

Seminar Constraint Programmierung

Kombination von Constraint-Lösern

Marcos dos Santos Rocha
Matrikelnummer 204270
Betreuer: Frank Zartmann

24. Juni 1998

Literatur

- [1] E.Lamma, M.Milano, P.Mello “A Multi-Level CLP Architecture for Consistency Techniques”, Constraint’96 Workshop, 1996.
- [2] E.Lamma, M.Milano, P.Mello “A Meta Constraint Logic Programming Scheme”, DEIS Technical Report DEIS-LIA-95-005, <http://www.deis.unibo.it/reasearch/>.
- [3] E.Lamma, M.Milano, P.Mello “An Incremental Consistency Algorithm for Adaptive Constraint Satisfaction”, DEIS Technical Report DEIS-LIA-97-001, <http://www.deis.unibo.it/reasearch/>.
- [4] T.Frühwirth, S.Abdennadher “Constraint-Programmierung: Grundlagen und Anwendungen”, Springer-Verlag Berlin Heidelberg, 1997.
- [5] J.Jaffar, M.J.Maher, “Constraint Logic Programming: a Survey”, in Journal of Logic Programming on 10 years of Logic Programming, 1994, pp.503-581.

Inhaltsverzeichnis

1	Einleitung	3
2	Multi-Level CLP-Schema	4
2.1	Grundlagen	4
3	Operationale Semantik	5
3.1	Die operationale Semantik einer einzelnen Komponente	5
3.2	Wechselwirkung zwischen zwei Komponenten	7
3.3	“Top-Down“-Ausführung	9
4	k-Konsistenz mit dem Meta-Level-CLP-Schema	10
4.1	Die Domänen	11
4.2	Beispiel	12
4.3	Die operationale Semantik	13
5	Qualitative Reasoning	15
5.1	Die Domänen	16
5.2	Beispiel	17
5.3	Die operationale Semantik	18
6	Zusammenfassung	19

1 Einleitung

In den letzten Jahren wurden mehrere CLP-Löser (constraint logic programming solvers) entwickelt, die in verschiedenen Anwendungen erfolgreich eingesetzt werden. Zur Lösung mancher Probleme reicht aber das reine CLP-Schema i.allg. nicht aus. In diesen Fällen ist eine Erweiterung des grundlegenden CLP-Schemas erforderlich.

Beim “qualitative reasoning” betrachtet man Disjunktionen von Constraints, d.h. Mengen von Relationen zwischen Variablen wie beispielsweise $X\{<, =\}Y$ (entspricht $(X < Y) \vee (X = Y)$). Dabei braucht man ein Werkzeug, das in der Lage ist, zu prüfen, ob eine bestimmte Menge von Constraints konsistent ist. Seien zwei Variablen X und Y mit einem nichtdefinierten Wertebereich und zwei inkonsistenten Constraints $X < Y$ und $X > Y$. Der Wertebereich der Variablen wird standardmäßig auf $[-max, max]$ gesetzt. Die Inkonsistenz wird von einem CLP(FD)-Löser (finite domain solver) aber erst nach einer Anzahl von Propagierungen, die proportional zu max ist, festgestellt. Wenn man direkt auf die Relationen schließt, kann diese Inkonsistenz in einem einzelnen Propagierungsschritt festgestellt werden. Es ist also ein CLP-System erforderlich, das auf Constraints schließen kann.

Oft will man bei einem gegebenen CSP (constraint satisfaction problem) stärkere (“tighter”¹) Constraints ableiten. Es seien drei Zeitpunkte T_1, T_2 und T_3 im Wertebereich $[1..10]$ mit den Constraints $T_1 \leq T_2$ und $T_2 < T_3$. Die Constraint-Propagierung vom CLP(FD)-Löser reduziert die Wertebereiche der Variablen auf $D_{T_1} = D_{T_2} = [1..9]$ und $D_{T_3} = [2..10]$. Durch die Anwendung von CLP(FD)-Läsern ist aber nicht möglich eine Relation zwischen T_1 und T_3 abzuleiten.

Ein weiteres Problem besteht darin, daß die meisten Constraint-Löser einen festen eingebetteten Konsistenz-Algorithmus implementieren, der nicht vom Benutzer geändert werden kann. Viele Constraint-Löser benutzen aus Effizienzgründen eine Kantenkonsistenz-Propagierungsmethode (arc-consistency propagation), die allerdings nicht immer ausreicht. Manche Anwendungen, wie beispielweise “temporal reasoning”, benötigen in manchen Fällen eine 4-Konsistenz.

Hier wird zunächst ein allgemeines Multi-Level-CLP-Schema präsentiert, das mehrere Constraint-Löser kombiniert. Danach werden zwei Spezialisierungen dieses Schemas vorgestellt.

Die erste Spezialisierung [1, 2, 3] ermöglicht einen beliebigen Konsistenzgrad zu erreichen ohne den Konsistenz-Propagierungs-Algorithmus der einzelnen Constraint-Löser zu verändern. Hierbei besteht das System aus mehreren CLP(FD)-Läsern, die eine beliebige k -Konsistenz mit $k > 1$ implementieren.

Die zweite Spezialisierung des allgemeinen Multi-Level-Schemas [2] ermöglicht das “qualitative reasoning”. Damit kann man z.B. das minimale Netzwerk be-

¹Eine Constraint-Menge C' ist “tighter” als eine Constraint-Menge C'' , wenn alle Tupel, die von C' erlaubt werden, auch von C'' erlaubt werden.

stimmen, wie es bei “temporal reasoning” notwendig ist. Hierbei besteht das System nur aus zwei Levels, wobei das erste ein CLP(FD)-Löser ist und das zweite ein Meta-Constraint-Löser, der auf die Constraints des ersten Levels schließt. Besonders interessant bei diesem Meta-CLP-Schema ist seine Modularität, Flexibilität, Skalierbarkeit und leichte Implementierung. Die Arbeitsweise des allgemeinen Multi-Level-Constraint-Lösers wird in den Abschnitten 2 und 3 beschrieben. Dabei wird die operationale Semantik und die Regeln, die die Wechselwirkung zwischen den verschiedenen Levels modellieren, erläutert. Diese Regeln verbinden die Constraintspeicher verschiedener Levels, um Informationen auszutauschen und um die Constraints zu propagieren. In den Abschnitten 4 und 5 werden dann die Spezialisierungen vorgestellt.

2 Multi-Level CLP-Schema

2.1 Grundlagen

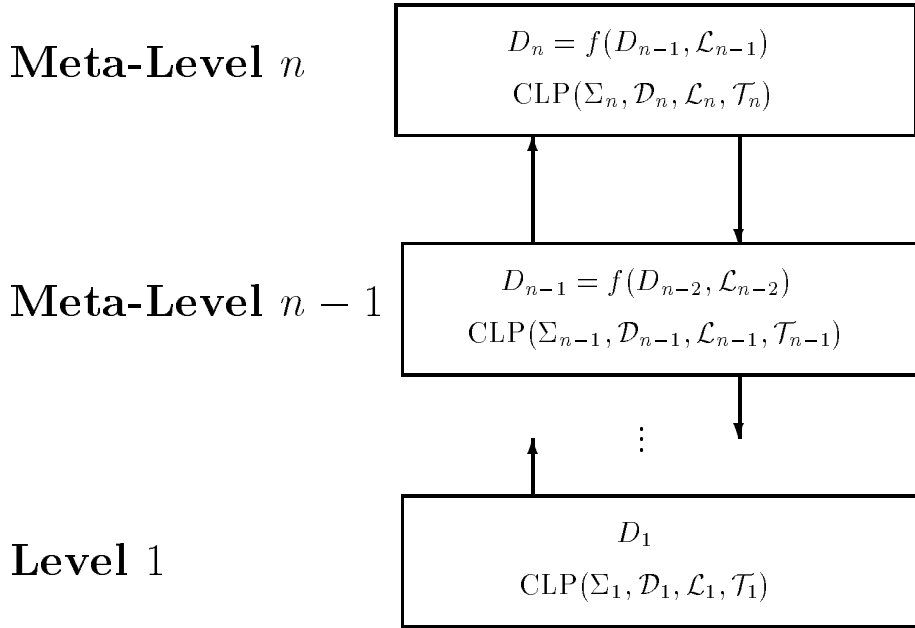
Jede CLP-Sprache kann als eine Instanz des allgemeinen $\text{CLP}(\Sigma, \mathcal{D}, \mathcal{L}, \mathcal{T})$ -Schemas betrachtet werden [5]. Hierbei ist Σ eine Signatur, die die vordefinierten Prädikaten- und Funktionssymbole darstellt, \mathcal{D} eine Σ -Struktur, die aus einer Menge D und einer Interpretation der Σ -Symbole in D besteht, \mathcal{L} eine Klasse von Σ -Formeln (die Constraints) und \mathcal{T} eine Σ -Theorie erster Ordnung, die die Eigenschaften von \mathcal{D} axiomatisiert.

CLP-Sprachen werden durch Festlegung der Komponente dieses 4-Tupels definiert. Um CLP-Systeme zu beschreiben, muß außerdem eine operationale Semantik bestimmt werden. Hier wird die operationale Semantik allgemein definiert und erst durch die Definition der Prädikate *infer* (Propagierung) und *consistent* (Erfüllbarkeit) spezialisiert.

Die CLP-Architektur, die hier präsentiert wird, besteht aus $n \geq 1$ Modulen (Abbildung 1). Diese haben die oben genannte Struktur $(\Sigma, \mathcal{D}, \mathcal{L}, \mathcal{T})$. Die wichtigste Eigenschaft dieses Schemas ist, daß jedes Meta-Level auf die Constraints des darunterliegenden Levels schließt. Wenn das Level i durch $(\Sigma_i, \mathcal{D}_i, \mathcal{L}_i, \mathcal{T}_i)$ definiert ist, so wird die Domäne D_{i+1} des Levels $i + 1$ durch eine Funktion f in Abhängigkeit der Domäne und der Constraints des i -ten Levels bestimmt:

$$D_{i+1} = f(D_i, \mathcal{L}_i)$$

Die Signatur des $(i+1)$ -ten Levels enthält eine implizite oder explizite Darstellung der Constraints in \mathcal{L}_i , die Operationen auf die Constraints von \mathcal{L}_i und die Meta-Constraints zwischen ihnen. Mit implizit ist gemeint, daß die Constraints als Symbole dargestellt werden, während mit explizit eine Aufzählung der Werte, die von den Constraints erlaubt werden, gemeint ist. Eine explizite Darstellung der Constraints ist allerdings nur möglich, wenn mit endlichen Bereichen (FD) gearbeitet wird.

Abbildung 1: Multi-Level-Architektur mit n CLP-Lösern

Die Repräsentation der Meta-Signatur wird gemäß der Anwendung bestimmt. Bei der ersten Spezialisierung (k -Konsistenz) werden die Constraints explizit dargestellt. Die Funktion $f(D_i, \mathcal{L}_i)$ liefert die Darstellung von D_{i+1} als das kartesische Produkt von D_i .

Bei der zweiten Spezialisierung (“qualitative reasoning”) werden die Constraints implizit repräsentiert. In diesem Fall liefert die Funktion f die Darstellung von D_{i+1} als die Potenzmenge der Constraints in \mathcal{L}_i .

3 Operationale Semantik

3.1 Die operationale Semantik einer einzelnen Komponente

Die Komponenten des Meta-CLP-Schemas sind Constraint-Systeme. In diesem Abschnitt wird die operationale Semantik dieser Constraint-Systeme durch ein Zustandsübergangssystem definiert. Das hier präsentierte Transitionssystem realisiert die sogenannte “Top-Down”-Ausführung [5] und beschreibt wie ein Ziel bearbeitet wird.

Ein Zustand ist ein Tupel $\langle A, C, S \rangle$, wobei A der Zielspeicher (eine Menge von Atomen und Constraints), C der aktive Constraint-Speicher und S der passive Constraint-Speicher ist. Ein Anfangszustand hat die Form $\langle Goal, \emptyset, \emptyset \rangle$. Der Endzustand ist ein Zustand, der keinen Folgezustand besitzt. Ein Endzustand

ist erfolgreich, wenn er die Form $\langle \emptyset, C, S \rangle$ hat. Ein erfolgloser Endzustand wird durch *failed* dargestellt.

Das Transitionssystem hat im wesentlichen 4 Reduktionsregeln:

- Resolution \mapsto_r :

Ist das ausgewählte Ziel a ein Atom und $h \leftarrow B$ eine Programmklausele (mit neuen, umbenannten Variablen), so daß h und a dasselbe Prädikatsymbol haben, dann wird das ausgewählte Atom durch den Rumpf der Klausel ersetzt und die Gleichheitsconstraints dem passiven Constraint-Speicher hinzugefügt:

$$\frac{a \text{ ist Atom, } h \leftarrow B \text{ Programmklausele}}{\langle A \cup a, C, S \rangle \mapsto_r \langle A \cup B, C, S \cup (a = h) \rangle}$$

Dabei ist $a = h$ die Abkürzung für die Konjunktion der Gleichheitsconstraints zwischen den entsprechenden Argumenten von a und h . Wenn im Programm keine Klausel definiert ist, deren Kopf dasselbe Prädikatsymbol wie a enthält, führt die Resolutionsregel zu einem Fehlschlag:

$$\frac{a \text{ ist Atom, } \nexists \text{ Programm-Klausele } h \leftarrow B \text{ mit } \textit{consistent}((a = h) \cup C)}{\langle A \cup a, C, S \rangle \mapsto_r \textit{fail}}$$

- Constraint \mapsto_c :

Ist das ausgewählte Ziel ein Constraint, dann wird es aus dem Zielspeicher entfernt und dem passiven Constraint-Speicher hinzugefügt:

$$\frac{c \text{ ist Constraint}}{\langle A \cup c, C, S \rangle \mapsto_c \langle A, C, S \cup c \rangle}$$

- Infer \rightarrow_i :

Die Transitionsregel Infer bestimmt, wie die Constraints propagiert werden. Diese Regel leitet aktive Constraints ab und modifiziert (normalerweise vereinfacht) passive Constraints:

$$\frac{(C', S') = \textit{infer}(C, S)}{\langle A, C, S \rangle \mapsto_i \langle A, C', S' \rangle}$$

Betrachten wir als Beispiel die FD-Sprache von CHIP. Hierbei sind alle unären Constraints aktiv (z.B. $X < 9$ oder $X \neq 0$). Für $S = \{X = Y + 1\}$ und $C = \{2 \leq X \leq 5 \wedge 0 \leq Y \leq 3\}$ ist $\textit{infer}(C, S) = (C', S')$ mit $C' = \{2 \leq X \leq 4 \wedge 1 \leq Y \leq 3\}$ und $S' = S$.

- Satisfiability \mapsto_s :

Testet mit dem Prädikat $consistent(c)$, ob die aktiven Constraints konsistent sind:

$$\frac{consistent(C)}{\langle A, C, S \rangle \mapsto_s \langle A, C, S \rangle}$$

sonst

$$\frac{\neg consistent(C)}{\langle A, C, S \rangle \mapsto_s fail}$$

Eine Ableitung ist eine Transitionsfolge $\langle A_1, C_1, S_1 \rangle \mapsto \dots \mapsto \langle A_i, C_i, S_i \rangle \mapsto \dots$, wobei die \mapsto 's eine der oben genannten Transitionen sind.

3.2 Wechselwirkung zwischen zwei Komponenten

Wir haben uns bis jetzt die Arbeitsweise einer einzelnen Komponente angesehen, nun betrachten wir das Zusammenspiel zweier Constraint-Systeme, die übereinander liegen. Dabei sprechen wir von einem Objekt- und einem Meta-Level (unteres bzw. oberes Level). Jedes dieser Levels ist ein Constraint-System und enthält einen aktiven und einen passiven Constraint-Speicher.

Die Constraint-Speicher des Meta-Levels enthalten implizit oder explizit eine Repräsentation der Constraint-Speicher des Objekt-Levels. Wenn z.B. der passive Objekt-Constraint-Speicher den Constraint $X \leq Y$ enthält, so muß der Meta-Constraint-Speicher seine Darstellung enthalten. Um das zu realisieren, kann man beispielsweise eine Meta-Variable XY definieren. Die Domäne dieser Variable könnte die Constraint-Symbole enthalten, die die Variablen X und Y im Objekt-Level binden. Eine weitere Möglichkeit diesen Constraint im Meta-Level darzustellen, wäre eine Aufzählung der Variablenbelegungen, die vom Objekt-Level erlaubt werden. Das ist allerdings nur möglich, wenn man mit endlichen Bereichen arbeitet.

Jede Änderung in einem Level muß auf das andere Level reflektiert werden. Diese Wechselwirkung zwischen zwei Levels wird durch drei Regeln bestimmt. Die Constraint-Speicher der zwei Systeme, d.h., die aktiven und passiven Objekt- und Meta-Level-Constraints $(C_{obj}, S_{obj}, C_m, S_m)$, werden wie in Abbildung 2 durch drei Regeln verbunden.

Die erste Regel (Pfeil Nummer (1) in Abbildung 2) verbindet den aktiven Meta-Constraint-Speicher C_m mit dem passiven Objekt-Level-Constraint-Speicher S_{obj} . Jede Änderung im aktiven Meta-Constraint-Speicher betrifft die passiven Constraints des darunterliegenden Levels. Deshalb muß der passive Objekt-Level-Constraint-Speicher modifiziert (erweitert oder einfach geändert) werden, immer dann wenn ein aktives Constraint im Meta-Level geändert wird. Die erste Regel kann wie folgt geschrieben werden:

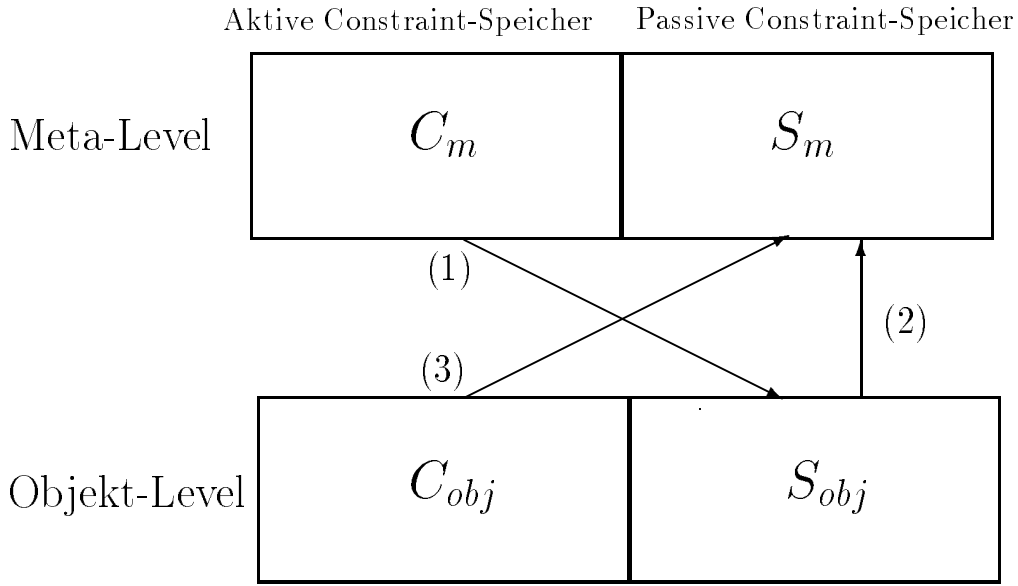


Abbildung 2: Wechselwirkung zwischen zwei Komponenten

$$\frac{C_m}{S_{obj} \mapsto_{(1)} S_{obj} \cup S_{obj-m}},$$

wobei der Constraint-Speicher S_{obj-m} eine Übersetzung der Meta-Constraints in Objekt-Level-Constraints enthält.

Die zweite Bindungsregel (Pfeil Nummer (2) in Abbildung 2) betrifft die Propagierung von Constraints auf das Meta-Level, wenn passive Objekt-Level-Constraints geändert werden. Bevor diese Regel ausgeführt wird, muß aber als erstes eine Transition \mapsto_{is} im Objekt-Level erfolgen, wenn die passiven Objekt-Level-Constraints geändert werden. Erst danach kann der Constraint in ein anderes Level propagiert werden. Die Regel (2) produziert dann ein aktives Meta-Constraint, das eine Meta-Variable erzeugt, die in ihrer Domäne die Relationen des Objekt-Llevels darstellt, die zwischen zwei Objekt-Variablen gelten. Diese Bindungsregel bezieht sich auf:

$$\frac{S_{obj}}{S_m \mapsto_{(2)} S_m \cup S_{m-obj}},$$

wobei $S_{m-obj} = reify(C_{obj}, S_{obj})$ berechnet wird, indem man die Objekt-Level-Constraints in Meta-Constraints reifiziert (konkretisiert).

Die Wechselwirkung zwischen dem Objekt- und dem Meta-Level umfaßt außerdem die Propagierung von Meta-Constraints, wenn aktive Objekt-Level-Constraints geändert werden (Pfeil Nummer (3) in Abbildung 2). Die Verbindungsregel von dem Objekt- in das Meta-Level fügt dem passiven Meta-Constraint-Speicher die

Constraints hinzu, die von den Werten der Objekt-Level-Variablen folgern. Diese Regel hat die Form:

$$\frac{C_{obj}}{S_m \mapsto_{(3)} S_m \cup S'_{m-obj}},$$

wobei S'_{m-obj} , der Constraint-Speicher ist, der aus den aktiven Objekt-Constraints folgt.

An dieser Stelle sollte erwähnt werden, daß neue Constraints immer als erste dem passiven Constraint-Speicher hinzugefügt werden, genauso wie es bei der Transition \mapsto_c passiert. Erst nachher können die Constraints von der Transition \mapsto_i in dem jeweiligen Level in aktive Constraints umgewandelt werden.

Jedes Mal, wenn ein Constraint-Speicher infolge einer Propagierung aus einem anderen Level geändert wird, muß das betroffene System seine eigene Propagierung veranlassen, um die Konsistenz zu prüfen und um neue Constraints abzuleiten.

3.3 “Top-Down”-Ausführung

Nun da die grundlegenden Regeln, die die Wechselwirkung zwischen zwei übereinanderliegenden Levels beschreiben, definiert worden sind, kann das Verhalten des Systems im ganzen durch ein Transitionssystem definiert werden.

Der Anfangszustand hängt von dem auszuführenden Programm ab. Da die Meta-Architektur für den Benutzer möglichst transparent sein sollte, werden die Programme weiterhin so geschrieben, als ob das System aus einem einzigen Level bestehen würde. Die Syntax eines Programms ist gleich der Syntax des ersten Levels. Der Anfangszustand des ersten Levels ist dann $\langle A, \emptyset, \emptyset \rangle$, wobei A die Anfangsmenge von Atomen und Constraints ist. Der Anfangszustand jedes Meta-Levels ist $\langle \emptyset, \emptyset, \emptyset \rangle$.

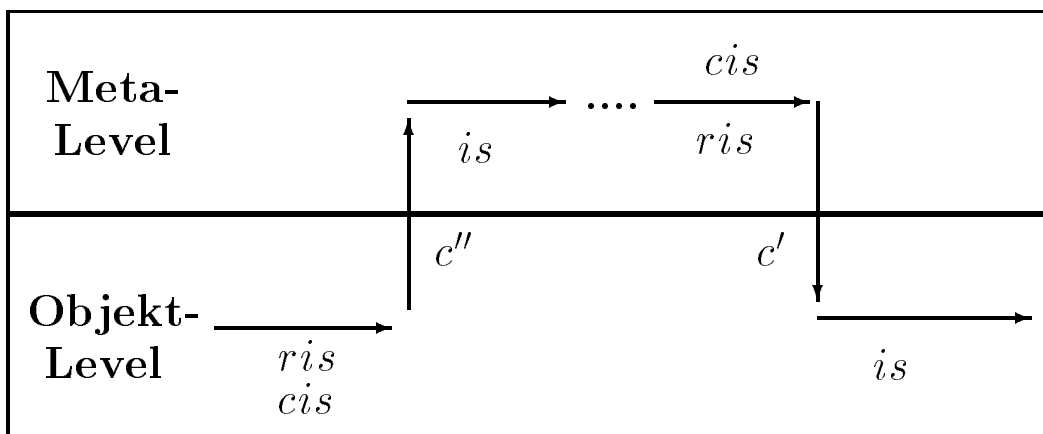


Abbildung 3: Transitionen zwischen Objekt- und Meta-Level

Das Verhalten des Meta- und Objekt-Level-Systems kann durch ihre Ableitungen bestimmt werden. Im folgenden werden die Transitionen bezüglich (1), (2) und (3) aus Abbildung 2 definiert.

Wenn das Meta-Level-System eine *infer*- und eine *consistent*-Transition ausführt, d.h. \mapsto_{is} , werden neue aktive konsistente unäre Constraints erzeugt, die im Objekt-Level mittels einer $\mapsto_{c'}$ -Transition reflektiert werden müssen. Diese Constraints müssen dem passiven Objekt-Level-Constraint-Speicher hinzugefügt werden. Die Transition $\mapsto_{c'}$ sieht der Transition \mapsto_c des Objekt-Systems sehr ähnlich. Der einzige Unterschied ist, daß der Constraint c , das dem passiven Constraint-Speicher hinzugefügt wird, aus einem anderen Level kommt. Nach einer $\mapsto_{c'}$ -Transition muß das Objekt-Level eine \mapsto_{is} -Transition ausführen, um die neuen Constraints zu behandeln. Damit kann die Transition $\mapsto_{c'}$ geschrieben werden als:

$$\frac{\langle A_m, C_m, S_m \rangle \mapsto_{is} \langle A_m, C'_m, S'_m \rangle}{\langle A_{obj}, C_{obj}, S_{obj} \rangle \mapsto_{c'} \langle A_{obj}, C_{obj}, S_{obj} \cup S_{obj-m} \rangle'}$$

wobei S_{obj-m} in 3.2 definiert wurde.

Die Regeln (2) und (3) können zu einer einzigen Transition $\mapsto_{c''}$ zusammengefaßt werden. Diese Transition erfolgt nach der Ausführung einer \mapsto_{is} -Transition im Objekt-Level. Die $\mapsto_{c''}$ -Transition übersetzt aktive und passive Objekt-Level-Constraints in passive Meta-Level-Constraints:

$$\frac{\langle A_{obj}, C_{obj}, S_{obj} \rangle \mapsto_{is} \langle A_{obj}, C'_{obj}, S'_{obj} \rangle}{\langle A_m, C_m, S_m \rangle \mapsto_{c''} \langle A_m, C_m, S_m \cup S_{m-obj} \cup S'_{m-obj} \rangle'}$$

wobei S_{m-obj} und S'_{m-obj} in 3.2 definiert wurden und die Reflektion von S'_{obj} bzw. C'_{obj} darstellen.

Genauso wie bei $\mapsto_{c'}$ kann die Transition $\mapsto_{c''}$ als eine Art von \mapsto_c betrachtet werden, mit dem einzigen Unterschied, daß die Constraints aus einem anderen Level stammen.

Es seien die Transitionen $\mapsto_{ris} = \mapsto_r \mapsto_i \mapsto_s$ und $\mapsto_{cis} = \mapsto_c \mapsto_i \mapsto_s$. Die (partielle) Reihenfolge, in der die Transitionen ausgeführt werden, beschreibt Abbildung 3. Das gesamte System kann durch die Transitionen \mapsto_{ris} , \mapsto_{cis} , $\mapsto_{c'is}$ und $\mapsto_{c''is}$ beschrieben werden. Das System ist also laut Definition in [5] *quick checking* und stellt Inkonsistenzen so früh wie möglich fest.

4 *k*-Konsistenz mit dem Meta-Level-CLP-Schema

Definition: (*k*-Konsistenz) Nehme alle Wertebelegungen von $k - 1$ Variablen, die alle Constraints über diese Variablen erfüllen. Ein Constraint heißt *k*-konsistent, falls für jede *k*-te Variable eine Wertbelegung existiert, so daß alle Constraints über diese *k* Variablen erfüllt sind.

Wir betrachten eine Multi-Level-Architektur, wobei jedes Level ein CLP(FD)-Löser ist.

4.1 Die Domänen

Das erste Level ist ein CLP(FD)-Löser mit der folgenden Domäne: $D = Z$ und

$$\Sigma = \{\{\in [m, n]\}_{m \leq n}, +, =, \neq, <\},$$

wobei $\in [m, n]_{m \leq n}$ bedeutet, daß für jedes Ganzzahlen-Paar (m, n) ($m \leq n$) der Interval-Constraint $x \in [m, n]$ als $m \leq x \leq n$ interpretiert wird. Die anderen Symbole in Σ haben ihre gewöhnliche Bedeutung. \mathcal{L} stellt die Constraints c dar, die von den primitiven Constraints erzeugt werden, wobei für jede Variable in c ein Intervall-Constraint definiert ist. Ein typischer Constraint, der in dieser Domäne ausgedrückt werden kann, ist $x \in [3, 7] \wedge y \in [1, 9] \wedge x \neq 5 \wedge 4x + y < 16$. Die Domäne eines Levels $n > 1$ hängt von den Konsistenz-Algorithmen ab, die von den darunterliegenden Systemen implementiert werden. Wenn alle Levels CLP(FD)-Löser sind, die einen Kantenkonsistenz-Algorithmus implementieren, dann hat das Level n die Domäne $D_n = Z^n$ (kartesisches Produkt von Z n -mal). Wenn man n Levels betrachtet, die jeweils einen Konsistenz-Grad c_i , mit $i = 1..n$, implementieren, dann erreicht das System insgesamt eine k -Konsistenz, mit

$$k = c_1 + \sum_{i=2}^n (c_i - 1).$$

(Beweis durch Induktion in [3]) Damit hat das $(n+1)$ -te Level die Domäne $D_{n+1} = Z^k$. Jedes Level realisiert eine Konsistenz-Technik auf den bereits konsistenten k -Tupeln des darunterliegenden Systems.

Die Signatur des n -ten Levels ist:

$$\Sigma_m = \{\{\in C\}, +_m, =_m, \neq_m, <_m\},$$

wobei die Symbole in Σ_m wie folgt von \mathcal{D}_m interpretiert werden:

$\in C$ ist ein unäres Constraint-Symbol, wobei C eine Teilmenge von D_m ist, d.h. eine Menge von k -Tupeln, die von den Constraints des darunterliegenden Systems erlaubt werden. Beispiel: $C = \{(1, 2), (2, 1)\}$;

Das Symbol $+_m$ summiert die k -Tupel des n -ten Levels komponentenweise;

Die Meta-Constraint-Symbole $=_m$, \neq_m und $<_m$ werden auf Paare von k -Tupeln angewendet, die $k - 1$ gemeinsame Variablen besitzen. Betrachten wir einen Meta-Constraint Δ_m auf zwei k -Tupeln: $(v_1, \dots, v_i, \dots, v_k) \Delta_m (v'_1, \dots, v'_j, \dots, v'_k)$. Dieser Constraint wird genau dann erfüllt, wenn

1. die $k - 1$ gemeinsamen Variablen dieser k -Tupeln gleich sind, d.h. formal $v_l = v'_p$ für $l, p = 1..k$, $l \neq i$, $p \neq j$;
2. die entsprechende Objekt-Level-Relation zwischen den zwei nicht-gemeinsamen (unshared) Variablen erfüllt ist, d.h. formal $v_i \Delta v'_j$.

4.2 Beispiel

Um die Arbeitsweise des Meta-CLP-Lösers zu veranschaulichen, betrachten wir zunächst ein Beispiel. Es sei ein CLP-System, das aus zwei CLP(FD)-Levels besteht, die eine Kantenkonsistenz-Technik implementieren. Es seien X, Y und Z drei Variablen im Wertebereich $[1..2]$ mit den Constraints $X \neq Y, X \neq Z$ und $Y \neq Z$. Intuitiv merkt man, daß diese Constraints mit den möglichen Werten der Variablen Pfad-inkonsistent sind, denn es stehen nur zwei Werte zur Verfügung, nämlich 1 und 2. Damit wäre eine Belegung der drei Variablen, so daß ihre Werte paarweise verschieden sind, unmöglich. Ein Kantenkonsistenz-Algorithmus ist aber nicht in der Lage diese Inkonsistenz festzustellen. Das liegt daran, daß bei diesem schwächeren Algorithmus nicht mehr als zwei Variablen gleichzeitig betrachtet werden können.

Das Meta-Level arbeitet auf konsistenten Paaren, die im Objekt-Level mittels eines Kantenkonsistenz-Algorithmus ermittelt wurden. Die Meta-Domäne ist $D_m = Z \times Z$. Die Variablen des Meta-Levels werden dann aus Paaren der Variablen des Objekt-Levels gebildet, die durch Objekt-Constraints gebunden sind.

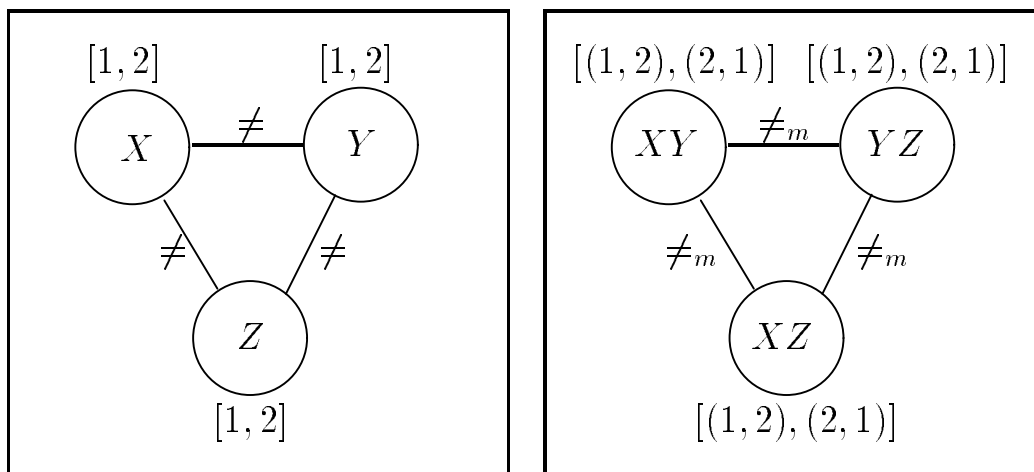


Abbildung 4: Constraint-Graph mit dem dazugehörigen Meta-Constraint-Graphen

In diesem Beispiel gibt es drei Meta-Variablen XY, XZ und YZ , die im Wertebereich $[(1, 2), (2, 1)]$ liegen. Dieser Wertebereich stellt die kantenkonsistenten Paare dar, die vom Objekt-System erlaubt werden (die Paare $(1, 1)$ und $(2, 2)$ sind kanteninkonsistent). Die drei Meta-Variablen werden durch den Constraint \neq_m gebunden, das wie folgt mit einem Kantenkonsistenz-Algorithmus propagiert wird. Schauen wir uns den Constraint $XY \neq_m YZ$ an. Für jeden Wert $xy \in D_{XY}$ prüft der Propagierungs-Algorithmus, ob es ein Wert $yz \in D_{YZ}$ existiert, der mit xy kompatibel ist. Die gesuchten Werte sind Paare (x, y) und (x, z) . Die Kompatibilität wird wie folgt definiert: der Wert der gemeinsamen Objekt-Variablen y

muß in beiden Paaren gleich sein, während die Werte von x und z den Constraint \neq erfüllen (Komposition von $x \neq y$ und $y \neq z$, und Schnitt mit $x \neq z$). Die Propagierung führt dann, wie erwartet, zu einem leeren Wertebereich, d.h. zu einem Fehlschlag.

Es wurde also ein CLP-System mit einer Kantenkonsistenz-Technik um ein Meta-Level erweitert, so daß zum Schluß eine Pfadkonsistenz, d.h. eine k -Konsistenz mit $k = 2 + 2 - 1 = 3$, erreicht wurde.

Hätten wir beispielsweise zwei Levels, die jeweils einen Pfadkonsistenz-Algorithmus implementierten, so würden wir insgesamt eine 5-Konsistenz erreichen. Das zweite Level schließt auf die konsistenten Tripel, die mittels eines Pfadkonsistenz-Algorithmus im ersten Level ermittelt werden. Das zweite Level implementiert eine Pfadkonsistenz auf diesen Tripeln und berechnet damit alle konsistenten Pfade, die drei Tripel verbinden (drei Pfade der Länge drei, d.h. fünf Variablen, denn die verbundenen Tripel haben 2 gemeinsame Variablen).

4.3 Die operationale Semantik

Die operationale Semantik des Meta-Schemas wird auf einer Seite durch die “Top-Down“-Ausführung der einzelnen Komponenten und auf der anderen Seite durch die Regeln, die die Komponenten verbinden, bestimmt.

Zuerst kommen wir zu den aktiven und passiven Constraint-Speichern der verschiedenen Levels. Die aktiven Constraint-Speicher enthalten auf jedem Level unäre Constraints, die die Werte der Variablen zu einer endlichen Domäne von k -Tupeln binden. Damit kann jeder Variable ein k -Tupel zugewiesen werden, wenn der Constraint gelöst wird. Die passiven Constraint-Speicher enthalten binäre, oder im allgemeineren Fall n -stellige Constraints. Jedes Modul i implementiert einen c_i -Konsistenz-Algorithmus, der durch das Prädikat *infer* bestimmt wird. Besonders interessant dabei sind die Regeln, die die Wechselwirkung zwischen zwei Komponenten beschreiben. Die erste Regel (Pfeil Nummer (1) in Abbildung 2) verbindet den aktiven Meta-Constraint-Speicher C_m mit dem passiven Objekt-Level-Constraint-Speicher S_{obj} :

$$\frac{C_m}{S_{obj} \mapsto_{(1)} S_{obj} \cup S_{obj-m}},$$

wobei der Constraint-Speicher S_{obj-m} wie folgt definiert wird:

$$S_{obj-m} = \bigcup_{\forall (X_m \in D_m) \in C_m} \left(\bigcup_{i=1}^k (x_i \in D_i) \right).$$

Dabei ist X_m eine Meta-Variable in der Meta-Domäne $D_m = D_1 \times \dots \times D_k$, die sich auf den k -stelligen Constraint zwischen x_1, \dots, x_k bezieht. Die Domäne $D_i = Proj_i(D_m)$ ($Proj_i : D_1 \times \dots \times D_k \rightarrow D_i$) projiziert k -Tupel auf die i -te Komponente.

Diese Bindungsregel fügt dem passiven Objekt-Level-Constraint-Speicher unäre Objekt-Level-Constraints hinzu, die Werte enthalten, die aus der Meta-Domäne folgern. Die Wertebereiche der Meta-Variablen werden also auf die Wertebereiche der betreffenden Objekt-Variablen reflektiert.

Nehmen wir an aus einer Meta-Propagierung ergebe sich, daß die Meta-Variable XY auf den Wertebereich $D_{XY} = \{(2, 8), (3, 4), (4, 6)\}$ eingeschränkt wird, während der Wertebereich von X die Werte $\{2, 3, 4, 7\}$ enthält. Der Wert 7 muß von D_X entfernt werden, denn der Durchschnitt von $\{2, 3, 4, 7\}$ mit der neuen Domäne von X ist gleich $Proj_1(D_{XY}) = \{2, 3, 4\}$.

Die zweite Bindungsregel (Pfeil Nummer (2) in Abbildung 2) betrifft die Propagierung aktiver Meta-Constraints, wenn passive Objekt-Level-Constraints verändert werden. Die neuen Constraints, die diese Regel erzeugt, werden dem passiven Meta-Constraint-Speicher hinzugefügt, auch wenn sie bereits aktiv sind. Sie werden erst von dem Propagierungsmechanismus des Meta-Levels, d.h. von der *Inference*-Regel des Meta-Levels, in aktive Meta-Level-Constraints umgewandelt. Diese Bindungsregel bezieht sich auf:

$$\frac{S_{obj}}{S_m \mapsto_{(2)} S_m \cup S_{m-obj}},$$

wobei S_{m-obj} unäre Meta-Constraints enthält. Das sind die k -Tupel, die im darunterliegenden Level konsistent sind. S_{m-obj} wird wie folgt definiert:

$$S_{m-obj} = \bigcup_{\substack{\forall (x_1 \in D_1, \dots, x_k \in D_k) \in C_{obj} \\ c(x_1, \dots, x_k) \in S_{obj} \text{ } k\text{-konsistent}}} X_m \in D_1 \times \dots \times D_k,$$

wobei $c(x_1, \dots, x_k)$ der k -stellige Constraint zwischen den Variablen x_1, \dots, x_k ist. Dieser k -stellige Constraint bestimmt eine Teilmenge des kartesischen Produkts der Wertebereiche der k Variablen.

Es seien drei Objekt-Level-Variablen X, Y und Z mit den dazugehörigen Wertebereichen $D_X = [3, 4]$, $D_Y = [3, 5]$ und $D_Z = [2, 3]$, und die Constraints $X < Y$, $Y \neq Z$ und $X > Z$. S_{m-obj} enthält die Constraints $XY \in D_{XY}$, $YZ \in D_{YZ}$ und $XZ \in D_{XZ}$, mit

$$\begin{aligned} D_{XY} &= \{(3, 4), (3, 5), (4, 5)\}, \\ D_{YZ} &= \{(3, 2), (4, 2), (4, 3), (5, 2), (5, 3)\}, \\ D_{XZ} &= \{(3, 2), (4, 2), (4, 3)\}. \end{aligned}$$

Diese Wertebereiche entsprechen den kantenkonsistenten Paaren des darunterliegenden Systems.

Die dritte Bindungs-Regel (Pfeil Nummer (3) in Abbildung 2) verbindet den aktiven Objekt-Constraint-Speicher C_{obj} mit dem passiven Meta-Level-Constraint-Speicher S_m :

$$\frac{C_{obj}}{S_m \mapsto_{(3)} S_m \cup S'_{m-obj}}$$

wobei S'_{m-obj} der Constraint-Speicher ist, der aus den aktiven Objekt-Constraints folgt. Mit dieser Regel werden die Wertebereiche der Meta-Variablen mit den Werten der Variablen des darunterliegenden Levels konsistent gehalten. Wenn eine Constraint-Propagierung dazu führt, daß Werte aus dem Wertebereich einer Objekt-Variablen gestrichen werden, müssen die k -Tupel des aktiven Meta-Constraint-Speichers, die diesen Wert enthalten, gelöscht werden. S'_{m-obj} wird wie folgt definiert:

$$S'_{m-obj} = \bigcup_{\substack{\forall (c(x_1, \dots, x_i, \dots, x_k), \\ c'(x'_1, \dots, x'_j, \dots, x'_k)) \in S_{obj} \\ x_l = x'_p \quad l, p = 1..k, \quad l \neq i, \quad p \neq j}} c_m(X, X'),$$

wobei $c(x_1, \dots, x_i, \dots, x_k)$ und $c'(x'_1, \dots, x'_j, \dots, x'_k)$ die k -stelligen Constraints zwischen den Variablen x_1, \dots, x_k bzw. x'_1, \dots, x'_k sind. Diese Constraints haben $k - 1$ gemeinsame Variablen. Ein k -stelliger Constraint definiert eine Teilmenge des kartesischen Produkts der Wertebereiche der k Variablen. X und X' sind Meta-Variablen, wobei $X \in D_1 \times \dots \times D_i \times \dots \times D_k$, $X' \in D'_1 \times \dots \times D'_j \times \dots \times D'_k$, mit $D_l = D'_p$, $l, p = 1..k$, $l \neq i$ und $p \neq j$. c_m ist der entsprechende Meta-Constraint zu dem Objekt-Level-Constraint, der x_i und x_j bindet.

Man betrachte weiterhin das obige Beispiel mit den Variablen X, Y, Z und ihren Constraints. S'_{m-obj} enthält den Constraint $XY >_m YZ$, der die Meta-Variablen XY und YZ bindet, weil ihre Domäne Paare enthalten, die sich eine Variable teilen. Der Constraint $>_m$ wird durch die Komposition von $X < Y$ und $Y \neq Z$ ($= \top$) und den Schnitt mit $X > Z$ abgeleitet. Andere Meta-Constraints sind $XY >_m XZ$ und $YZ >_m XZ$. Die Propagierung dieser Constraints (durch das Prädikat $infer_m = \text{Kantenkonsistenz}$) entfernt die Kanteninkonsistenten Paare $(3, 2)$ und $(4, 3)$ von D_{YZ} , denn es gibt kein Paar in D_{XY} , dessen zweites Element gleich 3 ist, bzw. es gibt kein Paar in D_{XY} , dessen zweites Element gleich 4 ist, und das erste Element kleiner als 3 ist.

5 Qualitative Reasoning

Bei dieser Spezialisierung betrachten wir eine Architektur mit zwei Levels, wobei das erste Level ein CLP(FD)-Löser ist, und das zweite Level ein Meta-Constraint-Löser ist, der auf die Constraints des ersten Levels schließt.

5.1 Die Domänen

Der Objekt-Level-Constraint-Löser ist ein CLP(FD)-Komponent, und seine Domäne ist mit Ausnahme des Constraints $<$, das hier durch \leq ersetzt wurde, genauso wie beim ersten Level der ersten Spezialisierung: $D = Z$ und

$$\Sigma = \{\{\in [m, n]\}_{m \leq n}, +, =, \neq, \leq\}.$$

Die Domäne des Meta-Levels muß die Constraints des Objekt-Levels enthalten. Der qualitative Constraint-Löser basiert also auf einem Meta-Constraint-Löser, der als Domäne Relationen hat. Jedes Mal, wenn ein neuer qualitativer Constraint zwischen die Objekt-Variablen X und Y eingefügt wird, wird eine Meta-Variable R_{XY} erzeugt.

Ein qualitativer Constraint ist eine Disjunktion von Constraints. Wenn zwischen den Variablen X und Y die Relation $(Xr_1Y) \vee \dots \vee (Xr_nY)$ definiert wird, die wir als $X\{r_1, \dots, r_n\}Y$ schreiben, wird eine Meta-Variable R_{XY} produziert, deren Domäne $D_{R_{XY}} = \mathcal{P}(\{r_1, \dots, r_n\})$ ist. Es sei der qualitative Constraint $X\{<, =, >\}Y$, $D_{R_{XY}}$ ist dann gleich $\{\{\}, \{<\}, \{=\}, \{>\}, \{=\}, \{<, =\}, \{<, >\}, \{>, =\}, \{<, =, >\}\}$.

Sei $D_m = \mathcal{L}$ und $V \subseteq \mathcal{P}(D_m)$, die Signatur des Meta-Level-Systems lautet:

$$\Sigma_m = \{\{\in V\}, +_m, =_m, <_m\},$$

wobei die Symbole in Σ_m wie folgt von \mathcal{D}_m interpretiert werden:
 $\in V$ ist ein unärer Constraint, wobei V eine Teilmenge von $\mathcal{P}(D_m)$ ist;

$+_m$ ist der Kompositionsoperator. Die Komposition von zwei Meta-Variablen R_{XZ} und R_{ZY} erlaubt nur die Werte R_{XY} , für die $R_{XZ} \in D_{R_{XZ}}$ und $R_{ZY} \in D_{R_{ZY}}$ existieren, so daß $R_{XY} = R_{XZ} \otimes R_{ZY}$, wobei das Symbol \otimes die Komposition primitiver Constraint-Symbole von D darstellt (Tabelle 1);

\otimes	$<$	\leq	$=$	\neq	\top
$<$	$<$	$<$	$<$	\top	\top
\leq	$<$	\leq	\leq	\top	\top
$=$	$<$	\leq	$=$	\neq	\top
\neq	\top	\top	\neq	\top	\top
\top	\top	\top	\top	\top	\top

Tabelle 1: Komposition von Constraints

Das Symbol $=_m$ stellt den Durchschnitt von Constraints dar;

Das Symbol $<_m$ wird als “Tighter”-Relation zwischen Constraints interpretiert. Eine Constraint-Menge C' ist “tighter” als eine Constraint-Menge C'' , wenn alle Tupel, die von C' erlaubt werden, auch von C'' erlaubt werden. In diesem Fall kann diese Relation durch \subseteq implementiert werden.

5.2 Beispiel

Wir betrachten ein Beispiel für das “qualitative reasoning”. Es seien drei Zeitpunkte T_1 = “Helen trifft John”, T_2 = “John betritt die Bank” und T_3 = “John parkt sein Auto” mit den Domänen $D_{T_1} = [10, 25]$, $D_{T_2} = [20, 25]$ bzw. $D_{T_3} = [10, 15]$ und den Constraints $T_1\{>, =\}T_2$ und $T_2\{>, =\}T_3$, d.h. T_1 erfolgt später als oder gleichzeitig mit T_2 , und T_2 erfolgt später als oder gleichzeitig mit T_3 . Das Meta-Level hat als Domäne die Potenzmenge der Constraint-Symbole des Objekt-Levels: $D_m = \mathcal{P}(\{>, =\})$.

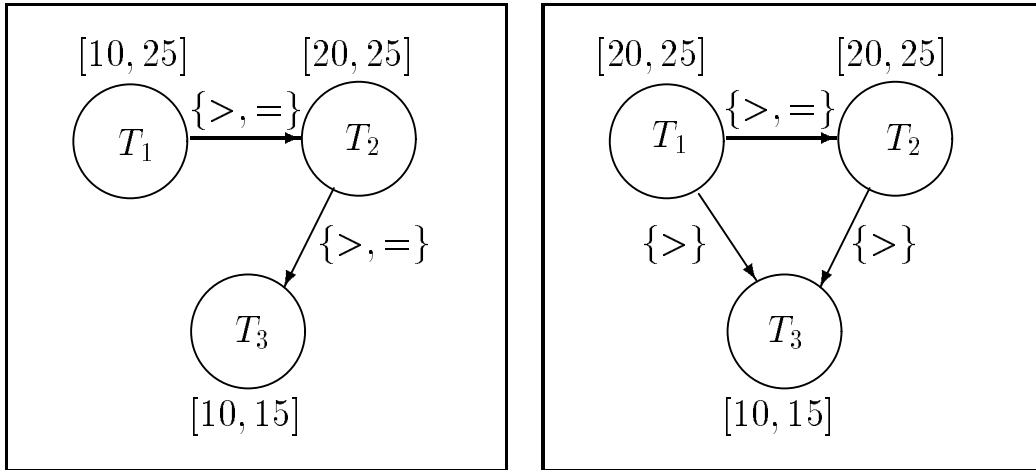


Abbildung 5: Ein- und Ausgabe des Beispiels als Constraint-Graphen.

Im Meta-Level werden drei Meta-Variablen für die Objekt-Level-Relationen erzeugt: $R_{T_1T_2}$, $R_{T_2T_3}$ und $R_{T_1T_3}$, die in den Wertbereichen $D_{R_{T_1T_2}} = D_{R_{T_2T_3}} = \{\{\}, \{>\}, \{=\}, \{>, =\}\}$ bzw. $D_{R_{T_1T_3}} = \{\{\top\}, \{\}\}$ liegen. Danach wird $D_{R_{T_1T_3}}$ durch Komposition von $D_{R_{T_1T_2}}$ und $D_{R_{T_2T_3}}$ gebildet. Es ergibt sich $D_{R_{T_1T_3}} = \{\{\}, \{>\}, \{=\}, \{>, =\}\}$. Der qualitative Constraint $T_1\{>, =\}T_3$ wird auf das Objekt-Level propagiert.

Da $T_1\{>, =\}T_2$ im Objekt-Level gelten muß, wird die Domäne von T_1 durch das Prädikat *infer* im Objekt-System auf $[20, 25]$ verkleinert. Weil sowohl in T_1 als auch in T_2 keinen Wert mehr gibt, der auch in T_3 enthalten ist ($T_1\{=\}T_3$ und $T_2\{=\}T_3$ sind unmöglich), werden die Constraints $R_{T_1T_3}, R_{T_2T_3} \in \{\{\}, \{>\}\}$ auf das Meta-Level propagiert, und damit werden die Domäne $D_{R_{T_1T_3}}$ und $D_{R_{T_2T_3}}$ verkleinert.

5.3 Die operationale Semantik

In diesem Abschnitt wird die operationale Semantik beider Systeme und die Wechselwirkung zwischen ihnen festgelegt.

Die operationale Semantik wird durch die Prädikate *infer* und *consistent* definiert. Sowohl im Objekt-Level als auch im Meta-Level implementiert *infer* eine Kantenkonsistenz-Propagierung, und das Prädikat *consistent* wird auf den aktiven Constraint-Speicher angewendet und gilt, wenn keine Variablendomäne leer ist.

Betrachten wir jetzt die Regeln, die die Wechselwirkung zwischen den zwei CLP-Lösern beschreiben. Sie sind in Abbildung 2 gezeichnet. Pfeil (1) bezieht sich auf die Regel:

$$\frac{C_m}{S_{obj} \mapsto_{(1)} S_{obj} \cup S_{obj-m}}$$

mit

$$S_{obj-m} = \bigcup_{\forall (R_{XY} \in D_{R_{XY}}) \in C_m} X \text{ lub}(R_{XY}) Y,$$

wobei $R_{XY} \in D_{R_{XY}}$ ein unärer Constraint ist und $D_{R_{XY}}$ eine Teilmenge von $\mathcal{P}(D_m)$.

Nehmen wir an, daß nach der Ausführung von $infer(C_m, S_m)$ im Meta-Level die Menge der aktiven Constraints C'_m den Constraint $R_{XY} \in [\{\}, \{<\}, \{=\}, \{<, =\}]$ enthält. Das *lub* (kleinste obere Schranke) der Variablen-Domäne ist $\{<, =\}$, und dieses *lub* stellt die Constraints dar, die zwischen den Variablen X und Y im Objekt-System gelten.

Wenn die Domäne der Meta-Variable R_{XY} nur die leere Menge enthält, d.h. $R_{XY} \in [\{\}]$, dann bedeutet das, daß ein Fehler im Objekt-System aufgetreten ist. Dieser Fehler kann entweder in das Meta-System propagiert werden, oder vom Meta-System behandelt werden um z.B. einige Constraints zu “contract”. Ein Fehler im Meta-System entspricht einer leeren Domäne einer Meta-Variable, d.h. $R_{XY} \in [\]$.

Der Pfeil (2) bezieht sich auf die Regel:

$$\frac{S_{obj}}{S_m \mapsto_{(2)} S_m \cup S_{m-obj}}$$

mit

$$S_{m-obj} = \bigcup_{\forall X, Y \in D, X \{r_1, r_2, \dots, r_n\} Y \in S_{obj}, r_1, r_2, \dots, r_n \in \mathcal{L}} R_{XY} \in \mathcal{P}(\{r_1, r_2, \dots, r_n\}).$$

Es wird also für je zwei Objekt-Variablen eine Meta-Variable erzeugt. Wenn die Menge der passiven Objekt-Constraints S_{obj} beispielsweise den Constraint $X \leq Y$

enthält, der als $X\{<,=\}Y$ geschrieben werden kann, erzeugt die Transition eine Meta-Variable R_{XY} mit dem Wertebereich $\mathcal{P}(\{<,=\}) = [\{\}, \{<\}, \{=\}, \{<,=\}]$. Die dritte Bindungsregel (Pfeil (3) in Abbildung 2) fügt dem passiven Meta-Constraint-Speicher die Constraints hinzu, die aus den Werten der Variablen des Objekt-Levels folgern. Sie hat die Form:

$$\frac{C_{obj}}{S_m \mapsto_{(3)} S_m \cup S'_{m-obj}},$$

wobei S'_{m-obj} , der Constraint-Speicher ist, der aus den aktiven Objekt-Constraints folgert. Das ist eine spezielle Regel, die von der Art der behandelten Constraints abhängt. Betrachten wir beispielsweise X und Y mit $D_X = [1..10]$ und $D_Y = [14..24]$ und mit dem Constraint $X\{<,=\}Y$. Der Constraint $=$ kann bei den Wertebereichen von X und Y nicht erfüllt werden. Es muß dann ein Constraint propagiert werden, der $=$ von den möglichen Werten der Meta-Variable R_{XY} entfernt. In diesem Fall wird S'_{m-obj} wie folgt definiert:

$$S'_{m-obj} = \bigcup_{\substack{\forall X, Y \in D, X\{r_{i_1}, \dots, r_{i_k}\}Y \in S_{obj}, r_{i_1}, \dots, r_{i_k} \in \mathcal{L} \\ \exists (X \in D_X, Y \in D_Y) \in C_{obj} \text{ mit } Xr_{i_j}Y}} R_{XY} \in \mathcal{P}(\{r_{i_1}, \dots, r_{i_k}\})$$

6 Zusammenfassung

In diesem Artikel wurde ein allgemeines Schema zur Kombination von Constraint-Lösern in einer Meta-Architektur präsentiert.

Die erste Spezialisierung ermöglicht einen beliebigen Konsistenzgrad zu erreichen ohne die Konsistenz-Algorithmen der grundlegenden Constraint-Löser zu verändern. Einer der größten Vorteile dieser Spezialisierung besteht darin, daß jeder Konsistenzgrad erreicht werden kann, ohne den grundlegenden Propagierungs-Algorithmus zu verändern. Man hat also eine hohe Flexibilität. Ein neuerer Artikel derselben Autorinnen [3] betrachtet einen anpaßungsfähigen Constraint-Löser, der diese Flexibilität des Meta-Schemas ausnutzt und mit Hilfe von Heuristiken erst zur Laufzeit den geeigneten Konsistenz-Algorithmus einstellt.

Die zweite Spezialisierung, die hier vorgestellt wurde, ermöglicht das "qualitative reasoning", indem das System um ein Meta-Level erweitert wird, das auf die Constraints des darunterliegenden Levels schließt.

Beide Spezialisierungen wurden mit Hilfe der Bibliothek von ECLⁱPS^e implementiert, Daten über die Effizienz wurden leider nicht veröffentlicht, so daß es nicht möglich ist die Laufzeit des hier vorgestellten Algorithmus für die k -Konsistenz und des "flachen" Algorithmus von Freuder zu vergleichen.