Adaptiv-inkrementelle Unifikation*

Armin Wolf

GMD — Forschungszentrum Informationstechnik GmbH
Forschungsinstitut für Rechnerarchitektur und Softwaretechnik
Rudower Chaussee 5, D-12489 Berlin

Tel.:	+49 30 63 92-18 64	Fax:	+49 30 63 92-18 05
E-mail:	`armin@first.gmd.de`	URL:	`http://www.first.gmd.de`

1 Einleitung

Der Einsatz constraintlogischer Programmierung zur Lösung praktischer dynamischer Constraintprobleme erfordert Lösungsverfahren, die sich im besonderen durch zwei Eigenschaften auszeichnen sollten:

Effizienz: Der Aufwand für die Lösungsanpassung soll proportional zur Größe der Änderung und im allgemeinen geringer sein als der Aufwand für eine vollständige Neuberechnung.

Stabilität: Die angepaßte Lösung soll möglichst ähnlich zur vorherigen Lösung sein.

Für den Einsatz constraintbasierter Problemlösungen in interaktiven oder reaktiven Systemen in dynamischen Umgebungen, wie zum Beispiel bei der Planung, Konfiguration und im Graphik/Design-Bereich, sind beide Punkte gleichermaßen wichtig: Reaktions- und Antwortzeiten nach Änderungen sind zu minimieren, damit das System *on-line* einsetzbar ist. Änderungen von Plänen, Entwürfen, Layouts und Graphiken sind möglichst zu vermeiden, um Umorganisationen, Umbestellungen, Umorientierungen zu reduzieren (siehe beispielsweise [5, 17]).

Um beiden Anforderungen gerecht zu werden, kommen im allgemeinen *inkrementelle* Verfahren [15] zur Bearbeitung dynamischer Constraintprobleme zum Einsatz. Diese versuchen, eine zuvor berechnete Lösung so anzupassen, daß sie

*Das diesem Bericht zugrundeliegende Forschungsvorhaben VERMEIL wird mit Mitteln des Bundesministeriums für Bildung, Wissenschaft, Forschung und Technologie (Förderkennzeichen 01 IN 507 B) gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt beim Autor.

der veränderten Problemstellung genügt. Idealerweise soll das inkrementelle Verfahren zur Lösung dynamischer Constraintprobleme auch *adaptiv* sein:

- Die angepaßten Lösungen sollen sich durch größtmögliche Stabilität auszeichnen, das heißt, sie sollen sich so wenig wie möglich von den ursprünglichen Lösungen unterscheiden.
- Die erzielbaren Effizienzsteigerungen des inkrementellen Verfahrens gegenüber einer vollständigen Neuberechnung sollen bei der Lösungsanpassung um so größer sein, je geringer die Veränderungen an der Problemstellung sind.

Erfolgt die Lösung der Constraintprobleme mit Hilfe von *Constraint Handling Rules* (CHRs) [1, 2, 10] kann eine Lösungsanpassung bei dynamischen Änderungen nur dann adaptiv erfolgen, wenn dies für die zugrundegelegten *built-in* Constraints möglich ist. Da syntaktische Gleichungen immer zu den *built-in* Constraints gehören (vergleiche [1]), wurde untersucht, ob eine möglichst stabile und effiziente Anpassung der Lösungen von syntaktischen Gleichungen erfolgen kann.

In dieser Arbeit wird ein adaptiv-inkrementelles Unifikationsverfahren vorgestellt, das für ein System syntaktischer Gleichungen entscheidet, ob es in der Menge der rationalen Bäume (*rational trees* [7]) lösbar ist. Bei Lösbarkeit wird das System in eine Lösungsform (*rational solved form* [12]) überführt. Wird das System um zusätzliche Gleichungen erweitert, wird bei Lösbarkeit auch die Lösungsform erweitert. Werden Gleichungen aus dem System entfernt, werden nur die davon betroffenen Gleichungen aus der Lösungsform entfernt und gegebenenfalls durch andere ersetzt.

2 Grundlegende Definitionen

Neben *syntaktischen Gleichungen* und deren Lösungen, *rationale Bäume*, definieren wir *Markierungen*, *markierte Gleichungen* und *markierte Terme* sowie zugehörige Operationen und Funktionen.

Syntaktische Gleichungen sind geordnete Paare der Gestalt $s \doteq t$, wobei s und t prädikatenlogische Terme sind.

Die Knoten der Bäume, die wir hier betrachten wollen, sind Funktionssymbole, wobei ein n-stelliges Funktionssymbol n geordnete Nachfolger hat. Somit bilden die Konstantensymbole die Blätter der Bäume. Neben endlichen Bäumen, die auch als Grundterme bezeichnet werden, gibt es unendliche Bäume. *Rationale Bäume* sind endliche/unendliche Bäume, die endlich viele Teilbäume enthalten. Zum Beispiel ist $f(f(f(\ldots)))$ ein rationaler Baum, da er nur sich selbst enthält.

Eine *Variablenbelegung* ist eine Menge von geordneten Paaren

$$\chi = \{x_1 := r_1, \ldots, x_n := r_n\}$$

wobei $x_1, \ldots, x_n$ paarweise verschiedene Variablen und $r_1, \ldots, r_n$ rationale Bäume sind. Falls ein Term t nur Variablen aus $x_1, \ldots, x_n$ enthält, ist

$$t/\chi = \begin{cases} r_i & \text{falls } t = x_i \\ f(t_1/\chi, \ldots, t_n/\chi) & \text{falls } t = f(t_1, \ldots, t_n) \end{cases}$$

wohldefiniert und ein rationaler Baum.

Eine syntaktische Gleichung $s \doteq t$ ist *lösbar*, wenn es eine Variablenbelegung χ gibt, so daß $s/\chi = t/\chi$ gilt (siehe auch [7]).

Markierungen sind Teilmengen einer nichtleeren, abzählbaren Grundmenge (zum Beispiel die natürlichen Zahlen). Für zwei *Listen von Markierungen* $M := [M_1, \ldots, M_k]$ und $N := [N_1, \ldots, N_l]$ definieren wir

$$M \otimes N \quad := \quad M_1 \cup \ldots \cup M_k \cup N_1 \cup \ldots \cup N_l.$$

Markierte Gleichungen sind Konstrukte der Form g^P oder $\langle s \doteq t\rangle^P$, wobei g und $s \doteq t$ syntaktische Gleichungen sind und P eine Markierung ist. Eine markierte Gleichung g^P heißt *lösbar*, wenn g lösbar ist.

Markierte Terme sind Konstrukte der Form t_V^L, wobei t ein prädikatenlogischer Term ist, V eine Liste von Variablen[1] ist und L eine Liste von Markierungen ist; beide Listen haben die gleiche Länge. Im folgenden schreiben wir $f_V^L(t_1, \ldots, t_k)$ statt $f(t_1, \ldots, t_k)_V^L$. Markierte Terme der Form $t_{[x]}^{[P]}$ nennen wir *Variablenbindungen*, die wir auch als markierte Gleichungen $\langle x \doteq t\rangle^P$ betrachten.

Eine Menge von Variablenbindungen heißt *zirkular*, wenn sie die Form

$$\{\langle x_1 \doteq x_2\rangle^{P_1}, \langle x_2 \doteq x_3\rangle^{P_2}, \ldots, \langle x_{n-1} \doteq x_n\rangle^{P_{n-1}}, \langle x_n \doteq x_1\rangle^{P_n}\}$$

hat. Eine Menge von Variablenbindungen heißt *Lösungsform*, wenn sie für jede Variable x höchstens eine markierte Gleichung $\langle x \doteq t\rangle^P$ enthält und keine zirkulare Teilmenge enthält (siehe auch [7, 12]).

Eine Lösungsform T ist *zyklenfrei* wenn der gerichtete Graph mit den prädikatenlogischen Termen als Knotenmenge und der Kantenmenge

$$\{x \to t \mid t_{[x]}^{[P]} \in T\} \cup \{t \to x \mid t \notin \mathcal{V}, x \in \mathsf{var}(t)\}$$

keine Zyklen enthält.[2]

Markierte Terme repräsentieren im folgenden Variablenbindungen und Substitutionen. Dazu seien zwei Funktionen definiert, die einen beliebigen prädikatenlogischen Term r und eine Menge von Variablenbindungen T auf einen (markierten)

[1] Variablen bezeichnen wir mit Kleinbuchstaben $x, y, z, \ldots$. Die Menge aller Variablen bezeichnen wir mit $\mathcal{V}$.

[2] $\mathsf{var}(t)$ ist die Menge aller Variablen im Term t.

Term abbilden:[3]

$$\mathsf{deref}(r,T) \;:=\; \mathsf{deref'}(r^{[]}_{[]},T)$$

$$\mathsf{deref'}(r^{L}_{V},T) \;:=\; \begin{cases} \mathsf{deref'}(y^{L\cdot[P]}_{V\cdot[r]},T) & \text{falls } r,y\in\mathcal{V} \text{ und } y^{[P]}_{[r]}\in T,\\ f^{L\cdot[P]}_{V\cdot[r]}(t_1,\ldots,t_k) & \text{falls } r\in\mathcal{V} \text{ und } f^{[P]}_{[r]}(t_1,\ldots,t_k)\in T,\\ r^{L}_{V} & \text{sonst.} \end{cases}$$

$$\mathsf{subst}(r,T)$$
$$:= \begin{cases} \mathsf{subst}(y,T) & \text{falls } r,y\in\mathcal{V} \text{ und } y^{[P]}_{[r]}\in T,\\ f(\mathsf{subst}(t_1,T),\ldots,\mathsf{subst}(t_k,T)) & \text{falls } r\in\mathcal{V} \text{ und } f^{[P]}_{[r]}(t_1,\ldots,t_k)\in T,\\ r & \text{sonst.} \end{cases}$$

Die beiden Funktionen sind nur für bestimmte Mengen von Variablenbindungen wohldefiniert. Die Funktion **deref** ist nur für Lösungsformen wohldefiniert: Enthält T mehr als eine Variablenbindung für eine Variable x, zum Beispiel $f(a)^{[P]}_{[x]}$ und $g(b)^{[Q]}_{[x]}$, oder eine zirkulare Teilmenge, wie zum Beispiel $\{y^{[P]}_{[x]}, x^{[Q]}_{[y]}\}$, dann ist $\mathsf{deref}(x,T)$ nicht definiert. Die Funktion **subst** ist nur für zyklenfreie Lösungsformen wohldefiniert: Enthält eine Lösungform T einen Zyklus, zum Beispiel $f(y)^{[P]}_{[x]}$ und $g(x)^{[Q]}_{[y]}$, dann sind weder $\mathsf{subst}(x,T)$ noch $\mathsf{subst}(y,T)$ definiert.

Die Funktion **deref** dient wie die gleichnamige Funktion in [4] der *Dereferenzierung*. Sie folgt einer möglichen Bindungskette, bis sie entweder zu einer Variablen gelangt, für die keine Bindung besteht (das heißt, *ungebunden* ist), oder zu einem Funktionsterm gelangt. Beispielsweise ist

$$\mathsf{deref}\left(x, \left\{y^{[L]}_{[x]}, f(z)^{[P]}_{[y]}, a^{[Q]}_{[z]}\right\}\right) = f(z)^{[L,P]}_{[x,y]}.$$

Analog zu **deref** folgt die Funktion **subst** einer möglichen Bindungskette, bis sie entweder zu einer Variablen gelangt, die ungebunden ist, oder zu einen Funktionsterm gelangt. Im zweiten Fall wird rekursiv den Bindungsketten der Argumente gefolgt. Beispielsweise ist

$$\mathsf{subst}\left(x, \left\{y^{[L]}_{[x]}, f(z)^{[P]}_{[y]}, a^{[Q]}_{[z]}\right\}\right) = f(a).$$

Ist eine Lösungsform T zyklenfrei, kann offensichtlich mit Hilfe von **subst** eine idempotente Substitution

$$\sigma_T \;:=\; \{x/t \mid \mathsf{subst}(x,T)=t, x\neq t\} \tag{1}$$

konstruiert werden.

Dynamische Constraintprobleme sind charakterisiert durch das Hinzufügen und Löschen von Constraints. Wir gehen davon aus, daß markierte Gleichungen und

[3] Listen repräsentieren wir in PROLOG-Notation. Die Konkatenation zweier Listen L und L' bezeichnen wir jedoch mit $L\cdot L'$.

Variablenbindungen in Abhängigkeit von Markierungen gelöscht werden und definieren eine entsprechende Operation del: Es sei D eine Markierung, G ist eine Liste von markierten Gleichungen und T eine Menge von Variablenbindungen gegeben. Wir definieren:

$$\begin{aligned}
\mathsf{del}(D,[\,]) &:= [\,],\\
\mathsf{del}(D,[\langle s \doteq t\rangle^P|G]) &:= \begin{cases} \mathsf{del}(D,G) & \text{falls } P\cap D\neq\emptyset,\\ [\langle s \doteq t\rangle^P|\mathsf{del}(D,G)] & \text{sonst.}\end{cases}\\
\mathsf{del}(D,\emptyset) &:= \emptyset,\\
\mathsf{del}(D,\{t_{[x]}^{[P]}\}\cup T) &:= \begin{cases} \mathsf{del}(D,T) & \text{falls } P\cap D\neq\emptyset,\\ \{t_{[x]}^{[P]}\}\cup\mathsf{del}(D,G) & \text{sonst.}\end{cases}
\end{aligned}$$

3 Adaptiv-inkrementelle Unifikation

Es wird ein Unifikationsverfahren vorgestellt, das auf der *UNION/FIND*-Methode [3] beruht und Lösungformen aufbaut. Das Verfahren terminiert, da durch Verwendung eines *lokalen Cache* [14] garantiert ist, daß die zu unifizierenden Gleichungen nur endlich oft betrachtet werden.

3.1 Problemstellung

Es sei eine endliche Folge (repräsentiert durch eine Liste) markierter Gleichungen $G = [\langle s_1 \doteq t_1\rangle^{P_1},\ldots,\langle s_n \doteq t_n\rangle^{P_n}]$ gegeben. Es ist zu entscheiden, ob die markierten Gleichungen in G (beziehungsweise die Gleichungen $s_1 \doteq t_1,\ldots,s_n \doteq t_n$) *unifizierbar* sind. Das heißt, es ist zu entscheiden, ob es eine Variablenbelegung χ gibt, so daß $s_1/\chi = t_1/\chi,\ldots s_n/\chi = t_n/\chi$ gilt — χ ist dann eine *Lösung* von G. Bei Unifizierbarkeit ist eine Lösungform T zu bestimmen, die *äquivalent* zu G ist, das heißt, die gleiche Lösungsmenge hat.

Zur Entscheidung der Unifizierbarkeit und zur Berechnung eines allgemeinsten Unifikators sind bereits viele *Unifikationsverfahren* bekannt [4, 6, 11, 13], die zwar effizient sind, jedoch wenig geeignet sind, wenn die Gleichungsmengen ständig verändert werden, da sie nicht inkrementell arbeiten. Das in [4] realisierte Unifikationsverfahren arbeitet inkrementell und effizient beim Hinzufügen von syntaktischen Gleichungen und bei dem durch das *Backtracking* bedingte chronologische Löschen, ist jedoch ungeeignet bei beliebigen Änderungen. — Im folgenden wird ein neues inkrementelles Verfahren vorgestellt, das uneingeschränkt bei beliebigen Änderungen der Gleichungsmengen eingesetzt werden kann und sowohl beim Hinzufügen als auch beim Löschen von markierten Gleichungen effizient arbeitet.

3.2 Definition des Verfahrens

Es seien zwei Varianten eines Unifikationsverfahrens definiert:

$$\begin{aligned}
\mathsf{unify}(G) &:= \mathsf{unify'}([\,], G, (\top, [\,], [\,])) \quad \text{und} \\
\mathsf{re\text{-}unify}(D, \mathsf{unify}(G)) &:= \mathsf{unify'}([\,], \mathsf{del}(D, R), (\top, \mathsf{del}(D, T), [\,])), \\
& \text{wobei } \mathsf{unify}(G) = (\Box, T, R) \text{ gilt.}
\end{aligned}$$

Dabei ist G eine Liste markierter Gleichungen, deren Unifizierbarkeit zu entscheiden ist und für die im positiven Fall eine Lösungsform zu berechnen ist. Die zweite Variante ist speziell zur Anpassung einer zuvor berechneten Lösungsform nach dem Löschen von syntaktischen Gleichungen bestimmt. Die Eingabe besteht jedoch nicht aus der reduzierten Liste markierter Gleichungen $\mathsf{del}(D, G)$, sondern aus der Markierung D und dem ursprünglichen Ergebnis der Unifikation $\mathsf{unify}(G)$.

Das Zeichen $\Box$ ist entweder $\top$ oder $\perp$ und gibt Auskunft über die Unifizierbarkeit der syntaktischen Gleichungen in G; $\top$ steht für (unendlich) unifizierbar und $\perp$ für nicht unifizierbar. Bei Unifizierbarkeit ist die berechnete Lösungsform T äquivalent zu G. Die Liste markierter Gleichungen R enthält eine Teilfolge der markierten Gleichungen in G, die zur Lösungsanpassung nach einem Löschen erneut berücksichtigt werden müssen, um einerseits eventuell zuviel gelöschte Variablenbindungen in der Lösungsform erneut zu etablieren oder um andererseits Gleichungen, die wegen Nicht-Unifizierbarkeit noch nicht bearbeitet wurden, zu berücksichtigen. — Das Tripel $(\Box, T, R)$ bezeichnen wir im folgenden als *Lösungstripel*.

Das bei den Definitionen der Verfahren unify und $\mathsf{re\text{-}unify}$ zugrundegelegte Verfahren $\mathsf{unify'}$ sei durch die folgenden Festlegungen definiert:

$$\mathsf{unify'}(H, [\,], (\Box, T, R)) := (\Box, T, R)$$

$$\begin{aligned}
&\mathsf{unify'}(H, [\langle s_1 \doteq t_1\rangle^{P_1}, \langle s_2 \doteq t_2\rangle^{P_2}, \ldots, \langle s_n \doteq t_n\rangle^{P_n}], (\top, T, R)) \\
&\quad := \quad \mathsf{unify'}(H, [\langle s_2 \doteq t_2\rangle^{P_2}, \ldots, \langle s_n \doteq t_n\rangle^{P_n}], \\
&\qquad\qquad\qquad \mathsf{i\text{-}unify}(H, \langle s_1 \doteq t_1\rangle^{P_1}, (T, R)))
\end{aligned}$$

$$\begin{aligned}
&\mathsf{unify'}([\,], [\langle s_1 \doteq t_1\rangle^{P_1}, \langle s_2 \doteq t_2\rangle^{P_2}, \ldots, \langle s_n \doteq t_n\rangle^{P_n}], (\perp, T, R)) \\
&\quad := \quad (\perp, T, R \cdot [\langle s_1 \doteq t_1\rangle^{P_1}, \langle s_2 \doteq t_2\rangle^{P_2}, \ldots, \langle s_n \doteq t_n\rangle^{P_n}])
\end{aligned}$$

$$\begin{aligned}
&\mathsf{unify'}(H, [\langle s_1 \doteq t_1\rangle^{P_1}, \langle s_2 \doteq t_2\rangle^{P_2}, \ldots, \langle s_n \doteq t_n\rangle^{P_n}], (\perp, T, R)) \\
&\quad := \quad (\perp, T, R) \quad \text{falls } H \neq [\,]
\end{aligned}$$

Die Eingabe des Verfahrens $\mathsf{unify'}$ besteht aus einem *lokalen Cache* H, der neben der aktuell bearbeiteten Gleichung alle Gleichungen enthält, die sich durch Gleichsetzen der Argumente zweier Funktionsterme ergeben und bereits bearbeitet wurden. Es folgt eine Liste der noch zu bearbeitenden Gleichungen und ein Lösungstripel, das Ergebnis der bereits bearbeiteten Gleichungen. Die Ausgabe ist wiederum ein Lösungstripel.

3.2.1 Ein Unifikationsschritt — i-unify

Das Unifikationsverfahren für eine Liste markierter Gleichungen besteht aus einer schrittweisen Bearbeitung der einzelnen Gleichungen. Das dazu verwendete Verfahren ist durch die folgenden Definitionen festgelegt:

$$\begin{aligned}
\textsf{i-unify}(H, \langle s \doteq t\rangle^P, (T,R)) &:= (\top, T, R) \quad \text{falls } s = t \\
\textsf{i-unify}(H, \langle s \doteq t\rangle^P, (T,R)) &:= (\top, T, R) \quad \text{falls } s \neq t \text{ und } \langle s \doteq t\rangle^Q \in H \\
\textsf{i-unify}(H, \langle s \doteq t\rangle^P, (T,R)) &:= \textsf{bind}([\langle s \doteq t\rangle^P | H], \textsf{deref}(s,T), \textsf{deref}(t,T), (T,R)) \\
&\ \text{falls } s \neq t \text{ und } \langle s \doteq t\rangle^Q \notin H
\end{aligned}$$

Der fehlende *Occur Check* wurde durch den Test $\langle s \doteq t\rangle^Q \in H$ ersetzt: Wurde die aktuell zu bearbeitende Gleichung $s \doteq t$ bereits bearbeitet, ist eine Bearbeitung überflüssig und zu vermeiden, da eine weitere Bearbeitung zur Nichtterminierung des Verfahren führen würde.

3.2.2 Bindungen — bind

In den einzelnen Unifikationsschritten werden alle möglichen Fälle der zu unifizierenden Termpaare unterschieden und dabei entweder weitere Variablenbindungen etabliert, das Unifikationsverfahren rekursiv auf die Subterme von Funktionstermen angewendet oder Nicht-Unifizierbarkeit festgestellt und vermerkt.

Gleiche Variablen: Aufgrund der ersten definierten Gleichheit für i-unify gilt in den nachfolgenden Fällen $L \neq [\,]$ oder $Q \neq [\,]$.

$$\begin{aligned}
\textsf{bind}([g^P|H], z_V^L, z_W^Q, (T,R)) &:= (\top, T, R) \quad \text{falls } z \in \mathcal{V} \text{ und } H \neq [\,] \\
\textsf{bind}([g^P], z_V^L, z_W^Q, (T,R)) &:= (\top, T, R \cdot [g^P]) \quad \text{falls } z \in \mathcal{V}
\end{aligned}$$

Unterschiedliche Variablen:

$$\begin{aligned}
&\textsf{bind}([g^P|H], x_V^L, y_W^Q, (T,R)) \\
&\quad := \ (\top, T \cup \{y_{[x]}^{[L \otimes [P] \otimes Q]}\}, R) \quad \text{falls } x \in \mathcal{V}, y \in \mathcal{V},\ x \neq y \text{ und } H \neq [\,]
\end{aligned}$$

$$\begin{aligned}
&\textsf{bind}([g^P], x_{[]}^{[]}, y_{[]}^{[]}, (T,R)) \\
&\quad := \ (\top, T \cup \{y_{[x]}^{[P]}\}, R) \quad \text{falls } x \in \mathcal{V}, y \in \mathcal{V},\ x \neq y \text{ und } g^P = \langle x \doteq y\rangle^P
\end{aligned}$$

$$\begin{aligned}
&\textsf{bind}([g^P], x_V^L, y_W^Q, (T,R)) \\
&\quad := \ (\top, T \cup \{y_{[x]}^{[L \otimes [P] \otimes Q]}\}, R \cdot [g^P]) \\
&\qquad\quad \text{falls } x \in \mathcal{V}, y \in \mathcal{V},\ x \neq y \text{ und } g^P \neq \langle x \doteq y\rangle^P
\end{aligned}$$

Funktionsterm und Variable:

$$\begin{aligned}&\mathsf{bind}(H, f_W^Q(t_1,\ldots,t_k), x_V^L, (T,R))\\ :=\ &\mathsf{bind}(H, x_V^L, f_W^Q(t_1,\ldots,t_k), (T,R)) \quad \text{falls } x \in \mathcal{V}\end{aligned}$$

Variable und Funktionsterm: Im folgenden gelte $x \in \mathcal{V}$.

$$\begin{aligned}&\mathsf{bind}([g^P|H], x_V^L, f_W^Q(t_1,\ldots,t_k), (T,R))\\ :=\ &(\top, T \cup \{f_{[x]}^{L\otimes[P]\otimes Q}(t_1,\ldots,t_k)\}, R) \quad \text{falls } H \neq [\,]\end{aligned}$$

$$\begin{aligned}&\mathsf{bind}([g^P], x_{[]}^{[]}, f_{[]}^{[]}(t_1,\ldots,t_k), (T,R))\\ :=\ &(\top, T \cup \{f_{[x]}^{[P]}(t_1,\ldots,t_k)\}, R) \quad \text{falls } g^P = \langle x \doteq f(t_1,\ldots,t_k)\rangle^P\end{aligned}$$

$$\begin{aligned}&\mathsf{bind}([g^P], x_V^L, f_W^Q(t_1,\ldots,t_k), (T,R))\\ :=\ &(\top, T \cup \{f_{[x]}^{L\otimes[P]\otimes Q}(t_1,\ldots,t_k)\}, R\cdot[g^P]) \quad \text{falls } g^P \neq \langle x \doteq f(t_1,\ldots,t_k)\rangle^P\end{aligned}$$

Zwei Funktionsterme mit identischen Funktionssymbolen:

$$\begin{aligned}&\mathsf{bind}([g^P|H], f_V^L(s_1,\ldots,s_k), f_W^Q(t_1,\ldots,t_k), (T,R))\\ :=\ &\mathsf{unify'}([g^P|H], [\langle s_i \doteq t_i\rangle^{L\otimes[P]\otimes Q} \mid i = 1,\ldots,k], (\top, T, R)) \quad \text{falls } H \neq [\,]\end{aligned}$$

$$\begin{aligned}&\mathsf{bind}([g^P], f_V^L(s_1,\ldots,s_k), f_W^Q(t_1,\ldots,t_k), (T,R))\\ :=\ &\mathsf{unify'}([g^P], [\langle s_i \doteq t_i\rangle^{L\otimes[P]\otimes Q} \mid i = 1,\ldots,k], (\top, T, R\cdot[g^P]))\end{aligned}$$

Funktionsterme mit unterschiedlichen Funktionssymbolen:

$$\begin{aligned}&\mathsf{bind}([g^P|H], f_V^L(s_1,\ldots,s_k), g_W^Q(t_1,\ldots,t_l), (T,R))\\ :=\ &(\bot, T, R)) \quad \text{falls } f \neq g \text{ und } H \neq [\,]\end{aligned}$$

$$\begin{aligned}&\mathsf{bind}([g^P], f_V^L(s_1,\ldots,s_k), g_W^Q(t_1,\ldots,t_l), (T,R))\\ :=\ &(\bot, T, R\cdot[g^P])) \quad \text{falls } f \neq g\end{aligned}$$

Der Cache H dient neben der Speicherung der aktuell bearbeiteten Gleichungen einem weiteren Zweck: Enthält er nur eine markierte Gleichung, $H = [g^P]$, dann ist g^P in der Liste der markierten Gleichungen G enthalten und muß bei einer Lösungsanpassung nach einem Löschen gegebenenfalls erneut berücksichtigt werden.

3.3 Wesentliche Eigenschaften

Ist bei Unifizierbarkeit die berechnete Lösungsform T zyklenfrei, dann ist σ_T (siehe Definition (1)) ein *allgemeinster (idempotenter) Unifikator* der syntaktischen Gleichungen $s_1 \doteq t_1, \ldots, s_n \doteq t_n$.

Sowohl beim Hinzufügen von markierten Gleichungen als auch beim Löschen markierter Gleichungen läßt sich die vorherige Lösung anpassen. Es gilt:

$$\mathsf{unify}(G \cdot G') = \mathsf{unify'}([\,], G', \mathsf{unify}(G)) \tag{2}$$
$$\mathsf{unify}(\mathsf{del}(D, G)) \simeq \mathsf{re\text{-}unify}(D, \mathsf{unify}(G)) \tag{3}$$

für zwei beliebige Listen G und G' markierter Gleichungen und eine beliebige Markierung D und zwar unabhängig davon, ob nun die Gleichungen in G unifizierbar sind oder nicht. Die Äquivalenz $\simeq$ in Gleichung (3) besagt, daß bei Unifizierbarkeit die Lösungformen äquivalent sind, das heißt, die gleiche Lösungsmenge haben, und daß bei Zyklenfreiheit der Lösungsformen die konstruierbaren allgemeinsten Unifikatoren Varianten voneinander sind.

3.3.1 Ein Beispiel

Die Liste markierter Gleichungen $G = [\langle x \doteq y\rangle^{\{1\}}, \langle y \doteq x\rangle^{\{2\}}, \langle y \doteq z\rangle^{\{3\}}]$ ist offensichtlich unifizierbar, und es gilt

$$\mathsf{unify}(G) = (\top, \{y_{[x]}^{[\{1\}]}, z_{[y]}^{[\{3\}]}\}, [\langle y \doteq x\rangle^{\{2\}}]).$$

Da die berechnete Lösungsform $T = \{y_{[x]}^{[\{1\}]}, z_{[y]}^{[\{3\}]}\}$ zyklenfrei ist, existiert die idempotente Substitution $\sigma_T = \{x/z, y/z\}$, die ein allgemeinster Unifikator der syntaktischen Gleichungen $\{x \doteq y, y \doteq x, y \doteq z\}$ ist.

Eine Lösungsanpassung nach dem Löschen der markierten Gleichung $\langle x \doteq y\rangle^{\{1\}}$ in G kann auf zwei Arten erfolgen:

$$\begin{aligned}
\mathsf{unify}(\mathsf{del}(\{1\}, G)) &= (\top, \{x_{[y]}^{[\{2\}]}, z_{[x]}^{[\{2,3\}]}\}, [\langle y \doteq z\rangle^{\{3\}}]) \quad \text{oder} \\
\mathsf{re\text{-}unify}(\{1\}, \mathsf{unify}(G)) &= \mathsf{unify'}([\,], [\langle y \doteq x\rangle^{\{2\}}], (\top, \{z_{[y]}^{[\{3\}]}\}, [\,])) \\
&= (\top, \{z_{[y]}^{[\{3\}]}, x_{[z]}^{[\{2,3\}]}\}, [\langle y \doteq x\rangle^{\{2\}}]).
\end{aligned}$$

Die beiden konstruierbaren, allgemeinsten Unifikatoren $\{x/z, y/z\}$ und $\{y/x, z/x\}$ von $\{y \doteq x, y \doteq z\}$ sind Varianten voneinander.

3.4 Nicht-Unifizierbarkeit — Ursachen und Behebung

Mit geringem Zusatzaufwand lassen sich Ursachen für eine Nicht-Unifizierbarkeit feststellen: Dazu ist bei den obigen Definitionen von bind das Zeichen $\perp$ an den entsprechenden Stellen durch $\perp(L \otimes [P] \otimes Q)$ zu ersetzen. Betrachtet man die Markierungen der markierten Gleichungen und Variablenbindungen als deren Begründungen, dann ist die Markierung in $\perp(L \otimes [P] \otimes Q)$ eine Begründung für die Nicht-Unifizierbarkeit. Die mit $\perp$ aufgezeichneten Markierungen können zu einem späteren Zeitpunkt zur Behebung der Nicht-Unifizierbarkeit genutzt werden. Dazu sind die entsprechenden markierten Gleichungen aus der nicht unifizierbaren

Gleichungsmenge G zu löschen und die Lösung ist anzupassen. Dies erfolgt am einfachsten mit einem erneuten Aufruf von $\mathsf{re\text{-}unify}(D, \mathsf{unify}(G))$, wobei D eine Teilmenge von $L \otimes [P] \otimes Q$ ist. Dies ist solange zu wiederholen, bis eine der reduzierten Gleichungsmengen $\mathsf{del}(D, G)$ unifizierbar ist.

3.5 Implementierung und Resultate

Das vorgestellte Unifikationsverfahren wurde prototypisch in ECL^{i}PS^e [9] implementiert, ausgetestet und mit einer analogen Implementierung einer stark vereinfachten Form des Verfahrens verglichen, wobei bei einer Veränderung der Problemstellung in naiver Weise die Ergebnisse neu berechnet werden. Im vereinfachten Verfahren werden Markierungen weder berücksichtigt, berechnet noch verwaltet und es werden keine nach dem Löschen erneut zu berücksichtigenden Gleichungen bestimmt. Das heißt, der zusätzliche Verwaltungsaufwand, der für das vorgestellte Verfahren notwendig ist, entfällt.

Der Vergleich hat gezeigt, daß sich das inkrementelle Verfahren tatsächlich adaptiv verhält: Einerseits werden in der Lösungsform nur die Variablenbindungen gelöscht, deren Berechtigungen nicht eindeutig gesichert sind. Andererseits steigen gerade beim Löschen von Gleichungen die mit dem inkrementellen Verfahren erzielten Effizienzsteigerungen (die Verhältnisse der Laufzeiten der beiden verglichenen Verfahren) mit sinkenden Graden der Veränderung (die Anteile der gelöschten Gleichungen von der Gesamtzahl der betrachteten Gleichungen).

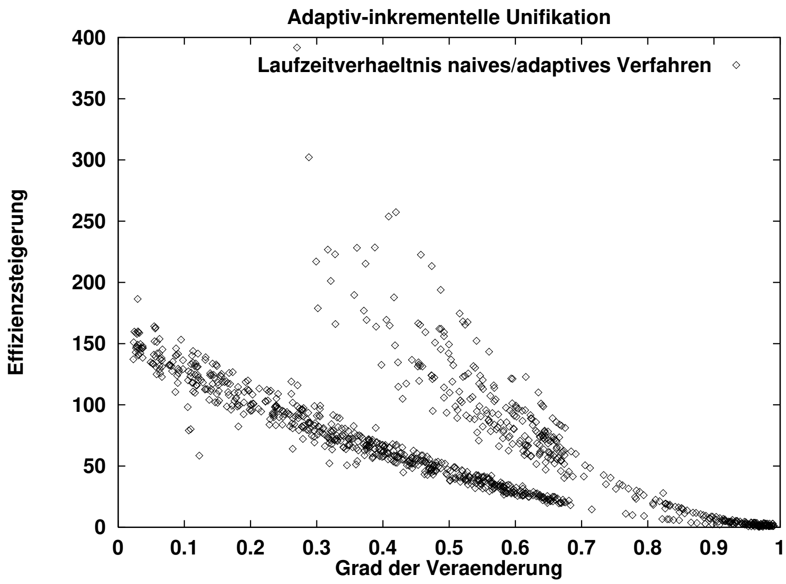

Abbildung 1: Effizienzsteigung von unify gegenüber einer naiven Neuberechnung.

Es wurden zufällig 100 unifizierbare Gleichungsmengen generiert, die aus jeweils 1999 markierten Gleichungen bestehen. Für jede dieser Mengen wurden

100 zufällig bestimmte Löschungen vorgenommen und sowohl die erzielten Effizienzsteigerungen als auch die Grade der Veränderung ermittelt. Das Ergebnis der Untersuchung ist in Abbildung 1 dargestellt.

Offenbar hängen die erreichten Effizienzsteigerungen stark von der Implementierung und den Problemstellungen ab. In einer früheren Implementierung wurden allgemeinere, aufwendiger zu verwaltende Markierungen verwendet, die ähnlich den *Labels* in ATMS [8, 16] sind; diese wurden unter Verwendung von Listen realisiert. In der jetzigen Implementierung werden Markierungen durch Bit-Vektoren repräsentiert, was zu einer weiteren, vier- bis fünffachen Effizienzsteigung führte. Die Resultate der Untersuchungen mit beiden Implementierungen lassen vermuten, daß die Effizienz des inkrementellen Verfahrens gegenüber einer naiven Neuberechnung bei geringer werdenden Änderungen der Problemstellung sogar überproportional wächst.

Es ist zu erwarten, daß mit einer maschinennahen Implementierung noch größere Effizienzsteigerungen möglich sind, da insbesondere Variablenbindungen als verzeigerte Strukturen realisierbar sind, wodurch beim Dereferenzieren die bisherige Suche in Listen entfällt. Die Suche erfordert in der jetzigen Implementierung gerade bei der Lösungsanpassung nach einem Löschen den größten Aufwand.

Literatur

[1] Slim Abdennadher. Operational semantics and confluence of constraint handling rules. In *Proceedings of the Third International Conference on Principles and Practice of Constraint Programming – CP97*, Lecture Notes in Computer Science. Springer Verlag, 1997.

[2] Slim Abdennadher, Thom Frühwirth, and Holger Meuss. On confluence of constraint handling rules. In Eugene C. Freuder, editor, *Proceedings of the Second International Conference on Principles and Practice of Constraint Programming – CP96*, number 1118 in Lecture Notes in Computer Science, pages 1–15. Springer Verlag, 1996.

[3] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison Wesley, Reading MA, 1974. Chapter 4: Data Structures for Set Manipulation Problems.

[4] Hassan Ait-Kaci. *Warren's Abstract Machine: a tutorial reconstruction.* The MIT Press, 1991.

[5] Karl-Friedrich Böhringer and Francis Newberry-Paulisch. Using constraints to achieve stability in automatic graph layout algorithms. In *Proceedings of the CHI — Conference on Human Factors in Computing Systems*, pages 43–51. ACM, April 1990.

[6] Christian Codognet and Philippe Codognet. Non-deterministic stream AND-parallelism based on intelligent backtracking. In *Proceedings of the 6th International Conference on Logic Programming*, pages 63–79, Lisbon, 1989. The MIT Press.

[7] Alain Colmerauer. Prolog and infinite trees. In K.L. Clark and S.-A. Tarnlund, editors, *Logic Programming*, pages 231–251. Academic Press, New York, 1982.

[8] Johan de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28:127–162, 1986.

[9] ECRC, München. *ECL^{i}PS^e 3.5, User Manual*, 1995.

[10] Thom Frühwirth. Constraint Handling Rules. In Andreas Podelski, editor, *Constraint Programming: Basics and Trends*, number 910 in Lecture Notes in Computer Science, pages 90–107. Springer Verlag, March 1995.

[11] Joxan Jaffar. Efficient unification over infinite terms. *New Generation Computing*, 1984.

[12] Michael J. Maher. Complete axiomatizations of algebras of finite, rational and infinite trees. In *Proc. of the 3rd IEEE Annual Sumposium on Logic in Computer Science*, pages 348–357, 1988.

[13] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 1982.

[14] Viswanath Ramachandran and Pascal van Hentenryck. Incremental algorithms for constraint solving and entailment over rational trees. *FSTTCS: Foundations of Software Technology and Theoretical Computer Science*, 13, 1993.

[15] G. Ramalingam and Thomas Reps. A categorized bibliography on incremental computation. In *Proceedings of the 20th annual ACM Symposium on Principles of Programming Languages*, pages 502–510. ACM, 1993.

[16] Michael Reinfrank. *Fundamentals and Logical Foundations of Truth Maintenance.* Dissertation no. 221, Department of Computer and Information Science, Linköping University, Sweden, 1989.

[17] Armin Wolf. Flexiblere Planung mit Constrainthierarchien. In J. Sauer, A. Günter, and J. Hertzberg (Herausgeber), *Planen und Konfigurieren 96 — PuK-96*, Band 3 der Reihe *Proceedings in Artificial Intelligence*, Seiten 104–110. Infix Verlag, 1996.