Constraints – Konzepte und Systeme sowie Lösungsverfahren für hybride Problemstellungen

Wolfgang Runte

Fachbereich Mathematik / Informatik
Universität Bremen

15. Oktober 2007

Übersicht

- Einführung
- Konzepte zur Constraint-Verarbeitung
- Verfügbare Constraint-Systeme
- Constraint-Lösungsverfahren
- Constraint-Framework YACS
- Zusammenfassung

Einführung

Constraints (1)

- *algebraische* Constraints:
  - Intensionale Relationen beschrieben durch Gleichungen oder Ungleichungen zur Einschränkung der möglichen Belegungen von Variablen.
- Beispiel:
  - **Variablen:** v_1 und v_2 mit je dem Wertebereich {0,1,2,3,4,5,6,7,8,9}
  - **Constraints:** $v_1 + v_2 = 10$ und $v_1 - v_2 = 2$
  - **Lösung:** $v_1 = 6$ und $v_2 = 4$
  - Neben arithmetischen sind auch symbolische Wertebereiche möglich.

Einführung

Constraints (2)

- Constraint-Propagation:
  - Fortpflanzung von Beschränkungen in einem *Constraint-System*.
  - Belegungen einzelner Variablen üben Beschränkungen aufeinander aus, die zu Wertebereichseinschränkungen in den Domänen der jeweiligen Variablen führen.
  - Einschränkungen breiten sich durch wiederkehrende Propagation über eine Vielzahl Variablen aus.
- Constraint-System:
  - Ein System, welches eine Problemreduktion durch schrittweise Einschränkung der möglichen Belegungen von Variablen erzeugt und ggf. mittels Suchverfahren Problemlösungen generiert.

Einführung

CSP – Definition

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

CSP(V,D,C):

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

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

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

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

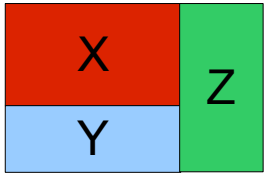
Lösungsraum für $c_i(V_i)$: $\{D_{i_1} \times \ldots \times D_{i_k}\}$

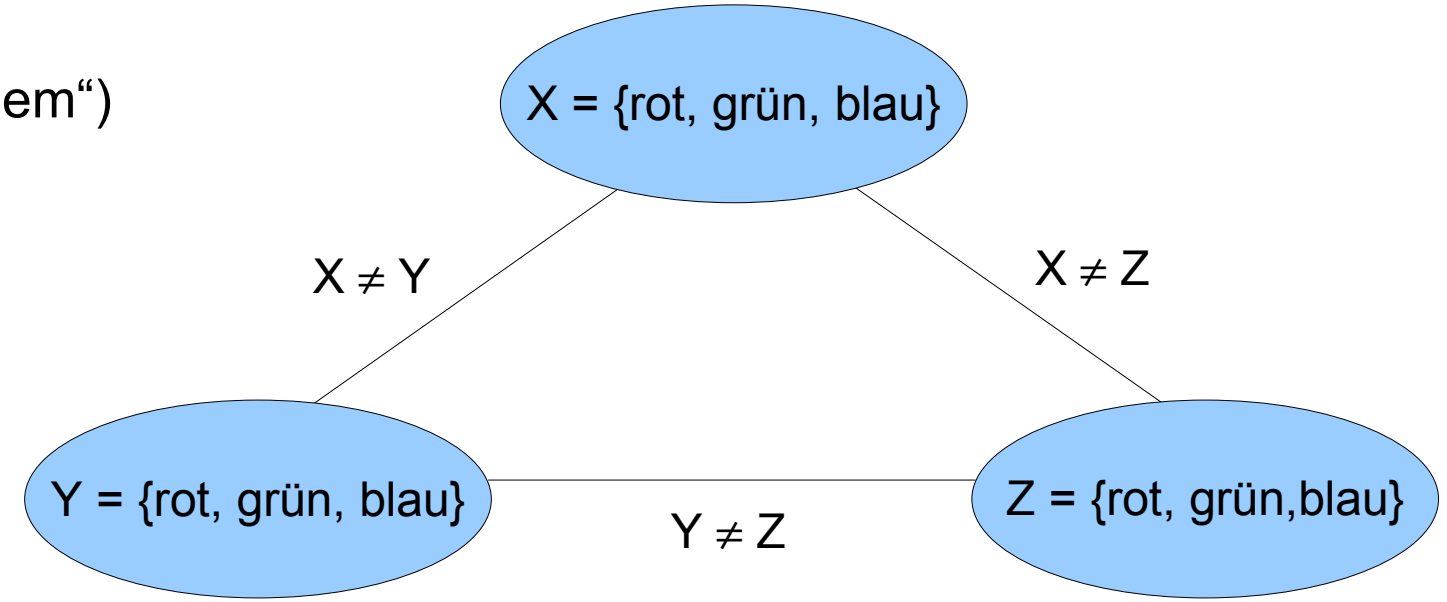
Einführung

CSP – Beispiel

Beispiel für einen binären Constraint-Graphen:

(„Kartenfärbeproblem“)





Einführung

Eigenschaften von Constraints

- finite und infinite Domänen
- arithmetische und symbolische Wertebereiche
- Größe eines Problems (Anzahl Variablen/Constraints)
- Stelligkeit eines Constraints (unär, binär, ternär, ..., *n*-är)
- Struktur eines Constraint-Netzes (CN):
  - 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

Einführung

EngCon: 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

Einführung

EngCon: Constraint-Relationen

- **Algebraische Constraints**
  Als (Un-)Gleichung formalisierte komplexe, algebraische 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 (allg. *global 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;
}
```

Übersicht

- Einführung
- **Konzepte zur Constraint-Verarbeitung**
- Verfügbare Constraint-Systeme
- Constraint-Lösungsverfahren
- Constraint-Framework YACS
- Zusammenfassung

Konzepte

Allgemeine Konzepte

- **„Klassisches“ Constraint Satisfaction Problem (CSP)**
  - endliche und diskrete Wertebereiche (*finite domain, FD*)
  - arithmetische und symbolische Wertebereiche
  - Montanari (1974), Waltz (1972, 1975), Mackworth (1977a)
- **Interval Constraint Satisfaction Problem (ICSP)**
  - unendliche und kontinuierliche Wertebereiche (infinite Domänen)
  - reellwertige Intervalle als Wertebereiche
  - Davis (1987), Cleary (1987), Hyvönen (1992), Lhomme (1993)
- **Temporal Constraint Satisfaction Problem (TCSP)**
  - Zeitplanungsprobleme als CSP formuliert (*scheduling*)
  - Betrachtung von Zeit*intervallen* vs. Zeit*punkten*
  - Allen (1983), Dechter et al. (1991)

Konzepte

Constraint-Verfahren zur Konfigurierung

- **Dynamic/Conditional Constraint Satisfaction Problem (DCSP, CondCSP)**
  - Variablen und Constraints erhalten Status „aktiviert“ bzw. „deaktiviert“
  - Bedingungen bzgl. des Aktivierungsstatus durch *Aktivierungs-Constraints*
  - Mittal und Falkenhainer (1990), Sabin und Freuder (1998, 1999)
  - DCSP als Sequenz P_0 ... P_α von statischen CSPs: Dechter (1988)
- **Composite Constraint Satisfaction Problem (CompCSP)**
  - Abbildung von Aggregatstrukturen
  - hierarchische Organisation von Komponententypen
  - Sabin und Freuder (1996a, b)
- **Generative Constraint Satisfaction Problem (GCSP)**
  - Constraints als Relation zwischen Komponenten*typen*
  - *generische* Constraints mit Metavariablen als Platzhalter für Komponenten-Variablen
  - Stumptner und Haselböck (1993), Stumptner et al. (1998)

Konzepte

Überbestimmte Constraint-Probleme (1)

- **Over-Constrained Systems (OCS)**
  - inkonsistente Zustände durch sich widersprechende oder zu starke Anforderungen
  - überspezifizierte Situation, z. B. innerhalb eines interaktiven Konfigurierungsprozesses
  - Ausweg: suboptimale Lösungen durch Teilerfüllung der Beschränkungen
- **Partial Constraint Satisfaction Problem (PCSP)**
  - vier existierende Möglichkeiten, Inkonsistenzen in einem OCS zu entschärfen:
    1. Wertebereiche einer Variable um geeignete Werte erweitern
    2. Relationstupel der kompatiblen Wertekombinationen eines Constraints erweitern
    3. Eliminierung von Variablen und der zugehörigen Constraints
    4. Eliminierung von Constraints
  - Bewertungsfunktion („Metrik“) zur Auswahl der bestgeeigneten (Teil-)Lösung nötig
  - Metrik definiert eine Ordnung auf den unterschiedlichen, suboptimalen Lösungen
  - Freuder (1989), Freuder und Wallace (1992)

Konzepte

Überbestimmte Constraint-Probleme (2)

- **Constraint Relaxierung / Constraint Retraction**
  - Oberbegriff zur Eliminierung von Constraints
    a) entweder: Fokussierung auf bestimmte Constraints (anhand einer Metrik)
    b) oder: Umkehroperation und Abhängigkeitswissen zum inkrementellen Zurücknehmen von Constraints erforderlich
  - Günter (1992), Günter (1995b), Syska und Cunis (1991), Ringwelski (2003)
- **Maximal Constraint Satisfaction Problem (MaxCSP)**
  - simple Metrik: je mehr Constraints erfüllt sind, umso besser ist die Problemlösung
  - Lösungsverfahren: *Branch & Bound*
  - Freuder und Wallace (1992)
- **Soft Constraint Satisfaction Problem (SoftCSP)**
  - Unterscheidung: *weiche* Constraints müssen nicht, *harte* Constraints müssen erfüllt werden, um zu einer „konsistenten“ Lösung zu gelangen
  - Ruttkay (1994)

Konzepte

Überbestimmte Constraint-Probleme (3)

- **Hierarchical Constraint Satisfaction Problem (HCSP)**
  - differenzierte Unterscheidung der Lösungen anhand deren Qualität
  - Gewichtung anhand einer endlichen Menge unterschiedlicher Prioritätsstufen
  - eingeschränkte Lösungsalgorithmen
    a) entweder: zyklische Constraint-Netze können nicht oder nur mit Einschränkungen verarbeitet werden (*local propagation*)
    b) oder: ausschließlich lineare Constraints möglich (*linear programming*)
  - besondere Ausprägung: Hierarchical Constraint Logic Programming (HCLP)
  - Borning et al. (1987, 1992, 1994)
- **Fuzzy Constraint Satisfaction Problem (Fuzzy CSP)**
  - Bestimmung des Grads der Erfüllung eines Constraints für jeweilige Wertebelegung
  - Abweichung vom „Sollwert“ als Wert zwischen 0 und 1
  - angestrebt wird die Maximierung des Erfüllungsgrades für das gesamte Problem
  - Ruttkay (1994)

Konzepte

Weiterführende Konzepte (1)

- **Structural Constraint Satisfaction Problem (SCSP)**
  - dynamische Strukturänderungen des Constraint-Netzes auf Basis der Regeln einer Graph-Grammatik (z. B. zur automatischen Ablaufplanung von Workflows)
  - Suche nach der Struktur des Constraint-Netzes als Teil des Lösungsprozesses
  - Nareyek (1999a, b)
- **Open Constraint Satisfaction Problem (OCSP)**
  - für *offene* und verteilte Umgebungen (Internet: *Web Services*, *Semantic Web*)
  - anstatt *closed world assumption* (CWA): *open-world*-Szenarien (z. B. für Anwendungen der *innovativen Konfigurierung*)
  - Informationen und deren Menge im Vorfeld unbekannt (Bündelung durch *Mediatoren*)
  - anstatt *Brute-Force*-Ansatz: nur benötigte Informationen „einsammeln“ (inkrementell)
  - erste gefundene Lösung ausgeben, weitere nach Bedarf generieren
  - Optimierung: Gewichtung unterschiedlicher Lösungen durch „gewichtete“ Constraints
  - Faltings und Macho-Gonzalez (2002, 2003)

Konzepte

Weiterführende Konzepte (2)

- **Distributed Constraint Satisfaction Problem (DisCSP)**
  - Verteiltheit, weil
    a) CSP logisch oder physikalisch verteilt ist
    b) verteilte bzw. parallele Verarbeitung Effizienzvorteile mit sich bringt
  - Behandlung verteilter Abhängigkeiten in *Multi-Agenten-Systemen* (MAS)
  - MAS: Variablen und Constraints auf eine Vielzahl automatisierter Agenten bzw. Prozesse verteilt (verteilte bzw. asynchrone Lösungsverfahren notwendig)
  - Yokoo und Hirayama (2000)

- **Concurrent Constraint Programming (CCP)**
  - Konzept für eine einfache Sprache zu effizienten/parallelen Constraint-Verarbeitung
  - zentrale Kontrollinstanz zur Koordinierung der Kommunikation der Agenten/Prozesse
  - Van Hentenryck und Saraswat (1996), Frühwirth und Abdennadher (1997)

Konzepte

Weiterführende Konzepte (3)

- **Asynchronous Constraint Solving (ACS)**
  - Zusammenführung von verteilter und dynamischer Constraint-Verarbeitung (DynDCSP)
  - „Ausführungsmodell“ für asynchrone Constraint-Lösungsverfahren
  - unterstützt inkrementelles Hinzufügen und Zurücknehmen von Constraints
  - Ringwelski (2001a, 2003), Ringwelski und Schlenker (2002)

- **Adaptive Constraint Satisfaction Problem (ACSP)**
  - automatisiert unterschiedlich komplexe Constraint-Lösungsverfahren anwenden (der Reihe nach auswählen, Beginnen mit dem einfachsten Verfahren)
  - einfache Probleme werden ohne viel Overhead gelöst, „harte“ Probleme können mit entsprechend komplexen/effizienten Lösungsverfahren behandelt werden
  - alternativ: Analyse zu Beginn des Lösungsprozesses (z. B. Maschinelles Lernverfahren)
  - Borrett et al. (1995, 1996a, b), Kwan (1997)

Konzepte

Weiterführende Konzepte (4)

- **Mixed Constraint Satisfaction Problem (Mixed CSP)**
  - Zusammenführung von finiten und infiniten Domänen
  - für Problemstellungen mit Constraints, die *gleichzeitig* Variablen mit finiten und infiniten Wertebereichen in Relation setzen
  - *mixed constraints* oder „heterogene“ Constraints
  - Erweiterung des Konsistenzbegriffs derart, dass sich diskrete und kontinuierliche Domänen gleichzeitig behandeln lassen
  - Integration unterschiedlicher Lösungsverfahren (finit/infinit) innerhalb eines Suchverfahrens zum Auffinden von „gemischten“ Lösungen
  - Anwendung im Bereich der Konfigurierung
  - Erweiterung um Aktivitäts-Constraints: Mixed Dynamic/Conditional CSP
  - Benhamou (1996), Gelle (1998), Gelle und Faltings (2003)

Übersicht

- Einführung
- Konzepte zur Constraint-Verarbeitung
- **Verfügbare Constraint-Systeme**
- Constraint-Lösungsverfahren
- Constraint-Framework YACS
- Zusammenfassung

Verfügbare Systeme

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)

Verfügbare Systeme

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.

Verfügbare Systeme

Constraint-Bibliotheken

	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

Übersicht

- Einführung
- Konzepte zur Constraint-Verarbeitung
- Verfügbare Constraint-Systeme
- **Constraint-Lösungsverfahren**
- Constraint-Framework YACS
- Zusammenfassung

Constraint-Lösungsverfahren

Konsistenztechniken (1)

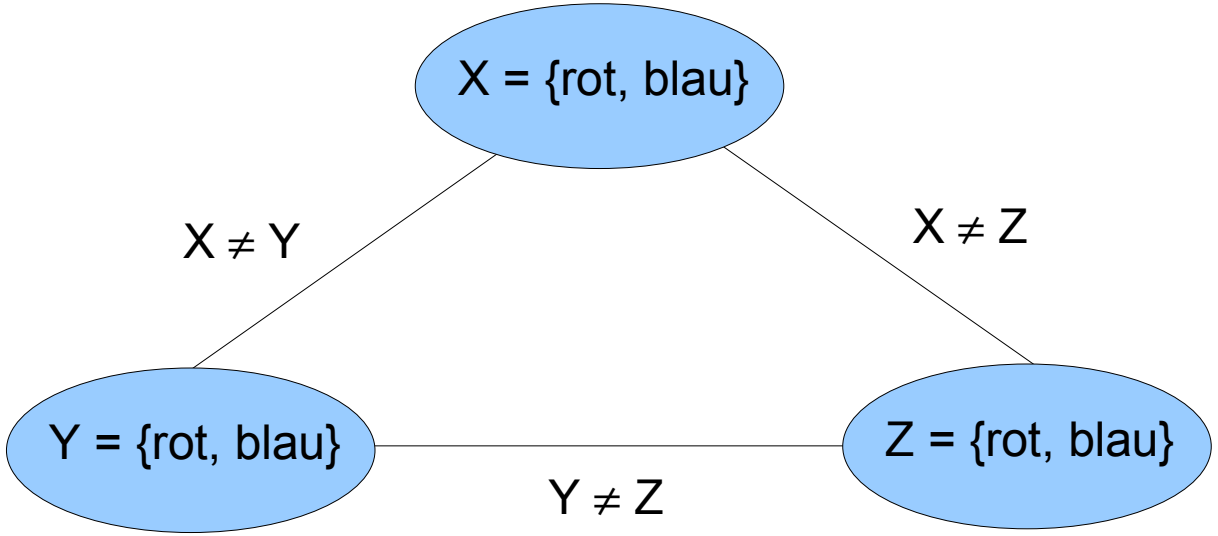
- Problemreduktion eines (klassischen) CSP mittels **Konsistenz-techniken** (Ursprung: Montanari 1974; Walz 1975; Mackworth 1977a)
- inkonsistente Werte aus den Domänen der Variablen entfernen
- erreichen im Regelfall lediglich *lokale Konsistenz*
- mögliche Konsistenzgrade:
  - **Knotenkonsistenz** (*node consistency*, NC)
  - **Kantenkonsistenz** (*arc consistency*, AC)
  - **Pfadkonsistenz** (*path consistency*, PC)
  - ***k*-Konsistenz** (*k-consistency*)

Constraint-Lösungsverfahren

Konsistenztechniken (2)

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

In diesem Fall:
Keine *globale* Lösung vorhanden!



Constraint-Lösungsverfahren

Weitere Konsistenzgrade

- *(i,j)-consistency*
  - verallg. der *k*-Konsistenz: (k-1,1)-Konsistenz; AC: (1,1)-Konsistenz
- *inverse consistency*
  - *k-inverse* Konsistenz: (1,k-1)-Konsistenz; inverse Pfadkonsistenz: (1,2)-Konsistenz (PIC)
- *lazy arc consistency* (LAC)
  - Ziel: Erkennen von Inkonsistenzen durch *domain wipe out* (DWO).
  - Erste gefundene, kantenkonsistente Belegung ausreichend.
- *restricted path consistency* (RPC)
  - AC erweitert um PC in speziellen Fällen.
- *directional arc consistency* (DAC)
  - gerichtete AC für geordnete Variablenpaare (REVISE)
  - globale Konsistenz für zyklenfreie CN
- *directional path concistency* (DPC)
  - gerichtete PC (vgl. DAC)
- *singleton consistency*
  - prüfen ob „Singleton“ zum DWO führt
  - kombinierbar mit beliebigen Konsistenzalgorithmen (z. B. LAC)
- *adaptive consistency*
  - gerichtete, globale Konsistenz
  - aufsteigender Konsistenzgrad (Variablen)

Constraint-Lösungsverfahren

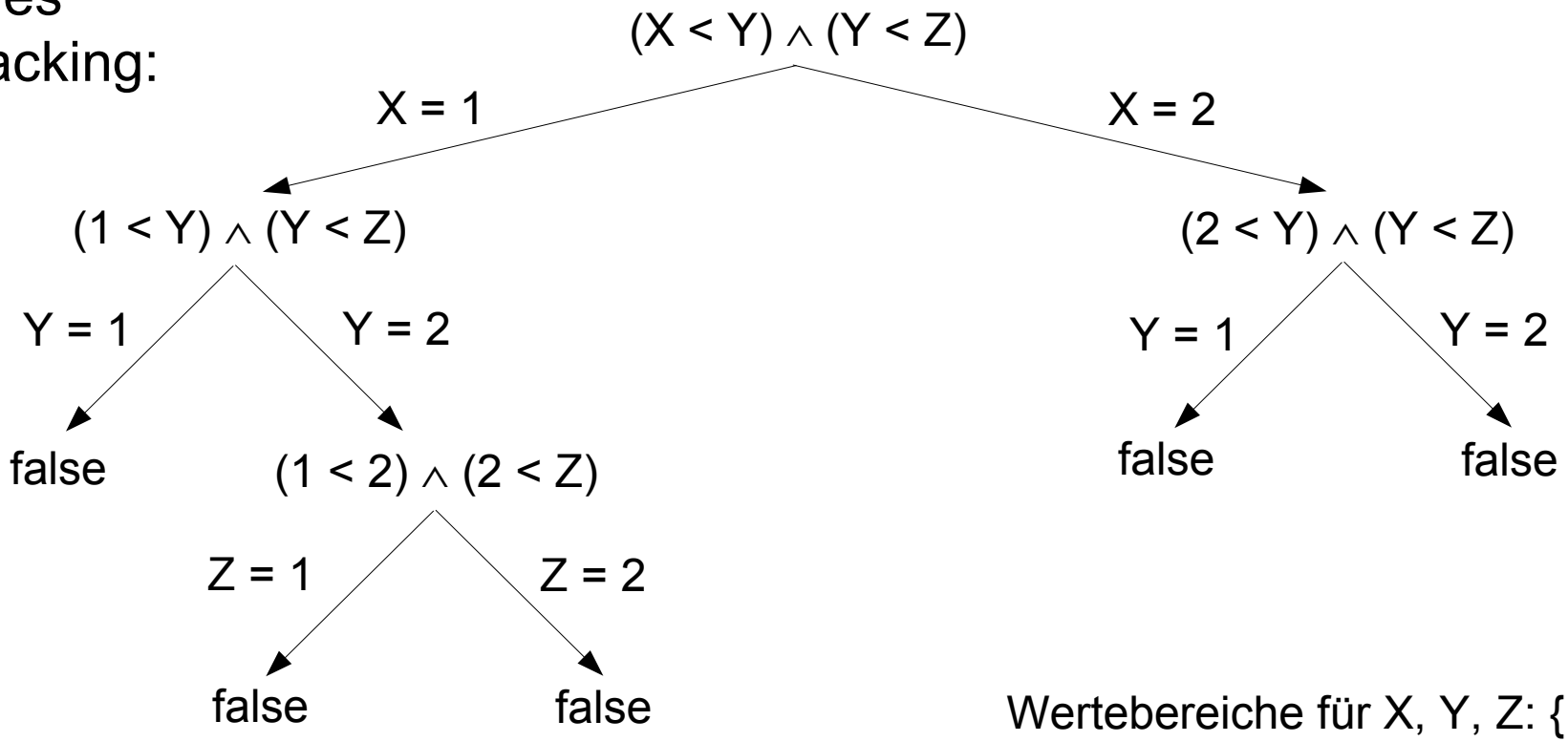
Lösungssuche (1)

- Ein klassisches CSP kann als **Suchproblem** gesehen werden (endliche & diskrete Domänen).
- Zur Auflösung kann **Generate & Test** (GT) zum Einsatz kommen:
  - systematisches Generieren von Lösungen
  - Nachteil: Der gesamte Suchraum muss getestet werden (kartesische Produkt der Wertebereiche aller Variablen).
- Besser: (chronologisches) **Backtracking** (BT).
  - inkrementelles Verfahren: frühzeitige Erkennung von Inkonsistenzen
  - Nachteil: Immer noch ineffizient, exponentieller Aufwand (vgl. Haralick u. Elliot 1980; Dechter u. Frost 2002)
  - weitere Schwäche: sog. *trashing* $\rightarrow$ wiederholtes Fehlschlagen aufgrund derselben Inkonsistenzen (vgl. Mackworth 1977a)

Constraint-Lösungsverfahren

Lösungssuche (2)

Beispiel-Suchbaum für einfaches Backtracking:



Wertebereiche für X, Y, Z: {1, 2}

Constraint-Lösungsverfahren

Lösungssuche (3)

Look-Back:

- *Backjumping* (BJ)
  - anstatt chronologischem BT Rücksprung zur konfliktauslösenden Variable
- *Backchecking* (BC)
  - merkt sich getestete inkompatible Belegungen
- *Backmarking* (BM)
  - merkt sich ebenfalls getestete kompatible Belegungen

Look-Ahead:

- *Forward Checking* (FC)
  - BT mit REVISE-Routine
- *Partial Look-Ahead* (PLA)
  - ähnlich DAC als *look-ahead*
- *Full Look-Ahead* (FLA)
  - annähernd AC als *look-ahead*
- *Maintaining Arc Consistency* (MAC)
  - vollständig AC als *look-ahead*

Constraint-Lösungsverfahren

„Hybride“ Lösungsverfahren

- Kombination von *look-back*- und *look-ahead*-Strategien.
- Gegenseitige Beeinflussung: Je höher der *look-ahead*-Konsistenzgrad, umso weniger wirksam sind *look-back*-Strategien ($\rightarrow$ *Overhead*).
- Gängige Beispiele: FC-BM-CBJ, FC-CBJ, MAC-CBJ.
- Lang anhaltende Diskussion, wieviel *look-ahead* sinnvoll ist.
- *„Lookahead to the future in order not to worry about the past.“* (Haralick u. Elliot 1980)

Constraint-Lösungsverfahren

Variablenordnungsheuristiken (1)

- Verhindern von *trashing:* „kritische Variablen“, von denen viele andere Belegungen abhängig sind, zu Beginn der Suche belegen.
- Prinzip: *most constrained first* – *oder* allgemein: *most critical variables first*.
- *„To succeed, try first, where you are most likely to fail.“* (Haralick u. Elliot 1980)
- Unterscheidung zwischen *static variable ordering* (SVO) und *dynamic variable ordering* (DVO).
- Kombinationen unterschiedlicher Heuristiken möglich.

Constraint-Lösungsverfahren

Variablenordnungsheuristiken (2)

- *fail first* (FF)
  - Variable mit kleinstem Wertebereich wird als erstes belegt.
- *minimal width ordering* (MWO)
  - Reihenfolge mit minimaler „Weite“ generieren: Variablen mit größerer Beschränkung weiter vorn in der Belegungsreihenfolge (Verringerung von Backtracking).
- *maximum cardinality ordering* (MCO)
  - Approximation von MWO: beginnend mit beliebiger Variable, der Reihe nach diejenigen hinzufügen, die mit den meisten bisherigen über ein Constraint verbunden sind.
- *maximum degree ordering* (MDO)
  - Ebenfalls Approximation von MWO: Variablen absteigend nach dem Grad ihres Vorkommens in den Constraints.
- *minimal bandwidth ordering* (MBO)
  - Verringerung der Abstände sich gegenseitig beschränkender Variablen („Bandbreite“), und Minimierung der Strecke, die BT im Konfliktfall zurücklegen muss.

Constraint-Lösungsverfahren

Werteordnungsheuristiken (1)

- engl.: *look-ahead value ordering* (LVO)
- Zuerst Zweige im Suchbaum durchsuchen, deren Wahrscheinlichkeit zu einer konsistenten Lösung zu führen größer ist als die anderer → eine erste Lösung kann früher gefunden werden.
- Prinzip: *least constraining values first* – oder allgemein: *most promising values first*.
- Kein Einfluss auf die Effizienz eines Suchverfahrens, wenn alle Lösungen für das CSP gesucht werden, bzw. wenn es für das CSP keine Lösung gibt, da alle Werte berücksichtigt werden müssen.
- Allgemein: viele Konsistenztests notwendig; Overhead lohnt bei einfachen Problemen nicht; werden weniger häufig eingesetzt als Variablenordnungsheuristiken.

Constraint-Lösungsverfahren

Werteordnungsheuristiken (2)

- *min-conflicts* (MC)
  - Variablen als erstes mit den Werten belegen, die die wenigsten Konflikte mit den Werten noch unbelegter Variablen aufweisen.
- *max-domain-size* (MD)
  - Bevorzugt Werte, die den größten minimalen Wertebereich für zukünftig zu belegende Variablen garantieren.
- *weighted-max-domain-size* (WMD)
  - Erweiterung von MD: Bei gleich großen minimalen Wertebereichen den Wert auswählen, der weniger Variablen mit dem minimalen Wertebereich erzeugt.
- *point-domain-size* (PDS)
  - Vergibt Punkte für Wertebereiche mit einer bestimmten Anzahl Elementen für jeden Wert der aktuell zu belegenden Variable. Der Wert mit z. B. der geringsten Punktsumme als erstes zur Belegung der aktuellen Variable ausgewählt.

Constraint-Lösungsverfahren

Binäre vs. *n*-äre Constraints

- Unterscheidung: unäre, binäre, ternäre, ..., *n*-äre Constraints
- Lösungsverfahren existieren häufig nur für binäre Constraints.
- Begründung: Verfahren für binäre Constraints können für *n*-äre Constraints verallgemeinert werden.
- In der Praxis werden häufig *n*-äre Constraints benötigt.
- Nachteil: *n*-äre Constraints weisen höhere Komplexität auf.
- Möglichkeiten:
  - direkte Verarbeitung *n*-ärer Constraints (BT, FC, Hyperkantenkonsistenz)
  - „Binärisierung“ (*Hidden-Variable-, Dual-Graph-* und *Double*-Repräsentation)
  - Zur Komplexitätsreduktion ausschließliche Verarbeitung der Grenzen der Wertebereiche: *bounds consistency* (Informationsverlust!)

Constraint-Lösungsverfahren

Intervall CSP

- *Intervall CSP* (ICSP), auch *Numeric CSP* (NCSP) und *Continuous CSP* (CCSP): Constraint-Variablen besitzen reellwertige Intervall-Domänen (unendlich & kontinuierlich).
- klassisches CSP (NP-vollständig) *versus* ICSP (unentscheidbar)
- ICSPs geeignet für folgende Szenarien:
  - Behandlung unscharfer Informationen (Wissen über Parameter nur als Werteintervall).
  - Aufzählung aller Lösungen nicht möglich (da unendlich viele).
  - Unterbestimmte Systeme mit mehr als einer Lösung (Bsp. optimaler Drehzahlbereich).
  - Statt kombinatorischer Verfahren für einzelne Werte, gleichzeitig unendliche viele Werte einschränken.
- Kombination von mathematischen Verfahren der Intervallarithmetik sowie Konsistenztechniken und Suchverfahren (*domain splitting*).

Constraint-Lösungsverfahren

Intervall-Lösungsverfahren (1)

- Intervall-Splitting / *interval splitting* (Cleary 1987)
  - Zerteilen der Wertebereiche, bis eine Lösung gefunden wird.
  - Optimierung: *intelligent domain splitting* und Clusteranalyse (für die Analyse zusammenhängender Bereiche des Lösungsraums).
  - Bisher ausschließlich fest implementierte Constraints (*global constraints*).
  - Nachteil: ggf. „kombinatorische Explosion“ (Lhomme 1993).
- *Label Inference* (Davis 1987)
  - Approximierte Kantenkonsistenz bezogen auf die Intervallgrenzen der Wertebereiche.
  - Ebenfalls ausschließlich fest implementierte Constraints (*global constraints*).
  - Nachteil: Verhalten bei nichtlinearen Constraints nicht vorhersagbar: (1) *early quiescence,* (2) *cycling,* (3) *slow convergence.*

Constraint-Lösungsverfahren

Intervall-Lösungsverfahren (2)

- Toleranzpropagation / *tolerance propagation* (Hyvönen 1992)
  - „Toleranzen“ als Synonym für Intervalle zur Modellierung unscharfen Wissens.
  - Berücksichtigung von diskontinuierlichen/unterbrochenen Intervallen.
  - Anstatt Constraints zu propagieren, werden von jedem Constraint die entsprechenden *solution functions* (implizite Funktionen) gebildet; zuvor Zerlegung in „primitive“ Constraints.
  - Neben lokaler auch globale Konsistenz durch Eliminierung von Zyklen im CN.
- 2B-, 3B-, *k*B-Konsistenz / 2B-, 3B-, *k*B-consistency (Lhomme 1993)
  - Erweiterungen zur Intervallpropagation: 2B-, 3B- und *k*B-Konsistenz analog zur Kanten- Pfad- und *k*-Konsistenz.
  - Berücksichtigt ausschließlich ununterbrochene Intervalle.
  - „Projektionen“ anstatt *solution functions;* Zerlegung in ternäre *basic constraints* (entsprechen primitiven Constraints von Hyvönen).

Constraint-Lösungsverfahren

Intervall-Lösungsverfahren (3)

- Box-Konsistenz / *box consistency* (Benhamou et al. 1994)
  - Ebenfalls Approximation von Kantenkonsistenz (ununterbrochene Intervalle).
  - Beinhaltet numerisch-mathematisches Verfahren zur Berechnung von Nullstellen und zur Einschränkung der Wertebereiche der Constraint-Variablen.
  - Newton-Intervallverfahren: Erweiterung des iterativen Newton-Verfahrens zur Nullstellenberechnung.
  - Erzeugung und Propagierung von speziellen Projektionen (Zerlegung entfällt).
- 2^k-Bäume / 2^k-*trees* (Sam-Haroud 1995, Sam-Haroud u. Faltings 1996)
  - Lösungsraum hierarchisch zerlegt als 2^k-Bäume: Bereiche ohne Lösungen (schwarz); Bereiche mit ausschließlich Lösungen (weiß); Bereiche, die beides enthalten (grau).
  - Herstellung unterschiedlicher Konsistenzgrade durch räumliche Projektion.
  - Ausschließlich logische anstatt numerische Operationen.

Übersicht

- Einführung
- Konzepte zur Constraint-Verarbeitung
- Verfügbare Constraint-Systeme
- Constraint-Lösungsverfahren
- **Constraint-Framework YACS**
- Zusammenfassung

YACS

Problemstellung

- 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).

YACS

Zielsetzung

- Entwicklung eines **objektorientierten Frameworks** zur flexiblen Anbindung unterschiedlicher Constraint-Solver resp. Constraint-Lösungverfahren.
- 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.

YACS

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

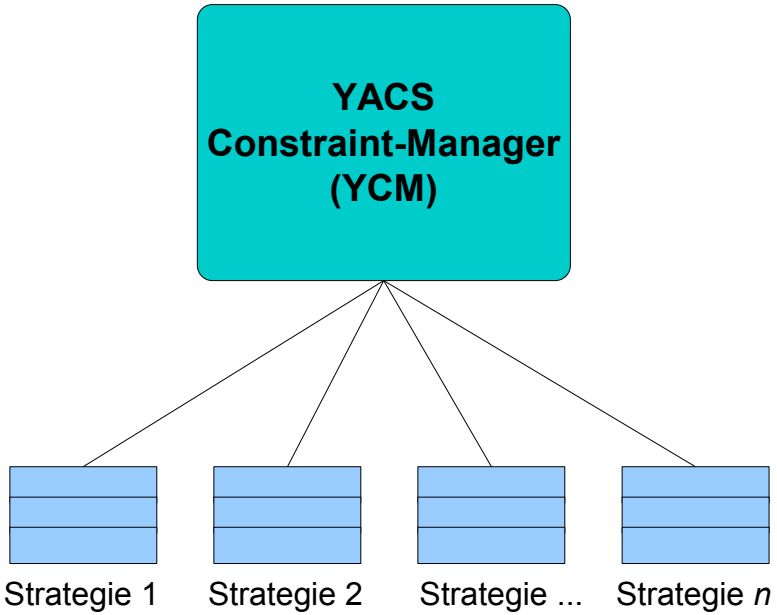
1	Binärisierung
2	(1) Knotenkonsistenz (2) Kantenkonsistenz
3	konfliktbasiertes Backjumping

1	Zerlegung in primitive Constraints
2	Hull-Konsistenz
3	-

YACS

Abstraktion durch Strategien (1)

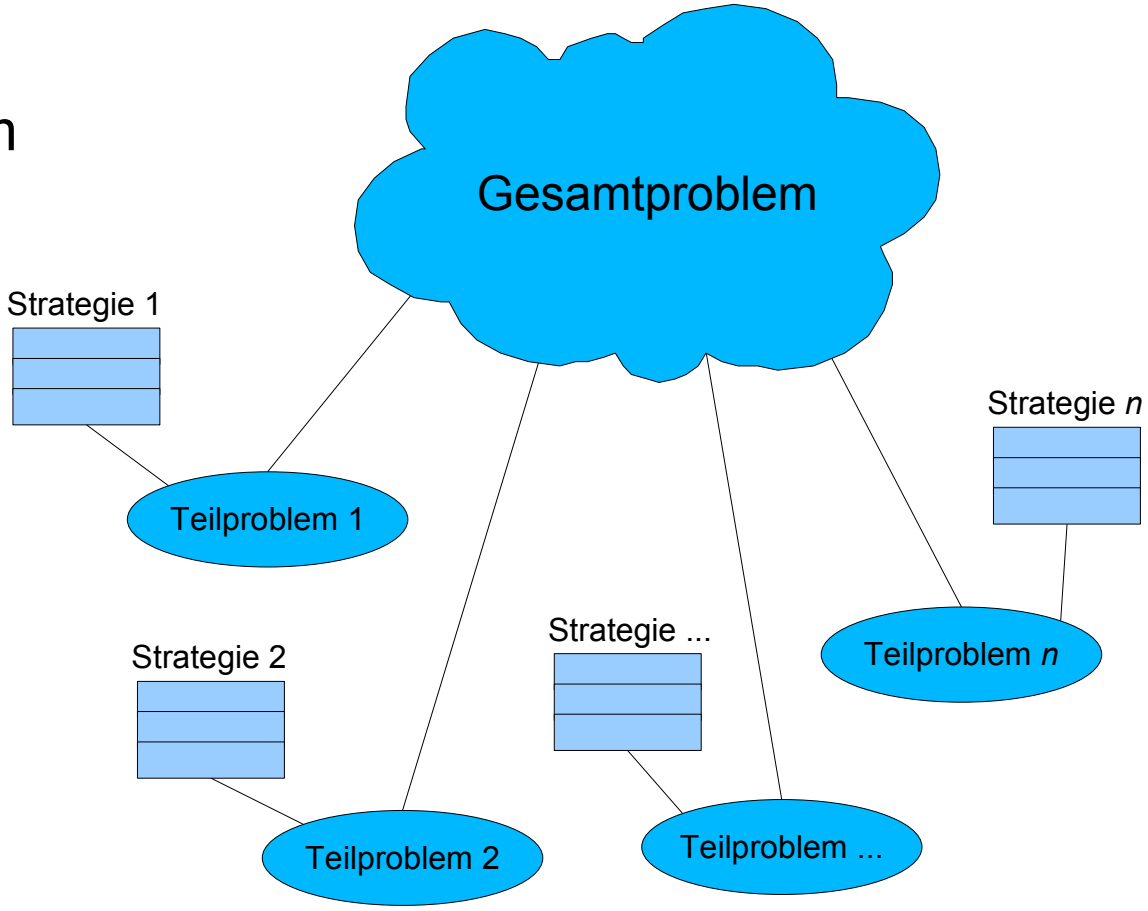
- *Constraint-Manager* verwaltet und steuert, welche Constraints von welcher Strategie verarbeitet werden sollen (*phasenweise*).
- Es müssen keine Lösungsalgorithmen sondern *Lösungsstrategien* zur Constraint-Verarbeitung ausgewählt werden.
- Einfache Austauschbarkeit von Lösungsverfahren ist gewährleistet.



YACS

Abstraktion durch Strategien (2)

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



YACS

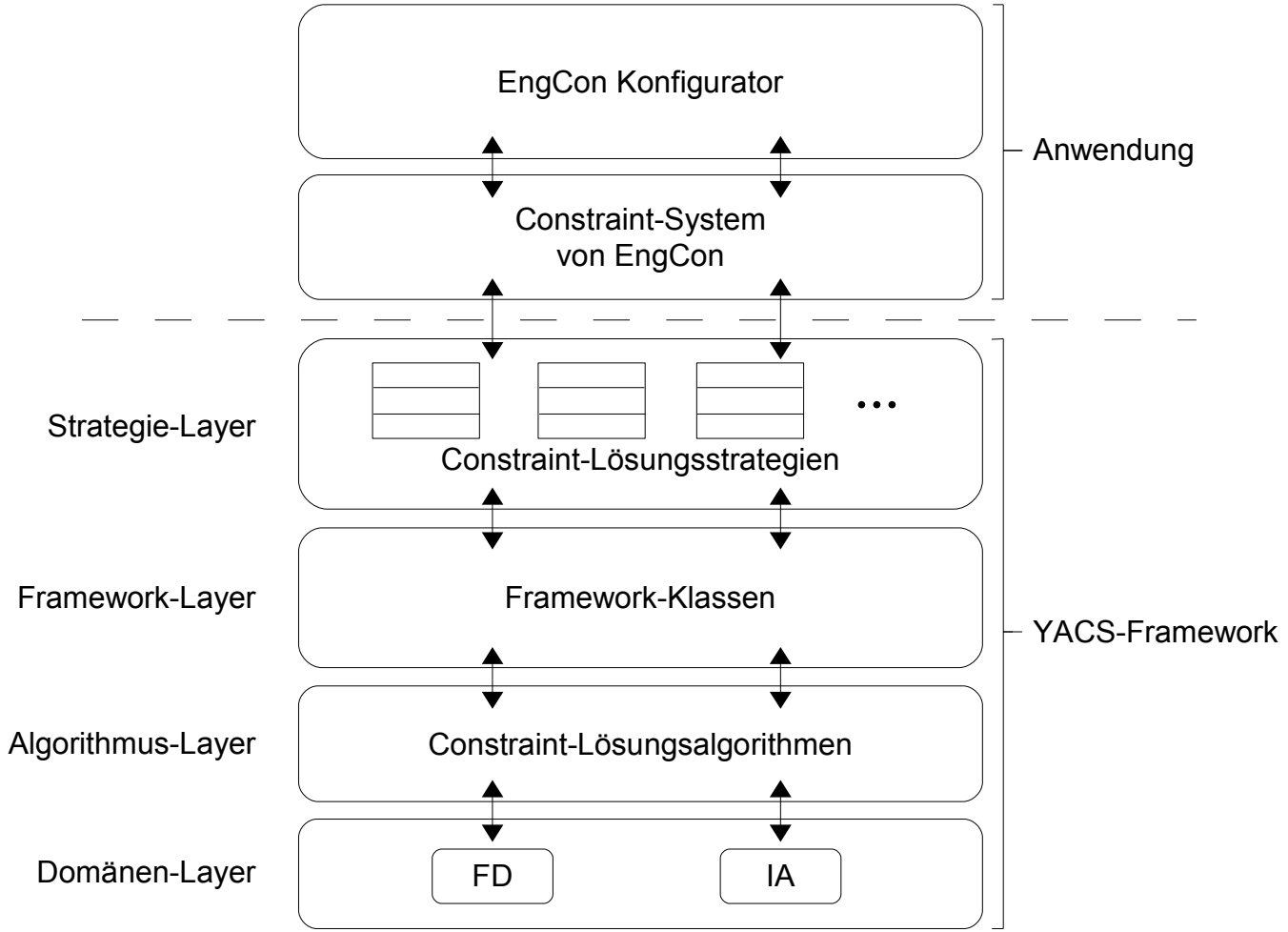
Modularität

- 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.

YACS

Systemarchitektur



YACS

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* Problem-stellungen zu Test- und Demonstrationszwecken *(YacsTester)*:
  - Testen von Constraint-Lösungsverfahren
  - Funktionalität von YACS aufzeigen

YACS

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

- Einführung
- Konzepte zur Constraint-Verarbeitung
- Verfügbare Constraint-Systeme
- Constraint-Lösungsverfahren
- Constraint-Framework YACS
- **Zusammenfassung**

Zusammenfassung (1)

- Constraints:
  - Relationen zur Beschreibung von Abhängigkeiten zwischen Variablen (intensionale/extensionale-Constraints, *global constraints*)
  - Constraint-Propagation
  - Constraint-System
- Konzepte:
  - Allgemeine Konzepte
  - Constraint-Verfahren zur Konfigurierung
  - überbestimmte Probleme
  - weiterführende Konzepte
- Constraint-Systeme: integrierte Systeme, Bibliotheken, Frameworks

Zusammenfassung (2)

- Lösungsverfahren:
  - Konsistenztechniken und Suchverfahren
  - Suche: *Look-Back*- und *Look-Ahead*-Strategien
  - Suche: Ordnungsheuristiken (für Variablen und einzelne Werte)
  - Lösungsverfahren für Probleme mit reellwertigen Intervalldomänen
- YACS:
  - hybrides, strategiebasiertes Framework für Constraint-Solver
  - inkrementell anwachsendes Constraint-Netz wird unterstützt
  - teilproblemübergreifende Metapropagation möglich
  - stringbasierte Schnittstelle vorhanden
  - Prototyp in JAVA bei SourceForge.net verfügbar

Danke für Ihre Aufmerksamkeit!

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
- **Cunis und Günter 1991** Cunis, Roman ; Günter, Andreas: PLAKON – Übersicht über das System. In: Cunis, Roman (Hrsg.) ; Günter, Andreas (Hrsg.) ; Strecker, Helmut (Hrsg.): Das PLAKON-Buch, Ein Expertensystemkern für Planungs- und Konfigurierungsaufgaben in technischen Domänen. Berlin, Heidelberg, New York : Springer Verlag, 1991 (Informatik-Fachberichte, Subreihe Künstliche Intelligenz 266), Kap. 4, S. 37–57. – ISBN 3-540-53683-3
- **Davis 1987** Davis, Ernest: Constraint Propagation with Interval Labels. In: Artificial Intelligence 32 (1987), Juli, Nr. 3, S. 281–331. – ISSN 0004-3702

Literatur (2)

- **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
- **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

Literatur (3)

- **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
- **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

Literatur (4)

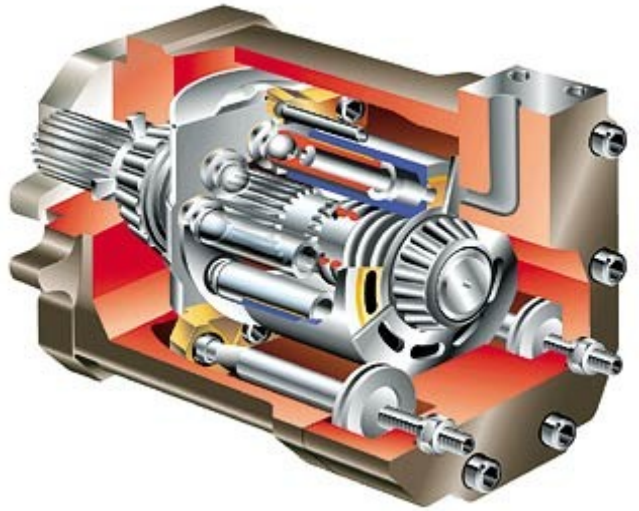
- **Sam-Haroud 1995** Sam-Haroud, Jamila: Constraint Consistency Techniques for Continuous Constraints. Lausanne, Switzerland, Swiss Federal Institute of Technology (EPFL), PhD Thesis No. 1423, 1995. – xviii + 178 S.
- **Sam-Haroud und Faltings 1996** Sam-Haroud, Djamila ; Faltings, Boi V.: Consistency Techniques for Continuous Constraints. In: Constraints, An International Journal 1 (1996), September, Nr. 1–2, S. 85–118. – ISSN 1383-7133
- **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

Konfigurierung (1)

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

Beispiele:

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



Konfigurierung (2)

Konfigurieren ist das Zusammenfügen von Einzelkomponenten in einer Sequenz von einzelnen Konfigurierungsschritten zu einer Gesamtlösung, genannt *Konfiguration* (vgl. Cunis und Günter 1991).

Gekennzeichnet durch:

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

Konfigurierung (2)

Konfigurieren ist das Zusammenfügen von Einzelkomponenten in einer Sequenz von einzelnen Konfigurierungsschritten zu einer Gesamtlösung, genannt *Konfiguration* (vgl. Cunis und Günter 1991).

Gekennzeichnet durch:

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

Methoden zur Konfigurierung

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

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

Konfigurierungssysteme

- R1/XCON
- SICONFEX
- MMC-CON
- ALL-RISE

- ConBaCon
- CAWICOMS

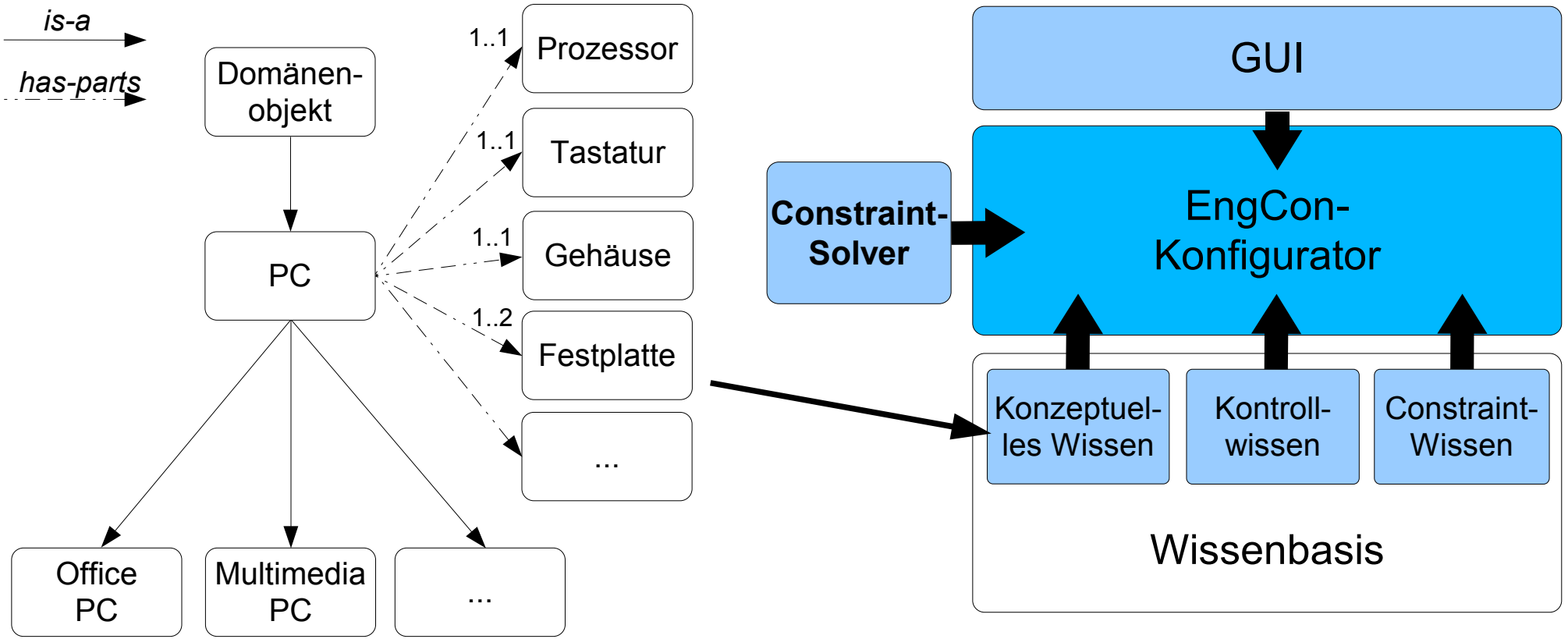
- PlaKon
- KonWerk
- EngCon

- CAS-Konfigurator
- Cameleon EPOS
- camos.Configurator
- COMIX
- COSMOS
- KIKon
- SAP SCE
- Baan SalesPlus
- Tacton Configurator
- Lava
- ILOG (J)Configurator

EngCon: Historie

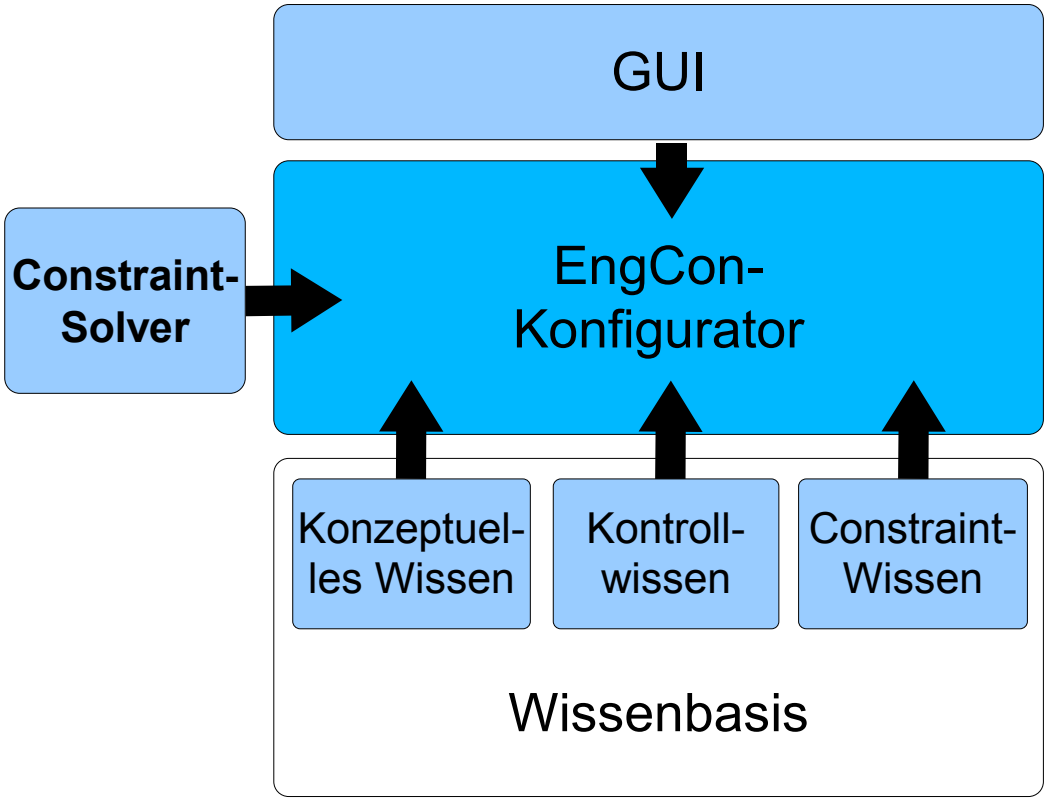
- Vorgänger: *Plakon*, *Konwerk*, entwickelt (u.a.) an der Universität Hamburg mit Unterstützung des BMBF
- prototypische Umsetzung auf moderne *JAVA*-Umgebung durch das TZI
- Überführung in ein Produkt durch die encoway GmbH: „Drive Solution Designer“ (DSD)
- Auszeichnung des DSD mit dem „Innovative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

Konfigurierung in EngCon



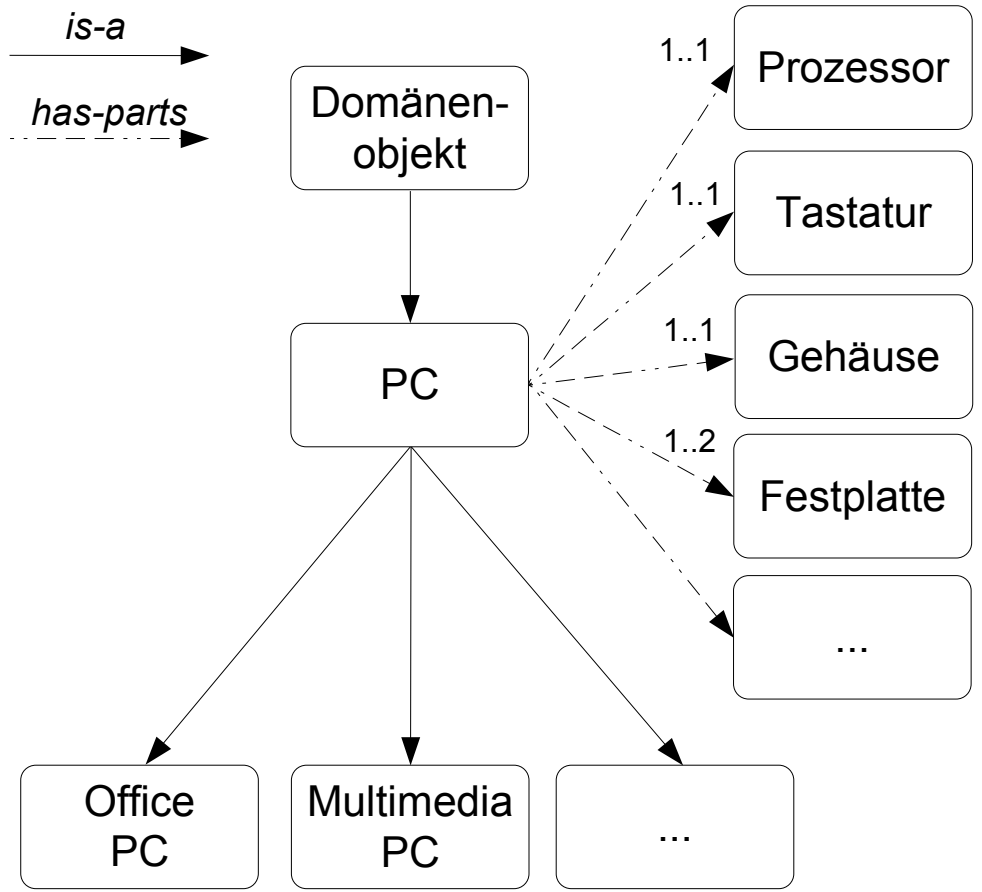
EngCon: Architektur

- domänenunabhängiges, *strukturbasiertes* Konfigurierungswerkzeug
- Bildung von Inferenzen aufgrund der wissens-basierten (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).

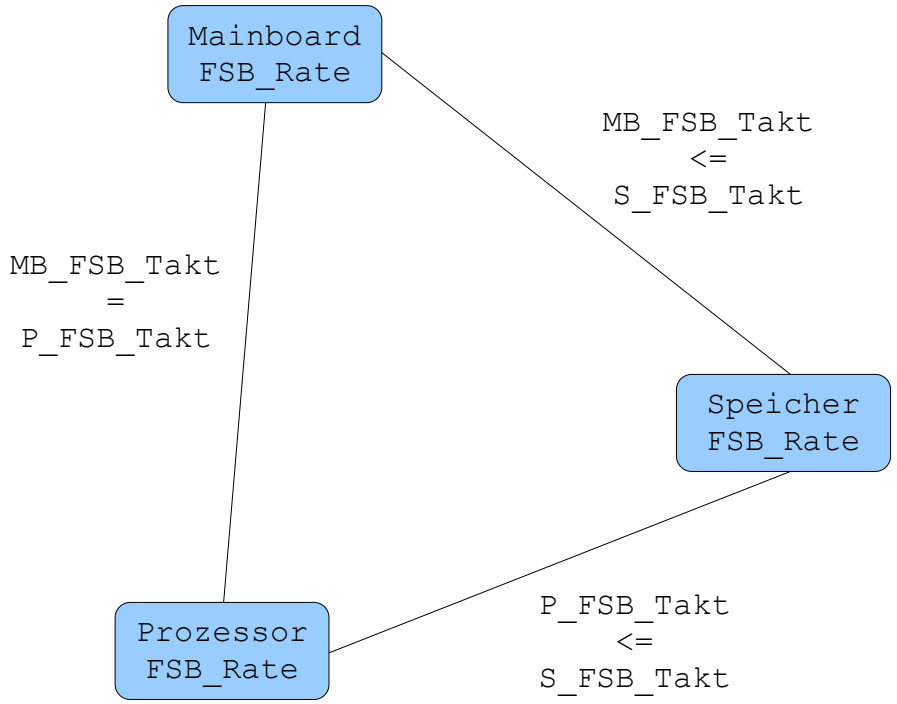
M_FSB	P_FSB	S_FSB
66	66	66
66	66	100
66	66	133
100	100	100
100	100	133
133	133	133

- **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.



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 infiniten 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

Anforderungen an die Problemlösung (2)

Nichtfunktionale Anforderungen:

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

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 '77a; Freuder '78; Mohr u. Henderson '86; Van Hentenryck et al. '92; Bessière u. Cordier '93; Bessière '94a; Bessière et al. '95, 1999a; Chmeiss u. Jégou '96a,b, '98; Bessière u. Régin '01a,b; Zhang u. Yap '01; van Dongen '02a,b; ...)
  - *(i,j)-consistency*, *inverse consistency*, *path inverse consistency*, *neigborhood 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. Régin '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. Régin '96; Chen u. van Beek '01; Grant u. Smith '96; Prosser 93a,b; Prosser 95a,b; ...)

Constraint-Lösungsverfahren (2)

... Fortsetzung: Klassische Constraint Satisfaction Probleme (CSP)

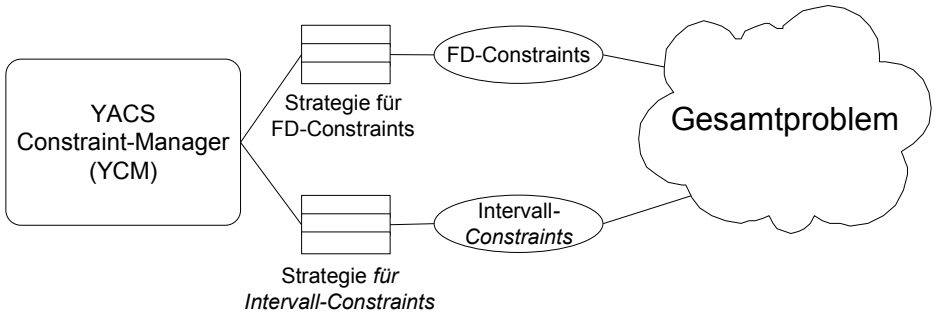
- **Ordnungsheuristiken:**
  - Variablenordnungsheuristiken: *fail first*, *minimal width ordering*, *maximum cardinality ordering*, *maximum degree ordering*, *minimal bandwidth ordering* (vgl. Haralick u. Elliot '80; Dechter u. Meiri '94; Frost u. Dechter '94, '95; ...)
  - Werteordnungsheuristiken: *min-conflicts*, *max-domain-size*, *weighted-max-domain-size*, *point-domain-size* (vgl. Minton et al. '92; Frost u. Dechter '95; ...)

Intervall Constraint Satisfaction Probleme (ICSP)

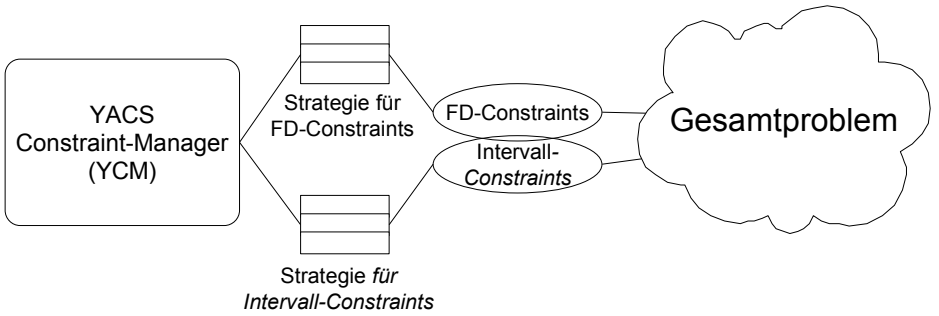
- *interval splitting* (vgl. Cleary '87)
- *label inference* (vgl. Davis '87)
- *tolerance propagation* (vgl. Hyvönen '89; Hyvönen '91; Hyvönen '92)
- *2B-, 3B-, kB-consistency* / *hull consistency* (vgl. Lhomme '93; Lhomme et al. '96, '98; Bordeaux et al. '01; Lebbah u. Lhomme '98, '02; ...)
- *box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

Hybridizität versus Heterogenität

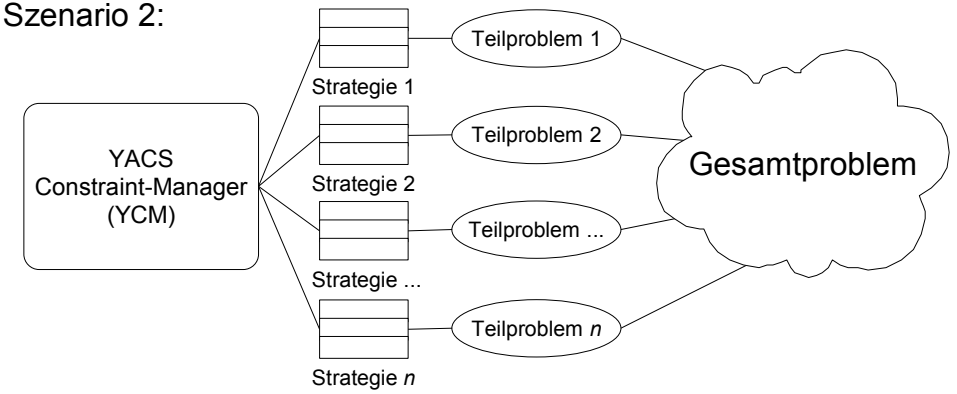
Szenario 1:



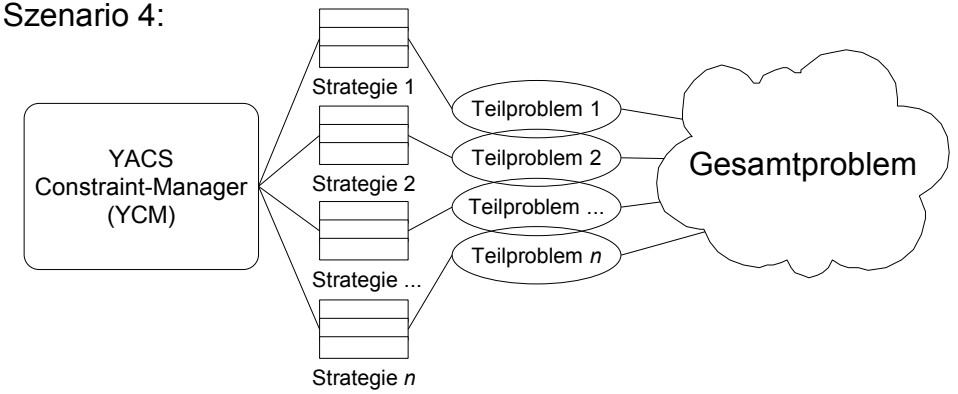
Szenario 3:



Szenario 2:

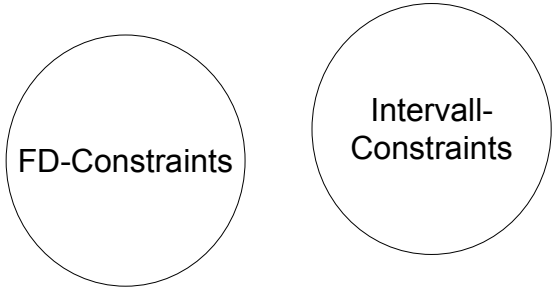


Szenario 4:

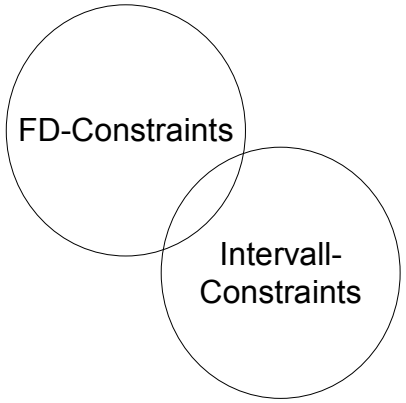


Hybridizität versus Heterogenität

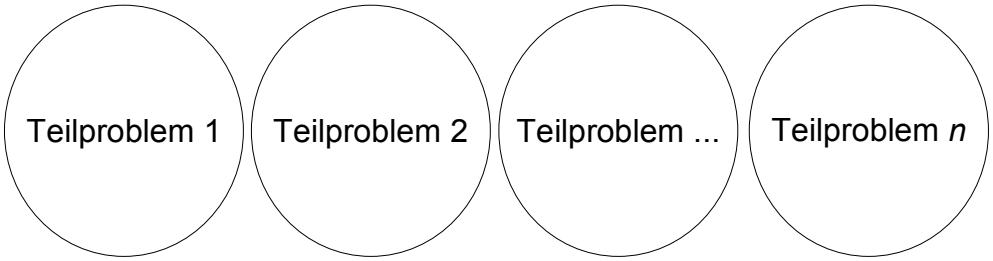
Szenario 1:



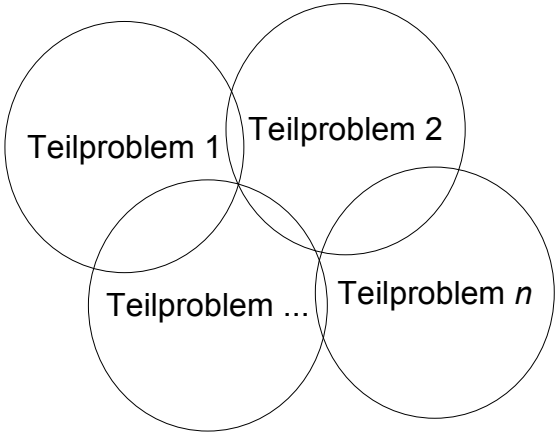
Szenario 3:



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)]

Constraints versus Regeln (1)

Constraints:

- *ungerichtet* → multidirektionale Auswertung
- *Black-Box*-Prinzip: deklarativer Formalismus, frei von Kontrollwissen
  - Reihenfolge nicht festgelegt und für das Ergebnis nicht relevant
  - Wissensingenieur spezifziert Abhängigkeiten und überlässt die Auswertung dem System
- wiederholte Propagation innerhalb eines Constraint-Netzes bis Konsistenz erreicht wird
  - frühe Konflikterkennung

Regeln:

- *gerichtet* → unidirektionale Auswertung
- Vermischung von Domänen- und Kontrollwissen innerhalb der Regeln
- einmalige Ausführung (wenn Bedingungsteil erfüllt) oder Kontrolle bestimmt Zeitpunkt der Ausführung (Vermischung von Randbedingungen und Kontrollwissen)
  - späte Konflikterkennung

Constraints versus Regeln (2)

Beispiel für *regelbasiertes* System:

- R1/XCon (80er Jahre), beinhaltete 11.500 Regeln, von denen jedes Jahr ca. die Hälfte aus Wartungsgründen modifiziert werden musste.

Beispiel für *constraint-basiertes* System:

- ILOG Configurator/JConfigurator, basiert auf ILOG Solver/JSolver und *Generative Constraint Satisfaction* (GCSP).

OOP-Frameworks

„Because frameworks require iteration and **deep understanding of an application domain**, it is hard to create them on schedule. Thus, framework design should never be on the critical path of an important project. This suggests that they should be developed by advanced development or research groups, not by product groups. On the other hand, framework design must be closely associated with the application developers because framework design requires **experience in the application domain**.“ Ralph E. Johnson, *Components, Frameworks, Patterns*, ACM SIGSOFT Software Engineering Notes, 22 (1997), Nr. 3, S. 10-17

Vorteile von YACS

- hybrides System
- inkrementelle Constraint-Verarbeitung
- Flexibilität durch Constraint-Lösungsstrategien
- Modularität durch Framework-Architektur
- stringbasierte Schnittstelle

Eigenschaften (wenn vorhanden) bei vergleichbaren Systemen nicht innerhalb eines Systems vereint.

Vorteile für den Anwender (1)

- vollwertige Constraint-Verarbeitung mit allen sich daraus ergebenen Vorteilen
  - *ungerichtet* → multidirektionale Auswertung
  - *Black-Box*-Prinzip
  - leistungsfähige dynamische und generische Constraint-Komponente in EngCon bereits vorhanden (u.a. *Pattern-Matching*)
  - größere Marktdurchdringung durch aktuelle Technologien
- strategische Vorteile einer Eigenlösung („Unabhängigkeit“)
- keine Lizenzkosten für externe Constraint-Komponente
  - Produkte lassen sich zu einem anderen Preis anbieten
- *Know-How* für Erweiterungen/Optimierungen vor Ort

Vorteile für den Anwender (2)

- Flexibilität/Modularität: beschleunigter Entwicklungsprozess
- einfache Nutzung der Implementierung (Akzeptanz):
  - stringbasierte Constraint-Schnittstelle
  - XML-Strategien
- YACS liegt im Quellcode vor:
  - erweiterbar
  - adaptierbar
  - skalierbar
- JAVA-Implementierung ist einfach integrierbar und plattformunabhängig

Vorteile für den Kunden

- niedrigere Kosten
  - weil Eigenlösung / keine Lizenzkosten (insbes. „Massenlizensierung“)
- beschleunigte Produktentwicklung
  - individualisierte, problemabhängige Solver (Flexibilität/Modularität)
- leistungsfähigeres System
  - angepasst an die jeweilige Problemstellung
  - effizientere Konfigurierung durch effiziente Solver
  - Verarbeitung spezieller Constraint-Domänen möglich
  - Skalierbarkeit

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).