Fachbereich Mathematik / Informatik
Studiengang Informatik

Vorschlag für eine Diplomarbeit

(Erweiterte Version)

YACS: Ein hybrides Framework für Constraint-Solver
zur Unterstützung wissensbasierter Konfigurierung
in technischen Domänen

Wolfgang Runte
Matrikelnr.: 1140989

Stand: 13. Juni 2005

Inhaltsverzeichnis

1 Motivation und Problemstellung **1**

2 Anforderungen an die Problemlösung **2**

3 Der Weg zur Lösung **3**
3.1 Klassische Constraint Satisfaction Probleme 4
3.2 Intervall Constraint Satisfaction Probleme 6
3.3 Der Framework-Ansatz 7
3.4 Constraint-Lösungsstrategien 8
3.5 Das YACS-Framework 9
3.5.1 Architektur 10
3.5.2 Ausführungsmodelle 11
3.5.3 Lösen von heterogenen Constraint-Problemen 19
3.6 Alternative Ansätze 21
3.7 Bewertung 24

4 Zusammenfassung **26**

Literaturverzeichnis **27**

1 Motivation und Problemstellung

Das wissensbasierte Konfigurierungswerkzeug EngCon verwendet u. a. Funktions- und Prädikat-Constraints[1] zur Beschreibung von Abhängigkeiten zwischen Konzepten der Wissensbasis (vgl. Hollmann et al. 2000; Ranze et al. 2002). Die Auflösung der Abhängigkeiten wird von einem Constraint-Solver geleistet (vgl. Syska und Cunis 1991). Dieser Constraint-Solver wird nicht von EngCon implementiert, sondern ist derzeit eine von einem Fremdhersteller eingebundene, externe Komponente.[2] Er ist ausschließlich für die Propagation und Auflösung von Constraints mit reellwertigen Intervalldomänen geeignet. Zudem ist die Anbindung an EngCon auf einen einzigen Constraint-Solver beschränkt und damit sehr unflexibel. Zur Erweiterung der Anwendungsmöglichkeiten des Konfigurierungswerkzeugs EngCon sowie zur Verbesserung der Effizienz ist ein Austausch oder eine Eigenimplementierung des Constraint-Solvers wünschenswert.

Der benötigte Constraint-Solver muss arithmetische Funktionen zur Berechnung der intensional in Form von algebraischen Ausdrücken formulierten Constraints innerhalb von EngCon bieten. Neben klassischen Constraint-Solvern zur Behandlung von finiten Domänen sind für die Constraint-Verarbeitung in EngCon Constraint-Lösungsmethoden für infinite, d. h. reellwertige Intervalldomänen erforderlich. Ein entsprechender Constraint-Solver muss einen hohen Präzisionsgrad durch Intervallarithmetik aufweisen (z. B. für Anwendungen im Maschinenbau) sowie unabhängig von der Constraint-Domäne ein inkrementell anwachsendes Constraint-Netz propagieren können.

Aufgrund der hohen Anforderungen von EngCon, insbesondere in Bezug auf unterschiedliche zu verarbeitende Wertedomänen, existiert kein Constraint-Solver, der sämtliche Anforderungen vollständig erfüllt. Darüber hinaus ergeben sich Anforderungen an den Constraint-Lösungsmechanismus häufig in Abhängigkeit von der Aufgabenstellung des jeweiligen Konfigurierungsproblems in EngCon. Spezielle benötigte Eigenschaften sind i. A. daher a priori nicht bekannt. Neben stabilen Constraint-Lösungsverfahren, die eine hohe Effizienz für möglichst viele Problemstellungen bieten, ist es deshalb erforderlich, problemabhängig unterschiedliche Verfahren nutzen zu können. Benötigt wird neben zusätzlichen Constraint-Lösungsmechanismen eine Komponente, an der sich flexibel unterschiedliche Constraint-Solver mit verschiedenen Eigenschaften, sowohl bezogen auf die Lösungsverfahren als auch auf die zu verarbeitenden Wertedomänen, einbinden lassen. Diese Constraint-Solver können eigenimplementierte aber auch Fremdsysteme sein. Eigene Implementierungen hätten den Vorteil der leichteren Erweiterbarkeit,[3] zudem entfällt

[1] *constraint* (engl.): Einschränkung, Beschränkung, Restriktion

[2] Die Vorgänger von EngCon, die Konfigurierungswerkzeuge KonWerk und PlaKon, wurden in Common Lisp bzw. CLOS (*C*ommon Lisp Object System) implementiert. Hier fand hier ein eigens ins Lisp implementiertes Modul zur Durchführung von arithmetischen Berechnungen Verwendung, welches die von Davis (1987) und Hyvönen (1992) beschriebenen Verfahren zur Propagation von Intervall-Constraints anwendet (vgl. Gülden 1993).

[3] Denkbare Erweiterungen, die allerdings nicht in dieser Arbeit behandelt werden, wäre z. B. die Möglichkeit während der Laufzeit des Systems einzelne Constraints zurücknehmen zu können („Constraint-Relaxierung“) oder die Möglichkeit zur Beschreibung von „Constraint-Hierarchien“, in denen „harte“ und „weiche“ Constraints definiert werden können. Je nachdem auf welcher Stufe innerhalb der Hierarchie sich ein Constraint befindet, muss oder kann es erfüllt sein, um zu einer gültigen Lösung zu gelangen.

der bei geeigneten Constraint-Solvern von Fremdherstellern ggf. hohe Integrationsaufwand der Komponenten.

2 Anforderungen an die Problemlösung

In vielen Anwendungsbereichen lassen sich Problemstellungen als Constraint-Problem formulieren. Aufgrund einer Vielzahl unterschiedlicher Domänen und der Vielfältigkeit der Anwendungsszenarien ist es notwendig, an die jeweilige Domäne angepasste Constraint-Lösungsverfahren einzusetzen. Zur Behandlung unterschiedlicher Constraint-Domänen innerhalb des Constraint-Systems von EngCon ist eine Kooperation mehrerer Constraint-Solver erforderlich. Diese Kooperation muss durch eine Komponente geleistet werden, mit der sich unterschiedliche Constraint-Solver je nach Bedarf und domänenspezifisch einsetzen lassen. Die Modularität des Entwurfs ist dabei entscheidend für die Austauschbarkeit einzelner Komponenten.

Im Detail stellen sich die folgenden Anforderungen an die in EngCon zu integrierende Constraint-Komponente:

Schnittstelle: Das Constraint-System muss unabhängig vom restlichen System funktionieren (*Black-Box*-Prinzip). Von außen darf lediglich beeinflusst werden, welche Constraint-Propagationsmethoden zum Einsatz kommen, nicht jedoch inhaltliche Details der Propagation. Die neue Komponente muss sich dazu in den vorhandenen Constraint-Mechanismus von EngCon einfügen (konzeptionelle Constraints, Pattern-Matcher). Die bereits existierenden Constraint-Propagationsmechanismen von EngCon bleiben unangetastet (Tupel-Constraints, Java-Constraints). Die Anbindung der Constraint-Solver muss über eine stringbasierte Schnittstelle erfolgen.

Dynamik: Die Constraint-Solver müssen in der Lage sein, ein inkrementell anwachsendes Constraint-Netz zu verarbeiten. Aufgrund der interaktiven Konfigurierung mit EngCon und der Möglichkeit von Konfigurierungskonflikten müssen sich zudem Zustände des Constraint-Netzes, die in der Vergangenheit liegen, wiederherstellen lassen.

Wertedomänen: Während einer Konfigurierung in EngCon müssen sowohl Elemente aus diskreten Werte*mengen* ausgewählt, als auch kontinuierliche Werte*bereiche* eingeschränkt werden können. Ein Constraint-System, welches ebensolche Konfigurierungsschritte unterstützt, muss infolgedessen Propagations- und Lösungsmechanismen sowohl für finite als auch infinite Domänen aufweisen.

Lösungsverfahren: Die Constraint-Lösungsverfahren müssen in der Lage sein, beliebige algebraische Constraint-Ausdrücke in Form von Gleichungen und Ungleichungen zu verarbeiten. Es sollte weiterhin möglich sein, n-stellige Constraints (bezogen auf die Anzahl der Variablen), nichtlineare Constraints und zyklische Strukturen des Constraint-Netzes zu behandeln.

Präzision und Effizienz: Der Erfordernis an die Präzision von Lösungen steht oftmals der Wunsch nach Effizienz bei den zum Einsatz kommenden Lösungsverfahren und

die Komplexität der Problemstellung entgegen. Das Constraint-System sollte demnach einen Kompromiss zwischen Effizienz und Qualität bzgl. der zu berechnenden Lösungen erlauben. Je nach Anwendungsdomäne, den vorhandenen Constraints, der bevorzugten Präzision und der benötigten Effizienz kann es sinnvoll sein, unterschiedliche Verfahren zur Auswertung der Constraints einzusetzen. Bei Einschränkungen bzgl. der Präzision von Lösungsverfahren ist zu beachten, dass keine möglichen Lösungen verloren gehen dürfen, da sonst die Vollständigkeit des Verfahrens nicht gewährleistet ist.

Hard- und Software-Umgebung: Die Anbindung bzw. Implementierung muss unter den von EngCon geforderten Bedingungen erfolgen, d. h. auf einer Microsoft-Windows-NT-Plattform auf x86-Hardware.[4] Vorzugsweise sollte eine Implementierung, ebenso wie die von EngCon, in der Programmiersprache Java erfolgen. Dies garantiert zudem weitestgehende Plattformunabhängigkeit der Constraint-Komponente und würde die Wiederverwendbarkeit auch für andere Projekte erhöhen.

Für eine Eigenentwicklung ist eine weitere Anforderung an den Entwurf die Anbindung eines ebenfalls austauschbaren Werkzeugs für intervallarithmetische Berechnungen. Derartige Operationen werden standardmäßig von den wenigsten Programmiersprachen unterstützt. Die Bibliothek IAMath[5] von *Timothy J. Hickey*[6] bietet sich für eine Anbindung an EngCon an. Die frei verfügbare Bibliothek ermöglicht grundlegende intervallarithmetische Operationen und Funktionen und ist ebenso wie EngCon vollständig in Java implementiert. Um eine eventuelle Weiterentwicklung nicht einzuschränken ist es auch hier erforderlich, sich nicht auf eine bestimmte Bibliothek festzulegen. Stattdessen ist im Entwurf eine Schnittstelle mit einer entsprechenden Kapselung vorzusehen.

Ferner sollte die eigenständige Wiederverwendbarkeit einer zu entwickelnden Constraint-Komponente gewährleistet sein. Dafür muss sie eine allgemeine Architektur und eine Schnittstelle aufweisen, die eine Nutzung auch in anderen Systemen losgelöst von EngCon ermöglicht.

3 Der Weg zur Lösung

Die in dieser Arbeit durchgeführte Evaluierung bestehender Werkzeuge zur Behandlung von Constraint-Problemen hat ergeben, dass diese nicht die konkreten Anforderungen bzgl. einer Integration in das Konfigurierungswerkzeug EngCon erfüllen. Es konnte kein System ermittelt werden, welches zusammen die relevantesten Forderungen erfüllt: Es muss „hybrid“ sein und damit sowohl finite Domänen als auch reellwertige Intervalle unterstützen, beliebige Relationen in Form von algebraischen Constraints sowie den inkrementellen Aufbau des Constraint-Netzes ermöglichen und von der Programmiersprache Java aus unter Microsoft Windows nutzbar sein. Eine einfache, stringbasierte Schnittstelle wird von keinem der betrachteten Systeme angeboten.

[4] Die Windows-NT-Produktfamilie von Microsoft für x86-Systeme besteht derzeit aus Windows XP, Windows 2000 und Windows NT 4.0.

[5] `http://interval.sourceforge.net/interval/java/ia_math/README.html`

[6] `http://www.cs.brandeis.edu/~tim`

Auch wenn im Bereich des *Constraint Logic Programming* (CLP) durchaus hybride Systeme existieren (z. B. BNR Prolog, CLP(BNR), ECL^{i}PS^e), besteht das Problem, dass sich diese Systeme i. d. R. nur schwer adaptieren lassen, da eine Java-Schnittstelle häufig nicht enthalten ist, bzw. weil diese Systeme für die (logische) Programmierung mit Constraints ausgelegt sind.[7] Damit wird zwar üblicherweise eine Schnittstelle zur Nutzung von in diesem Kontext benötigten, *global constraints* angeboten, aber keine einfach zu nutzende (stringbasierte) Schnittstelle für beliebige Relationen. Für die Integration in EngCon sind allerdings generische Constraint-Solver zur Auflösung derartiger Constraints erforderlich, anstatt generische, d. h. wiederverwendbare, Constraints aus dem Bereich *Constraint Programming* (CP).

Eine Schnittstelle zur Anbindung an EngCon muss in jedem Fall entwickelt und implementiert werden. Die durch diese Schnittstelle bereitgestellte Umgebung ist maßgeblich für die Integration von Constraint-Verfahren in EngCon, seien es Fremdsysteme oder Eigenentwicklungen. Der Nutzen einer möglicherweise komplizierten Anbindung eines Fremdsystems an diese Schnittstelle kann sich dagegen im Vergleich zur Eigenimplementierung von Lösungsalgorithmen als recht gering erweisen. Besonders wenn die im Fremdsystem vorhandenen Constraint-Lösungsverfahren nicht sehr anspruchsvoll sind, wie z. B. in JACK[8] (Abdennadher et al. 2001, 2002a, b) und GNU Prolog[9] (Diaz und Codognet 2001). Dabei würde von einem Fremdsystem ausschließlich der Zugriff auf dessen Constraint-Verarbeitung benötigt. Relevant für EngCon ist nicht die Deklarativität z. B. einer Prolog-Umgebung, sondern der stringbasierte Zugriff auf deren Constraint-Lösungsverfahren und die möglichst flexiblen Einbettung in EngCon.[10]

Um dies zu ermöglichen scheint eine Eigenentwicklung am Ehesten geeignet. Die Erfahrungen hinsichtlich der eingeschränkten Funktionalität und Skalierbarkeit, die bei einer prototypischen Integration von GNU Prolog in EngCon gemacht wurden, bestärken dies: Der Aufwand für die komplexe Integration eines Fremdsystems gestaltet sich u. U. größer, als die Implementierung eigener Algorithmen zur Constraint-Verarbeitung. Vorausgesetzt das entsprechende *Know-How* bzgl. der entsprechenden Constraint-Lösungsverfahren ist vorhanden. Ein Teil der Arbeit wird sich daher mit Verfahren zum effizienten Auflösen von Constraint-Problemen, sowohl über finite als auch über infinite Domänen, beschäftigen.

3.1 Klassische Constraint Satisfaction Probleme

Constraints über finite Domänen sind ein klassisches Gebiet der KI, auf dem bereits seit längerer Zeit geforscht wird. Sie kommen in vielen Anwendungen zum Einsatz, die Problemstellungen mit endlichen Wertebereichen beinhalten. Für klassische *Constraint Satis-*

[7] Zudem sind CLP-Systeme z. T. nur unter Unix-Systemen nutzbar, und nicht wie erforderlich unter Microsoft Windows. Ob diese Systeme in einer Umgebung wie Cygwin in der erforderlichen Stabilität nutzbar sind (vorausgesetzt der Quellcode für den notwendigen Kompilierungsvorgang ist verfügbar), kann nicht garantiert werden, wenn nicht entsprechende Pakete verfügbar sind.

[8] `http://www.pms.ifi.lmu.de/software/jack/`

[9] `http://gprolog.inria.fr`

[10] Eine deklarative Prolog-Schnittstelle zur Programmierung mit Constraints (CP) ist für den konkreten Einsatz in EngCon grundsätzlich eher ungeeignet und würde eine Nutzung des entsprechenden Constraint-Systems durch den entstehenden Overhead aufwendiger machen.

faction Probleme (CSP) mit finiten Domänen wurde daher eine Vielzahl an Constraint-Lösungsmethoden und Heuristiken entwickelt (vgl. Barták 1999a, b, 2001; Dechter 1992, 1999; Dechter und Rossi 2003; Güsgen 2000; Kumar 1992; Tsang 1993). Grundsätzlich wird zwischen Konsistenz- und (systematischen) Suchverfahren unterschieden. Durch Konsistenzverfahren allein ist es außer in Ausnahmefällen nicht möglich, Lösungen für CSPs zu finden. Stattdessen werden Konsistenzverfahren häufig als Preprozessing bzw. während eines Suchverfahrens, währenddessen die eigentlichen Lösungen generiert werden, als *look-ahead*-Mechanismus eingesetzt. Um eine weitere Optimierung des Laufzeitverhaltens zu erreichen, können Suchverfahren durch Heuristiken zur Variablen- und Werteauswahl unterstützt werden.

Eine besondere Stellung nehmen binäre CSPs ein, da viele der existierenden Lösungsalgorithmen auf binäre Constraints beschränkt sind. Verbunden wird dies häufig mit dem Hinweis, dass sich die Verfahren grundsätzlich auf n-äre CSPs verallgemeinern lassen. Aufgrund der hohen Komplexität n-ärer Constraints sind tatsächlich nur wenige Algorithmen verallgemeinert worden. Die Möglichkeit zur Binärisierung n-ärer Constraint-Netze verbessert die Situation nicht wirklich, da durch die Umwandlung in eine extensionale Repräsentation in Form von „umfassenden Variablen“ ein sehr hoher Platzbedarf verbunden mit relativ hohem Berechnungsaufwand entsteht. Die Binärisierung n-ärer CSPs, um anschließend binäre Lösungsalgorithmen verwenden zu können, ist daher nur für überschaubare Problemstellungen zu empfehlen.

Im Rahmen dieser Arbeit soll eine Grundausstattung an soliden Lösungsverfahren entstehen, die ein stabiles Laufzeitverhalten für möglichst viele Problemstellungen bieten. Verfahren zur Berechnung globaler Konsistenz sind i. A. zu aufwendig und zu ineffizient. Kantenkonsistenz dagegen ist ein weit verbreitetes Verfahren, welches einen angemessenen Kompromiss zwischen Berechnungsaufwand und der Menge der gefilterten Werte darstellt. Aufgrund der notwendigen extensionalen Repräsentation für den `AC-4`-Algorithmus und höher (durch *support*-Einträge), sowie aufgrund des durchschnittlich besseren Laufzeitverhaltens (vgl. Wallace 1993) sollte dem `AC-3`-Algorithmus bzw. einer der weiterentwickelten Varianten (`AC-8`, `AC-2000`, `AC-2001`, `AC-3.1`, `AC-`3_d) der Vorzug gegeben werden. Pfadkonsistenz ist für viele Problemstellungen zu aufwendig. Für die vorhandenen Algorithmen ist zudem eine Matrix-Repräsentation der Constraints erforderlich, was das Verfahren aufgrund des entstehenden Ressourcenbedarfs ungeeignet für umfangreiche Problemstellungen macht. Durch die Matrix-Repräsentation sind Algorithmen zur Herstellung von Pfadkonsistenz überdies auf binäre Constraints beschränkt. Ausgeschlossen werden soll ein derartiger Constraint-Solver allerdings nicht, denn u. U. ist ein Konsistenzgrad größer als Kantenkonsistenz für bestimmte, überschaubare Problemstellungen durchaus sinnvoll in Form einer Effizienzverbesserung. Weitere Konsistenzarten sind Verallgemeinerungen oder spezialisierte Varianten, die bspw. dazu dienen, möglichst frühzeitig Inkonsistenzen festzustellen oder geringfügig höhere bzw. niedrigere Konsistenzgrade anzubieten als die „Standardverfahren“. Diese Konsistenzen können bei speziellen Problemstellungen sinnvoll sein, sie werden allerdings nicht schwerpunktmäßig in dieser Arbeit behandelt.

Neben Filter- bzw. Konsistenzalgorithmen müssen durch die Constraint-Komponente verschiedene Suchverfahren angeboten werden, in deren Verlauf Lösungen ermittelt werden, sofern vorhanden. Für einfache Problemstellungen sollte, um unnötigen Overhead

zu vermeiden, einfaches, chronologisches Backtracking nutzbar sein. Außerdem sollte für die Verarbeitung komplexer Constraint-Netze eine (konfliktbasierte) Backjumping-Variante angeboten werden, denn Backjumping ist, im Gegensatz zu anderen Suchverfahren mit *look-back*-Strategie wie Backchecking und Backmarking, mit dynamischen und damit leistungsfähigen Heuristiken zur Variablenordnung einsetzbar. Diese Suchverfahren sollten außerdem mit Filteralgorithmen als *look-ahead*-Mechanismus kombiniert angeboten werden (*Forward Checking*, *Maintaining Arc Consistency*). Zur Variablenordnung sollte eine Kombination des *fail-first*-Prinzips und der *maximum-degree-ordering*-Heuristik ein stabiles Laufzeitverhalten bieten. Heuristiken zur Werteauswahl bringen, begründet durch den hohen Aufwand jeden einzelnen Wert zu betrachten, im Schnitt weniger Gewinn und müssen daher nicht vorrangig behandelt werden. Die Gefahr, durch derartige Ordnungsheuristiken unnötigen Overhead zu produzieren, ist ungleich größer.

Grundsätzlich sollten die oben genannten Verfahren auch für n-äre Constraint-Netze anwendbar sein, auch wenn der Einsatz aufgrund der hohen Komplexität derartig formulierter Problemstellungen nicht zu empfehlen ist.

3.2 Intervall Constraint Satisfaction Probleme

Die Domänen reellwertiger algebraischer Constraints innerhalb von *Intervall Constraint Satisfaction Problemen* (ICSP) werden in Form von Intervallen mit oberer und unterer Grenze definiert (vgl. Benhamou 2001; van Emden 2002). Kombinatorische Verfahren zur Aufzählung möglicher Lösungen versagen hier. Allerdings lassen sich basierend auf intervallarithmetischen Grundlagen Konsistenzalgorithmen in abgewandelter Form effizient anwenden. Die Wertebereiche der Constraint-Variablen werden hierbei möglichst weit eingeschränkt, ohne mögliche Lösungen des Problems zu verlieren. Durch ein Splitting der Wertebereiche können in einem weiterführenden Suchvorgang punktgenaue Lösungen ermittelt werden (vgl. Cleary 1987). Aufgrund einer Vielzahl von Splitting-Möglichkeiten führt dieses Verfahren bei umfangreicheren Problemen allerdings schnell zu einer kombinatorischen Explosion (vgl. Lhomme 1993, S. 236 f.).

Basierend auf dem Waltz-Filteralgorithmus zur Herstellung von Kantenkonsistenz (vgl. Waltz 1975) wurde von Davis (1987) das sog. Label Inference zur Einschränkung von Intervallgrenzen entwickelt. Fortführungen davon, die eine Zerlegung von Constraints in primitive Constraints vornehmen und dadurch eine generelle Anwendbarkeit ermöglichen, sind die Toleranzpropagation von Hyvönen (1989, 1991, 1992) und Hull-Konsistenz bzw. 2B-Konsistenz von Lhomme (1993).[11] Während im Rahmen der Toleranzpropagation von Hyvönen (1992) Möglichkeiten aufgezeigt werden, mit denen durch (*dynamic*) Splitting und der Eliminierung von Zyklen im Constraint-Netz ggf. globale Konsistenz erreicht werden kann, werden von Lhomme (1993) im Rahmen der Hull-Konsistenz höhere Konsistenzgrade durch 3B- bzw. kB-Konsistenz definiert.

Eine weiterführende Methode von Benhamou et al. (1994a, b) zur Herstellung von Box-Konsistenz setzt numerisch-mathematische Verfahren ein. Eingebettet in einen Waltz-ähnlichen Filteralgorithmus wird das Newton-Intervallverfahren, zur Einschränkung der

[11] Der durch lokale Toleranzpropagation erreichte Konsistenzgrad entspricht 2B-Konsistenz.

Intervallgrenzen genutzt. Auch hier lassen sich ggf. durch Splitting kanonische Lösungen generieren.

Ein weiterer Ansatz, die 2^k-Baum-Methode, ist in der Lage, für ununterbrochene Intervalle bzw. für Intervalle, die speziellen Konvexitätsbedingungen genügen, effizient globale Konsistenz sicherzustellen. Im Gegensatz zu Waltz-basierten Verfahren garantiert die 2^k-Baum-Methode durch ein binäres Suchverfahren Konvergenz. Sie hat allerdings im Gegensatz zu anderen numerischen Verfahren nicht die Berechnung eines einzigen, optimalen Fixpunktes zum Ziel, sondern ist eher dazu geeignet, Lösungsräume zu berechnen, die von Constraint-Systemen bestehend aus Ungleichungen gebildet werden. Für diese Fälle lässt sich eine kompakte Repräsentation des Lösungsraums berechnen (vgl. Haroud und Faltings 1994; Sam-Haroud und Faltings 1996a, b; Sam-Haroud 1995).

Für die zu entwickelnde Komponente zur Verarbeitung von Constraints mit infiniten Domänen in EngCon ist es vorrangig erforderlich, Algorithmen zum Einschränken der Intervallgrenzen zur Verfügung zu stellen. Berechnungsverfahren explizit für kanonische Lösungen stehen weniger im Fokus. Die Wertebereichseinschränkungen dagegen sollten so weit wie möglich vorgenommen, d. h. die Lösungen so dicht wie möglich umschlossen, werden. Dies lässt sich für eine Vielzahl an Problemstellungen effizient mit Verfahren ähnlich dem Waltz-Filteralgorithmus erreichen. Die (lokale) Toleranzpropagation und die Algorithmen zur Herstellung von Hull-Konsistenz (2B-Konsistenz, 3B-Konsistenz) implementieren dies für Intervalldomänen. Während für die Toleranzpropagation bzw. für Hull-Konsistenz lediglich eine Zerlegung der Constraints vorgenommen werden muss, ist für die Herstellung von Box-Konsistenz die Implementierung komplexer mathematischer Operationen und ein automatischer Gleichungsumformer erforderlich, durch den die Constraints in ein durch das Newton-Intervallverfahren verarbeitbares Format gebracht werden müssen. Für die 2^k-Baum-Methode sind eine komplexe Datenrepräsentation und aufwendige Projektionen zu implementieren. Im Gegensatz zu diesen Verfahren, die in ihrer Umsetzung zu Aufwendig sind, als dass sich dies im Rahmen dieser Arbeit bewerkstelligen ließe, ist eine Umsetzung der Toleranzpropagation bzw. von Algorithmen zur Herstellung von Hull-Konsistenz innerhalb eines vertretbaren Zeitaufwands möglich.

Aus Gründen der Komplexität sollten die in dieser Arbeit umgesetzten Verfahren ausschließlich konvexe Intervalldomänen verarbeiten. Zum Einen steigt bei der Berechnung mit diskontinuierlichen Intervallen, d. h. mit Vereinigungsmengen einzelner Teilintervalle, der Berechnungsaufwand stark an, und zum Anderen existiert für Berechnungen mit ununterbrochenen Intervallen mit der IAMath-Bibliothek bereits ein Werkzeug für intervallarithmetische Berechnungen für die Programmiersprache Java (vgl. Abschnitt 2, S. 3).

3.3 Der Framework-Ansatz

Durch das Aufkommen von objektorientierten Sprachen und objektorientierter Programmierung (OOP) entstanden Ansätze, welche verstärkt die Steigerung der Wiederverwendbarkeit von einmal entwickelten Software-Komponenten zum Ziel hatten. Im Besonderen sind dies Software-Frameworks, in denen ein Rahmenwerk für die Bewältigung eines bestimmten Aufgabenspektrums bereitgestellt wird. Sie bieten eine Architekturhilfe beim Aufteilen des Entwurfs in abstrakte Klassen und bei der Definition ihrer Zuständigkeiten

und Interaktionen. Ein objektorientiertes Framework wird für eine bestimmte Anwendung spezialisiert, indem der Entwickler anwendungsspezifische Unterklassen für abstrakte Framework-Klassen erstellt (vgl. Gamma et al. 1996, S. 37).

Objektorientierte Constraint-Frameworks im Speziellen dienen dazu, OOP-Sprachen und Techniken zur Constraint-Verarbeitung zu verbinden und in unterschiedlichen Szenarien nutzbar zu machen. Von Roy et al. (2000) wird ein Constraint-Framework als Möglichkeit beschrieben, eine flexible und erweiterbare Implementierung des Constraint-Mechanismus anzubieten. Anstatt wie bei CP- bzw. CLP-Systemen eine Domäne, nämlich die der (logischen) Programmierung mit Constraints zu fokussieren, bietet ein Constraint-Framework allgemeine Mechanismen, die eine Anpassung für spezielle Problemstellungen erlauben. In einem Framework werden demnach kollaborierende Komponenten integriert, die auf diese Weise eine wiederverwendbare Architektur für eine Vielzahl von Anwendungen zur Verfügung stellen. Weil ein Framework i. A. bereits funktionsfähige Mechanismen beinhaltet, kann es häufig auch *out of the box* wie eine Bibliothek eingesetzt werden.

Um je nach Problemstellung flexibel geeignete Lösungsverfahren einsetzen zu können, bietet sich für die zu entwickelnde Constraint-Komponente ein objektorientierter Framework-Ansatz an. Da für EngCon ein hybrides Constraint-System, sowohl für finite als auch infinite Constraint-Domänen benötigt wird, sind in jedem Fall unterschiedliche Constraint-Solver erforderlich. Neben eigenen implementierten Constraint-Solvern können in ein derartiges Framework über einen Wrapper-Mechanismus zudem bestehende Constraint-Systeme eingebunden werden, die, wenn sie die Anforderungen von EngCon auch nicht vollständig erfüllen, für bestimmte Problemstellungen ausreichend (oder gar notwendig) sind. Ein Framework bietet die notwendige Umgebung zur relativ einfachen Implementierung neuer Lösungsverfahren bzw. zur einfachen Integration von Fremdsystemen. Es bietet weiterhin den Vorteil der modularen Erweiterbarkeit und allgemeine Schnittstellen zum flexiblen Einsatz in unterschiedlichen Anwendungen auch außerhalb von EngCon.

3.4 Constraint-Lösungsstrategien

Flexibilität bzgl. der einzusetzenden Lösungsverfahren kann durch ein Konzept von modularen und austauschbaren Constraint-Lösungsstrategien erreicht werden. Zur Strukturierung des Constraint-Lösungsvorgangs wird dieser Prozess in drei Phasen eingeteilt: (1) Preprozessing, (2) Konsistenzherstellung und (3) Lösungssuche. Diese Phasen spiegeln sich innerhalb von Constraint-Lösungsstrategien wieder (vgl. Abbildung 1 auf der gegenüberliegenden Seite). In der ersten Phase wird ein Preprozessing des Constraint-Problems vorgenommen. Dies kann z. B. die Binärisierung eines Constraint-Netzes oder die Zerlegung von Constraints in primitive Constraints sein, um anschließend darauf aufbauende Lösungsalgorithmen anwenden zu können. In der zweiten Phase werden Filter- bzw. Konsistenzalgorithmen zur Einschränkung der Domänen der Constraint-Variablen angewendet. Da dies allein i. A. nicht zu einer Lösung des Constraint-Problems führt, können in einer dritten Phase Suchverfahren zum Auffinden von Lösungen in den reduzierten Wertebereichen eingesetzt werden.

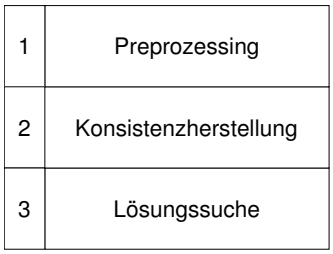

Abbildung 1: Aufbau einer Constraint-Lösungsstrategie

Zu beachten ist, dass in jeder Phase mehrere Einträge innerhalb einer Lösungsstrategie existieren können. So ist es z. B. möglich, mehrere Preprozessing-Schritte auf ein Problem anzuwenden, bevor Verfahren aus der nächsten Phase zum Einsatz kommen. Dies gilt ebenso für Konsistenz- und Suchverfahren.

Während es für Konsistenzverfahren durchaus sinnvoll erscheint bspw. Knotenkonsistenz herzustellen, bevor ein Algorithmus zum Herstellen von Kantenkonsistenz eingesetzt wird, erschließt sich der Sinn mehrerer Einträge im Fall von Suchverfahren nicht sofort. Für den Fall allerdings, dass für eine geeignete Anwendung aus Effizienzgründen anstatt systematischer Suchverfahren lokale bzw. stochastische Suchverfahren zum Einsatz kommen, kann hierdurch die Unvollständigkeit lokaler Verfahren abgemildert werden: Wenn ein (lokales) Suchverfahren keine Lösung für ein Constraint-Problem gefunden hat, sich aber mehrere Einträge in der Phase für die Lösungssuche befinden, kann entsprechend das nächste Suchverfahren angewendet werden.[12]

Ebenso ist es möglich, dass für eine Phase innerhalb einer Strategie keine Einträge existieren. Nicht für alle Konsistenz- und Suchverfahren ist ein Preprozessing erforderlich. Gleichfalls kann ein Suchverfahren i. A. auch ohne vorherigen Filteralgorithmus angewendet werden, insbesondere wenn das Suchverfahren bereits Filtermechanismen enthält. Sind für eine Anwendung keine exakten Lösungen sondern nur eingeschränkte Wertebereiche erforderlich, kann auf ein Suchverfahren in der dritten Phase verzichtet werden. Mehrere Beispiele für mögliche Constraint-Lösungsstrategien sind in Abbildung 2 auf der nächsten Seite zu sehen.

Zur Anwendung von derartigen Constraint-Lösungsstrategien ist eine Komponente bzw. eine Schnittstelle erforderlich, welche die Erstellung und Nutzung modularer Lösungsstrategien zur Verarbeitung von hybriden Constraint-Problemen, und basierend auf einer in eine Framework-Architektur integrierte Bibliothek von Constraint-Solvern erlaubt.

3.5 Das YACS-Framework

Aufgrund einer Vielzahl von Constraint-Lösungsverfahren und möglicher Kombinationen dieser, deren unterschiedlichen Eigenschaften und der problemabhängigen bzw. anwendungsspezifischen Effizienz unterschiedlicher Verfahren, ist eine Komponente notwendig,

[12] Lokale Suche wird aufgrund der Unvollständigkeit dieser Verfahren nicht in Zusammenhang mit ENGCON umgesetzt.

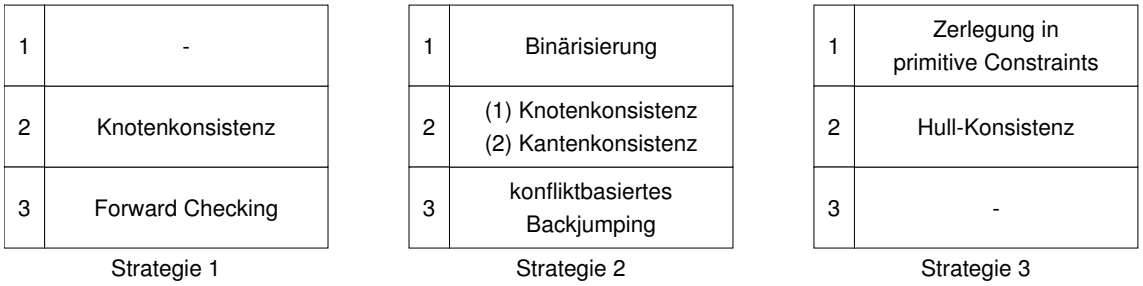

Abbildung 2: Beispiele für Constraint-Lösungsstrategien

mit der sich flexibel, je nach Problemstellung unterschiedliche Constraint-Lösungsmechanismen einsetzen lassen.

Das im Rahmen dieser Arbeit zu entwickelnde YACS-Framework stellt eine modulare und wiederverwendbare Constraint-Lösungskomponente dar. Der Name YACS steht für *Yet Another Constraint Solver*. YACS ist allerdings mehr als nur ein einzelner Constraint-Solver. Es bezeichnet vielmehr ein hybrides System zum flexiblen Einsatz von Constraint-Lösungsverfahren für finite und infinite Domänen. Sie sind eingebettet innerhalb einer modularen Framework-Architektur.

Der flexible Einsatz von Constraint-Lösungsverfahren wird über ein Strategiekonzept realisiert. Abstrahiert von den eigentlichen Lösungsalgorithmen können von dem Wissensingenieur problemabhängig bzw. anwendungsspezifisch unterschiedliche Constraint-Lösungsstrategien eingesetzt werden. Diese Lösungsstrategien müssen vorab in Abhängigkeit von den vorhandenen Lösungsverfahren definiert werden.

Durch die Framework-Architektur wird sichergestellt, dass flexibel Lösungsverfahren ausgetauscht und zudem auf einfache Weise neue Lösungsalgorithmen implementiert bzw. Fremdsysteme integriert werden können. Das YACS-Framework stellt hierfür einen geeigneten Rahmen mit einheitlichen Schnittstellen bereit.

3.5.1 Architektur

In Abbildung 3 auf der gegenüberliegenden Seite ist eine Übersicht über die Systemarchitektur von YACS, angebunden an das Konfigurierungswerkzeug EngCon, zu sehen. Aufsetzend auf einem *Domänen-Layer*, einer Umgebung zur arithmetischen Verarbeitung von finiten Domänen (FD) und reellwertigen Intervallen (IAMath), werden die eigentlichen Algorithmen zum Auflösen von Constraint-Problemen implementiert (*Algorithmus-Layer*). Constraint-Verfahren aus Fremdsystemen können an dieser Stelle über Wrapper-Klassen eingebunden werden. Die Algorithmen bzw. die sie umschließenden (Wrapper)-Klassen müssen wiederum den Schnittstellen des *Framework-Layers* von YACS genügen.

Der durch das Framework vereinheitlichte Zugriff auf Constraint-Lösungsverfahren ermöglicht es dem *Strategie-Layer*, dem Anwender bzw. dem übergeordnetem System (in diesem Fall das Constraint-System von EngCon) eine flexible Auswahl an Lösungsver-

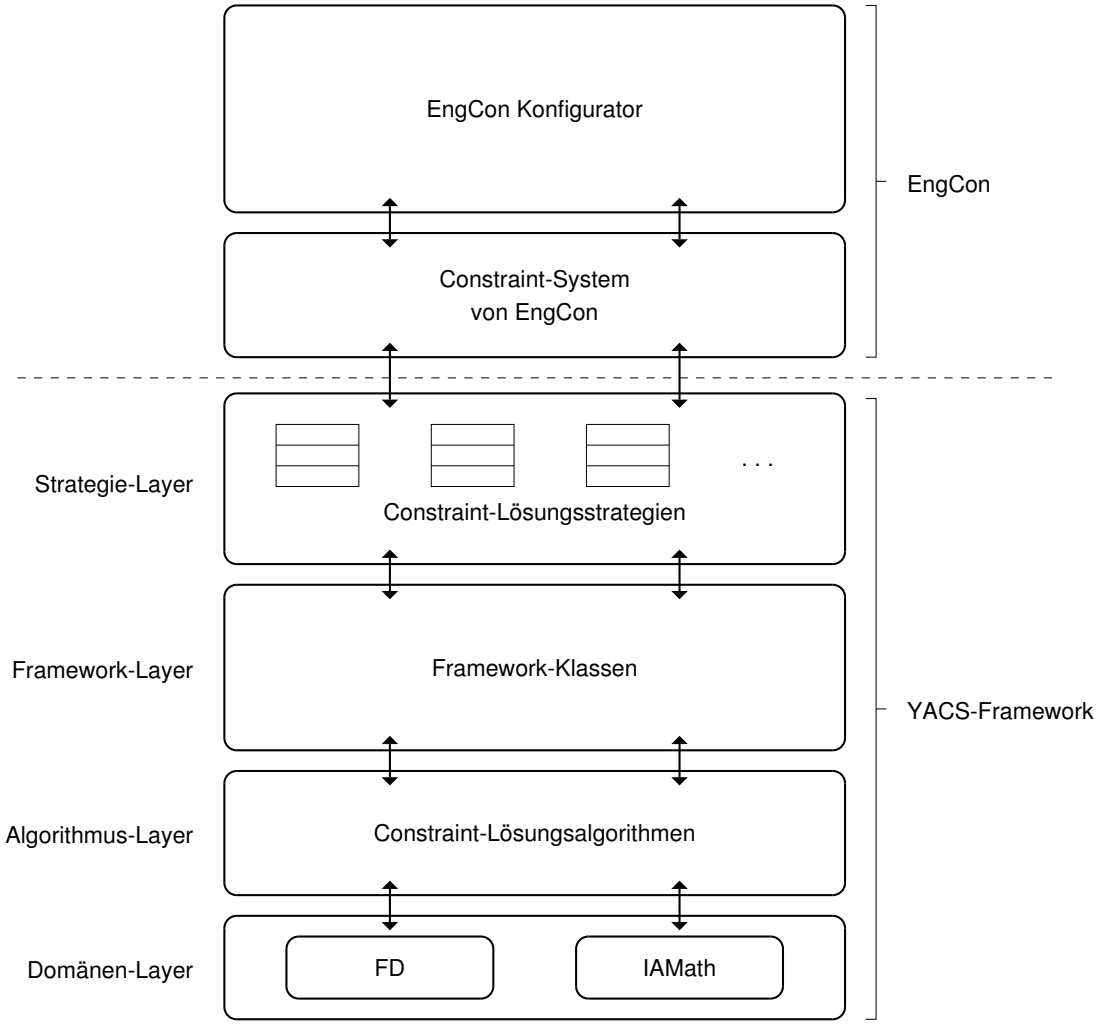

Abbildung 3: Systemarchitektur von YACS

fahren anbieten zu können.[13] Abstrahiert von den Lösungsverfahren können auf dieser Ebene aus einer Reihe vordefinierter Constraint-Lösungsstrategien problemabhängig die für die jeweilige Anwendung geeigneten Strategien ausgewählt werden.

3.5.2 Ausführungsmodelle

Benötigt wird eine Komponente an der Schnittstelle zwischen der internen Constraint-Verwaltung von EngCon und dem YACS-Framework, der die Verwaltung der unterschiedlichen Constraint-Netze, der Constraint-Lösungsstrategien, der Constraint-Solver und letztendlich der Steuerung des Lösungsprozesses obliegt. Diese Verwaltungs- und

[13] Auf eine detaillierte Darstellung der weiteren Systembestandteile von EngCon wurde in Abbildung 3 aus Gründen der Übersichtlichkeit verzichtet.

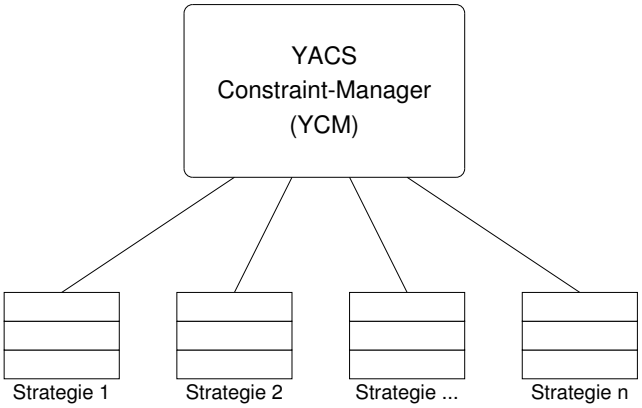

Abbildung 4: Verwaltung von Constraint-Lösungstrategien durch den YCM

Steuerungsaufgabe wird von dem „YACS Constraint-Manager“ (YCM) wahrgenommen (vgl. Abbildung 4).

Von dem aufrufenden System, in diesem Fall EngCon, erhält der Constraint-Manager die Informationen, welche Constraints mit welcher Strategie aufzulösen sind. Der Constraint-Manager aktiviert die entsprechenden Constraint-Lösungskomponenten und übergibt in der jeweiligen Bearbeitungsphase das entsprechende Constraint-Netz zur Verarbeitung an die dafür vorgesehene Komponente.

Der Prozess des Constraint-Lösens ist analog zum Aufbau der Constraint-Lösungsstrategien in drei Phasen unterteilt. In jeder Phase werden sequentiell jeweils die Constraint-Netze aller Strategien der Reihe nach bearbeitet.[14] Das heißt die in den Strategien angegebenen Constraint-Verfahren werden in den entsprechenden Phasen auf die zugehörigen Constraint-Netze angewendet:

- **Phase 1 (Preprozessing):** Das jeweilige Constraint-Netz wird vollständig konvertiert oder es wird festgestellt, dass es sich mit den gegebenen Algorithmen nicht umformen lässt.
- **Phase 2 (Konsistenzherstellung):** Es wird solange propagiert, bis keine Änderungen der Wertebereiche mehr eintreten (d. h. Konsistenz hergestellt ist) oder eine Inkonsistenz auftritt.
- **Phase 3 (Lösungssuche):** Die Suchalgorithmen finden die geforderten Lösungen oder stellen fest, dass keine Lösung existiert.

Eine Ausnahme führt zum Abbruch des Lösungsvorgangs. Ein Beispiel für einen Lösungsprozess mit unterschiedlichen Strategien, deren Constraint-Verfahren in den jeweiligen Phasen der Reihe nach abgearbeitet werden, ist in Abbildung 5 auf der gegenüberliegenden Seite zu sehen. Für eine Realisierung dieses Strategiekonzepts kommen zwei unterschiedliche Szenarien in Frage:

[14] An dieser Stelle bietet sich Optimierungspotential bzgl. einer verteilten Constraint-Verarbeitung.

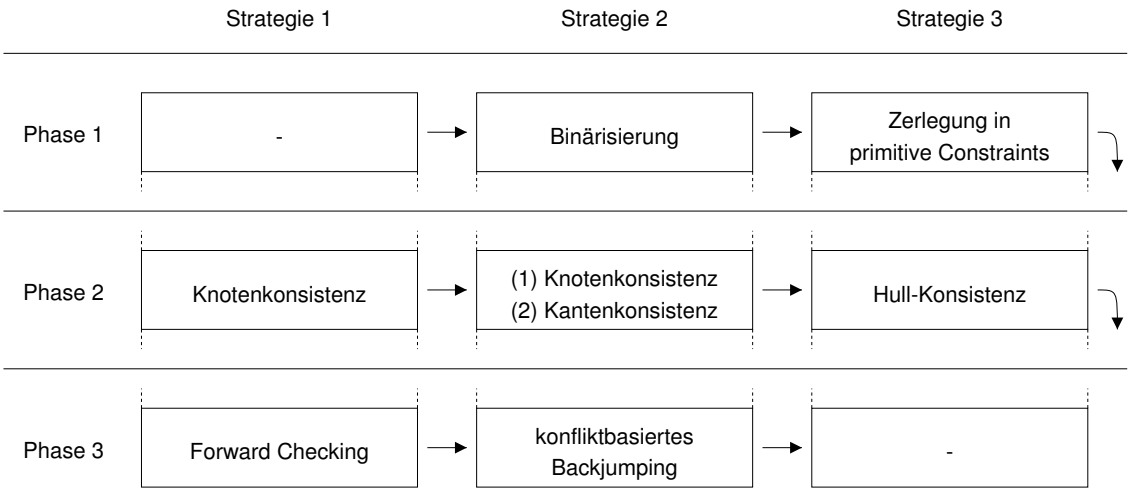

Abbildung 5: Beispiel für phasenweisen Lösungsprozess

Szenario 1 Ein vereinfachtes Szenario sieht vor, dass für jede Wissensbasis jeweils lediglich eine Strategie für finite und eine für intervallwertige Domänen vorgegeben werden kann. Um die Unabhängigkeit von YACS sicherzustellen, sollte dies außerhalb der EngCon-Wissensbasis in einer separaten Definitionsdatei geschehen. In diesem Fall würde der Constraint-Manager die Constraint-Lösungsstrategien auslesen und anschließend anwenden. Der Constraint-Lösungsprozess gestaltet sich in diesem Szenario folgendermaßen:

- Zuerst werden für die beiden Constraint-Netze (finite und infinite Domänen) die in den Strategien definierten Preprozessingschritte ausgeführt.
- Als nächstes werden für beide Constraint-Netze die Konsistenzverfahren so lange angewendet, bis keine Wertebereichsänderungen mehr auftreten.
- Als letztes werden die Lösungsverfahren der Strategien auf beide Constraint-Netze angewendet.

Abschließend erfolgt die Rückmeldung der neuen Wertebereiche der Constraint-Variablen sowie die möglichen Lösungen an das Constraint-System von EngCon.

Vorteile dieses einfachen Szenarios sind die schnelle und einfache Realisierbarkeit und der geringe Overhead, der in Bezug auf die Verwaltung innerhalb des Constraint-Managers von YACS entsteht. Flexibilität bzgl. des Einsatzes von Constraint-Lösungsverfahren ist dadurch gewährleistet, dass für jede Wissensbasis und dem darin enthaltenen Konfigurierungs- und Constraint-Problem erneut entschieden werden kann, welche Strategien respektive Constraint-Verfahren angewendet werden sollen. Nachteilig hinsichtlich der Flexibilität wirkt sich dieses Szenario möglicherweise bei komplexen Constraint-Problemen aus. Innerhalb von umfangreichen Problemstellungen kann es sinnvoll sein, Unterprobleme, d. h. Teile des Constraint-Netzes, aus Effizienzgründen mit speziellen Constraint-Lösungstechniken zu verarbeiten. Für andere Bereiche des Constraint-Problems wiederum können dieselben Constraint-Verfahren unnötigen Overhead bedeuten.

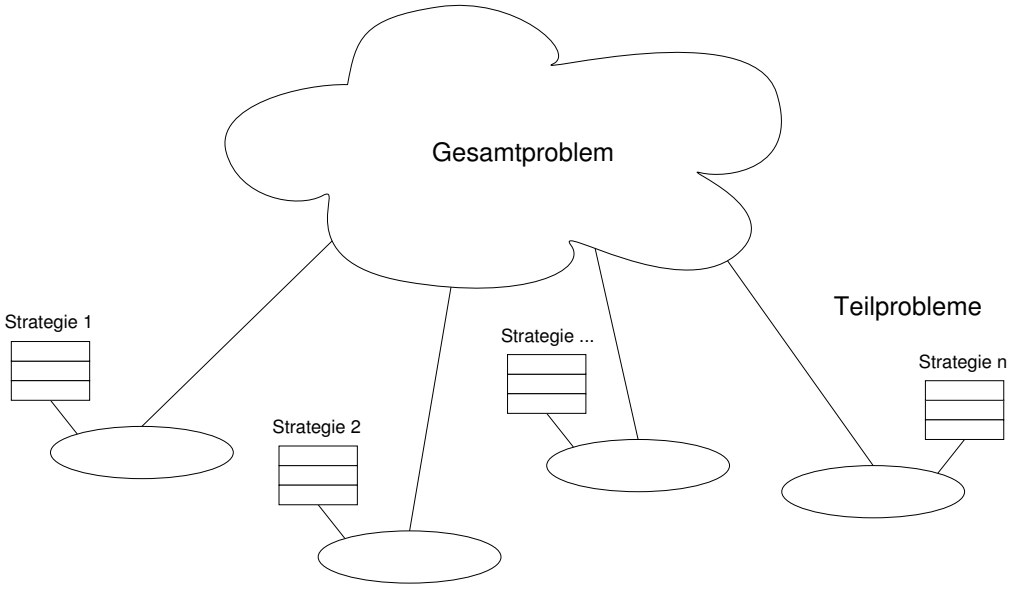

Abbildung 6: Zuständigkeiten unterschiedlicher Strategien für Teilbereiche des Constraint-Problems

Ob dieses Szenario eine angemessene Umsetzung darstellt, ist abhängig von der Komplexität des Constraint-Netzes innerhalb der Wissensbasis. Bei überschaubaren Problemstellungen werden ggf. nicht mehr als zwei Strategien (je eine für finite und infinite Domänen) benötigt.

Szenario 2 Erweiterte Flexibilität wird erreicht, wenn in der Wissensbasis für jedes Constraint der Name einer entsprechenden Strategie zu dessen Lösung angegeben werden kann. Der Name der jeweiligen Strategie entspricht dabei einem eigenen Teilbereich des ursprünglichen Constraint-Problems. Jede Strategie ist für die Verarbeitung eines (Sub-)Constraint-Netzes vorgesehen (vgl. Abbildung 6). Bei der Initialisierung wird durch den Constraint-Manager von YACS sichergestellt, dass alle geforderten Strategien existieren und angewendet werden können.

Die Strategien werden separat von EngCon definiert. EngCon übergibt die Constraints jeweils mit dem Namen der zugehörigen Strategie an den YACS Constraint-Manager. Dieser generiert daraus die unterschiedlichen Constraint-Netze und wendet die entsprechenden Constraint-Lösungsstrategien an:

- Zuerst werden für sämtliche Constraint-Netze die in den Strategien definierten Preprozessingschritte ausgeführt.

- Als nächstes werden für alle Constraint-Netze die jeweils entsprechenden Konsistenzverfahren so lange angewendet, bis in den jeweiligen Netzen keine Wertebereichsänderungen mehr auftreten.

- Als letztes werden die Lösungsverfahren der Strategien auf die jeweils entsprechenden Constraint-Netze angewendet.

Abschließend erfolgt auch hier die Rückmeldung der neuen Wertebereiche der Constraint-Variablen sowie die möglichen Lösungen an das Constraint-System von EngCon.

Da für jedes Constraint theoretisch eine eigene Constraint-Lösungsstrategie angegeben werden kann, ist in diesem Szenario die maximale Flexibilität bzgl. des Einsatzes unterschiedlicher Constraint-Lösungsverfahren für unterschiedliche Topologien von Constraint-Netzen sichergestellt. Nachteilig ist der höhere Aufwand für die Realisierung dieses Szenarios. Zudem ist mit höherem Overhead als in dem erstgenannten Szenario zu rechnen. Dies betrifft sowohl die Verwaltung im Constraint-Manager von YACS, als auch die Erstellung und Pflege der Wissensbasis von EngCon.

Der zusätzliche Overhead in diesem Szenario ist allerdings nicht sehr hoch und kann weiter verringert werden, indem bei überschaubaren Problemstellungen weniger Strategien eingesetzt werden. So gesehen ist dieses Szenario in Abhängigkeit von der Problemstellung „skalierbar“ , und kann ggf. auf das Szenario 1 mit lediglich zwei unterschiedlichen Strategien reduziert werden. Das Szenario 2 stellt demnach eine Verallgemeinerung von Szenario 1 dar, und umgedreht das Szenario 1 eine Spezialisierung von Szenario 2.

Semantische Aspekte Klassischerweise wird ein Constraint-Problem ausschließlich von einem einzigen Constraint-Solver mit einem eindeutigen Namensraum und eindeutigen Variablenbelegungen bzw. Wertebereichen bearbeitet. Unterschiedliche Constraint-Lösungsstrategien bedingen getrennt voneinander zu verarbeitende Teilbereiche des ursprünglichen Constraint-Problems. Unterschiedliche Constraint-Solver kooperieren in diesem Fall, um gemeinsam mögliche Lösungen für das Constraint-Problem zu generieren.

Ein wesentliches Problem in diesem Szenario ist, wie aus den einzelnen Teillösungen vollständige Gesamtlösungen (globale Lösungen) generiert werden. Ein in diesem Zusammenhang zentraler Punkt betrifft die Frage, ob die Constraint-Netze der unterschiedlichen Constraint-Solver innerhalb von YACS denselben Namensraum aufweisen oder nicht, d. h. die Frage ob „Überlappungen“ der Constraint-Netze möglich sein sollen oder nicht, und wie diese zu behandeln sind. Überlappungen treten an den Stellen auf, an denen dieselben Variablen in den Constraint-Netzen mehrerer Strategien auftauchen.

Das Szenario mit voneinander getrennten, disjunkten Constraint-Netzen mit jeweils eigenem Namensraum ohne Überlappungen wird *lokale Sicht* genannt. Eine derartige Verarbeitung innerhalb von YACS würde eine Vereinfachung bedeuten. Gleichzeitig wird das Problem allerdings verlagert, wenn für die übergeordnete Anwendung globale Lösungen für das Constraint-Problem benötigt werden. Hierfür ist ein einheitlicher Namensraum erforderlich. Existiert nur ein einziger Namensraum und sind Überlappungen von Constraint-Netzen unterschiedlicher Strategien zulässig, so wird dies *globale Sicht* genannt.

Für eine genauere Untersuchung erfolgt an dieser Stelle wiederum eine phasenweise Betrachtung des Constraint-Lösungsprozesses. Dies geschieht jeweils in Bezug auf voneinander getrennte Namensräume (lokale Sicht) und in Bezug auf denselben Namensraum für alle beteiligten Constraint-Verfahren (globale Sicht):

- **Phase 1 (Preprozessing)**

Die Preprozessing-Phase ist weitgehend unkritisch bzgl. überlappender Constraint-Netze von kooperierenden Constraint-Solvern in unterschiedlichen Strategien.

Lokale Sicht: Jedes Constraint-Netz wird für sich und unabhängig vom restlichen Problem konvertiert und damit für den weiteren Lösungsprozess vorbereitet.

Globale Sicht: Umformungsprozesse fügen i. A. Variablen und zusätzliche Constraints zum ursprünglichen Problem hinzu. Beispielsweise werden sowohl bei einer Binärisierung (umfassende) Variablen hinzugefügt (finite Domänen), als auch bei einer Zerlegung der Constraints in primitive Constraints (infinite Domänen). Diese Konvertierungen des Constraint-Netzes sind unkritisch, da bei Überschneidungen der Constraint-Netze der Zugriff auf die ursprünglichen Variablen gesichert bleibt. Wenn Umformungsprozesse allerdings Variablen ersetzen oder der Zugriff auf die ursprünglichen Variablen aus irgendeinem Grund nicht möglich sein sollte, wird dadurch in Kauf genommen, dass Einschränkungen von deren Wertebereichen YACS-intern nicht übergreifend registriert und propagiert werden können.

- **Phase 2 (Konsistenzherstellung)**

  Die Phase der Konsistenzherstellung profitiert von einem einheitlichen Namensraum der Constraint-Netze aller Strategien. Es wird direkt innerhalb von YACS eine höhere Filterung inkonsistenter Werte erreicht, wodurch für nachfolgende Suchverfahren weniger Suchaufwand notwendig wird.

  Lokale Sicht: Wertebereichseinschränkungen werden von jeder Strategie für das dem entsprechenden Teilproblem zugehörige Constraint-Netz separat vorgenommen. Es werden daher i. A. weniger inkonsistente Werte entfernt bzw. die Intervallgrenzen weniger stark beschränkt, als wenn Einschränkungen im vollständigen Constraint-Problem propagiert werden. Sollte in den Strategien kein abschließendes Suchverfahren definiert sein, können beim Zurückschreiben der eingeschränkten Wertebereiche in das interne Constraint-System von EngCon unterschiedliche Filterstärken anhand unterschiedlich stark eingeschränkter Wertebereiche registriert werden. An dieser Stelle kann durch EngCon eine erneute Propagation initiiert werden, wobei für jede Variable der jeweils kleinste Wertebereich berücksichtigt werden sollte, um möglichst viele inkonsistente Werte auszuschließen. In einem weiteren Durchlauf der Propagation werden diese Einschränkungen auch für andere Namensräume sichtbar.

  Globale Sicht: Wertebereichseinschränkungen, die durch Konsistenzverfahren einer Strategie hervorgerufen werden, haben in diesem Fall direkt Auswirkungen auf alle überlappenden Constraint-Netze anderer Strategien. Einschränkungen werden augenblicklich in allen Constraint-Netzen sichtbar und haben entsprechende Auswirkungen auf die Filterung inkonsistenter Werte bzw. die Einschränkung der Intervallgrenzen. Sollte kein abschließendes Suchverfahren erfolgen, sind die an EngCon zurückgegebenen Wertebereiche bereits so weit

wie möglich eingeschränkt. Eine erneute Propagation würde keine zusätzlichen Einschränkungen bewirken und ist damit unnötig.

- **Phase 3 (Lösungssuche)**

  Werden ausschließlich Constraint-Verfahren zur Einschränkung der Wertebereiche und Herstellung bestimmter Konsistenzgrade eingesetzt, ist die Kooperation unterschiedlicher Constraint-Solver relativ unproblematisch. Werden dagegen von der Anwendung konkrete Lösungen für das Constraint-Problem benötigt, so müssen aus den einzelnen Teillösungen in einem „Meta-Constraint-Problem“ globale Lösungen generiert werden.

  Lokale Sicht: Für jedes Constraint-Netz der unterschiedlichen Strategien werden wie bei der klassischen Verarbeitung von Constraint-Problemen separate Lösungen berechnet und an EngCon zurückgemeldet. Lösungsverfahren einer Strategie profitieren in diesem Fall nicht von möglichen Wertebereichseinschränkungen anderer Strategien. Es besteht zudem innerhalb von YACS keine Möglichkeit, globale Lösungen zu generieren, wenn die Constraint-Netze unterschiedlicher Strategien Überlappungen aufweisen. Der Wissensingenieur muss daher bei der Spezifikation der Wissensbasis beachten, dass ausschließlich abgeschlossene Teile des Constraint-Problems jeweils an eine Strategie übergeben werden. Ansonsten sind die Ergebnisse lediglich lokal konsistente Teillösungen. In der Anwendung (in diesem Fall EngCon) würde ein übergeordneter Mechanismus notwendig, der aus den separaten Teillösungen globale Lösungen generiert.

  Globale Sicht: Die Wertebereiche sind durch mögliche Konsistenzverfahren in allen Constraint-Netzen gleich stark eingeschränkt worden. Im Rahmen der Lösungssuche berechnen auch hier zunächst wie bei lokalen Sicht unterschiedliche Constraint-Solver in unterschiedlichen Strategien für jedes Constraint-Netz separat die jeweils möglichen Teillösungen. In diesem Fall allerdings können anschließend *innerhalb* von YACS globale Lösungen für das Problem generiert werden. Dafür ist nach der Berechnung der Teillösungen eine weitere Constraint-Lösungskomponente erforderlich, welche in dem entstandenen Meta-Constraint-Problem nach möglichen Lösungen sucht. Die Menge der Lösungen für ein Teilproblem stellt analog zu einer n-dimensionalen Matrix jeweils eine vollständige Relation zwischen den beteiligten Variablen dar.[15] Jede einzelne Teillösung steht für eine gültige Belegung der Relation. Das aufzulösende Constraint-Problem ist entsprechend ein extensional repräsentiertes, n-stelliges CSP.

Diskussion Während Preprozessingverfahren von der globalen Sicht i. A. nicht profitieren, ergibt sich für Konsistenzverfahren der Vorteil, dass bei überlappenden Constraint-

[15] Sämtliche Teillösungen – auch unterschiedlicher Strategien – welche sich auf dieselbe Variablenmenge beziehen, müssen hierbei ggf. zusammen betrachtet werden, um eine vollständige Relation mit allen Abhängigkeiten zwischen diesen Variablen zu erhalten (vgl. *totales Constraint*).

Netzen Wertebereichseinschränkungen innerhalb von YACS global propagiert werden können, anstatt nur innerhalb des durch die jeweilige Strategie definierten Teilproblems. Ebenso ermöglicht die globale Sicht YACS das Generieren globaler Lösungen für überlappende Constraint-Netze.

Wenn innerhalb von YACS hingegen eine lokale Sichtweise angenommen wird, führt dies zu einer Vereinfachung, da die Constraint-Verfahren jeder Strategie ausschließlich ihr zugewiesenes Teilproblem bearbeiten müssen. Übergreifende Wertebereichseinschränkungen könnten bei einer erneuten Propagierung, angestoßen durch EngCon, verarbeitet werden. Globale Lösungen allerdings können von YACS in diesem Fall nur dann berechnet werden, wenn sichergestellt ist, dass sich die Constraints auch im internen Constraint-Netz von EngCon nicht überlappen.

Werden in einer globalen Sichtweise überlappende Constraint-Lösungsstrategien, die Suchverfahren einsetzen, gleichzeitig mit solchen Strategien verwendet, in denen auf Suchverfahren verzichtet wird (Strategien, die lediglich die Wertebereiche durch Konsistenzverfahren einschränken), so muss die Lösungssuche für die betroffenen Variablen auch von einem übergeordnetem Constraint-Lösungsmechanismus nicht geleistet werden. An den Stellen allerdings, wo es zu Überlappungen kommt, müssen für die betreffenden Constraints Lösungen berechnet werden, die konsistent mit den entsprechenden Wertebereichen sind.[16]

Überlappungen der Constraint-Netze unterschiedlicher Strategien können das Vermischen unterschiedlicher Wertedomänen bedeuten. Während dies in Szenario 1 aufgrund der begrenzten Anzahl Strategien bei Überlappungen grundsätzlich der Fall ist, kann dies auch in der Verallgemeinerung des Problems in Szenario 2 gegeben sein: Bei Überlappungen ist es möglich, dass intervallwertige Domänen infiniter Variablen von diskreten Werten finiter Variablen beschränkt werden und umgekehrt. Das in diesem Fall vorliegende CSP ist ein *Mixed CSP* (vgl. Gelle 1998; Gelle und Faltings 2003). Die Einschränkung der Wertebereiche und die Lösungssuche hat in Bezug auf die globale Sicht analog zu anderen Überlappungen zu erfolgen. Zu beachten ist allerdings, dass in diesem Fall durch YACS automatische Konvertierungen der Wertebereiche vorgenommen werden müssen. Dies bedeutet eine Diskretisierung von intervallwertigen Domänen und umgekehrt die Propagation der Grenzen von Wertebereichen, wenn FD-Variablen von Intervall-Constraint-Solvern verarbeitet werden.[17] Wenn Konvertierungen der Wertebereiche vorgenommen werden, ist von dem Wissensingenieur zu beachten, dass nach Möglichkeit keine „ungünstigen“ Domänen konvertiert werden. Das heißt es sollte vermieden werden, dass z. B. eine Domäne mit den Werten $\{0, 10000, 1000000\}$ in eine (konvexe) Intervalldomäne konvertiert wird. Umgedreht sollte eine Variable mit einer intervallwertigen Domäne von bspw. $[0, 1000000]$ nicht innerhalb eines FD-Constraints verwendet werden.[18]

[16] Durch eine erneute Propagation kann es bei Übernahme der Werte aus den Lösungsverfahren als neue Domänenwerte auch durch Konsistenzverfahren zu eindeutigen Lösungen kommen.

[17] Eine intelligente Diskretisierungsmethode würde bspw. eine FD-Variable mit dem Wertebereich $\{3, 17, 27\}$, welcher durch Intervallpropagationsverfahren auf das Intervall $[3, 20]$ beschränkt wurde, bezogen auf den ursprünglichen Wertebereich auf die Werte $\{3, 17\}$ beschränken, und das Constraint anschließend für eine erneute Propagation vorsehen.

[18] Alternativ könnte in letzterem Fall zur Effizienzverbesserung innerhalb der betreffenden Constraint-Lösungsstrategie für FD-Constraints ein Intervallverfahren als Preprozessing integriert werden.

Die Überlappung von Constraints mit finiten und infiniten Wertedomänen kann dazu führen, dass wenn eine Intervallvariable von einem FD-Constraint beschränkt wird, dies (in Abhängigkeit von den Beschränkungen durch die Constraints) bei Auftauchen von Punktintervallen eine Kettenreaktion auslöst, an deren Ende ggf. alle Intervallvariablen auf einzelne Punktintervalle beschränkt sind. Wenn das System diskontinuierliche Intervalle unterstützt, tritt dieser Effekt bei überlappenden Constraint-Netzen verstärkt auf. Wenn hingegen ausschließlich konvexe Intervalle verarbeitet werden, tritt ein Informationsverlust auf, denn der FD-Solver hat ggf. mehr Werte aus einer Domäne herausgefiltert, als von den konvexen Intervallgrenzen umschlossen werden.

3.5.3 Lösen von heterogenen Constraint-Problemen

Für die Berechnung von globalen Lösungen ist in jedem Fall ein übergeordneter Constraint-Lösungsmechanismus erforderlich. Wenn aus den möglicherweise teils überlappenden Teillösungen unterschiedlicher Strategien Gesamtlösungen generiert werden sollen, stellt dies ein neues kombinatorisches Problem dar: ein extensionales, n-äres Meta-CSP. Jede Teillösung umfasst sämtliche involvierte Variablen des Teilproblems und stellt dadurch eine extensional repräsentierte Relation zwischen diesen Variablen dar.[19] Aufgabe eines übergeordneten Constraint-Lösungsverfahrens ist es, aus den n-ären Relationen konsistente Lösungen zu generieren.

Wenn durch die Constraint-Verfahren einer Lösungsstrategie im Rahmen eines Teilproblems ein Wert einer Variable aus deren Domäne entfernt wird, geschieht dies deshalb, weil dieser Wert inkonsistent mit einem Constraint ist, und auf keinen Fall Teil einer Lösung sein kann. Das heißt die Teillösungen unterschiedlicher Strategien ergänzen sich nicht, sondern schließen sich gegenseitig aus. Eine mögliche Lösung des Meta-CSPs besteht deshalb darin, die Teillösungen einer Strategie jeweils mit den Teillösungen der anderen Strategien zu kombinieren und auf widersprüchliche oder konsistente Wertebelegungen zu überprüfen. Dies entspräche allerdings einem *Generate-&-Test*-Vorgehen. Aus Effizienzgründen sollte stattdessen ein systematisches Suchverfahren zur Anwendung kommen. Ein vollständige Suche muss

- durch sukzessives, systematisches Belegen aller Variablen,
- durch Testen, ob die jeweils aktuelle (Teil-)Belegung von allen Teillösungen unterstützt wird und
- durch Backtracking im Fall von Inkonsistenzen

systematisch Lösungen anhand der existierenden Relationen in den Teillösungen generieren. Das Lösungsverfahren ist umso aufwendiger, je mehr Teillösungen das Constraint-Problem aufweist. In Problemen mit niedriger Lösungsdichte, bezogen sowohl auf die Teillösungen als auch auf die Gesamtlösungen, ist die Lösungssuche effizienter. Gegebenenfalls

[19] Anstatt einer n-dimensionalen Matrix kann eine Teillösung zur Vereinfachung und analog zur Repräsentation von extensionalen (Tupel-)Constraints in EngCon als Tabelle mit der Anzahl Spalten aufgefasst werden, wie Variablen im jeweiligen Teilproblem existieren, und mit der Anzahl Zeilen, wie gültige Relationen in Form von einzelnen Teillösungen existieren.

sollte die Suche dieses Meta-Constraint-Solvers durch intelligente Suchverfahren (z. B. eine Backjumping-Variante) verbessert werden.

Die globale Lösungssuche kann außerdem optimiert werden, indem weitere bestehende Inkonsistenzen vor der Initiierung des Suchvorgangs durch Ausnutzung des in den Teillösungen vorhandenen Wissens eliminiert werden. Bezogen auf ein spezifisches Problem stellt die Aufzählung aller möglichen Lösungen globale Konsistenz hinsichtlich der Wertebereiche dar. Für die vorhandenen Teillösungen bedeutet dies, dass sich die Wertebereiche der Constraint-Variablen ggf. weiter einschränken lassen. Dies resultiert daraus, dass sich mögliche Teillösungen nicht addieren, sondern sich gegenseitig ausschließen und ggf. Inkonsistenzen identifizieren: Werte, die nicht innerhalb einer Teillösung als gültige Belegung auftauchen, sind als inkonsistent herausgefiltert worden und dürfen dementsprechend bei Überlappungen der Constraint-Netze auch in keiner anderen Teillösung respektive einer globalen Lösung vorkommen. Entsprechend können aus Teillösungen eingeschränkte Wertebereiche abgeleitet werden. Ein Algorithmus müsste folgendes leisten:

1. Die ursprünglichen Wertebereiche der Variablen werden in einer temporären Struktur zwischengespeichert.

2. Für alle Teilprobleme und jeweils für alle Variablen v des aktuellen Teilproblems gilt: Alle Werte für v, die im Vergleich zur ursprünglichen Domäne herausgefiltert worden sind, können ebenfalls nicht Teil einer Lösung in anderen Teillösungen sein. Die entsprechenden, einzelnen Teillösungen (anderer Constraint-Lösungsstrategien bzw. Teilprobleme), die trotzdem eine derartige Belegung für v enthalten, sind zu entfernen.[20]

3. Zur Vermeidung redundanter Such- und Vergleichsvorgänge sind jeweils Wertebelegungen, nach denen bereits gesucht worden ist, aus den zwischengespeicherten ursprünglichen Wertebereichen zu entfernen. Anschließend erfolgt die Überprüfung der nächsten Wertebelegung, bis sämtliche Variablen in allen Teilproblemen überprüft worden sind.

4. Weil durch das Entfernen von einzelnen Teillösungen immer auch Wertebelegungen anderer Variablen als Teil der jeweiligen (extensionalen) Relation betroffen sind, müssen nach jedem Entfernen für die jeweils betroffenen Variablen ebenfalls die unter Punkt 2 bzw. Punkt 3 geschilderten Überprüfungen vorgenommen werden.

Durch diese Vorgehensweise wird allerdings noch kein spezifischer Konsistenzgrad hergestellt, da keine Wertekombinationen verglichen und auf Konsistenz überprüft werden. Durch einen Algorithmus, der oben beschriebene (Teil-)Relationen entfernt, werden lediglich die Wertebereiche der Variablen der einzelnen Teillösungen angeglichen – in diesem Fall anhand extensional repräsentierter Relationen.

Ebenso können in Bezug auf das Meta-CSP in weiteren Algorithmen Konsistenzverfahren zur Vereinfachung eingesetzt werden. Die Anwendung geht allerdings mit einer hohen

[20] Entspricht dem Löschen einer Zeile aus der Tabelle mit der Menge der Teillösungen für ein Teilproblem.

Komplexität einher, da das Meta-CSP durch n-äre, extensionale Relationen – den (vollständigen) Teillösungen – repräsentiert wird. Die Komplexität des Problems ist abhängig davon, wieviele Variablen an den einzelnen Teillösungen beteiligt sind. Eine Teillösung, die zwanzig Variablen umfasst, würde entsprechend die Behandlung eines 20-stelligen Constraints bedeuten.

In Bezug auf das Meta-CSP anzuwendende Konsistenz- und Suchverfahren ist weiterhin zu beachten, dass ggf. hybride Constraints innerhalb eines Mixed CSP verarbeitet werden müssen. Dies sind Constraints, die sowohl Variablen mit finiten als auch infiniten Wertebereichen gleichzeitig beschränken. Die entsprechenden Konvertierungsmethoden, d. h. Diskretisierungen und Überführungen von finiten Wertebereichen in kontinuierliche Intervalldomänen, sind zu berücksichtigen und die entsprechenden Lösungsalgorithmen anzupassen.

Problematisch sind Lösungsverfahren, die unvollständig sind. Dies sind Suchverfahren, die nicht alle Lösungen eines (Teil-)Problems berechnen, sondern z. B. aus Effizienzgründen lediglich die Erstbeste. Werden solche Lösungsverfahren eingesetzt, kann insgesamt die Vollständigkeit nicht garantiert werden, d. h. es können mögliche Lösungen für ein Problem verloren gehen. Bezogen auf das Gesamt problem bedeutet dies, das ggf. keine Lösung gefunden wird, obwohl welche existieren.

Weiterhin ist zu beachten, dass in allen Phasen des Lösungsvorgangs und in allen Strategien die Inkrementalität der Lösungsverfahren gewährleistet sein muss. Das heißt, dass bei einer erneuten Propagation mit zusätzlichen Constraints und Variablen die vormals erstellten Constraint-Netze nicht erneut instantiiert werden müssen. In Bezug auf Preprozessingverfahren sind hinzukommende Variablen und Constraints unkritisch. Das Hinzufügen neuer (umfassender) Variablen für eine Binärisierung von FD-Constraints ist ebenso wie das Hinzufügen neuer Variablen bei einer Zerlegung von (Intervall-)Constraints in primitive Constraints voneinander unabhängig in jedem Teilproblem inkrementell möglich. Auf Konsistenz- und Suchverfahren trifft dies ebenfalls zu. Die anwachsenden Constraint-Netze müssen innerhalb der Strategien von einer Phase zur nächsten „durchgereicht“ werden. Ob Inkrementalität dabei von den einzelnen Constraint-Verfahren selbst angeboten wird, oder ob dies ggf. eine geeignete (Wrapper-)Schnittstelle simuliert (z. B. für eingebundene Fremdsysteme), ist für YACS unerheblich. Einzelne Constraint-Solver werden als Black Box aufgefasst. Die im Rahmen dieser Arbeit umzusetzenden Constraint-Lösungsverfahren sollten allerdings inkrementell anwachsende Constraint-Netze von sich aus unterstützen.

3.6 Alternative Ansätze

Die Kooperation unterschiedlicher Constraint-Solver zur gemeinsamen Verarbeitung und Auflösung von Constraint-Problemen wurde in der Vergangenheit bereits mehrfach untersucht (vgl. Granvilliers et al. 2001a, b, c). Die vorliegenden Arbeiten lassen sich grundsätzlich unterscheiden in Ansätze, die eine fixe Kooperation mehrerer Constraint-Lösungsverfahren vorsehen, und Ansätze, die sich flexibel bzgl. der einzusetzenden Verfahren und z. T. auch bzgl. der Art und Weise der Kooperation verhalten.

Ansätze, die eine fixe Kooperation vorsehen, auch „Ad-hoc-Kooperationen“ genannt, sind in der Anwendung beschränkt auf eine bestimmte Domäne, fest vorgegebene Constraint-Solver und/oder eine bestimmte Lösungsstrategie, wie die Constraint-Solver zusammenwirken (vgl. Granvilliers et al. 2001a, b, S. 3; Granvilliers et al. 2001c, S. 159 ff.; Monfroy 1998, S. 349; Monfroy 2000, S. 212). Beispiele für derartige Ansätze sind die in (Benhamou und Granvilliers 1996) und (Granvilliers 2001) vorgestellten Kooperationen. Benhamou und Granvilliers (1996) kombiniert algebraische Berechnungsmethoden mit Box-Konsistenz, Granvilliers (2001) Algorithmen zur Herstellung von Hull- und Box-Konsistenz. Ziel ist in diesen Fällen die effizientere Verarbeitung der Problemstellungen.

Marti und Rueher (1995) und Rueher (1995) kombinieren (reellwertige) algebraische und intervallarithmetische Lösungsverfahren. Sie entwickeln ein verteiltes System mit asynchron kommunizierenden Agenten, basierend auf dem Paradigma des *Concurrent Constraint Programming* (CCP). Das System ermöglicht es Probleme zu lösen, die ein einzelner der eingesetzten Constraint-Solver nicht bewältigen könnte.

Kooperationen von Solvern, die Constraints mit vermischten Domänen verarbeiten können, sind die bereits genannten Ansätze zur Verarbeitung von Mixed CSPs (Benhamou 1996; Gelle 1998; Gelle und Faltings 2003). Sie bieten die Möglichkeit Constraint zu verarbeiten, die Variablen sowohl mit finiten als auch infinite Domänen beschränken, sind allerdings ebenfalls fixe Ad-hoc-Kooperationen bzgl. der eingesetzten Lösungsverfahren und der Lösungsstrategie.

Zur Erhöhung der Flexibilität ist die Anwendung strategiebasierter Ansätze ein gängiges Mittel zur Steuerung der Propagation und Lösungssuche von kooperierenden Constraint-Solvern. Der konkrete Einsatz von Strategien in den vorhandenen Ansätzen unterscheidet sich allerdings von der spezifischen Nutzung innerhalb von YACS, daher erfolgt an dieser Stelle eine nähere Betrachtung zweier Ansätze zur flexiblen Kooperation.

In ihrer Dissertation stellt *Petra Hofstedt* ein allgemeines und formales Schema für die Kombination von Constraint-Systemen und die Kooperation von Constraint-Solvern zur Behandlung von Constraints mit vermischten Wertedomänen vor (vgl. Hofstedt 2001). Strategien werden hier in Form von frei modellierbaren „Kooperationsstrategien“ eingesetzt. Mit diesen kann beschrieben werden, welche Constraint-Verfahren wie und in welcher Reihenfolge (sequentiell/parallel) kombiniert werden. Ebenfalls ist in diesem Ansatz ein Meta-Constraint-Solver vorgesehen, der allerdings im Wesentlichen zur Verwaltung des Constraint-Netzes und zur Koordinierung der einzelnen Constraint-Solver eingesetzt wird (vgl. Hofstedt 2000). Das theoretische Schema von Hofstedt (2000, 2001) wurde im Rahmen einer Diplomarbeit (vgl. Godehardt und Seifert 2001; zit. nach Hofstedt 2001, S. 127 ff.; Hofstedt et al. 2001, S. 18 u. S. 21) prototypisch in der Programmiersprache Java umgesetzt. Als Constraint-Solver kommen der IASolver[21] für reellwertige Intervalle, eine Reimplementierung der C-Lib[22] für finite Domänen und ein eigenimplementierter Simplex-Solver für linear-arithmetische Constraints zum Einsatz (vgl. Hofstedt et al. 2001, S. 13 ff.). Das von Hofstedt (2000, 2001) vorgestellte formale Schema ist sehr flexibel allerdings auch sehr komplex, so dass die Implementierung dagegen recht einfach gehalten ist

[21] `http://www.cs.brandeis.edu/~tim/Applets/`
[22] `http://ai.uwaterloo.ca/~vanbeek/software/software.html`

und nur eine einzige, fixe Kooperationsstrategie enthält (vgl. Hofstedt et al. 2001, S. 16). Neuere Arbeiten zielen daher darauf ab, mögliche Strategien mittels einer Beschreibungssprache (engl. *strategy description language*) dynamisch spezifizieren und flexibel einsetzen zu können, sind allerdings bisher ausschließlich prototypisch für Common Lisp verfügbar (vgl. Frank et al. 2003a, b).

Eric Monfroy widmet sich in seiner Dissertation bzw. Habilitation ebenfalls dem Themenbereich kooperierender Constraint-Solver (vgl. Monfroy 1996, 2002). Er entwickelt eine Sprache namens BALI zur Steuerung von Kooperationen. BALI erlaubt es, die Kooperation unterschiedlicher Constraint-Solver auf hoher Ebene durch eine eigene Sprache zu beschreiben, und auf diese Weise effizient neue Prototypen von kooperierenden Solvern zu erstellen (vgl. Monfroy 1998, 2000). Neben unterschiedlichen Kooperationsprimitiven, welche die sequentielle, die unabhängig parallele und die nebenläufige Ausführung von Constraint-Lösungsmechanismen erlauben, bietet BALI mehrere Lösungsstrategien an, darunter eine inkrementelle Variante (vgl. Monfroy 2000, S. 215 u. S. 224). Die Implementierung von BALI wurde innerhalb des CLP-Systems ECL^{i}PS^e[23] vorgenommen, welches dementsprechend als „Host-Sprache“ fungiert (vgl. Monfroy 1996, S. 163). Ein neuerer Ansatz besteht in der Nutzung von speziellen Koordinierungssprachen. Eine Reimplementierung von BALI wurde innerhalb der Koordinierungssprache Manifold[24] umgesetzt (vgl. Arbab und Monfroy 1998a, b). Koordinierungssprachen unterstützen die Interaktionen und das kooperative Zusammenwirken mehrerer Komponenten. Sie erlauben die saubere Trennung zwischen den eigentlichen Berechnungen eines Systems und der Koordinierung von Kooperationen (vgl. Arbab 1998a, b).

Die hier genannten Ansätze weisen z. T. eine enge Verwandtschaft mit Distributed CSP und CCP auf. Die Systeme von Hofstedt (2000) und Monfroy (2000) nutzen formale Sprachen zur Definition von Ausführungsvorschriften bzw. zur Definition von Kooperationsstrategien. Die Art und Weise der Kooperation ist dadurch in beiden Ansätzen weitestgehend frei definierbar. Im Gegensatz dazu ist das Ausführungsmodell innerhalb von YACS fest vorgegeben. Es kann lediglich gewählt werden, welche Komponenten für die jeweilige Kooperation genutzt werden sollen. Der Ansatz von Hofstedt (2000) berücksichtigt allerdings nicht den Faktor eines inkrementell anwachsenden Constraint-Netzes. In das von Monfroy (2000) entwickelte System BALI sind zwar Constraint-Solver und Preprozessing-Mechanismen für reelle Zahlen und reellwertige Intervalle integriert, es ist allerdings kein Solver für finite Domänen vorgesehen.

Das Konzept für YACS wurde aus der Idee heraus geboren, größtmögliche Flexibilität hinsichtlich der einzusetzenden Constraint-Lösungsmechanismen zu bieten. Weniger im Vordergrund steht der Aspekt der flexiblen *Steuerung* von Kooperationen unterschiedlicher Lösungsverfahren. Die Art der Kooperation ist innerhalb von YACS grundsätzlich durch den Aufbau der Constraint-Lösungsstrategien und deren bisher ausschließlich *sequentiellen* Ausführung vorgegeben. Flexibilität wird innerhalb von YACS durch unterschiedliche und auswechselbare Constraint-Solver für finite und infinite Domänen sichergestellt. Das YACS-Framework stellt somit einen Mittelweg dar zwischen einer flexiblen Kooperation,

[23] `http://www.icparc.ic.ac.uk/eclipse/`
[24] `http://www.cwi.nl/projects/manifold/manifold.html`

wie sie die Kooperationssprachen von Hofstedt (2000) und Monfroy (2000) bieten, und einer starren Ad-hoc-Kooperation.

3.7 Bewertung

Das Framework-Konzept von YACS bietet eine Architektur zur Implementierung von Constraint-Lösungsverfahren und abstrahiert von den tatsächlich eingesetzten Verfahren (vgl. Abschnitt 3.3, S. 7). Die Framework-Architektur wiederum wird in Bezug auf die Flexibilität und Nutzerfreundlichkeit optimal ergänzt durch das in Abschnitt 3.4 auf Seite 8 beschriebene Strategiekonzept.

In den Constraint-Lösungsstrategien werden die tatsächlich einzusetzenden Lösungsverfahren definiert. Dies geschieht problemabhängig und flexibel je nach Anwendung und Einsatzzweck. Der Wissensingenieur kann sich somit bei der Erstellung der Wissensbasis auf vordefinierte und dokumentierte Constraint-Lösungsstrategien stützen, und diese zur Behandlung der im Rahmen der Konfigurierung entstehenden Constraint-Probleme gezielt einsetzen. Sollten Anpassungen notwendig werden, so ist eine einfache Wartung, Pflege und auch Neuimplementierung oder -anbindung von Constraint-Lösungsverfahren durch eine modulare Struktur und die Framework-Architektur gewährleistet.

In Abschnitt 3.5.2 auf Seite 11 wurden in zwei Szenarien unterschiedliche Ausführungsmodelle dargelegt. In Verbindung mit den diskutierten Aspekten bzgl. Überlappungen von Constraint-Netzen ergeben sich insgesamt vier Möglichkeiten hinsichtlich der Funktionalität einer Realisierung von YACS:

1. **Zwei Strategien und getrennte Constraint-Netze (lokale Sichtweise):**

   Es werden für jedes Konfigurierungsproblem ausschließlich zwei Constraint-Lösungsstrategien – je eine für finite und infinite Domänen – vorgesehen. Die Constraint-Netze der beiden Teilprobleme dürfen sich nicht überlappen.

   Diese Voraussetzung resultiert in zwei sauber getrennte Constraint-Netze, die unabhängig voneinander propagiert und aufgelöst werden können, ähnlich wie zurzeit innerhalb von EngCon die unterschiedlichen Constraint-Arten (Tupel-, Java- und Funktions-/Prädikat-Constraints) separat behandelt werden. Die „Meta-Propagation“ zur Erstellung von konsistenten Konfigurierungslösungen wird in diesem Fall von EngCon geleistet.

2. **Beliebig viele Strategien und getrennte Constraint-Netze (lokale Sichtweise):**

   Diese Annahme dehnt obiges auf beliebig viele Constraint-Netze aus, die sich ebenfalls nicht überschneiden dürfen. Für jedes Konfigurierungsproblem können beliebig viele Strategien definiert und eingesetzt werden (theoretisch für jedes Constraint eine andere). Die Constraint-Netze der jeweils entstehenden Teilprobleme dürfen sich allerdings auch hier aus Gründen der Vereinfachung nicht überlappen.

   Jedes Teilproblem wird damit separat propagiert, ebenso wie die unterschiedlichen Constraint-Arten (Tupel-Constraints, Java-Constraints, etc.) innerhalb von Eng-

Con. Auch in diesem Fall kommt EngCon die Aufgabe zu, die einzelnen Teillösungen innerhalb einer konsistenten Gesamtkonfiguration zusammenzuführen.

3. **Zwei Strategien und überlappende Constraint-Netze (globale Sichtweise):**

   In diesem Fall werden wiederum lediglich zwei Strategien (finite und infinite Domänen) für ein Konfigurierungsproblem zugelassen. Da Überlappungen möglich sein sollen, ist ein Meta-Constraint-Solver erforderlich, der aufgrund der begrenzten Anzahl unterschiedlicher Teilprobleme relativ überschaubaren Verwaltungsaufwand zu leisten hat.

   Trotzdem müssen bei einer Realisierung dieser Möglichkeit die in Abschnitt 3.5.2 auf Seite 11 und insbesondere Abschnitt 3.5.3 auf Seite 19 beschriebenen Methoden bzgl. überlappender Constraint-Netze und zum Auflösen des entstehenden Mixed CSP umgesetzt werden. Dadurch wird YACS in die Lage versetzt, globale Lösungen für die zu verarbeitenden algebraischen Constraints zu generieren.

4. **Beliebig viele Strategien und überlappende Constraint-Netze (globale Sichtweise):**

   Diese Möglichkeit bietet maximale Flexibilität hinsichtlich des Einsatzes unterschiedlicher Constraint-Lösungsstrategien und in Bezug auf das in Abschnitt 3.5.2 auf Seite 11 geschilderte, phasenweise Ausführungsmodell. Es können beliebig viele Strategien zur Verarbeitung unterschiedlicher Teilprobleme genutzt werden, wobei wiederum Überlappungen der Constraint-Netze erlaubt sind. Es ist daher wie bei der vorhergehenden Möglichkeit ein Meta-Constraint-Solver erforderlich. In diesem Fall allerdings ist ein generischer Ansatz erforderlich, d. h. für beliebig viele Teilprobleme.

   Hierbei ist es insbesondere erforderlich, die in Abschnitt 3.5.2 auf Seite 11 und Abschnitt 3.5.3 auf Seite 19 angesprochenen Aspekte und Lösungsmöglichkeiten bzgl. Mixed CSP und überlappender Constraint-Netze zu berücksichtigen. YACS wird es dadurch ermöglicht, globale Lösungen für das vorliegende heterogene Constraint-Problem, bestehend aus unterschiedlichen Teilproblemen, zu erzeugen.

Im Rahmen der Diplomarbeit soll eines dieser vier Szenarien umgesetzt werden. Je flexibler das realisierte Szenario ist, umso eher besteht die Möglichkeit, dass YACS in EngCon und anderen Projekten tatsächlich eingesetzt wird. Wenn sich auch das erstgenannte Szenario mit lediglich zwei voneinander unabhängigen Constraint-Netzen am einfachsten realisieren lässt, würden hier weitergehende Möglichkeiten zur Flexibilisierung des Einsatzes von Constraint-Lösungsverfahren nicht wahrgenommen. Trotzdem ist dieses einfache Szenario für eine rudimentäre Nutzung in EngCon bereits ausreichend und würde eine deutliche Verbesserung hinsichtlich des derzeitigen Zustands bzgl. des flexiblen Einsatzes unterschiedlicher Constraint-Verfahren darstellen.

So gesehen sind die aufgezeigten Szenarien als aufeinander aufbauende Ausbaustufen des YACS-Frameworks innerhalb eines Entwicklungsprojekts zu sehen. Mit dieser Arbeit sollte daher mindestens das erstgenannte Szenario realisiert werden. In Abhängigkeit von dem bis dahin benötigten Aufwand können danach die Szenarien zwei und drei angegangen werden, wobei diese alternativ oder parallel entwickelt werden können. Ob das

letztgenannte Szenario im Rahmen des zur Verfügung stehenden Zeitaufwands innerhalb dieser Arbeit umgesetzt werden kann ist fraglich. Dies ist allerdings auch unbedingt nicht notwendig, um für das derzeit grundsätzliche Problem, der statischen Anbindung eines einzigen Constraint-Solvers an EngCon ausschließlich für Intervall-Constraints, Abhilfe zu schaffen.

4 Zusammenfassung

Für das Konfigurierungswerkzeug EngCon werden Constraint-Lösungsmethoden für finite und infinite Domänen benötigt. Finite Domänen werden durch diskrete Wertebereiche, infinite Domänen durch kontinuierliche, reellwertige Intervalle repräsentiert. Bestehende Constraint-Systeme erfüllen die Anforderungen für EngCon nur teilweise, zudem ist die Anbindung bestehender Systeme mit relativ hohem Integrationsaufwand verbunden. Die Eigenimplementierung geeigneter Constraint-Verfahren zum Auflösen von klassischen CSPs mit diskreten Domänen und ICSPs mit intervallwertigen Domänen scheint eine angemessene Lösung zu sein.

Aufsetzend auf einer mehrschichtigen Architektur, dem YACS-Framework, ist der Wissensingenieur für EngCon in der Lage, flexibel und abstrahiert von den eigentlichen Lösungsalgorithmen zwischen unterschiedlichen Verfahren zum Auflösen von Constraint-Problemen mit unterschiedlichen Wertedomänen auszuwählen. Das YACS-Framework umfasst eine hierfür geeignete, hybride Schnittstelle basierend auf Constraint-Lösungsstrategien. Eingebettet in eine Framework-Architektur und gesteuert über definierte Lösungsstrategien erhält der Benutzer Zugriff auf eine modulare Bibliothek von Constraint-Lösungsalgorithmen.

Das YACS-Framework bietet somit eine flexible und erweiterbare Plattform für den problemabhängigen und anwendungsspezifischen Einsatz unterschiedlicher Constraint-Lösungsmethoden für finite und infinite Domänen.

Literaturverzeichnis

Hinweis: Die Zahlen am Ende eines jeden Eintrags kennzeichnen die Seitenzahlen der vorliegenden Ausarbeitung, auf denen die jeweilige Quelle zitiert wird.

1. **Abdennadher et al. 2001** Abdennadher, Slim ; Krämer, Ekkerhard ; Saft, Matthias ; Schmauss, Matthias: JACK: A Java Constraint Kit. In: *Proceedings of the International Workshop on Functional and (Constraint) Logic Programming (WFLP'01)*, Universität Kiel (veröffentlicht als technischer Bericht Nr. 2017), 13.–15. September 2001. – URL `http://www.pms.ifi.lmu.de/publikationen/PMS-FB/PMS-FB-2001-10.pdf`. – Zugriffsdatum: 16. März 2005. – Zugl.: (Abdennadher et al. 2002a). – Gekürzte Fassung: (Abdennadher et al. 2002b) 4, 27

2. **Abdennadher et al. 2002a** Abdennadher, Slim ; Krämer, Ekkerhard ; Saft, Matthias ; Schmauss, Matthias: JACK: A Java Constraint Kit. In: *Electronic Notes in Theoretical Computer Science (ENTCS)* 64 (2002), September, S. 1–17. – URL `http://www.cs.guc.edu.eg/faculty/sabdennadher/Publikationen/paperENTC.ps.gz`. – Zugriffsdatum: 16. März 2005. – Zugl.: (Abdennadher et al. 2001). – Gekürzte Fassung: (Abdennadher et al. 2002b). – ISSN 1571-0661 4, 27

3. **Abdennadher et al. 2002b** Abdennadher, Slim ; Krämer, Ekkerhard ; Saft, Matthias ; Schmauss, Matthias: JACK: A Java Constraint Kit. In: *ALP Newsletter (Association for Logic Programming)* 15 (2002), Februar, Nr. 1. – URL `http://www.cs.kuleuven.ac.be/~dtai/projects/ALP/newsletter/feb02/nav/monfroy/monfroy_toprint.pdf`. – Zugriffsdatum: 2. Februar 2005. – Ausführliche Version: (Abdennadher et al. 2001, 2002a) 4, 27

4. **Arbab 1998a** Arbab, Farhad: The Coordination Language Manifold. In: *ERCIM News, online edition* (1998), Oktober, Nr. 35. – URL `http://www.ercim.org/publication/Ercim_News/enw35/arbab.html`. – Zugriffsdatum: 6. Juni 2005 23

5. **Arbab 1998b** Arbab, Farhad: What Do You Mean, Coordination? In: Brune, Mieké (Hrsg.) ; Willem Klop, Jan (Hrsg.) ; Rutten, Jan (Hrsg.): *Nieuwsbrief van de Nederlandse Vereniging voor Theoretische Informatica*. Amsterdam, The Netherlands, März 1998, S. 11–22. – URL `http://www.cwi.nl/pub/manifold/NVTIpaper.ps.Z`. – Zugriffsdatum: 31. Mai 2005. – March'98 Issue of the Bulletin of the Dutch Association for Theoretical Computer Science 23

6. **Arbab und Monfroy 1998a** Arbab, Farhad ; Monfroy, Eric: Coordination of Heterogeneous Distributed Cooperative Constraint Solving. In: *ACM SIGAPP Applied Computing Review* 6 (1998), September, Nr. 2, S. 4–17. – URL `http://www.sciences.univ-nantes.fr/info/perso/permanents/monfroy/Papers/acr98.ps.gz`. – Zugriffsdatum: 1. Juni 2005. – Special Issue on Coodination Languages and Models. – Zugl.: CWI Technical Report SEN-R9828, ISSN 1386-369X. – Vorhergehende Version: (Arbab und Monfroy 1998b) 23, 28

7. **Arbab und Monfroy 1998b** Arbab, Farhad ; Monfroy, Eric: Using Coordination for Cooperative Constraint Solving. In: George, K. M. (Hrsg.) ; Lamong, Gary B. (Hrsg.): *Proceedings of the ACM Symposium on Applied Computing (SAC'98), Atlanta, Georgia, USA, 27. Februar – 1. März 1998*. New York, NY, USA : ACM Press, 1998, S. 139–148. – URL `http://www.sciences.univ-nantes.fr/info/perso/permanents/monfroy/Papers/sac98.ps.gz`. – Zugriffsdatum: 1. Juni 2005. – Erweiterte Fassung: (Arbab und Monfroy 1998a). – ISBN 0-89791-969-6 23, 27

8. **Barták 1999a** Barták, Roman: Constraint Programming: In Pursuit of the Holy Grail. In: *Proceedings of the Week of Doctoral Students (WDS'99), Part IV (Invited Lecture)*. Prague, Czechia : MatFyzPress, Juni 1999, S. 555–564. – URL `http://ktilinux.ms.mff.cuni.cz/~bartak/downloads/WDS99.pdf`. – Zugriffsdatum: 14. September 2004. – Vorhergehende Version: (Barták 1999b) 5, 28

9. **Barták 1999b** Barták, Roman: Constraint Programming: What is Behind? In: *Proceedings of the Workshop on Constraint Programming for Decision and Control (CPDC'99), Invited Talk*. Gliwice, Poland, Juni 1999, S. 7–15. – URL `http://ktilinux.ms.mff.cuni.cz/~bartak/downloads/CPDC99.pdf`. – Zugriffsdatum: 22. September 2004. – Erweiterte Fassung: (Barták 1999a) 5, 28

10. **Barták 2001** Barták, Roman: Theory and Practice of Constraint Propagation. In: *Proceedings of the 3rd Workshop on Constraint Programming for Decision and Control (CPDC'01), Invited Lecture*. Gliwice, Poland, Juni 2001, S. 7–14. – URL `http://ktilinux.ms.mff.cuni.cz/~bartak/downloads/CPDC2001.zip`. – Zugriffsdatum: 14. September 2004 5

11. **Benhamou 1996** Benhamou, Frédéric: Heterogeneous Constraint Solving. In: Hanus, Michael (Hrsg.) ; Rodríguez-Artalejo, Mario (Hrsg.): *Proceedings of the 5th International Conference on Algebraic and Logic Programming (ALP'96), Aachen, 25.–27. September 1996* Bd. 1139. Berlin, Heidelberg, New York : Springer Verlag, 1996, S. 62–76. – URL `http://www.sciences.univ-nantes.fr/info/perso/permanents/benhamou/PAPERS/Ben_ALP96.pdf`. – Zugriffsdatum: 15. Mai 2005. – ISBN 3-540-61735-3 22

12. **Benhamou 2001** Benhamou, Frédéric: Interval Constraints. In: Floudas, Christodoulos A. (Hrsg.) ; Pardalos, Panos M. (Hrsg.): *Encyclopedia of Optimization* Bd. 3. Dordrecht, Netherlands : Kluwer Academic Publishers, August 2001, S. 45–48. – URL `http://www.sciences.univ-nantes.fr/info/perso/permanents/benhamou/PAPERS/Ben99_EoO.pdf`. – Zugriffsdatum: 14. September 2004. – ISBN 0-7923-6932-7 6

13. **Benhamou und Granvilliers 1996** Benhamou, Frédéric ; Granvilliers, Laurent: Combining Local Consistency, Symbolic Rewriting and Interval Methods. In: Calmet, Jacques (Hrsg.) ; Campbell, John A. (Hrsg.) ; Pfalzgraf, Jochen (Hrsg.): *Proceedings of the 3rd International Conference on Artificial Intelligence and Symbolic Mathematical Computation (AISMC-3), Steyr, Austria, 23.–25. September*

1996 Bd. 1138. Berlin, Heidelberg, New York : Springer Verlag, 1996, S. 144–159. – URL `http://www.sciences.univ-nantes.fr/info/perso/permanents/granvil/papers/bgaismc96.pdf`. – Zugriffsdatum: 9. Juni 2005. – ISBN 3-540-61732-9 22

14. **Benhamou et al. 1994a** 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. – URL `http://www.sciences.univ-nantes.fr/info/perso/permanents/benhamou/PAPERS/BenMcAlVHen94.pdf`. – Zugriffsdatum: 14. September 2004. – Vorhergehende Version: (Benhamou et al. 1994b). – ISBN 0-262-52191-1 6, 29

15. **Benhamou et al. 1994b** Benhamou, Frédéric ; McAllester, David ; Van Hentenryck, Pascal: CLP(Intervals) Revisited / University of Marseille. France, April 1994 (Technical Report CS-94-18). – Forschungsbericht. – 16 S. – URL `ftp://ftp.cs.brown.edu/pub/techreports/94/cs94-18.ps.Z`. – Zugriffsdatum: 22. September 2004. Überarb. Fassung: (Benhamou et al. 1994a) 6, 29

16. **Cleary 1987** Cleary, John G.: Logical Arithmetic. In: *Future Computing Systems* 2 (1987), Nr. 2, S. 125–149. – ISSN 0266-7207 6

17. **Cunis et al. 1991** 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). – 279 S. – ISBN 3-540-53683-3 35

18. **Davis 1987** Davis, Ernest: Constraint Propagation with Interval Labels. In: *Artificial Intelligence* 32 (1987), Juli, Nr. 3, S. 281–331. – ISSN 0004-3702 1, 6

19. **Dechter 1992** Dechter, Rina: Constraint Networks. In: Shapiro, Stuart C. (Hrsg.): *Encyclopedia of Artificial Intelligence* Bd. 1. 2. Aufl. Chichester, London, New York : John Wiley & Sons, Februar 1992, S. 276–285. – URL `http://www.ics.uci.edu/~csp/r17-survey.pdf`. – Zugriffsdatum: 16. September 2004. – ISBN 0-4715-0307-X 5

20. **Dechter 1999** Dechter, Rina: Constraint Satisfaction. In: Wilson, Robert A. (Hrsg.) ; Keil, Frank C. (Hrsg.): *The MIT Encyclopedia of the Cognitive Sciences (MITECS)*. Cambridge, Massachusetts, USA : Bradford Books/The MIT Press, 1. Juni 1999, S. 195–197. – URL `http://www.ics.uci.edu/~csp/R68.pdf`. – Zugriffsdatum: 16. September 2004. – ISBN 0-262-73124-X 5

21. **Dechter 2000** Dechter, Rina (Hrsg.): *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming (CP 2000), Singapore, 18.–21. September 2000*. Berlin, Heidelberg, New York : Springer Verlag, 2000. (LNCS 1894). – ISBN 3-540-41053-8 32

22. **Dechter und Rossi 2003** Dechter, Rina ; Rossi, Francesca: Constraint Satisfaction. In: Nadel, Lynn (Hrsg.): *Encyclopedia of Cognitive Science (ECS)* Bd. 1. London, New York, Tokyo : Nature Publishing Group/Macmillan Publishers, Juli 2003, S. 793–780. – URL `http://www.ics.uci.edu/~csp/r85.pdf`. – Zugriffsdatum: 16. September 2004. – ISBN 0-333-79261-0 5

23. **Diaz und Codognet 2001** Diaz, Daniel ; Codognet, Philippe: Design and Implementation of the GNU Prolog System. In: *Journal of Functional and Logic Programming (JFLP)* 2001 (2001), Oktober, Nr. 6. – URL `http://danae.uni-muenster.de/lehre/kuchen/JFLP/articles/2001/S01-02/JFLP-A01-06.pdf`. – Zugriffsdatum: 13. Januar 2005. – ISSN 1080-5230 4

24. **van Emden 2002** Emden, Maarten H. van: Interval Constraints / University of Victoria, Computer Science Department. British Columbia, Canada, Stand: 26. Dezember 2002. – Tutorial. – URL `http://csr.uvic.ca/~vanemden/research/intConstrInd.html`. – Zugriffsdatum: 19. April 2005 6

25. **Frank et al. 2003a** Frank, Stephan ; Hofstedt, Petra ; Mai, Pierre R.: A Flexible Meta-Solver Framework for Constraint Solver Collaboration. In: Günter, Andreas (Hrsg.) ; Kruse, Rudolf (Hrsg.) ; Neumann, Bernd (Hrsg.): *Proceedings of the 26th Annual German Conference on AI (KI'03): Advances in Artificial Intelligence, Hamburg, Germany, 15.–18. September 2003.* Berlin, Heidelberg, New York : Springer Verlag, 2003 (LNCS 2821), S. 520–534. – URL `http://uebb.cs.tu-berlin.de/~ph/ph.papers/ki2003.pdf`. – Zugriffsdatum: 24. Mai 2005. – ISBN 3-540-20059-2 23

26. **Frank et al. 2003b** Frank, Stephan ; Hofstedt, Petra ; Mai, Pierre R.: Meta-S: A Strategy-Oriented Meta-Solver Framework. In: Russell, Ingrid (Hrsg.) ; Haller, Susan M. (Hrsg.): *Proceedings of the 16th International Florida Artificial Intelligence Research Society Conference (FLAIRS'03), Special Track on Constraint Solving and Programming, Casa Monica Hotel, St. Augustine, Florida, USA, 12.–14. Mai 2003.* Menlo Park, California, USA : AAAI Press, 2003, S. 177–181. – URL `http://uebb.cs.tu-berlin.de/~ph/ph.papers/flairs2003.pdf`. – Zugriffsdatum: 24. Mai 2005. – ISBN 1-57735-177-0 23

27. **Freuder 1996** Freuder, Eugene C. (Hrsg.): *Proceedings of the 2nd International Conference on Principles and Practice of Constraint Programming (CP'96), Cambridge, Massachusetts, USA, 19.–22. August 1996.* Berlin, Heidelberg, New York : Springer Verlag, 1996. (LNCS 1118). – 572 S. – ISBN 3-540-61551-2 34

28. **Freuder und Mackworth 1994** Freuder, Eugene C. (Hrsg.) ; Mackworth, Alan K. (Hrsg.): *Constraint-Based Reasoning.* Cambridge, Massachusetts, USA : Bradford Books/The MIT Press, Februar 1994 (Special Issues of Artificial Intelligence). – 409 S. – ISBN 0-262-56075-5 33

29. **Gamma et al. 1996** Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Entwurfsmuster – Elemente wiederverwendbarer objektorientierter Software.* 1.

Aufl. München, Boston, San Francisco : Addison-Wesley, 1996. – xx + 479 S. – ISBN 3-89319-950-0 8

30. **Gelle 1998** Gelle, Esther: *On the Generation of Locally Consistent Solution Spaces in Mixed Dynamic CSPs*. Lausanne (Switzerland), Swiss Federal Institute of Technology (EPFL), PhD. Thesis No. 1826, 1998. – xx + 180 S. – URL `http://liawww.epfl.ch/Publications/Archive/Gelle1998.pdf`. – Zugriffsdatum: 18. November 2004 18, 22

31. **Gelle und Faltings 2003** Gelle, Esther ; Faltings, Boi V.: Solving Mixed and Conditional Constraint Satisfaction Problems. In: *Constraints, An International Journal* 8 (2003), April, Nr. 2, S. 107–141. – URL `http://liawww.epfl.ch/Publications/Archive/Gelle2003.pdf`. – Zugriffsdatum: 10. November 2004. – ISSN 1383-7133 18, 22

32. **Godehardt und Seifert 2001** Godehardt, Eicke ; Seifert, Dirk: *Kooperation und Koordination von Constraint Solvern – Implementierung eines Prototyps*, Technische Universität Berlin, Diplomarbeit, Januar 2001 22

33. **Granvilliers 2001** Granvilliers, Laurent: On the Combination of Interval Constraint Solvers. In: *Reliable Computing* 7 (2001), Dezember, Nr. 6, S. 467–483. – ISSN 1385-3139 22

34. **Granvilliers et al. 2001a** Granvilliers, Laurent ; Monfroy, Eric ; Benhamou, Frédéric: Cooperative Solvers in Constraint Programming: A Short Introduction. In: *ALP Newsletter (Association for Logic Programming)* 14 (2001), Mai, Nr. 2. – URL `http://www.cs.kuleuven.ac.be/~dtai/projects/ALP/newsletter/may01/nav/cooperation/cooperation.ps`. – Zugriffsdatum: 12. Juni 2005. – Zugl.: (Granvilliers et al. 2001b) 21, 22, 31

35. **Granvilliers et al. 2001b** Granvilliers, Laurent ; Monfroy, Eric ; Benhamou, Frédéric: Cooperative Solvers in Constraint Programming: A Short Introduction. In: Monfroy, Eric (Hrsg.) ; Granvilliers, Laurent (Hrsg.): *Proceedings of the Workshop on Cooperative Solvers in Constraint Programming (CoSolv'01) at the 7th International Conference on Principles and Practice of Constraint Programming (CP'01)*. Paphos, Zypern, 1. Dezember 2001, S. 1–6. – URL `http://www.sciences.univ-nantes.fr/info/recherche/Theme_Contraintes/cosolv/Papers/Granvilliers_Monfroy_Benhamou.pdf`. – Zugriffsdatum: 1. Juni 2005. – Zugl.: (Granvilliers et al. 2001a) 21, 22, 31

36. **Granvilliers et al. 2001c** Granvilliers, Laurent ; Monfroy, Eric ; Benhamou, Frédéric: Symbolic-Interval Cooperation in Constraint Programming (Survey Paper). In: *Proceedings of the 2001 International Symposium on Symbolic and Algebraic Computation (ISSAC'01), London, Ontario, Canada, 22.–25. Juli 2001*. New York, NY, USA : ACM Press, 2001, S. 150–166. – URL `http://www.sciences.univ-nantes.fr/info/perso/permanents/benhamou/PAPERS/GraMonBen_ISSAC2001.pdf`. – Zugriffsdatum: 9. Juni 2005. – ISBN 1-58113-417-7 21, 22

37. **Gülden 1993** GÜLDEN, Oliver: *Entwurf und Implementierung eines prototypischen Constraintsystems für das Konfigurierungswerkzeug KONWERK*, Universität Hamburg, Fachbereich Informatik, Studienarbeit, Dezember 1993. – iii + 76 S. – (PROKON-Memo Nr. 46) 1

38. **Güsgen 2000** GÜSGEN, Hans W.: Constraints. In: GÖRZ, Günther (Hrsg.) ; ROLLINGER, Claus-Rainer (Hrsg.) ; JOSEF, Schneeberger (Hrsg.): *Handbuch der Künstlichen Intelligenz*. 3., vollst. überarb. Aufl. München : Oldenbourg Verlag, 2000, Kap. 8, S. 267–287. – ISBN 3-486-25049-3 5

39. **Haroud und Faltings 1994** HAROUD, Djamila ; FALTINGS, Boi V.: Global Consistency for Continuous Constraints. In: BORNING, Alan (Hrsg.): *Proceedings of the 2nd International Workshop on Principles and Practice of Constraint Programming (PPCP'94), Rosario, Orcas Island, Washington, USA, 2.–4. Mai 1994*. Berlin, Heidelberg, New York : Springer Verlag, 1994 (LNCS 874), S. 40–50. – URL `http://liawww.epfl.ch/Publications/Archive/Haroud1994.pdf`. – Zugriffsdatum: 14. September 2004. – Zugl.: Cohn, Anthony G. (Hrsg.): Proceedings of ECAI'94, John Wiley & Sons, 1994, S. 115–119. – ISBN 3-540-58601-6 7

40. **Hofstedt 2000** HOFSTEDT, Petra: Cooperating Constraint Solvers. In: (Dechter 2000), S. 520–524. – URL `http://uebb.cs.tu-berlin.de/~ph/ph.papers/cp2000.pdf`. – Zugriffsdatum: 24. Mai 2005. – ISBN 3-540-41053-8 22, 23, 24

41. **Hofstedt 2001** HOFSTEDT, Petra: *Cooperation and Coordination of Constraint Solvers*. Aachen : Shaker Verlag, September 2001 (Berichte aus der Informatik 124). – xiv + 222 S. – URL `http://uebb.cs.tu-berlin.de/~ph/ph.papers/Hofstedt.PhDThesis.2001.pdf`. – Zugriffsdatum: 24. Mai 2005. – Zugl.: Dresden, Technische Universität, Fakultät Informatik, Dissertation, 2001. – ISBN 3-8265-9351-0 22

42. **Hofstedt et al. 2001** HOFSTEDT, Petra ; SEIFERT, Dirk ; GODEHARDT, Eicke: A Framework for Cooperating Constraint Solvers – A Prototypic Implementation. In: MONFROY, Eric (Hrsg.) ; GRANVILLIERS, Laurent (Hrsg.): *Proceedings of the Workshop on Cooperative Solvers in Constraint Programming (CoSolv'01) at the 7th International Conference on Principles and Practice of Constraint Programming (CP'01)*. Paphos, Zypern, 1. Dezember 2001, S. 7–21. – URL `http://uebb.cs.tu-berlin.de/~ph/ph.papers/cosolv2001.pdf`. – Zugriffsdatum: 24. Mai 2005 22, 23

43. **Hollmann et al. 2000** HOLLMANN, Oliver ; WAGNER, Thomas ; GÜNTER, Andreas: EngCon – A Flexible Domain-Independent Configuration Engine. In: *Proceedings of the Workshop on Configuration at the 14th European Conference on Artificial Intelligence (ECAI 2000)*. Humboldt-Universität zu Berlin, 21.–22. August 2000, S. 94–96. – URL `http://www.hitec-hh.de/ueberuns/home/aguenter/literatur/ecai2000.pdf`. – Zugriffsdatum: 15. September 2004 1

44. **Hyvönen 1989** HYVÖNEN, Eero: Constraint Reasoning Based on Interval Arithmetic. In: (Sridharan 1989), S. 1193–1198. – ISBN 1-55860-094-9 6

45. **Hyvönen 1991** Hyvönen, Eero: Global Consistency in Interval Constraint Satisfaction. In: Mayoh, Brian H. (Hrsg.): *Proceedings of the 3rd Scandinavian Conference Conference on Artificial Intelligence (SCAI'91), Roskilde, Denmark, 21.–24. Mai 1991* Bd. 12. Amsterdam, The Netherlands : IOS Press, 1991, S. 241–251. – ISBN 90-5199-056-1 6

46. **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. – Special Volume on Constraint Based Reasoning. – Zugl.: (Freuder und Mackworth 1994), S. 71–112. – ISSN 0004-3702 1, 6

47. **Kumar 1992** Kumar, Vipin: Algorithms for Constraints Satisfaction Problems: A Survey. In: *AI Magazine* 13 (1992), Nr. 1, S. 32–44. – URL `http://www-users.cs.umn.edu/~kumar/papers/csp-aimagazine.ps`. – Zugriffsdatum: 21. September 2004. – ISSN 0738-4602 5

48. **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 6

49. **Marti und Rueher 1995** Marti, Philippe ; Rueher, Michel: A Distributed Cooperating Constraints Solving System. In: *International Journal on Artificial Intelligence Tools (IJAIT)* 4 (1995), Juni, Nr. 1–2, S. 93–113. – URL `http://www.essi.fr/~rueher/Publis/IJAIT.ps`. – Zugriffsdatum: 12. Juni 2005. – ISSN 0218-2130 22

50. **Monfroy 1996** Monfroy, Eric: *Solver Collaboration for Constraint Logic Programming.* France, Université Henri Poincaré – Nancy I, Centre de Recherche en Informatique de Nancy, INRIA-Lorraine, PhD Thesis, 8. November 1996. – viii + 221 S. – URL `http://www.sciences.univ-nantes.fr/info/perso/permanents/monfroy/Papers/vext_abs.ps.gz`. – Zugriffsdatum: 26. Mai 2005 23

51. **Monfroy 1998** Monfroy, Eric: A Solver Collaboration in BALI. In: Jaffar, Joxan (Hrsg.): *Proceedings of the Joint International Conference and Symposium on Logic Programming (JICSLP'98), Manchester, UK, 15.–19. Juni 1998.* Cambridge, Massachusetts, USA : The MIT Press, 1998, S. 349–350 (Poster). – URL `http://www.sciences.univ-nantes.fr/info/perso/permanents/monfroy/Papers/poster_jicslp.ps.gz`. – Zugriffsdatum: 1. Juni 2005. – ISBN 0-262-60031-5 22, 23

52. **Monfroy 2000** Monfroy, Eric: The Constraint Solver Collaboration Language of BALI. In: Gabbay, Dov (Hrsg.) ; Rijke, Maarten de (Hrsg.): *Proceedings of the 2nd International Workshop Frontiers of Combining Systems (FroCoS'98), Amsterdam, The Netherlands, 2.–4. Oktober 1998.* Baldock, Hertfordshire, UK : Research Studies Press, 2000 (Studies in Logic and Computation Vol. 7), S. 211–230. – URL `http://www.sciences.`

univ-nantes.fr/info/perso/permanents/monfroy/Papers/frocos98.ps.gz. – Zugriffsdatum: 27. Mai 2005 22, 23, 24

53. **Monfroy 2002** Monfroy, Eric: *Cooperative Constraint Solving.* France, Université de Nantes, Institut de Recherche en Informatique de Nantes, Habilitation Thesis, 22. November 2002. – viii + 212 S. – URL http://www.sciences.univ-nantes.fr/info/perso/permanents/monfroy/Papers/hdr.pdf. – Zugriffsdatum: 26. Mai 2005 23

54. **Podelski 1995** Podelski, Andreas (Hrsg.): *Constraint Programming: Basics and Trends, Châtillon Spring School, Châtillon-sur-Seine, France, 16.–20. Mai 1994, Selected Papers.* Berlin, Heidelberg, New York : Springer Verlag, 1995 (LNCS 910). – 90–107 S. – ISBN 3-540-59155-9 34

55. **Ranze et al. 2002** Ranze, Christoph ; Scholz, Thorsten ; Wagner, Thomas ; Günter, Andreas ; Herzog, Otthein ; Hollmann, Oliver ; Schlieder, Christoph ; Arlt, Volker: A Structure-Based Configuration Tool: Drive Solution Designer – DSD. In: Dechter, Rina (Hrsg.) ; Sutton, Rich (Hrsg.) ; Kearns, Michael (Hrsg.) ; Chien, Steve (Hrsg.) ; Riedl, John (Hrsg.): *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI'02) and 14th Conference on Innovative Applications of Artificial Intelligence (IAAI'02), Edmonton, Alberta, Canada, 28. Juli – 1. August 2002.* Menlo Park, California, USA/Cambridge, Massachusetts, USA : AAAI Press/The MIT Press, September 2002, S. 845–852. – URL http://www.hitec-hh.de/ueberuns/home/aguenter/literatur/IAAI2002.pdf. – Zugriffsdatum: 15. September 2004. – ISBN 0-262-51129-0 1

56. **Roy et al. 2000** Roy, Pierre ; Liret, Anne ; Pachet, François: The Framework Approach for Constraint Satisfaction. In: *ACM Computing Surveys (CSUR)* 32 (2000), März, Nr. 1es. – URL http://www.csl.sony.fr/downloads/papers/1998/roy98a.pdf. – Zugriffsdatum: 22. Februar 2005. – CSUR Electronic Symposium on Object-Oriented Application Frameworks – Artikel Nr. 63. – ISSN 0360-0300 8

57. **Rueher 1995** Rueher, Michel: An Architecture for Cooperating Constraint Solvers on Reals. In: (Podelski 1995), S. 231–250. – URL http://www.essi.fr/~rueher/Publis/SpringSchool.ps. – Zugriffsdatum: 12. Juni 2005. – Zugl.: Research Report 94-54, Informatique, Signaux et Systèmes de Sophia Antipolis (I3S), Université de Nice Sophia Antipolis, France, 1994. – ISBN 3-540-59155-9 22

58. **Sam-Haroud und Faltings 1996a** 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. – URL http://liawww.epfl.ch/Publications/Archive/Sam-Haroud1996a.pdf. – Zugriffsdatum: 14. September 2004. – ISSN 1383-7133 7

59. **Sam-Haroud und Faltings 1996b** Sam-Haroud, Djamila ; Faltings, Boi V.: Solving Non-Binary Convex CSPs in Continuous Domains. In: (Freuder 1996), S. 410–424. – URL http://liawww.epfl.ch/Publications/Archive/Sam-Haroud1996.pdf. – Zugriffsdatum: 14. September 2004. – ISBN 3-540-61551-2 7

60. **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. – URL `http://liawww.epfl.ch/~haroud/Thesis-SamHaroud.ps.gz`. – Zugriffsdatum: 14. September 2004 7

61. **Sridharan 1989** SRIDHARAN, Natesa S. (Hrsg.): *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI'89), Detroit, Michigan, USA, 20.–26. August 1989*. San Mateo, California, USA : Morgan Kaufmann Publishers, 1989. – ISBN 1-55860-094-9 32

62. **Syska und Cunis 1991** SYSKA, Ingo ; CUNIS, Roman: Constraints in PLAKON. In: (Cunis et al. 1991), Kap. 6, S. 77–91. – ISBN 3-540-53683-3 1

63. **Tsang 1993** TSANG, Edward P. K.: *Foundations of Constraint Satisfaction*. London, San Diego, New York : Academic Press, 1993 (Computation in Cognitive Science). – 421 S. – URL `http://cswww.essex.ac.uk/CSP/papers/Tsang-Fcs1993.pdf/`. – Zugriffsdatum: 20. September 2004. – ISBN 0-12-701610-4 5

64. **Wallace 1993** WALLACE, Richard J.: Why AC-3 is Almost Always Better than AC-4 for Establishing Arc Consistency in 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. 239–245. – ISBN 1-55860-300-X 5

65. **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 6