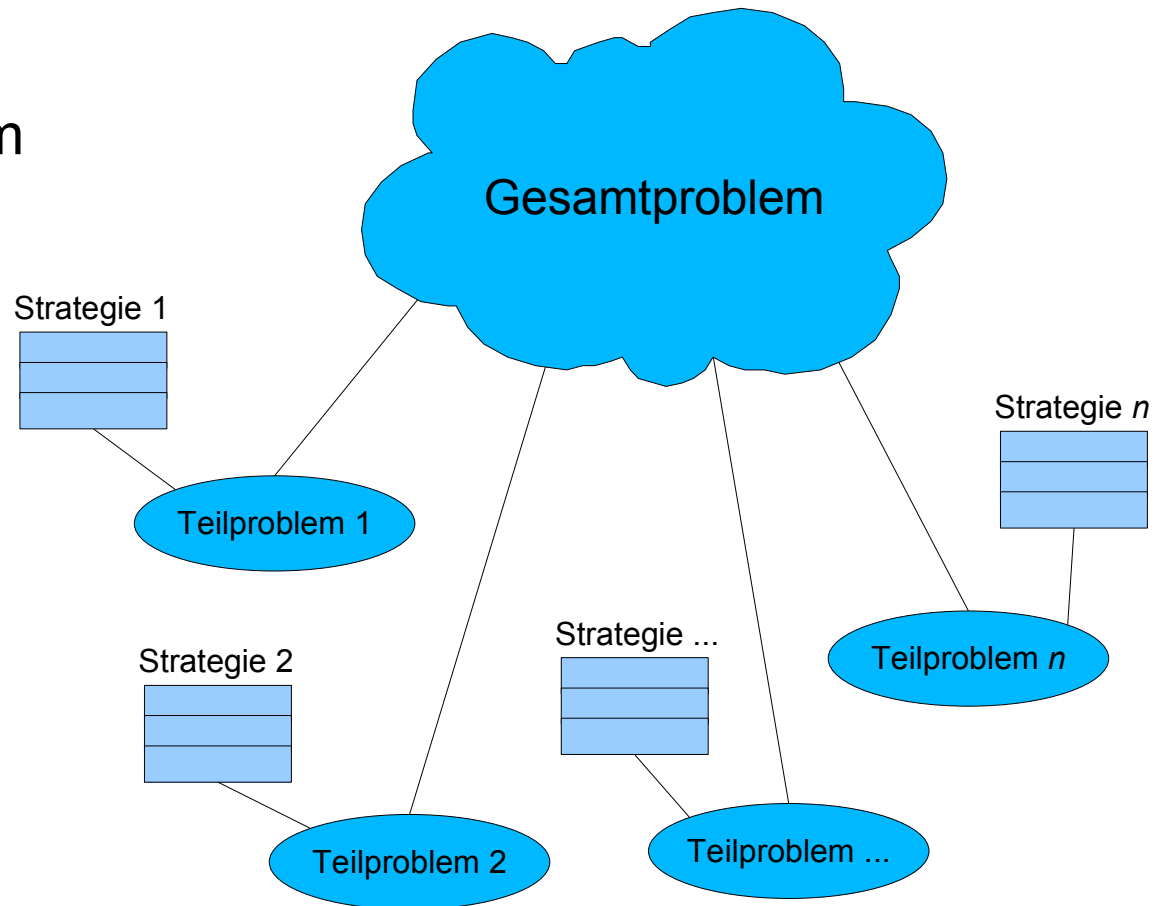
- ▶ *Constraint-Manager* verwaltet und steuert, welche Constraints von welcher Strategie verarbeitet werden sollen (*phasenweise*).
- ▶ Es müssen keine Lösungsalgorithmen sondern *Lösungsstrategien* zur Constraint-Verarbeitung ausgewählt werden.
- ▶ Einfache Austauschbarkeit von Lösungsverfahren ist gewährleistet.



Lösungsansatz (Konzeption)

# Abstraktion durch Strategien (2)

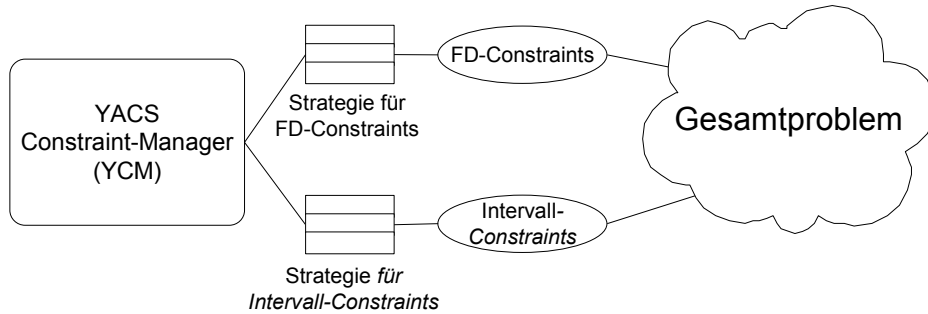
- ▶ Jedes Constraint wird vom Wissensingenieur mit einer geeigneten Lösungsstrategie assoziiert.
- ▶ Zugehörigkeit einzelner Constraints zu Lösungsstrategien führt zur Bildung von Teilproblemen.



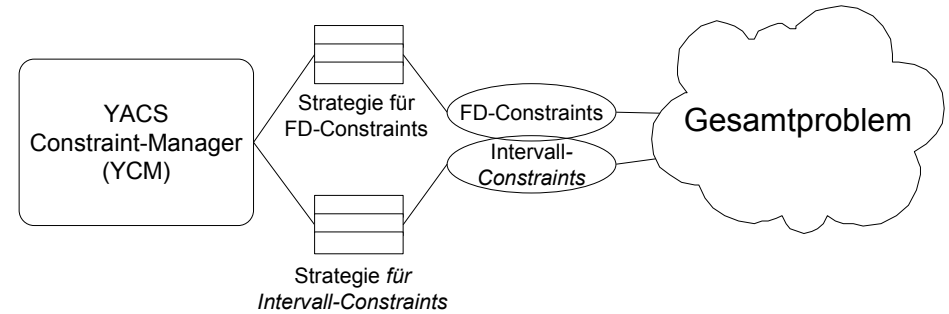
Lösungsansatz (Konzeption)

# Hybridizität versus Heterogenität

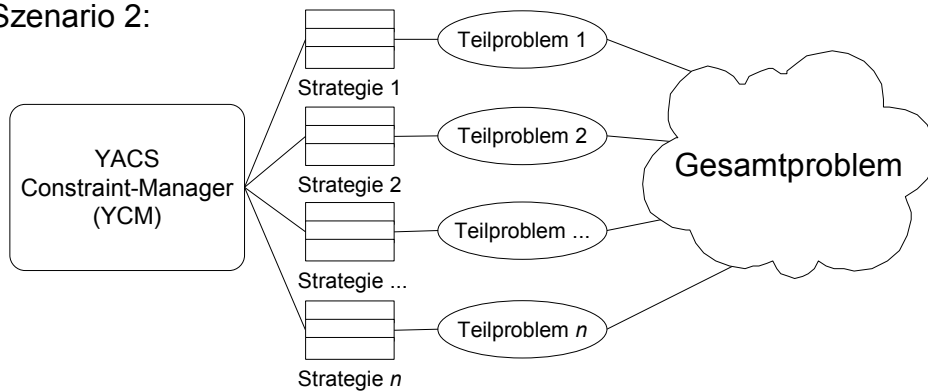
Szenario 1:



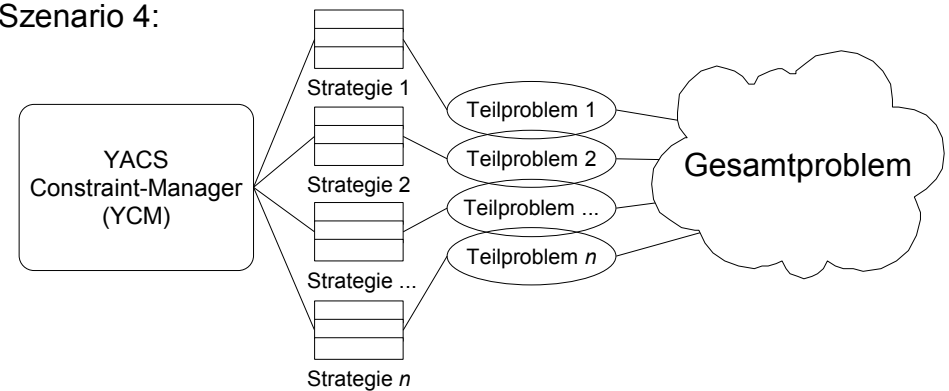
Szenario 3:



Szenario 2:



Szenario 4:

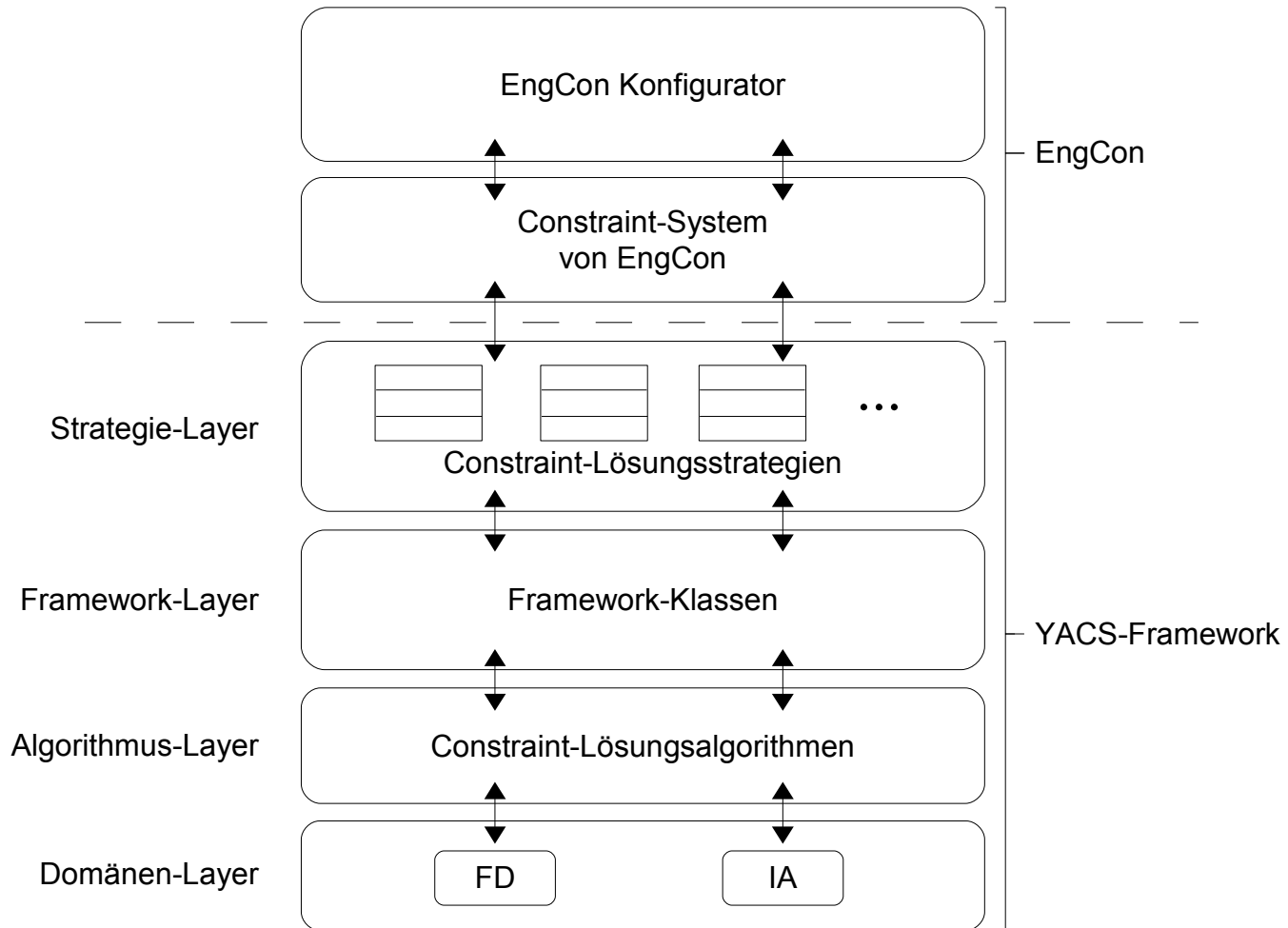


Lösungsansatz (Konzeption)

# Modularität

- ▶ Modularität durch *objektorientierte Framework-Architektur*.
- ▶ Objektorientierte Frameworks: Steigerung der Wiederverwendbarkeit durch OOP-Techniken. (Johnson '97a, b; Gamma et al. '96)
- ▶ Code- und Designwiederverwendung      Frameworks = (Components + Patterns)
- ▶ Spezialisierung abstrakter Klassen für konkrete Anwendungen (Constraint-Solver, Variablen, Domänen, etc.) → Erweiterbarkeit.
- ▶ Allgemeiner Kontrollzyklus wird durch das Framework vorgegeben.
- ▶ Einheitliche Schnittstellen erlauben flexiblen Austausch von Constraint-Lösungsverfahren.

# Lösungsansatz (Konzeption) Systemarchitektur



# Übersicht

- ▶ Motivation
- ▶ Problemstellung
- ▶ Lösungsansatz
- ▶ **Evaluierung**
- ▶ Zusammenfassung

## Evaluierung Umsetzung

- ▶ JAVA-Implementierung: „YACS“ (*Yet Another Constraint Solver*)
- ▶ Prototypische Integration in *EngCon V0*:
  - Propagation des Constraint-Netzes während der Konfigurierung möglich (finite/infinite Domänen)
  - minimale Anpassung vorhandener Constraints (Erweiterung um den Namen der jeweiligen Constraint-Lösungsstrategie)
- ▶ Implementierung einer Reihe von *synthetischen* Problemstellungen zu Test- und Demonstrationszwecken (*YacsTester*):
  - Testen von Constraint-Lösungsverfahren
  - Funktionalität von YACS aufzeigen



## Evaluierung Prototyp

### ► Prototyp „YACS“:

- unterstützt inkrementell anwachsendes Constraint-Netz
- erlaubt teilproblemübergreifende Metapropagation (ausschließlich innerhalb derselben Domäne)
- beinhaltet eine modulare Bibliothek von Lösungsalgorithmen (NC, AC, BT, MAC, Hull-Konsistenz, Werteordnungsheuristik *dom/deg*)
- stringbasierte Schnittstelle für Constraints (JLex/CUP-Parser)
- Constraint-Lösungsstrategien lassen sich unabhängig vom Programmcode innerhalb einer XML-Datei verwalten
- im Internet verfügbar (LGPL):  
<http://www.sourceforge.net/projects/constraints>

# Übersicht

- ▶ Motivation
- ▶ Problemstellung
- ▶ Lösungsansatz
- ▶ Evaluierung
- ▶ **Zusammenfassung**

# Zusammenfassung

- ▶ Ersetzung des externen Constraint-Solvers für algebraische Constraints des strukturbasierten Konfigurierungswerkzeugs EngCon.
- ▶ Entwicklung eines *objektorientierten Frameworks* für Constraint-Solver für den problemabhängigen und anwendungsspezifischen Einsatz von Constraint-Lösungsmethoden (auch außerhalb von EngCon).
- ▶ Unterstützung von Constraints über finite und infinite Domänen:
  - endliche & diskrete Wertebereiche
  - unendliche & kontinuierliche Wertebereiche (reellwertige Intervalle)
- ▶ *Modularität* durch objektorientierte Framework-Architektur (schlanke, einheitliche Schnittstellen, Erweiterbarkeit).
- ▶ *Flexibilität* durch strategiebasierten Ansatz (einfache Austauschbarkeit der Constraint-Lösungsverfahren, phasenweiser Lösungsprozess).

# Diskussion

# Literatur (1)

- ▶ **Benhamou et al. 1994** Benhamou, Frédéric ; McAllester, David ; Van Hentenryck, Pascal: CLP(Intervals) Revisited. In: Bruynooghe, Maurice (Hrsg.): Logic Programming, Proceedings of the 1994 International Symposium (ILPS'94), Ithaca, New York, USA, 13.–17. November 1994. Cambridge, Massachusetts, USA : The MIT Press, 1994, S. 124–138. – ISBN 0-262-52191-1
- ▶ **Cleary 1987** Cleary, John G.: Logical Arithmetic. In: Future Computing Systems 2 (1987), Nr. 2, S. 125–149. – ISSN 0266-7207
- ▶ **Davis 1987** Davis, Ernest: Constraint Propagation with Interval Labels. In: Artificial Intelligence 32 (1987), Juli, Nr. 3, S. 281–331. – ISSN 0004-3702
- ▶ **Dechter und Frost 2002** Dechter, Rina ; Frost, Daniel: Backjump-based Backtracking for Constraint Satisfaction Problems. In: Artificial Intelligence 136 (2002), April, Nr. 2, S. 147–188. – ISSN 0004-3702
- ▶ **Freuder 1978** Freuder, Eugene C.: Synthesizing Constraint Expressions. In: Communications of the ACM (CACM) 21 (1978), November, Nr. 11, S. 958–966. – ISSN 0001-0782

# Literatur (2)

- ▶ **Gamma et al. 1996** Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: Entwurfsmuster – Elemente wiederverwendbarer objektorientierter Software. 1. Aufl. München : Addison-Wesley, 1996. – xx + 479 S. – ISBN 3-89319-950-0
- ▶ **Johnson 1997a** Johnson, Ralph E.: Components, Frameworks, Patterns. In: Harandi, Medhi (Hrsg.): Proceedings of the 1997 Symposium on Software Reusability (SSR'97), Boston, Massachusetts, USA, 17.–20. Mai 1997. New York, NY, USA : ACM Press, 1997, S. 10–17. – ISBN 0-89791-945-9
- ▶ **Johnson 1997b** Johnson, Ralph E.: Frameworks = (Components + Patterns). In: Communications of the ACM (CACM) 40 (1997), Oktober, Nr. 10, S. 39–42. – ISSN 0001-0782
- ▶ **Haralick und Elliot 1980** Haralick, Robert M. ; Elliot, Gordon L.: Increasing Tree Search Efficiency for Constraint Satisfaction Problems. In: Artificial Intelligence 14 (1980), Oktober, Nr. 3, S. 263–313. – ISSN 0004-3702
- ▶ **Hyvönen 1992** Hyvönen, Eero: Constraint Reasoning Based on Interval Arithmetic: The Tolerance Propagation Approach. In: Artificial Intelligence 58 (1992), Dezember, Nr. 1–3, S. 71–112. – ISSN 0004-3702

# Literatur (3)

- ▶ **Lhomme 1993** Lhomme, Olivier: Consistency Techniques for Numeric CSPs. In: Bajcsy, Ruzena (Hrsg.): Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI'93), Chambéry, France, 28. August – 3. September 1993. San Mateo, California, USA : Morgan Kaufmann Publishers, 1993, S. 232–238. – ISBN 1-55860-300-X
- ▶ **Mackworth 1977** Mackworth, Alan K.: Consistency in Networks of Relations. In: Artificial Intelligence 8 (1977), Februar, Nr. 1, S. 99–118. – ISSN 0004-3702
- ▶ **Montanari 1974** Montanari, Ugo: Networks of Constraints: Fundamental Properties and Applications to Picture Processing. In: Information Sciences 7 (1974), S. 95–132. – ISSN 0020-0255
- ▶ **Sam-Haroud 1995** Sam-Haroud, Jamila: Constraint Consistency Techniques for Continuous Constraints. Lausanne, Switzerland, Swiss Federal Institute of Technology (EPFL), PhD Thesis No. 1423, 1995. – xviii + 178 S.
- ▶ **Sam-Haroud und Faltings 1996** Sam-Haroud, Djamilia ; Faltings, Boi V.: Consistency Techniques for Continuous Constraints. In: Constraints, An International Journal 1 (1996), September, Nr. 1–2, S. 85–118. – ISSN 1383-7133

# Literatur (4)

- ▶ **Waltz 1975** Waltz, David L.: Understanding Line Drawings of Scenes with Shadows. In: Winston, Patric Henry (Hrsg.): The Psychology of Computer Vision. New York, NY, USA : McGraw-Hill, 1975, S. 19–91. – ISBN 0-07-071048-1



# Methoden zur Konfigurierung

- ▶ regelbasiertes Konfigurieren
- ▶ strukturbasiertes Konfigurieren
- ▶ constraint-basiertes Konfigurieren
- ▶ ressourcenbasiertes Konfigurieren
- ▶ fallbasiertes Konfigurieren
- ▶ verhaltensbasiertes Konfigurieren

Verfahren sind selten in „Reinform“ anzutreffen – meistens Kombination mehrerer Methoden

# Konfigurierungssysteme

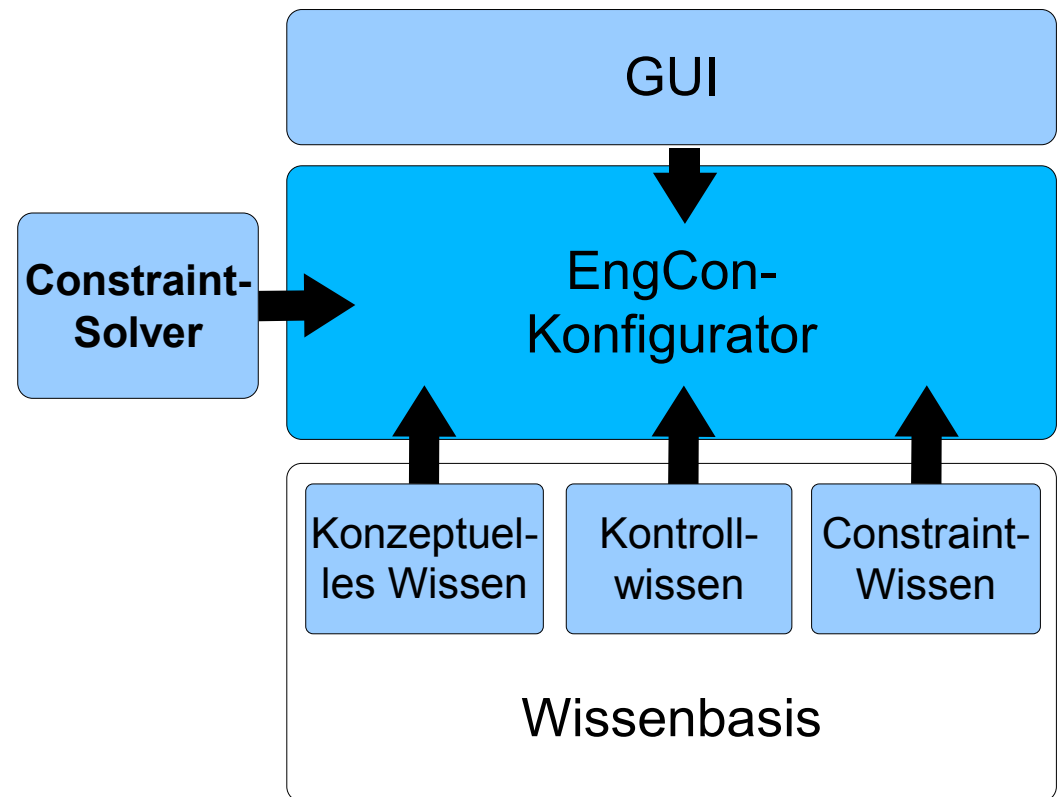
- ▶ R1/XCON
  - ▶ SICONFEX
  - ▶ MMC-CON
  - ▶ ALL-RISE
  
  - ▶ ConBaCon
  - ▶ CAWICOMS
  
  - ▶ PlaKon
  - ▶ KonWerk
  - ▶ EngCon
- ▶ CAS-Konfigurator
  - ▶ Cameleon EPOS
  - ▶ camos.Configurator
  - ▶ COMIX
  - ▶ COSMOS
  - ▶ KIKon
  - ▶ SAP SCE
  - ▶ Baan SalesPlus
  - ▶ Tacton Configurator
  - ▶ Lava
  - ▶ ILOG (J)Configurator

# EngCon: Historie

- ▶ Vorgänger: *Plakon*, *Konwerk*, entwickelt (u.a.) an der Universität Hamburg mit Unterstützung des BMBF
- ▶ prototypische Umsetzung auf moderne *JAVA*-Umgebung durch das TZI
- ▶ Überführung in ein Produkt durch die encoway GmbH: „Drive Solution Designer“ (DSD)
- ▶ Auszeichnung des DSD mit dem „Innovative Applications of Artificial Intelligence (IAAI) Award 2002“ der *American Association for Artificial Intelligence (AAAI)*
- ▶ Erfolgsmodell „Technologietransfer“

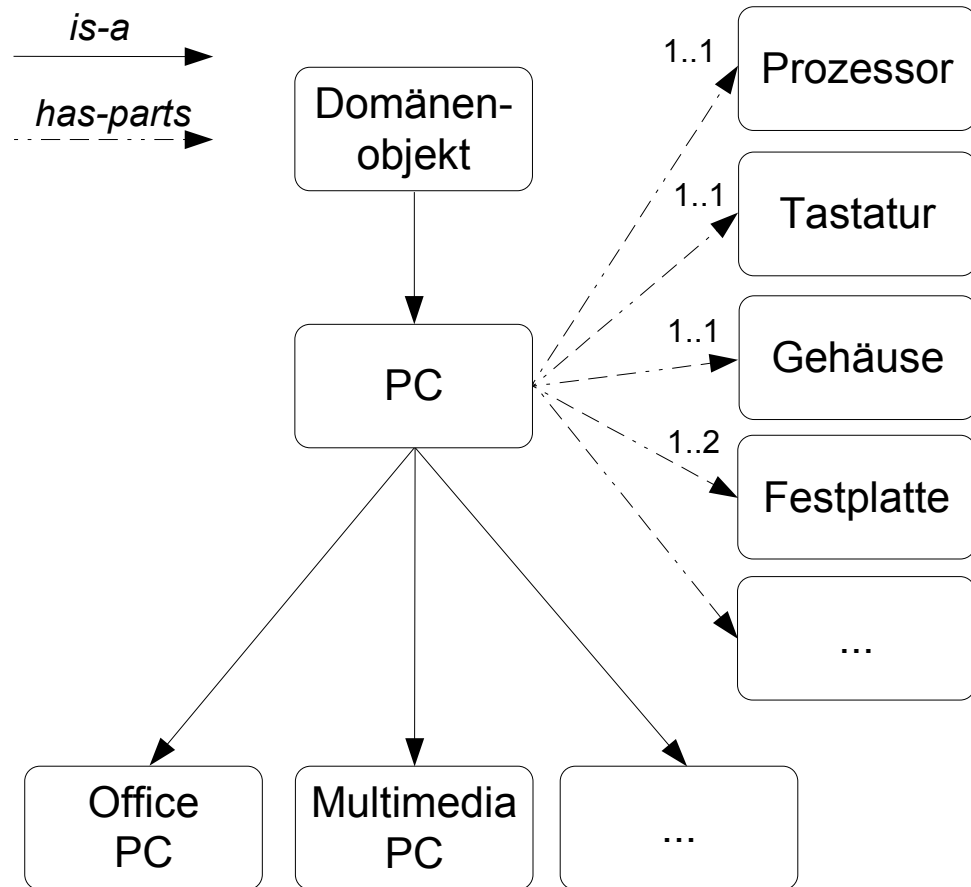
# EngCon: Architektur

- ▶ domänenunabhängiges, *strukturbasiertes* Konfigurierungswerkzeug
- ▶ Bildung von Inferenzen aufgrund der wissensbasierten (hybriden) Architektur des Systems
- ▶ inkrementeller, interaktiver Konfigurierungsverlauf, der zu *einer* Lösung führt (Tiefensuche)



# EngCon: Konzeptuelles Wissen

- ▶ Ontologie für die abstrakte Repräsentation der Struktur aller Lösungen des Konfigurierungsproblems
- ▶ *Closed-World-Assumption*
- ▶ Konzepte stehen über *is-a* und *has-parts* Beziehungen in Relation
- ▶ Spezialisierungshierarchie / Taxonomie
- ▶ Zerlegungshierarchie / Partonomie



# EngCon: Kontrollwissen

- ▶ Kontrollmechanismus „interpretiert“ die Begriffshierarchie (*Begriffshierarchie-orientierte Kontrolle*)
- ▶ agendabasierte Steuerung der Suche im Lösungsraum
- ▶ Unterteilung in Phasen mittels „Strategien“
- ▶ Kontrollzyklus:
  1. Analyse der aktuellen Teilkonfiguration
  2. Konfigurierungsschritt auswählen
  3. Bearbeitungsverfahren anwenden spezialisieren, zerlegen, parametrieren
  4. Propagation des Constraint-Netzes

# EngCon: Konzeptuelle Constraints

- ▶ Zuordnung der Constraint-Relationen zu den Instanzen des Konfigurierungsprozesses
- ▶ Instantiierungsregeln / Bindungsmuster in Form von *Variablen-Pattern-Paaren*
- ▶ *Ein Pattern-Matcher* überprüft für neue Konzept-Instanzen, ob ein Variablen-Pattern-Paar erfüllt wird und instantiiert ggf. die entsprechenden Constraint-Relationen.

```
(def-konzeptuelles-constraint
  :name                conc_AGP_Mainboard
  :variablen-pattern-paare ((?v :name VGA_Card
                               :parameter((Bus 'agp)))
                           (?m :name Mainboard))
  :constraint-aufrufe   ((func_AGP_Mainboard (?m AGP_Slot))))
```

# EngCon: Constraint-Relationen

▶ **Funktions-Constraints**

als (Un-)Gleichung formalisierte komplexe, funktionale Zusammenhänge

$$A = 150 * B$$

▶ **Extensionale Constraints / Tupel-Constraints**

Aufzählung von Tupel aller möglichen Wertebereiche (relationale Abhängigkeit)

M_FSB	P_FSB	S_FSB
66	66	66
66	66	100
66	66	133
100	100	100
100	100	133
133	133	133

▶ **Java-Constraints**

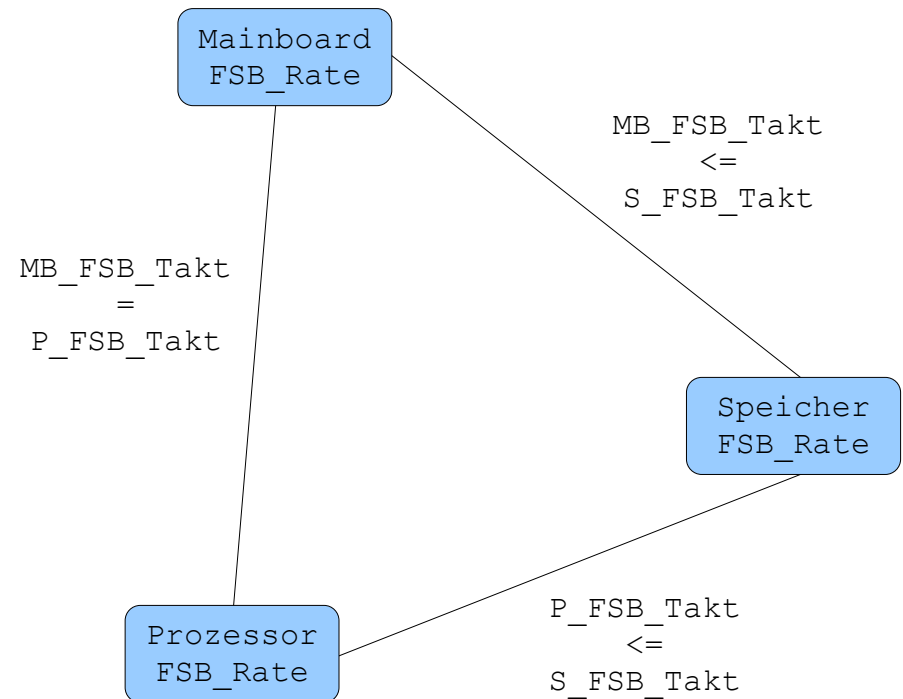
als *JAVA*-Methode implementierte „prozedurale Constraints“ für „beliebige“ Berechnungen

```
public static Vector setEqual(Vector a){
    if(a != null){
        if(a.size() != oldVector.size()){
            if(a.size() < oldVector.size())
                return a;
        }
    }
    return oldVector;
}
```



# EngCon: Constraint-Netz

- ▶ Inkrementeller Aufbau durch sukzessives instantiieren der Constraint-Relationen durch den *Pattern-Matcher*.
- ▶ Propagation der Wertebereiche der Constraint-Variablen bei jedem Konfigurierungszyklus.
- ▶ Sicherstellung der Konsistenz, d.h. die Domänen der Constraint-Variablen dürfen nur gültige Wertebereiche bzgl. der sie enthaltenen Constraints aufweisen.



# Eigenschaften von Constraints

- ▶ finite und infinite Domänen
- ▶ symbolische und numerische Wertebereiche
- ▶ Größe eines Problems (Anzahl Variablen/Constraints)
- ▶ Stelligkeit eines Constraints (unär, binär, ternär, ...,  $n$ -är)
- ▶ Struktur eines Constraint-Netzes:
  - Constraint-Dichte (*high density* vs. *low density*)
  - Beschränkungsgrad (*constrainedness*, *tightness*), wird durch die Lösungsdichte definiert (*loosely constrainedness* vs. *tightly constrainedness*)
  - Schwierigkeitsgrad (*hardness* vs. *easyness*), abhängig vom gewünschten Ergebnis und den eingesetzten Lösungsverfahren
- ▶ unterbestimmte, überbestimmte und wohlbestimmte Probleme

# CSP – Definition

Ein **Constraint Satisfaction Problem** (CSP) ist ein Tripel

$CSP(V, D, C)$ :

$V = \{v_1, \dots, v_n\}$  endliche Menge **Variablen**

$D = \{D_1, \dots, D_n\}$  assoziierte **Wertebereiche**  $\{v_1 : D_1, \dots, v_n : D_n\}$

$C$  endliche Menge **Constraints**  $c_i(V_i), i \in \{1, \dots, m\}$ ,

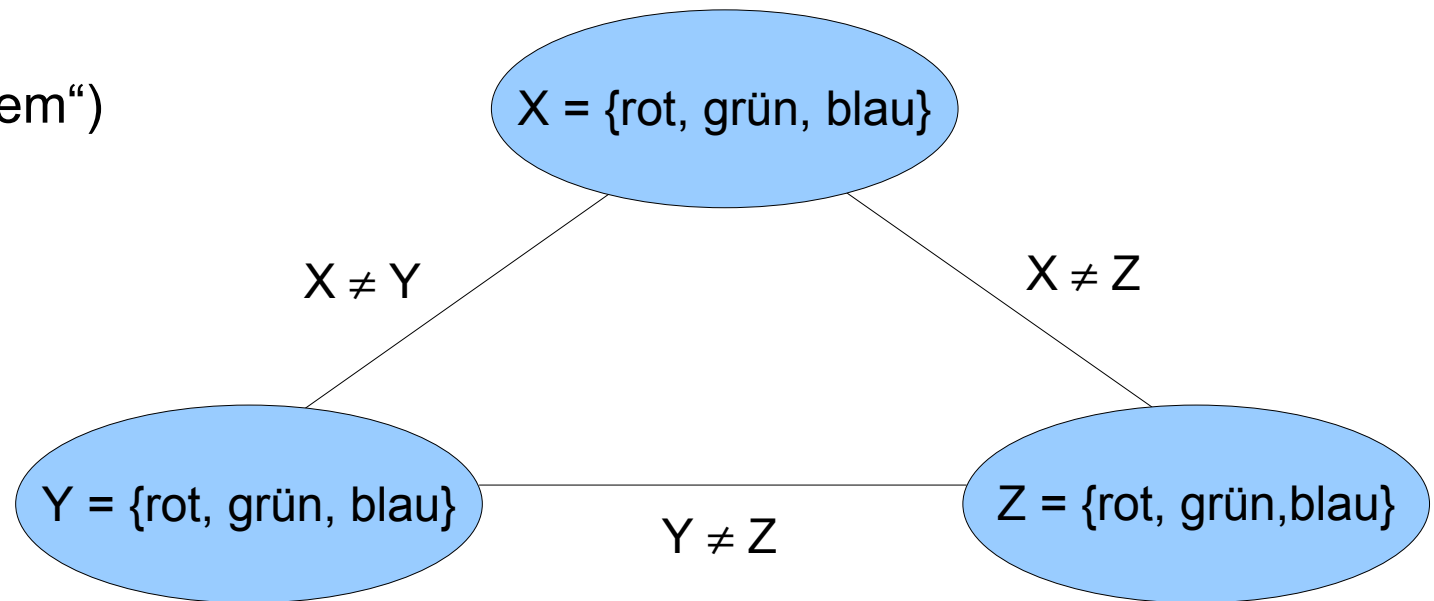
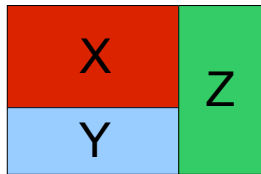
$c_i(V_i)$  setzt Teilmenge  $V_i = \{v_{i_1}, \dots, v_{i_k}\} \subseteq V$  in Relation,

Lösungsraum für  $c_i(V_i)$ :  $\{D_{i_1} \times \dots \times D_{i_k}\}$

# CSP – Beispiel

**Beispiel** für einen binären  
Constraint-Graphen:

(„Kartenfärbeproblem“)



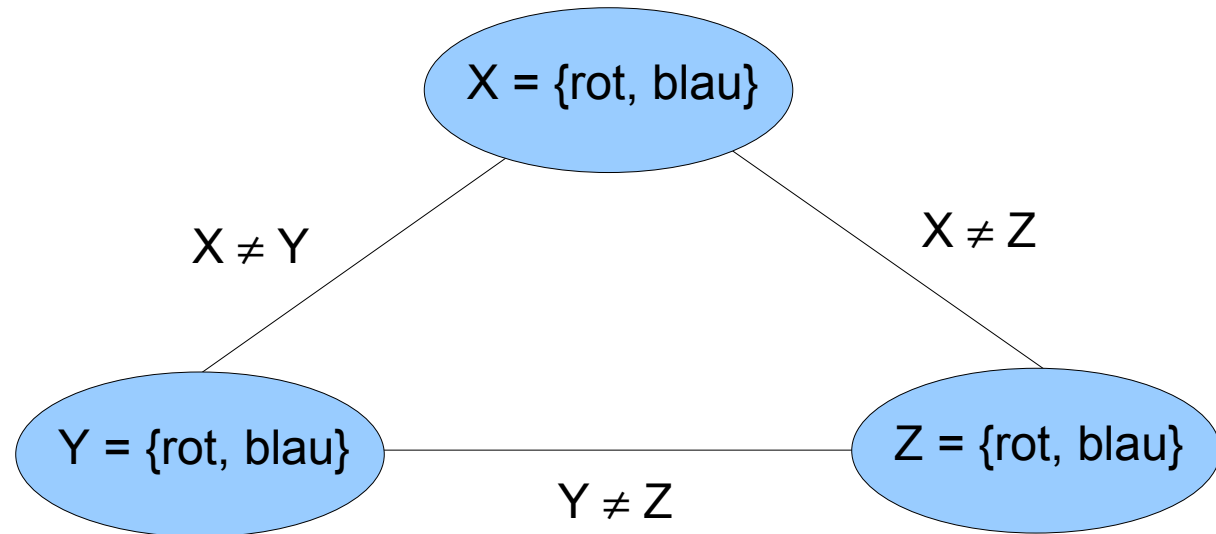
# CSP – Konsistenztechniken (1)

- ▶ Problemreduktion eines CSP mittels **Konsistenztechniken**  
(Ursprung: Montanari '74; Walz '75; Mackworth '77)
- ▶ inkonsistente Werte aus den Domänen der Variablen entfernen
- ▶ erreichen im Regelfall lediglich *lokale Konsistenz*
- ▶ mögliche Konsistenzgrade:
  - **Knotenkonsistenz** (*node consistency*, NC)
  - **Kantenkonsistenz** (*arc consistency*, AC)
  - **Pfadkonsistenz** (*path consistency*, PC)
  - **k-Konsistenz** (*k-consistency*)

# CSP – Konsistenztechniken (2)

**Beispiel** für einen *kantenkonsistenten* Graphen („Kartenfärbeproblem“):

In diesem Fall:  
Keine *globale* Lösung  
vorhanden!

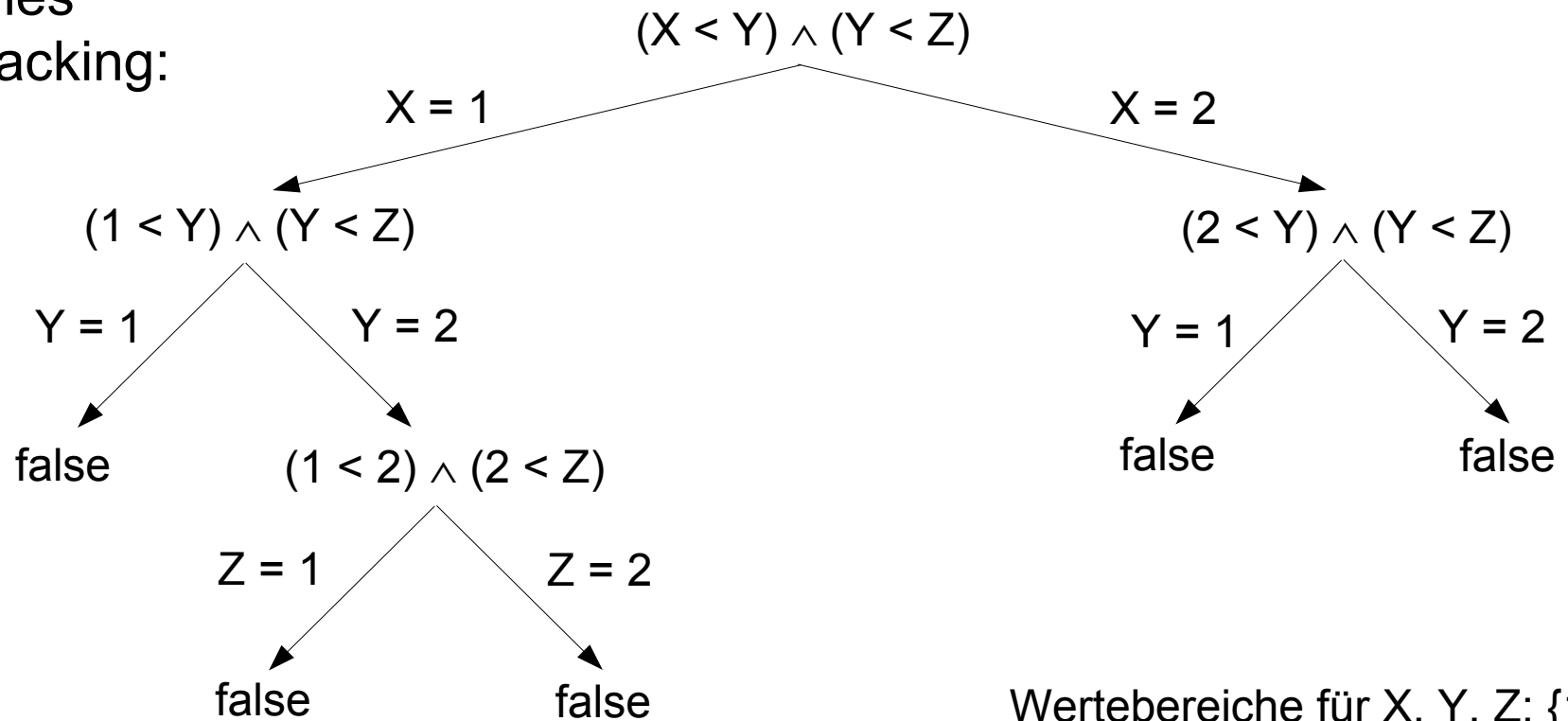


# CSP – Lösungssuche (1)

- ▶ Ein klassisches CSP kann als **Suchproblem** gesehen werden (endliche & diskrete Domänen)
- ▶ Zur Auflösung kann (chronologisches) **Backtracking** (BT) zum Einsatz kommen (vgl. Haralick u. Elliot '80; Dechter u. Frost 2002)
- ▶ Nachteil: ineffizient, exponentieller Aufwand
- ▶ Effizientere Verfahren zur systematischen Suche (Optimierung):
  - Look Back
    - Backjumping (BJ)
    - Backchecking (BC)
    - Backmarking (BM)
  - Look Ahead
    - Forward Checking (FC)
    - Partial Look Ahead (PLA)
    - Full Look Ahead (FLA) / Maintaining Arc Consistency (MAC)

# CSP – Lösungssuche (2)

**Beispiel-Suchbaum für einfaches Backtracking:**



Wertebereiche für X, Y, Z: {1, 2}



# Intervall CSP

- ▶ *Intervall CSP* (ICSP), auch *Numeric CSP* (NCSP) und *Continuous CSP* (CCSP), besitzen reellwertige Intervall-Domänen → unendlich & kontinuierlich
- ▶ Kombination von mathematischen Verfahren der Intervallarithmetik sowie Konsistenztechniken und Suchverfahren (*domain splitting*)
- ▶ Verfahren:
  - *interval splitting* (Cleary '87)
  - *label inference* (Davis '87)
  - *tolerance propagation* (Hyvönen '92)
  - *2B-, 3B-, kB-consistency* (Lhomme '93)
  - *box consistency* (Benhamou et al. '94)
  - *2<sup>k</sup>-trees* (Sam-Haroud '95, Sam-Haroud u. Faltings '96)

# Constraint-Systeme (1)

## Integrierte Constraint-Solver:

- ▶ Prolog II
- ▶ CLP(R)
- ▶ CHIP
- ▶ Prolog III
- ▶ BNR Prolog
- ▶ SICStus Prolog
- ▶ CAL, GDCC
- ▶ CLP(Intervals): CLP(BNR), Interlog, CIAL, Prolog IV, ECLIPSe, DeclIC, CLIP, Newton, Numerica
- ▶ CLP(FD), GNU Prolog

## Bibliotheken:

- ▶ Cassowary
- ▶ ILOG Solver / ILOG JSolver
- ▶ UniCalc

- ▶ ALIAS
- ▶ RealPaver
- ▶ C-Lib, Java Constraint Library (JCL)
- ▶ Declarative Java (DJ)
- ▶ Koalog Constraint Solver (KCS)
- ▶ J.CP
- ▶ IASover

## Frameworks:

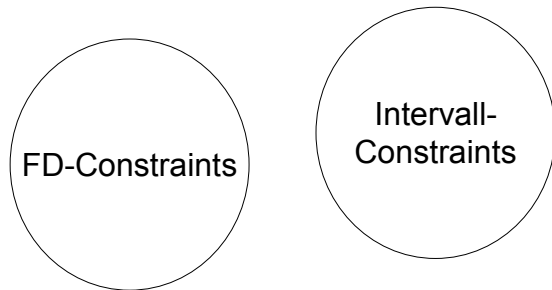
- ▶ BackTalk
- ▶ GIFT
- ▶ POOC
- ▶ Constraint Handling Rules (CHR)
- ▶ Java Constraint Kit (JACK)

# Constraint-Systeme (2)

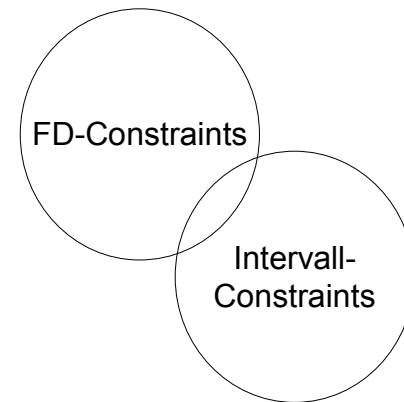
- ▶ **integrierte Solver / CLP-Systeme:** schwer adaptierbar, hoher Integrationsaufwand (deklarative Schnittstelle, Overhead), vorrangig für spezielle *global constraints*, oftmals keine Java-Schnittstelle
- ▶ **Bibliotheken:** kein System erfüllt vollständig die relevanten Anforderungen (vgl. Tabelle)
- ▶ **Frameworks:** nicht in Java/C/C++ verfügbar (*BackTalk*), reine Schnittstelle (*GIFT*) oder CLP bzw. deklarativ (*POOC*, *CHR*, *JACK*), zudem lediglich meist sehr simple Lösungsverfahren bzw. eingeschränkte Funktionalität
- ▶ **kooperative Ansätze:** entweder statisch und damit unflexibel oder flexibel, aber mit hohem Overhead durch Koordinierung
- ▶ **Fazit:** Eigenentwicklung eines OOP-Frameworks mit integrierten Constraint-Lösungsverfahren

# Hybridizität versus Heterogenität

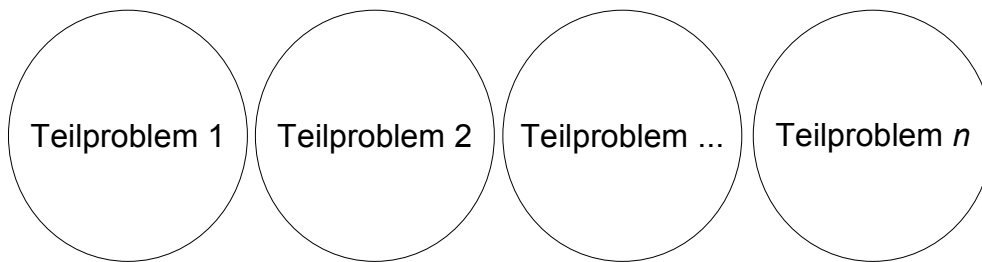
Szenario 1:



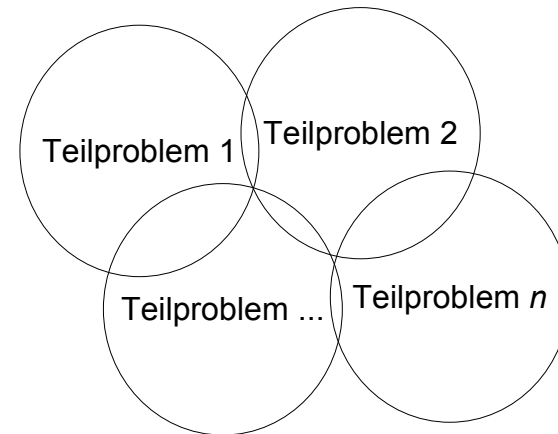
Szenario 3:



Szenario 2:



Szenario 4:



# Heterogenes Constraint-Lösen

- ▶ Unterschiedliche Constraint-Lösungsstrategien bedingen getrennt voneinander zu verarbeitende **Teilprobleme**.
- ▶ Variablen, die in unterschiedlichen Teilproblemen auftauchen, bewirken **Überlappung** von Teilproblemen (*lokaler vs. globaler Namensraum*).
- ▶ Bei **Vermischung** von Domänen ist zusätzlich heuristische Diskretisierung von reellwertigen Intervallen bzw. die intervallwertige Behandlung von finiten Domänen erforderlich (heterogenes Constraint-Problem).
- ▶ Problem: Aus einzelnen Teillösungen (*lokale Sicht*) müssen vollständige **Gesamtlösungen** (*globale Sicht*) generiert werden.
- ▶ Lösung: **Meta-Constraint-Solver**

# Heterogenes Constraint-Lösen

- ▶ „Vermischung“ von finiten und infiniten Wertebereichen
- ▶ Überschneidung von Constraint-Netzen mit Variablen unterschiedlicher Wertebereiche
- ▶ Heuristiken:
  - Intervall-Variable in finitem Constraint → Wertebereich diskretisieren (als Integer-Menge)
  - finite Variable in Intervall-Constraint → als Intervall [(1,1)(2,2)(3,3)]