SEKI-Working Paper

Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern 1, W. Germany

COMTES - Vervollständigung von Termersetzungssystemen

Joachim Steinbach

SEKI Working Paper SWP-89-07

COMTES
– Vervollständigung von Termersetzungssystemen

Joachim Steinbach

Universität Kaiserslautern
Fachbereich Informatik
Postfach 3049
D-6750 Kaiserslautern
F.R.G.

Diese Arbeit wurde unterstützt vom Sonderforschungsbereich 314,
"Künstliche Intelligenz - Wissensbasierte Systeme", D4-Projekt.

Abstract

COMTES is a parametric Knuth-Bendix completion procedure that is particularly suited for efficiency experiments: Various critical pair methods, different reduction strategies and several term orderings are available. This paper describes these methods.

COMTES ist eine Softwareumgebung zur Vervollständigung von Termersetzungssystemen. COMTES stellt ein Experimentalsystem dar, das von verschiedenen Parametern abhängt: von mehreren Methoden zur Auswahl kritischer Paare, einigen Reduktionsstrategien und von diversen Termordnungen. Der vorliegende Bericht beschreibt diese Verfahren anhand ihrer Implementierungen.

Vorwort

Termersetzungssysteme gewinnen immer mehr an Bedeutung, da sie ein allgemeines Modell für nicht-deterministische Berechnungen darstellen. Klassische Anwendungsbereiche sind u. a. das Theorembeweisen, Programmverifikation und die Spezifikation von abstrakten Datentypen. Zwei Eigenschaften von Termersetzungssystemen sind dabei von zentraler Bedeutung: die Eindeutigkeit und die Terminierung. Der Prozeß der Vervollständigung ermöglicht es in manchen Fällen, diese Charakteristika zu garantieren. Er hat jedoch den Nachteil, daß er nicht immer terminiert.

Die aktuelle und hier dokumentierte Version (2.2) des COMTES-Systems ist das Ergebnis intensiver Forschungen auf dem Gebiet der Termersetzungssysteme. Die Implementierung eines Prototyps war Aufgabe einer Projektarbeit im Sommersemester 1984, durchgeführt im Fachbereich Informatik der Universität in Kaiserslautern ([WS84]). In den Jahren 1985 und 1986 wurde diese Grundversion ausgebaut. Zwei Aspekte hatte man sich zum Ziel dieser Erweiterung gesetzt: Effizienzsteigerung und Aufbau eines Experimentalsystems. Im Rahmen zweier Diplomarbeiten wurden u.a. verschiedene Reduktionsstrategien ([Wa86]) und mehrere Termordnungen ([St86]) integriert. Zu dieser Zeit war das System noch in Pascal geschrieben und stand auf einer VAX unter VMS zur Verfügung. Danach wurde es, unter finanzieller Unterstützung des Sonderforschungsbereichs 314, auf Apollo unter Aegis portiert. Als Programmiersprache wurde Common-Lisp ausgewählt. Nach Beendigung dieser Installation, Ende 1987, kamen einige Verfahren zur Verringerung der Anzahl zu erzeugender kritischer Paare hinzu ([So88]). Die Untersuchung von Termordnungen ([St88]) lieferte einige Resultate zur Verbesserung der bereits implementierten Techniken. Diese wurden im System umgesetzt. Abschließend wurden, wiederum im Rahmen einer Projektarbeit ([Pa89], siehe auch [Ze89]), Anfang des Jahres Polynomordnungen zur Verfügung gestellt.

Dieser Bericht stellt eine Zusammenfassung der Charakteristiken des COMTES-Systems dar (Die Kapitel 4 und 5 sind teilweise den entsprechenden Projekt-/Diplomarbeiten entnommen). Er soll zum einen als Anleitung zum Gebrauch von COMTES dienen, zum anderen Unterstützung beim Entwurf einer neuen Vervollständigungsumgebung bieten.

Eine kompakte Darstellung theoretischer Grundlagen (Kapitel 1) weist auf die Problembereiche beim Vervollständigen von Termersetzungssystemen hin. Der Überblick über die Struktur von COMTES soll deutlich machen, welche Fragestellungen angegangen werden (Kapitel 2). Das dritte Kapitel stellt den globalen Rahmen (den Vervollständigungsalgorithmus) von COMTES vor. Den Parametern des Systems sind die Kapitel 4 bis 7 gewidmet. Das Protokoll einer Sitzung ist in Kapitel 8 enthalten (ein Katalog von Beispielen ist in Bearbeitung). Kapitel 9 können Details über die Implementierung (z.B. verwendete Datenstrukturen) entnommen werden.

An dieser Stelle möchte ich mich bei Manuela Gaß, Rita Kohl und Inger Sonntag bedanken, deren Unterstützung und kritischen Kommentare das Zustandekommen dieser Arbeit wesentlich erleichtert haben.

Inhalt

1 Theoretische Grundlagen

1.1 Motivation

Häufig werden Strukturen, in denen man rechnen möchte, durch definierende Gleichungen beschrieben. Die Beschreibung geschieht so, daß man Namen für Operationen angibt, die in der Struktur realisiert werden sollen, und daß man durch Gleichungen die Wirkungsweise der Operationen und ihre gegenseitige Abhängigkeit festlegt. Dabei ergibt sich die Frage, ob zwei Ausdrücke im Rahmen des erzeugten Gleichungssystems äquivalent sind. Dieses Problem heißt das Wortproblem und ist von fundamentaler Bedeutung für das effektive Rechnen in Strukturen.

Wie läßt es sich lösen? Offenbar gilt s = t genau dann, wenn sich s durch Anwendung der Gleichungen in t überführen läßt. Der Test, ob dies möglich ist, erscheint recht aufwendig. Da jede Gleichung von rechts nach links und von links nach rechts gelesen werden kann, muß auf jeden Fall verhindert werden, daß der Test in eine unendliche Schleife läuft. Wir lösen dieses Problem gewaltsam dadurch, daß wir die Gleichungen nur noch in einer Richtung anwenden, d.h. wir machen aus dem Gleichungssystem ein Regelsystem.

Anschaulich ist für jede Regel die rechte Seite einfacher als die linke. Wir nennen die Anwendung einer Regel auf einen Ausdruck einen Reduktionsschritt und definieren die Reduktionsrelation $\Longrightarrow$: s $\Longrightarrow$ t falls es eine Regel l $\rightarrow$ r gibt deren linke Seite in s vorkommt und t aus s entsteht, indem man l durch r ersetzt. Die Relation $\Longrightarrow$ ist so definiert, daß s und t die gleichen Ausdrücke darstellen.

Nun gilt für die Lösung des Wortproblems: Lassen sich s und t in endlich vielen $\Longrightarrow$-Schritten auf ein gemeinsames Element reduzieren, so ist s = t. Dieses Ergebnis ist noch unbefriedigend, da es unklar ist, wie man entscheidet, ob sich s und t auf ein gemeinsames Element reduzieren lassen. Könnte man gewährleisten, daß sich jeder Ausdruck s auf genau eine berechenbare 'Normalform' $\hat{s}$ (nicht mehr zu reduzierender Ausdruck) reduzieren läßt, so gilt s = t genau dann, wenn $\hat{s}$ und $\hat{t}$ übereinstimmen. Diese Eigenschaft kann garantiert werden, falls man das Regelsystem als terminierend (es gibt mindestens eine Normalform) und konfluent (es existiert höchstens eine Normalform) nachweist.

Eine Ordnung > - eine zweistellige, irreflexive und transitive Relation - ermöglicht den Nachweis der Terminierung, falls jede linke Seite einer Regel größer (bzgl. >) ist als die entsprechende rechte Seite.

Die Eigenschaft der Eindeutigkeit der Reduktionen kann mittels des Konzepts der kritischen Paare gezeigt werden, d.h. alle Ausdrücke (Überlappungen durch unifizierbare linke Seiten von Regeln) von denen jeder auf zwei verschiedene Ausdrücke (in einem Schritt) reduziert werden kann (kritisches Paar), müssen untersucht werden. Das Regelsystem ist konfluent, falls alle diese kritischen Paare auf jeweils einen gemeinsamen Nachfolger reduziert werden können.

Ist ein Regelsystem nicht konfluent, versucht man es in ein dazu äquivalentes, konfluentes zu transformieren. Dies ist mit Hilfe des sogenannten Vervollständigungsalgorithmus' von Knuth und Bendix möglich, indem nicht konfluente kritische Paare in (terminierende) Regeln umgewandelt und zum initialen Regelsystem aufgenommen werden. Anschließend müssen, unter Berücksichtigung dieser neu entstandenen Regeln, weitere kritische Paare generiert und der gesamte Prozeß wiederholt werden.

Der Knuth-Bendix Algorithmus ist für spezielle Regelsysteme, sogenannte Termersetzungssysteme, entwickelt worden. In den folgenden Abschnitten folgt deshalb die Übertragung (und konkrete Definition) der abstrakten Begriffe auf Termersetzungssysteme.

1.2 Terme

Ein Termersetzungssystem über einer Menge Γ von Termen ist eine Menge von Regeln $l_i \Rightarrow r_i$ und wird deshalb auch Regelsystem genannt. Die l_i und r_i sind Terme aus Γ. Ein Term besteht aus einer Aneinanderreihung von Funktionssymbolen und Variablen. Die Menge der Funktionssymbole sei durch $\mathfrak{F}$ gekennzeichnet, die der Variablen durch $\mathfrak{V}$. Infolgedessen ist die Menge Γ der Terme durch $\mathfrak{F}$ und $\mathfrak{V}$ bestimmt und wird deshalb im weiteren mit $\Gamma(\mathfrak{F},\mathfrak{V})$ bezeichnet. Die Teilmenge $\Gamma(\mathfrak{F},\emptyset)$ wird als Menge der Grundterme bezeichnet.

Das führende Funktionssymbol eines Terms t wird Topsymbol genannt: top(t). Args(t) ist die Menge der Argumente von t (direkte Teilterme). Die Teilterme eines Terms sind der Term selbst und die Teilterme seiner Argumente. Variablen und Konstanten haben nur sich selbst als Teilterm. Eine Teiltermposition oder Stelle ist eine endliche Folge von nicht negativen ganzen Zahlen, die durch '.' getrennt werden. Die leere Sequenz ε entspricht dem vollständigen Term. Ist $f(t_1,...,t_n)$ der Teilterm an der Position i, so steht t_j an der Stelle i.j. Die Menge aller Stellen eines Terms t wird durch O(t) abgekürzt. t/u ist der Teilterm von t an der Stelle u. t[u$\leftarrow$ s] bezeichnet den Term, der aus t entsteht, indem t/u durch s ersetzt wird.

Ein Termersetzungssystem legt fest, welche Terme durch welche anderen Terme ersetzt werden dürfen. Eine Regel $l \Rightarrow r$ kann auf einen Term t angewendet werden, falls es eine Substitution σ - Ersetzung von Variablen durch Terme aus $\Gamma(\mathfrak{F},\mathfrak{V})$ - und eine Stelle u in t gibt, sodaß $t/u = \sigma(l)$. Die Anwendung der Regel $l \Rightarrow r$ auf t wird ausgeführt, indem t/u durch $\sigma(r)$ ersetzt wird: $t := s[u \leftarrow \sigma(r)]$. Wir schreiben $s \Longrightarrow t$, falls t auf die beschriebene Art aus s herleitbar ist und sagen, daß s zu t reduziert werden kann. Kann t nicht weiter reduziert werden, so ist t irreduzibel und wird als Normalform bezeichnet.

1.3 Terminierung und Konfluenz

Es ist möglich, daß ein Term unendlich oft reduziert werden kann, also gar keine Normalform besitzt. Hat ein Termersetzungssystem (Regelsystem) $\mathfrak{R}$ die Eigenschaft, daß keine unendliche Kette von Reduktionen existiert, so heißt $\mathfrak{R}$ noethersch. Leider ist diese Eigenschaft im allgemeinen unentscheidbar, sogar für den Fall daß $\mathfrak{R}$ nur aus einer Regel besteht. Es gibt jedoch einige Verfahren die eine Lösung des Problems in vielen Fällen zulassen.

Sie basieren auf der Idee, daß die Reduktionsrelation $\Longrightarrow$ eines Regelsystems in einer wohlfundierten Ordnung auf Termen enthalten ist. Eine (Partial-) Ordnung auf $\Gamma(\mathfrak{F},\emptyset)$ ist eine transitive und irreflexive Binärrelation > und heißt wohlfundiert, falls es keine unendlichen Ketten von Termen gibt. Um die Inklusion '$\Longrightarrow \subseteq >$' zu gewährleisten, müssen alle (unendlich viele) Ableitungen (bzgl. $\Longrightarrow$) getestet werden. Um diesen Test zu vermeiden, verwenden wir Reduktionsordnungen. Eine Reduktionsordnung > ist eine wohlfundierte Partialordnung, die kompatibel mit der Termstruktur ist, d.h.

$$t_1 > t_2 \quad \text{impliziert} \quad s[u \leftarrow t_1] > s[u \leftarrow t_2],$$

für alle $s, t_1, t_2 \in \Gamma(\mathfrak{F},\emptyset)$ und $u \in O(s)$.

Auf der Basis von Reduktionsordnungen läßt sich die Terminierung von Termersetzungssystemen auf folgende Art beweisen: Ein Termersetzungssystem $\mathfrak{R}$ über $\Gamma(\mathfrak{F},\mathfrak{V})$ terminiert (d.h., ist noethersch) genau dann, wenn es eine Reduktionsordnung > auf $\Gamma(\mathfrak{F},\emptyset)$ gibt, sodaß $\sigma(l) > \sigma(r)$ für jede Regel $l \to r$ und jede Substitution σ. Dieses Theorem weist auf ein weiteres Problem hin: Stabilität gegenüber Substitutionen, d.h. es sind i.a. unendlich viele Tests durchzuführen. Deshalb fordern wir von einer Reduktionsordnung, daß sie außerdem noch abgeschlossen ist gegenüber Substitutionen, d.h.

$$s > t \quad \text{impliziert} \quad \sigma(s) > \sigma(t), \quad \text{für alle } \sigma.$$

Eine Klasse von Termordnungen, die diese Eigenschaften erfüllt, ist die Simplifikationsordnung. Eine Partialordnung heißt Simplifikationsordnung, falls sie folgende zwei Eigenschaften besitzt:

- Kompatibilität bzgl. der Termstruktur und
- Teiltermeigenschaft (jeder Term ist größer als jeder seiner echten Teilterme).

Eine Simplifikationsordnung hat gegenüber einer Reduktionsordnung den Vorteil, daß sie noethersch ist. Das COMTES-System stellt sieben verschiedene Simplifikationsordnungen zur Verfügung. Sie werden in Kapitel 6 ausführlich beschrieben.

Ein teminierendes Regelsystem kann also nur in Zusammenhang mit einer Ordnung gesehen werden. Wenn ein Regelsystem terminierend ist, kann man zu jedem Term mindestens eine Normalform bestimmen. Oft können Terme aber nicht eindeutig reduziert werden, sondern es gibt innerhalb des Terms mehrere Stellen, an denen eine Regel anwendbar ist. Dies führt häufig dazu, daß ein Term mehrere Normalformen besitzt. Die Konfluenz der Reduktionsrelation impliziert nun, daß das Ergebnis der Reduktion unabhängig von der Reduktion in den einzelnen Schritten ist.

Zwei Terme $t_1, t_2 \in \Gamma(\mathfrak{F},\mathfrak{V})$ heißen konfluent, wenn es einen Term s gibt, sodaß $t_1 \overset{*}{\Longrightarrow} s$ und $t_2 \overset{*}{\Longrightarrow} s$. (Schreibweise: $t_1 \downarrow t_2$; $\overset{*}{\Longrightarrow}$ ist die reflexive und transitive Hülle von $\Longrightarrow$).
$\Longrightarrow$ heißt konfluent, falls aus $s \overset{*}{\Longrightarrow} t_1$ und $s \overset{*}{\Longrightarrow} t_2$ folgt: $t_1 \downarrow t_2$ für alle $t_1, t_2, s \in \Gamma(\mathfrak{F},\mathfrak{V})$.
$\Longrightarrow$ heißt lokal konfluent, falls aus $s \Longrightarrow t_1$ und $s \Longrightarrow t$ folgt: $t_1 \downarrow t_2$ für alle $t_1, t_2, s \in \Gamma(\mathfrak{F},\mathfrak{V})$.

Das 1942 entwickelte Newman-Lemma legt die Beziehung zwischen den Begriffen 'Konfluenz' und 'lokaler Konfluenz' fest: Ein noethersches Regelsystem ist genau dann konfluent, wenn es lokal konfluent ist. Es zeigt sich, daß in vielen Fällen der Nachweis der lokalen Konfluenz einfacher ist als der der Konfluenz. Notwendig sind nun hinreichende Bedingungen für die lokale Konfluenz. Die, von Knuth und Bendix entwickelten, kritischen Paare stellen eine solche Möglichkeit dar. Das Knuth-Bendix Theorem legt folgende Beziehung fest: Ein noethersches Regelsystem ist genau dann lokal konfluent, wenn alle kritischen Paare konfluent sind. Im folgenden werden wir die Definition eines kritischen Paares angeben.

Zwei Regeln $l \to r$ und $l' \to r'$ überlappen sich, wenn es eine Stelle $u \in O(l)$ und eine Substitution σ (Unifikator) gibt mit: $\sigma(l)/u = \sigma(l')$. Der Term $\sigma(l)$ wird Überlappung (Superposition) genannt. Er kann mit der Regel $l \to r$ an der Stelle ε und mit der Regel $l' \to r'$ an der Stelle u reduziert werden. Das Termpaar $\sigma(r)$, $\sigma(l)[u \leftarrow \sigma(r')]$ wird als kritisches Paar bezeichnet.

Buchberger verfolgte das Ziel, die hinreichende Bedingung für die (lokale) Konfluenz noetherscher Regelsysteme noch weiter abzuschwächen als Newman. Aufbauend auf dem Begriff 'verbunden unterhalb' formulierte er diese Abschwächung. Zwei Terme $t_1, t_2 \in \Gamma(\mathfrak{F},\mathfrak{V})$ heißen verbunden unterhalb $s \in \Gamma(\mathfrak{F},\mathfrak{V})$, falls es Terme s_i gibt mit $t_1 = s_0 \Longleftrightarrow s_1 \Longleftrightarrow \ldots \Longleftrightarrow s_n = t_2$ und $s > s_i$ (> ist eine Partialordnung). Wir drücken diesen Sachverhalt durch $t_1 \overset{*}{\underset{<s}{\Longleftrightarrow}} t_2$ aus. Mittels dieses Konzeptes kann das Newman Lemma wie folgt abgeändert werden ('Generalized Newman Lemma'): Ein noethersches (> Partialordnung) Regelsystem ist genau dann (lokal) konfluent, wenn aus $s \Longrightarrow t_1$ und $s \Longrightarrow t_2$ folgt $t_1 \overset{*}{\underset{<s}{\Longleftrightarrow}} t_2$, für alle $s, t_1, t_2 \in \Gamma(\mathfrak{F},\mathfrak{V})$. Das zusätzliche Einbeziehen von kritischen Paaren ermöglicht eine neue Definition des Knuth-Bendix Theorems ('Generalized Knuth-Bendix Theorem'): Ein noethersches (> Partialordnung) Regelsystem ist genau dann (lokal) konfluent wenn für alle kritischen Paare (t_1, t_2) gilt, daß $t_1 \overset{*}{\underset{<s}{\Longleftrightarrow}} t_2$ wobei $s \in \Gamma(\mathfrak{F},\mathfrak{V})$ die zugehörige Überlappung ist.

Winkler und Buchberger, Küchlin, Kapur und Musser und Narendran entwickelten daraus Konfluenzkriterien, welche der Verminderung der zu bearbeitenden kritischen Paare dienen. Sowohl diese Methoden als auch die klassische Art nach Knuth und Bendix sind im COMTES-System implementiert und werden in Kapitel 4 vorgestellt.

In einem konfluenten und noetherschen Regelsystem $\mathfrak{R}$ ist die Normalform eines Terms eindeutig bestimmt. $\mathfrak{R}$ wird deshalb auch eindeutig terminierend genannt.

1.4 Vervollständigungsalgorithmus

Um Regelsysteme sinnvoll einsetzen zu können, ist es unbedingt notwendig, daß sie eindeutig terminierend sind. Da dies aber oft nicht der Fall ist, gibt es eine Möglichkeit, ein gegebenes System in ein eindeutig terminierendes zu transformieren.

Durch ein Regelsystem ist auch in eindeutiger Weise eine Äquivalenzrelation auf den Termen definiert. Danach sind zwei Terme äquivalent, wenn einer aus dem anderen durch mehrfaches Ersetzen von Gleichem durch Gleiches entsteht. Was bedeutet es nun,

daß jeder Term eine eindeutige Normalform besitzt? Es bedeutet, daß es in jeder Äquivalenzklasse ein Element gibt, das bzgl. der vorgegebenen Ordnung minimal ist. Dieses Element ist die Normalform aller Terme dieser Äquivalenzklasse.

Würde man ein nicht eindeutig terminierendes Termersetzungssystem so erweitern, daß man Regeln hinzunimmt, die alle Terme jeweils auf das kleinste Element ihrer Äquivalenzklasse reduzieren, so hätte man das System in ein äquivalentes, eindeutig terminierendes Termersetzungssystem transformiert. Diese Strategie verfolgt der sogenannte Vervollständigungsalgorithmus von Knuth und Bendix.

Die Aufgabe des Algorithmus' ist es, alle kritischen Paare zu bestimmen und zu überprüfen, ob sie konfluent sind, d.h. ob die Normalformen beider Seiten identisch sind. Sind alle kritischen Paare konfluent, so ist auch das Regelsystem konfluent. Sind die Normalformen $\hat{s}$, $\hat{t}$ eines kritischen Paares nicht identisch, so wird eine neue Regel gebildet: Ist $\hat{s} > \hat{t}$ (bzgl. der Ordnung >), so entsteht die neue Regel $\hat{s} \rightarrow \hat{t}$. Ist $\hat{t} > \hat{s}$, so heißt die neue Regel $\hat{t} \rightarrow \hat{s}$. Sind die Terme $\hat{s}$ und $\hat{t}$ unvergleichbar, so stoppt der Algorithmus.

Die, auf diese Weise aus allen kritischen Paaren generierten, neuen Regeln werden zum initialen System hinzugefügt und der ganze Prozeß wiederholt. Dies bedeutet, daß das eindeutig terminierende Regelsystem schrittweise aus dem initialen Regelsystem gewonnen wird, indem Systeme $\mathfrak{R}_i$ generiert werden, die sich von $\mathfrak{R}_{i-1}$ durch Hinzunahme neuer Regeln unterscheiden. Diese Iteration wird erst abgebrochen, wenn keine kritischen Paare mehr gebildet werden können (d.h. $\mathfrak{R}_{i-1} = \mathfrak{R}_i$).
Dieser Algorithmus kann in einfacher Art und Weise als Inferenzsystem dargestellt werden (siehe Bachmair). Im folgenden sei > eine beliebige Reduktionsordnung, G eine Menge von Gleichungen (kritische Paare) und $\mathfrak{R}$ ein Termersetzungssystem ($\Longrightarrow_{\mathfrak{R}} \subseteq >$):

- Orientieren einer Gleichung :

$$\frac{G \cup \{s = t\}\ ,\ \mathfrak{R}}{G,\ \mathfrak{R} \cup \{s \rightarrow t\}} \qquad \text{falls} \quad s > t$$

- Hinzufügen eines kritischen Paares :

$$\frac{G\ ,\ \mathfrak{R}}{G \cup \{s = t\}\ ,\ \mathfrak{R}} \qquad \text{falls} \quad s \ {}_{\mathfrak{R}}\!\Longleftarrow r \Longrightarrow_{\mathfrak{R}} t$$

- Simplifizieren einer Gleichung :

$$\frac{G \cup \{s = t\}\ ,\ \mathfrak{R}}{G \cup \{r = t\}\ ,\ \mathfrak{R}} \qquad \text{falls} \quad s \Longrightarrow_{\mathfrak{R}} r$$

- Löschen einer trivialen Gleichung :

$$\frac{G \cup \{t = t\}\ ,\ \mathfrak{R}}{G\ ,\ \mathfrak{R}}$$

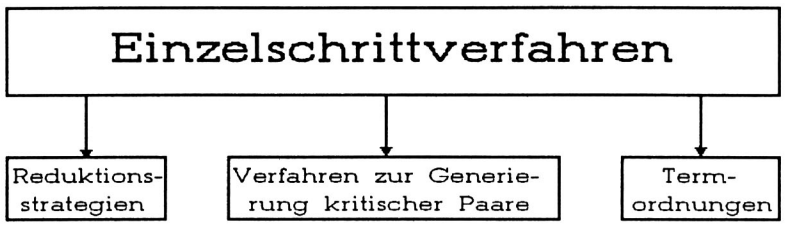
Im COMTES-System ist die Version von Huet implementiert. Eine Beschreibung folgt in Kapitel 3.

2 Aufbau des Systems

2.1 Die Komponenten von COMTES

COMTES steht für 'COMpletion of TErm rewriting Systems'. Der Kern des COMTES-Systems ist ein Vervollständigungsalgorithmus a la Huet, d.h. ein Einzelschrittverfahren. In diesem Verfahren entsteht ein neues Regelsystem aus einem vorhandenen durch Hinzunahme einer einzigen neuen Regel, die aus einem kritischen Paar gewonnen wurde. Im Gegensatz dazu, erzeugt das Gesamtschrittverfahren alle kritischen Paare eines Regelsystems.

COMTES wurde als Experimentalversion konzipiert und verfügt somit über eine Reihe von Parametern. Diese sind im folgenden Diagramm aufgelistet:



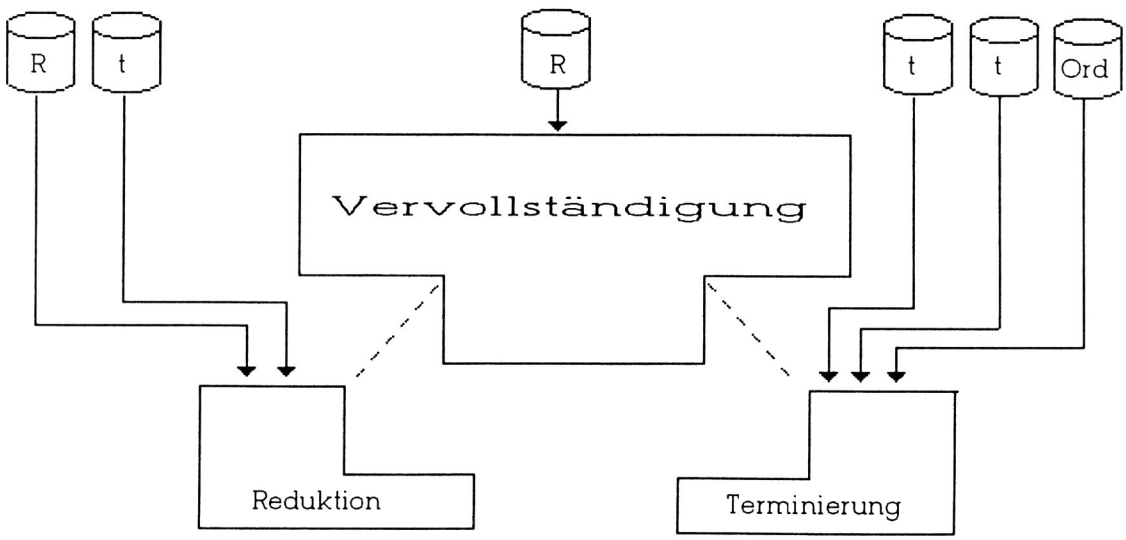
Neun verschiedene Reduktionsstrategien: Will man zu einem gegebenen Term eine Normalform bilden, wird dieser solange reduziert, bis keine Regel mehr anwendbar ist. Eine Reduktionsstrategie legt dabei genau fest, in welcher Reihenfolge die einzelnen Reduktionsschritte ausgeführt werden. In Abhängigkeit dieser Reihenfolge, werden fünf verschiedene (davon eine triviale, nicht deterministische) Strategien unterschieden. Die Effizienz von vier dieser Methoden wird noch durch Hinzunahme von Marken (Kennzeichnung irreduzibler Teilterme) gesteigert.

Sechs Verfahren zur Generierung kritischer Paare: Ein Hauptbestandteil des Vervollständigungsalgorithmus' ist die Bildung kritischer Paare. Es wird versucht, die linke Seite einer Regel mit der linken Seite einer anderen an irgendeiner Stelle zu überlappen. Existiert eine solche Überlappung, so wird das dazugehörige kritische Paar gebildet. Es stehen die klassische Methode (nach dem 'Newman Lemma') und drei weitere Methoden (nach dem 'generalized Newman Lemma') von Kapur, Küchlin und Winkler zur Verfügung. Neben diesen bekannten Kriterien wurden jeweils Weiterentwicklungen des Küchlinschen bzw. Winklerschen Ansatzes realisiert.

Sieben Termordnungen: Es wurden bereits viele Verfahren zur Überprüfung der Terminierung entwickelt. Simplifikationsordnungen repräsentieren einen wesentlichen Teil dieser Methoden. Die Implementierung umfaßt sieben Termordnungen aus den Bereichen der Pfadordnungen, Dekompositionsordnungen und Polynomordnungen.

Weitere Parameter: Die Grundversion des Knuth-Bendix Algorithmus' kann verbessert werden, indem das Regelsystem immer so klein wie möglich gehalten wird. Das Regelsystem soll aus einer minimalen Anzahl von Regeln bestehen und die linke und rechte Seite jeder Regel soll bzgl. der Regelmenge irreduzibel sein. Ein Regelsystem mit dieser Eigenschaft heißt interreduziert (normiert). Der Prozeß der Interreduktion kann optional dazugeschaltet werden. Ist er zugeschaltet, so sind die beiden Seiten aller Regeln irreduzibel. Dieses Phänomen wird ausgenutzt um alle Teilterme der beiden Seiten jeder Regel zu markieren. Bildet man ein kritisches Paar zwischen zwei Regeln eines interreduzierten Regelsystems, so kann dieses nur an den nicht markierten Stellen reduzierbar sein. Diese Einschränkung der Testmenge der zu reduzierenden Terme (Markierung kritischer Paare) wird bei eingeschalteter Interreduktion automatisch durchgeführt.
Wenn ein kritisches Paar gebildet wurde, so werden Normalformen der beiden Seiten bestimmt und diese mit der vorgegebenen Ordnung verglichen. Das sogenannte Postponing läßt die Möglichkeit zu, unvergleichbare Paare zunächst zurückzustellen und erst zu einem späteren Zeitpunkt wieder zu verarbeiten. Dieses Konzept kann somit als Verfeinerung der Ordnung angesehen werden. Es vermeidet in manchen Fällen den Abbruch des Vervollständigungsalgorithmus'.
Ein weiteres Hilfsmittel zur Vermeidung des Abbruchs ist die Verwendung neuer Operatoren. Sind zwei Terme t_1 und t_2 eines kritischen Paares unvergleichbar, so werden sie auf einen Term t abgebildet. D.h., die Regeln $t_1 \Rightarrow t$ und $t_2 \Rightarrow t$ werden aufgenommen. Der Term t wird, mittels eines neuen Funktionssymbols, so konstruiert, daß er mit t_1 und t_2 vergleichbar ist.

Eine effektive Ausnutzung aller vorgestellten Verfahren legt nahe, COMTES nicht nur zur Vervollständigung zu benutzen. COMTES liefert außerdem auch unabhängige Werkzeuge zum Vergleich und zur Reduktion von Termen. Das gesamte COMTES-System hat somit folgende Struktur:



2.2 Die Eingabe von COMTES

Der Vervollständigungsalgorithmus transformiert ein beliebiges Regelsystem in ein eindeutig terminierendes. Folglich besteht die Eingabe für das COMTES-System aus einem Termersetzungssystem. Um die Korrektheit der Syntax dieses Systems überprüfen zu können, müssen weitere Informationen vom Benutzer zur Verfügung gestellt werden. COMTES verlangt als Eingabe eine Spezifikation. Eine Spezifikation ist ein Paar (sig, G) wobei G eine Menge von Gleichungen und sig eine Signatur bezeichnet. Ein Tripel (S, $\mathfrak{F}$, τ) heißt Signatur, falls S eine Menge von Sorten, $\mathfrak{F}$ die Menge der zugeordneten Funktionssymbole und τ eine Stelligkeitsfunktion ist. τ gibt die Stelligkeit sowie den Definitions- und Wertebereich der Funktionssymbole an, d.h. $\tau : \mathfrak{F} \rightarrow S^+$. $\tau(f) = s_1 \ldots s_n s$ genau dann, wenn $f \in \mathfrak{F}$ n Eingabeelemente der Sorten $s_1,\ldots,s_n$ und als Ausgabe ein Element der Sorte s besitzt.

Anstatt eines Gleichungssystems G benötigt COMTES ein Regelsystem R. Dadurch ist es dem Benutzer möglich, direkt gewünschte Richtungen von Gleichungen festzulegen. Diese Art einer Spezifikation muß in Form einer Datei vorliegen (die Namen der Spezifikation und der Datei müssen übereinstimmen), die von einem separaten Werkzeug - dem Parser - auf eventuelle Fehler hin durchsucht wird. Der Parser geht von einer speziellen Eingabesprache aus. Er übersetzt eine Spezifikation dieser Sprache in eine interne Darstellung, die dem COMTES-System als Eingabe dient. Die Eingabesprache sieht folgendes Aussehen einer Spezifikation vor:

SPEC <spec>

SORTS <sorts>

OPS <ops>

RULES <rules>

ENDSPEC

SPEC ist der Name der Spezifikation. Dieser muß mit dem Namen (ohne Suffix '.spec') der zugehörigen Datei identisch sein. SORTS gibt alle verwendeten Sorten an. OPS enthält die Namen und Stelligkeit sowie die jeweils benötigten Sorten der Funktionssymbole (Signatur). RULES ist die Menge der Regeln (das Termersetzungssystem). Das Schlüsselwort ENDSPEC schließt eine Spezifikation ab. Danach können Kommentare angegeben werden.

Im folgenden wird die Syntax der Eingabesprache beschrieben:

<spec> := <atom>

<sorts> := <sort> [, <sorts>]
<sort> := <sexpr>

<ops> := <fun-def>*
<fun-def> := <atom> : <sort>* --> <sort>

<rules> := <rule>*
<rule> := <term> --> <term>

<term> := <atom> | <atom> (<term> [, <term>])

Die Ausdrücke <atom> und <sexpr> sind der Programmiersprache LISP entnommen. Die kleinste syntaktische Einheit in LISP wird mit <atom> bezeichnet, d.h. <atom> := <literal> | <nat> | <nat> . <nat>. Atome sind also alle Objekte die keine CONS-Zellen sind. <sexpr> beschreibt einen LISP-Ausdruck (s-expression) der Form <atom> | (<sexpr>*) | (<sexpr>* . <sexpr>).

Die Syntax der eingegebenen Spezifikation wird gemäß den oben angegebenen Richtlinien überprüft. Ferner werden folgende Tests durchgeführt:

- Existiert die im Aufruf angegebene Spezifikation?
- Stimmt der Spezifikationsname (der SPEC-Klausel) mit dem Namen der Datei überein?
- Sind alle Schlüsselwörter syntaktisch korrekt und ist ihre Reihenfolge fehlerlos?
- Wurden verwendete Sorten definiert?
- Ist jeder Operator genau einmal deklariert worden?
- Für jedes, in den Regeln vorkommende, Funktionssymbol f wird folgendes getestet:
  - Ist f in der Signatur definiert? Ist es nicht definiert, so handelt es sich um eine Variable.
  - Stimmt die Anzahl der Argumente von f mit der in der Signatur definierten überein?
  - Stimmen die Eingabesorten der Argumente mit denen in der Signatur angegebenen überein?
  - Stimmen die Sorten der linken und rechten Seite jeder Regel überein (d.h. ist die Ausgabesorte korrekt)?

Wurden alle Tests erfolgreich beendet, d.h. ist kein Fehler aufgetreten, so wird die interne Darstellung der Spezifikation generiert. Dabei wird als wesentliche Idee die LISP-Liste verwendet. Die geparste Spezifikation liegt als *eine* Liste vor. Die Terme werden wie folgt dargestellt:

intern(x)	= x	falls $x \in \mathfrak{V}$
intern(a)	= (a)	falls a eine Konstante ist
intern(f(t_1,...,t_n))	= (f intern(t_1) ... intern(t_n))	sonst

Eine vollständige Spezifikation hat als interne Darstellung folgende Gestalt (die entsprechende externe Darstellung ist auf der linken Seite angegeben):

SPEC specification	((SPEC specification)
SORTS $sort_1$,..., $sort_n$	(SORTS $sort_1$ ... $sort_n$)
OPS f_1: s_1 ... s_m --> s_{m+1}	(OPS (f_1 s_1 ... s_m s_{m+1})
f_2: s_1' ... s_k' --> s_{k+1}'	(f_2 s_1' ... s_k' s_{k+1}')
.	.
.	.
.	.)

RULES t_1 --> t_1'	(RULES (intern(t_1) intern(t_1') (specification 1))
t_2 --> t_2'	(intern(t_2) intern(t_2') (specification 2))
.	.
.	.
.	.))

An einem Beispiel werden die Eingabeanforderungen deutlich gemacht. Das Gleichheitsprädikat über den natürlichen Zahlen wird spezifiziert. Dazu sind die Sorten boolean und natural numbers notwendig, sowie die Operatoren true und false, 0 und s (successor: s(x) = x+1). Die Spezifikationsdatei *nat-equal.spec* enthält die folgenden Angaben:

```
SPEC        nat-equal

SORTS       bool , nat

OPS         true , false :          -->  bool
            0            :          -->  nat
            s            : nat      -->  nat
            eq           : nat nat  -->  bool

RULES       eq(0 , 0)        -->  true
            eq(0 , s(y))     -->  false
            eq(s(x) , 0)     -->  false
            eq(s(x) , s(y))  -->  eq(x , y)

ENDSPEC
```

Der Parser liefert folgende Ausgabe:

```
( (SPEC NAT-EQUAL)
  (SORTS NAT BOOL)
  (OPS    (TRUE BOOL)
          (FALSE BOOL)
          (0 NAT)
          (S NAT NAT)
          (EQ NAT NAT BOOL))
  (RULES  (((EQ (0) (0))      (TRUE))    (NAT-EQUAL 1))
          (((EQ (0) (S Y))    (FALSE))   (NAT-EQUAL 2))
          (((EQ (S X) (0))    (FALSE))   (NAT-EQUAL 3))
          (((EQ (S X) (S Y)) (EQ X Y))  (NAT-EQUAL 4))))
```

Außer dem initialen Regelsystem, das auf Syntaxfehler überprüft wird, muß der Benutzer zu Beginn jedes Vervollständigungsprozesses über Tastatur einige Parameter vorbelegen. Dies sind im einzelnen:

- completion | comparison | reduction :
  Soll ein System vervollständigt werden oder sollen Terme verglichen bzw. reduziert werden ?

- Auswahl der Reduktionsstrategie (siehe Kapitel 5).

- Auswahl der Ordnung, der Vorordnung, des Status' bzw. der Gewichtsfunktion (siehe Kapitel 6).

- Verfahren zur Bildung kritischer Paare (siehe Kapitel 4).

- Wird Interreduktion gewünscht (siehe 7.1) ?

- Sollten unvergleichbare kritische Paare generiert werden, so kann der Benutzer in den Vervollständigungsprozess eingreifen (siehe 7.2 - 7.4).

- Interrupt:
  Der Benutzer hat zu jeder Zeit die Möglichkeit, die laufende Vervollständigung zu unterbrechen, durch Eingabe von 'i <return>'. Er kann dann einige Informationen über die derzeitige Situation der Vervollständigung abrufen (siehe auch 2.3 und Kapitel 8).

2.3 Die Ausgabe von COMTES

Das COMTES-System produziert sowohl während als auch nach Beendigung jedes Vervollständigungsprozesses eine Vielfalt von Ausgabedaten.

Ausgabe auf Bildschirm

Zu Beginn jeden Laufs (nach Angabe der Termordnung und einigen anderen Parametern) wird das initiale Regelsystem ausgegeben:

```
+-----------------------+
|  initial rule system  |
+-----------------------+

(rule 1):   l1 -> r1
    :
```

Die Berechnung kritischer Paare wird durch Ausgabe der aktuellen Regel angezeigt:

```
¤¤¤  computing critical pairs with  ¤¤¤
(rule 1):  l1 -> r1
```

Wurde aus einem kritischen Paar eine Regel erzeugt, so erscheint auf dem Bildschirm folgende Meldung:

+++ new rule +++
rule l, rule 2 $\Longrightarrow$ (rule 4): $l_4 \rightarrow r_4$

wobei genau spezifiziert wird, aus welchen Regeln (rule l und rule 2) die neue Regel (rule 4) entstanden ist.

Interreduktion kann eventuell dazu führen, daß (linke Seiten von) Regeln reduziert und somit aus dem aktuellen Regelsystem entfernt werden (siehe 7.1). Diese Aktion hat die Meldung

--- delete rule ---
(rule l): $l_1 \rightarrow r_1$

zur Folge.

Stoppt der Vervollständigungsprozeß mit Erfolg, so werden das initiale und das kanonische Regelsystem, sowie die vollständigen Daten der verwendeten Ordnung (Vorordnung, Status, Gewichtsfunktion) angegeben:

initial rule system

(rule l): $l_1 \rightarrow r_1$
⋮

canonical rule system

rule l, rule 2 $\Longrightarrow$ (rule 4): $l_4 \rightarrow r_4$
⋮

information

name of the ordering
precedence
status
weight function

Anschließend wird eine Statistik erstellt, die folgende Informationen liefert (mit entsprechenden prozentualen Anteilen):

statistic

Total run time used
Number of rules generated
Number of critical pairs generated

Time spent in normalization
Time spent in unification
Number of unifiers generated
Time spent in reduction
Number of reduction steps
Time spent in ordering

Hinweis: Es steht eine Help-Funktion zur Verfügung, die - wann immer eine Eingabe vom Benutzer zu erfolgen hat - Informationen über die Art der Eingabe bereitstellt.

Ausgabe auf Datei

Alle Ausgaben, die nach Beendigung des Vervollständigungsprozesses über Bildschirm erfolgen, werden auch auf verschiedenen Dateien gespeichert:

Initiales und kanonisches Regelsystem, Informationen über die Termordnung } <name der Spezifikationsdatei>.kb in der Directory comtes

Statistik } statistic/trace

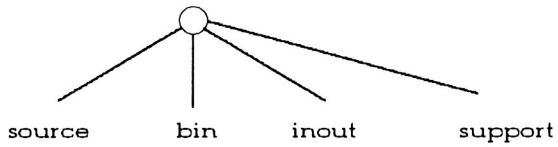
Ferner kann auf Wunsch des Benutzers während des Laufs ein Schnappschuß der aktuellen Situation erfolgen (siehe auch 2.2). Die Daten über

- das aktuelle Regelsystem
- die unvergleichbaren kritischen Paare
- die aktuelle Termordnung

werden in der Datei comtes/<name der Spezifikationsdatei>.out abgelegt.

2.4 Technische Details

COMTES ist in LUCID-COMMON-LISP implementiert und läuft auf Apollo Domain Systemen unter dem Betriebssystem AEGIS (das UNIX sehr ähnlich ist). Da die Möglichkeit besteht, Dateien baumartig anzulegen, haben wir die Eingabedateien, das eigentliche System, sowie die Ausgabe voneinander getrennt. Wir haben folgende Baum-Struktur vorgegeben:

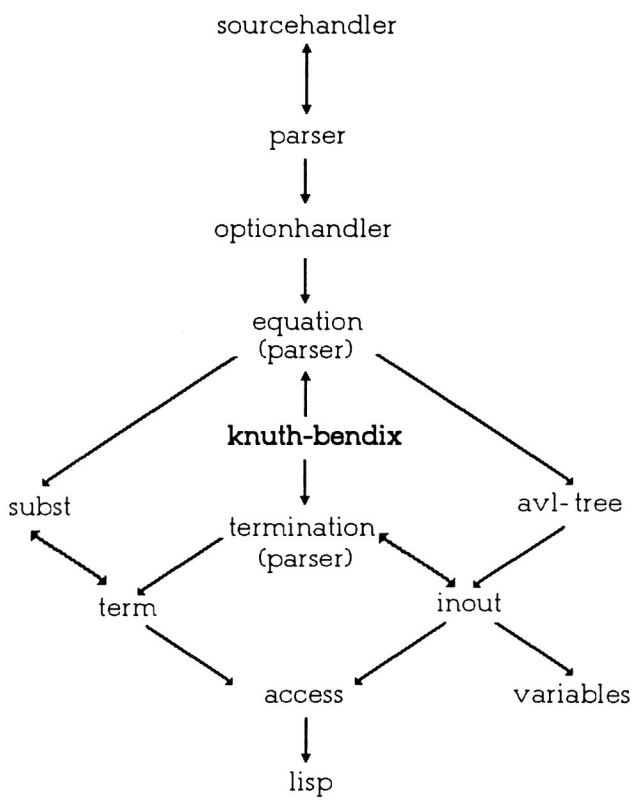


Das System ist modular aufgebaut: die 13 Quelldateien von COMTES befinden sich im Verzeichnis 'source'. Die dazugehörigen Binärcodes wurden in 'bin' abgelegt. Die Liste der Module hat folgendes Aussehen:

Modulname	**Beschreibung**	**Größe**		
		Quellcode		Binärcode
		in Zeilen	in kByte	in kByte
access	Konstruktor- und Selektorfunktionen für eine Reihe von Datenstrukturen, wie z.B. Argumente eines Terms, Generierung einer Regel aus zwei Termen und einem Label	394	8	14
avl-tree	Funktionen zur Generierung und Verwaltung von AVL-Bäumen; sie dienen der Speicherung bereits gebildeter Überlappungen	557	37	25
equation	Funktionen zur Behandlung von Gleichungen (Regeln): Generierung kritischer Paare, Reduktion eines Terms, etc.	1273	71	44
inout	Funktionen, die der Ein- bzw. Ausgabe von Daten dienen. z.B.: Bestimmen der Vorordnung, Ausgabe eines Regelsystems	687	41	40
knuth-bendix	Toplevelmodul, d.h. Funktionen die den Ablauf des Knuth-Bendix Algorithmus' steuern	758	46	31
lisp	elementare Lisp-Funktionen; stellt eine Erweiterung der 'Lisp-primitives' dar	829	24	22
option-handler	Funktionen zur Bereitstellung von ERROR-Optionen (Parser)	238	13	14

parser	Toplevelmodul des Parsers; überprüft die Syntax der Eingabe	519	29	25
source-handler	Funktionen zum Einlesen von Zeichenketten (Parser)	80	5	5
subst	Funktionen die Substitutionen verwenden, wie z.B. Matching und Unifikation	393	18	13
term	Funktionen auf Termen, wie z.B. Berechnung von Stellen und ihre Sortierung bzgl. bestimmter Reduktionsstrategien	572	20	16
termination	Funktionen, die einen Vergleich zweier Terme bzgl. verschiedenen Ordnungsmethoden ausführen	2199	101	83
variables	alle globalen Variablen	88	2	3

Die Modul-Struktur, d.h. die Abhängigkeiten der Dateien untereinander, ist durch folgendes Diagramm beschrieben ($modul_1 \Rightarrow modul_2$ bedeutet, daß $modul_1$ Funktionen aus $modul_2$ benötigt):



Es existiert zu jedem Modul eine Dokumentation seiner Funktionen sowie eine Testdatei (im Verzeichnis support/doc bzw. support/testfiles enthalten).

Abschließend folgt eine Übersicht über technische Daten von COMTES:

Systemname:	COMTES 'Completion of term rewriting systems'
Programmiersprache:	Lucid-Common Lisp
Rechner:	Apollo Domain System unter AEGIS
Größe:	13 Module (entspricht 398 Funktionen, 68 Makros und 33 globalen Variablen), 402484 Bytes Quelltext (entspricht 8587 Zeilen bzw. 332632 Bytes Binärcode)
Designer:	Werner Engeln, Doerthe Paul, Harald Sohns, Inger Sonntag, Elvira Steinbach, Joachim Steinbach

3 Einzelschrittverfahren

Der im COMTES-System implementierte Vervollständigungsalgorithmus basiert auf der Idee des Einzelschrittverfahrens. Dieses Verfahren ist eine günstige Alternative zum Gesamtschrittverfahren.
Das sogenannte Gesamtschrittverfahren erzeugt *alle* kritischen Paare eines Regelsystems und vereinigt alle daraus entstehenden Regeln mit dem vorhandenen Regelsystem. Um aus der Menge der kritischen Paare neue Regeln zu bilden, werden diese bezüglich der Regeln des aktuellen Regelsystems auf Normalform gebracht. Dabei kann es vorkommen, daß aus einem kritischen Paar eine Regel gebildet wird, mit der weitere kritische Paare so reduziert werden könnten, daß sie konfluent sind. Daher ist es oft sinnvoll, eine neuerzeugte Regel direkt in das Regelsystem aufzunehmen, um sie zur Reduktion verwenden zu können. Diese Idee liegt dem Einzelschrittverfahren zugrunde.

Das Einzelschrittverfahren generiert nun eine Folge von Regelsystemen, die im günstigsten Fall zu einem konfluenten Regelsystem führt. Der Übergang von dem aktuellen Regelsystem der Folge zum neuen System erfolgt durch Hinzunahme neuer Regeln. Diese neuen Regeln entstehen aus den vergleichbaren kritischen Paaren, welche zu den Überlappungen *einer einzigen* Regel mit allen anderen des aktuellen Systems gehören. Unvergleichbare kritische Paare erzwingen i.a. den Abbruch des Algorithmus' (siehe Kapitel 7). Entstehen dagegen keine unvergleichbaren Paare, so stoppt der Algorithmus nur, falls ein endliches, eindeutig terminierendes Regelsystem gefunden wird. Interreduktion beschleunigt den Ablauf des Programms, da dies i.a. die Anzahl der zu überlappenden Regeln verringert (siehe 7.1).

Bei diesem Algorithmus kann es zu folgender Situation kommen: Es existiert ein kritisches Paar, aus dem eine Regel erzeugt werden kann, mit deren Hilfe alle weiteren kritischen Paare konfluent werden. Dieses kritische Paar wird aber nicht gebildet, weil immer andere neue Regeln erzeugt werden, mit denen zuerst kritische Paare gebildet werden. D.h., der Algorithmus läuft unendlich, obwohl er terminieren könnte. Die Fairneßbedingung gewährleistet, daß jedes mögliche kritische Paar in endlicher Zeit betrachtet wird. Um diese Fairneßbedingung zu garantieren, müssen die Regeln mit zusätzlichen Informationen versehen werden, die darüber Auskunft geben, welche Regeln mit welchen Regeln bereits überlappt wurden. Hier bietet sich eine Markierung der Regeln an[1], mit denen bereits kritische Paare gebildet wurden.

Der Algorithmus[2] erhält als Eingabe das - nach der Länge der Regeln - sortierte und unmarkierte, initiale Regelsystem $\mathfrak{R}$. Er wählt aus dem aktuellen Regelsystem $\mathfrak{R}$ die kürzeste (hinsichtlich der Anzahl der in der Regel enthaltenen Funktionssymbole und Variablen),

---

[1] vgl. EQ-REDUCE-EX-INNERMOST, Kapitel 9
[2] vgl. KB-SINGLE, Kapitel 9

Algorithmus:

```
MR := ∅
A: WHILE es gibt eine unmarkierte Regel in ℜ
   DO    Rl := kürzeste, unmarkierte Regel aus ℜ
         markiere Rl
         MR := MR ∪ Rl

B:       WHILE es gibt eine Regel in MR, die noch nicht mit Rl überlappt wurde
         DO    R2 := kürzeste, unbearbeitete Regel aus MR
               FOR alle u ∈ O'(linke Seite von Rl)
               DO  kritische-Paare := KRIT-PAAR(Rl , R2 , u)
                   IF      kritische-Paare ≠ ∅
                   THEN ℜ  := NORMALIZE(ℜ , kritische-Paare)
                        MR := MR ∩ ℜ
                        IF Rl ∉ ℜ
                        THEN GOTO A
                        ELSE IF R2 ∉ ℜ
                             THEN GOTO B
               IF    Rl ≠ R2
               THEN FOR alle v ∈ O'(linke Seite von R2)
                    DO  kritische-Paare := KRIT-PAAR(R2 , Rl , v)
                        IF kritische-Paare ≠ ∅
                        THEN ℜ  := NORMALIZE(ℜ , kritische-Paare)
                             MR := MR ∩ ℜ
                             IF Rl ∉ ℜ
                             THEN GOTO A
                             ELSE IF R2 ∉ ℜ
                                  THEN GOTO B
```

unmarkierte Regel (Rl) aus und markiert diese (MR ist die Menge der markierten Regeln). Es werden alle kritischen Paare zwischen Rl und den bereits markierten Regeln aus $\mathfrak{R}$ gebildet. Sobald ein kritisches Paar erzeugt wurde (KRIT-PAAR), wird es an die Prozedur NORMALIZE weitergereicht, die es in $\mathfrak{R}$ einreiht oder das gesamte Verfahren abbricht. Der Algorithmus stoppt mit einem konfluenten Regelsystem, falls es keine unmarkierte Regel mehr in $\mathfrak{R}$ gibt.
O'(t) bezeichnet die Menge der Stellen des Terms t ohne die Stellen der Variablen: $O'(t) = O(t) \setminus \{u \in O(t) \mid t/u \in \mathfrak{V}\}$.
Die Prozedur NORMALIZE[1] wählt jeweils ein Paar (t_1,t_2) aus der Menge "kritische-Paare". Die Terme t_1 und t_2 werden zu Normalformen reduziert. Sind diese unvergleichbar, so bricht der Algorithmus ab (siehe Kapitel 7). Für den Fall, daß sie vergleichbar sind, wird eine neue Regel erzeugt und in das aktuelle Regelsystem $\mathfrak{R}$ aufgenommen. Zusätzlich werden alle rechten Seiten der Regeln in $\mathfrak{R} \setminus \{\text{neue Regel}\}$ bezüglich der neuen Regel auf Normalform gebracht (siehe auch 7.1).

---

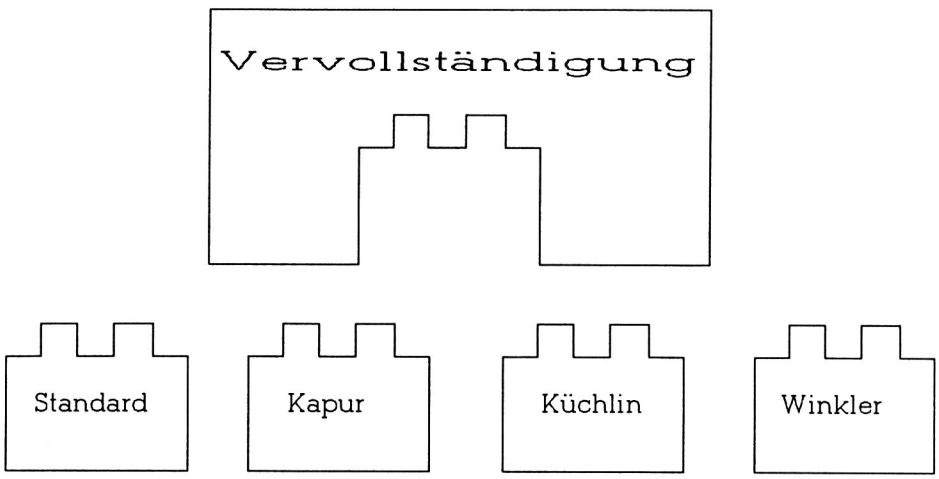
[1] vgl. KB-NORMALIZE bzw. KB-NORMALIZE-OHNE-INTERRED, Kapitel 9

Die Prozedur KRIT-PAAR[1] versucht die linke Seite der längeren Regel (zweiter Parameter) mit der linken Seite der kürzeren Regel (erster Parameter) an einer Stelle (dritter Parameter) zu überlappen. Existiert eine solche Überlappung, so wird das dazugehörige kritische Paar gebildet und an den Algorithmus des Einzelschrittverfahrens zurückgeliefert.

[1] vgl. EQ-CP-ONE-LIST, Kapitel 9

4 Bildung kritischer Paare

Es gibt zwei verschiedene Ansätze, um die Konfluenz eines Regelsystems zu garantieren. Der eine Ansatz basiert auf dem "Newman Lemma" von 1942, der zweite auf dem "Generalized Newman Lemma" von Buchberger aus dem Jahr 1983. Die implementierten Verfahren von Winkler, Küchlin und Kapur wurden aus dem "Generalized Newman Lemma" hergeleitet, während die Standardstrategie das "Newman Lemma" als Grundlage hat.
Diese Strategien wurden unabhängig voneinander implementiert und mit der gleichen Schnittstelle versehen. Diese Vorgehensweise garantiert die problemlose, wechselweise Verwendung der einzelnen Verfahren (die Prozedur KRIT-PAAR aus Kapitel 3 wird instanziiert durch das gewünschte Kriterium):



4.1 Klassische Methode

Im weiteren Verlauf dieses Kapitels wird die Variablendisjunktheit der linken Seiten zweier Regeln vorausgesetzt.
Die Definition eines **kritischen Paares** bildet die Grundlage aller nachfolgend vorgestellten Verfahren:

Sei $\mathfrak{R}$ ein Regelsystem, $l_1 \to r_1$ und $l_2 \to r_2 \in \mathfrak{R}$.
Gibt es eine Überlappung

$$(l_1 \to r_2,\ \sigma,\ u,\ l_2 \to r_2)$$

(d.h. $\sigma(l_1/u) = \sigma(l_2)$), so heißt

$$(\ \sigma(l_2)[u \leftarrow \sigma(r_1)]\ ,\ \sigma(r_2)\ )$$

ein kritisches Paar zu den beiden Regeln.

Die klassische Methode zur Bildung eines kritischen Paares[1] verwendet ausschließlich oben angegebene Definition:

```
PROC  KRIT-CLASSICAL(l1 → r1, l2 → r2, u)
IF    es gibt eine Überlappung (l1 → r1, σ, u, l2 → r2)
THEN  t1 := σ(r2)
      t2 := σ(l2)[u ← σ(r1)]
      IF t1 = t2
      THEN  RETURN ∅
      ELSE  RETURN (t1,t2)
ELSE  RETURN ∅
```

4.2 Verfahren von Kapur

Das von Kapur, Musser und Narendran entwickelte Verfahren[2] liest direkt von einer Überlappung ab, ob das daraus entstehende kritische Paar behandelt werden muß. Ein noethersches Regelsystem ist genau dann konfluent, wenn alle kritischen Paare vom Typ prime konfluent sind.

Eine Überlappung $(l_1 \to r_1, \sigma, u, l_2 \to r_2)$ heißt prime, falls $\sigma(l_2)/u$ keinen echten, reduziblen Teilterm besitzt.

Kapurs Lemma impliziert, daß nur die kritischen Paare zu prime Überlappungen im Verlauf des Vervollständigungsalgorithmus' gebildet und bearbeitet werden müssen. Daher reicht in dem in 4.1 beschriebenen Algorithmus eine Erweiterung um einen Test auf prime aus. Es ergibt sich die folgende Prozedur:

```
PROC  KRIT-KAPUR(l1 → r1, l2 → r2, u)
IF  es gibt eine Überlappung super = (l1 → r1, σ, u, l2 → r2)
THEN  IF  super ist vom Typ prime
      THEN  t1 := σ(r2)
            t2 := σ(l2)[u ← σ(r1)]
            IF t1 = t2
            THEN  RETURN ∅
            ELSE  RETURN (t1,t2)
      ELSE  RETURN ∅
ELSE  RETURN ∅
```

Der Test, ob eine Überlappung vom Typ prime ist, wird durch den (negativen) Test auf composite (= nicht prime) ersetzt: Eine Überlappung $(l_1 \to r_1, \sigma, u, l_2 \to r_2)$ wird als composite bezeichnet, falls es eine Regel $l_i \to r_i$ gibt, die auf einen echten Teilterm von $\sigma(l_2)/u$ anwendbar ist.

---

[1] vgl. EQ-CP-ONE-LIST, Kapitel 9

[2] vgl. EQ-CP-KAPUR, Kapitel 9

Wie bereits erwähnt, sieht COMTES eine optionale Zuschaltung des Interreduktionsprozesses vor. Dies hat zur Folge, daß i.a. bedeutend weniger Überlappungen (und somit kritische Paare) erzeugt werden. Unter den wegfallenden Superpositionen befinden sich speziell solche vom Typ composite. Die Anzahl der Überlappungen, bei denen das Kriterium von Kapur greift, wird also eingeschränkt. Es gibt jedoch auch Fälle, in denen das Verfahren von Kapur - trotz Interreduktion - anwendbar ist.

Beispiel: Sei $\mathfrak{R}$ folgendes noethersches Regelsystem:

f(a,b)	→	c	(R1)
h(y,f(a,y))	→	a	(R2)
g(b,h(b,x))	→	i(b)	(R3)

Die Überlappung zwischen R2 und R3 hat folgende Gestalt:

$(l_2 \rightarrow r_2,\ \{\sigma(y) = b,\ \sigma(x) = f(a,b)\},\ 2,\ l_3 \rightarrow r_3)$,
wobei $\sigma(l_3) = g(b,h(b,f(a,b)))$.

Jedoch läßt sich $\sigma(l_3)/22$ mit R1 reduzieren, d.h. diese Überlappung ist composite und braucht nicht weiter betrachtet zu werden. Bei der Vervollständigung entstehen noch zwei weitere Überlappungen (zwischen R1 und R2, R3 und der neuen Regel), die nicht composite sind.

4.3 Verfahren von Winkler

Das von Winkler entwickelte Kriterium[1] bietet ein Mittel, mit dessen Hilfe anhand der bereits bearbeiteten Überlappungen entschieden werden kann, ob das kritische Paar der aktuellen Überlappung betrachtet werden muß. Im Gegensatz zum Kriterium von Kapur (4.2) ist hierbei eine Kennzeichnung bzw. Sammlung der bis zum aktuellen Zeitpunkt abgearbeiteten Superpositionen erforderlich. Die dazu notwendige Erweiterung des Standardalgorithmus' hat daher einen größeren Verwaltungsaufwand zur Folge.
Zur Beschreibung des Kriteriums wird zunächst der Begriff "Winkler-Überlappung" (oder Überlappung vom Typ "Winkler") eingeführt:

$(l_1 \rightarrow r_1,\ \sigma,\ u,\ l_2 \rightarrow r_2)$ heißt "Winkler-Überlappung", falls es Positionen v, w und eine Regel $l_3 \rightarrow r_3$ gibt mit Überlappungen $(l_1 \rightarrow r_1,\ \sigma_1,\ v,\ l_3 \rightarrow r_3)$ und $(l_3 \rightarrow r_3,\ \sigma_2,\ w,\ l_2 \rightarrow r_2)$, sodaß gilt:

- $u = w.v$
- $\sigma_1(l_3) \leq \sigma(l_2)/w$
- $\sigma_2(l_2) \leq \sigma(l_2)$
- $\sigma_1(l_2)$ und $\sigma(l_2)$ wurden bereits bearbeitet.

---

[1] vgl. EQ-CP-WINKLER, Kapitel 9

Die Beziehung $s \leq t$ gilt, falls es eine Substitution σ gibt, sodaß $\sigma(s) = t$ (Subsumptionsordnung).
Mit Hilfe obiger Definition ergibt sich das Lemma: Ein noethersches Regelsystem ist genau dann konfluent, wenn alle kritischen Paare, deren zugehörige Überlappungen nicht vom Typ "Winkler" sind, konfluent sind.
Diesem Lemma entsprechend, wird die Standardprozedur aus 4.1 um einen Test erweitert, der angibt, ob eine Überlappung vom Typ "Winkler" ist. Ist dies der Fall, so liefert die Prozedur kein kritisches Paar zurück, da es nicht betrachtet werden muß.

```
PROC  KRIT-WINKLER(l1 → r1, l2 → r2, u)
IF    es gibt eine Überlappung super = (l1 → r1, σ, u, l2 → r2)
THEN  t1 := σ(r2)
      t2 := σ(l2)[u ← σ(r1)]
      IF   t1 = t2
      THEN RETURN ∅
      ELSE IF   TEST-WINKLER(super) = true
           THEN RETURN ∅
           ELSE RETURN (t1,t2)
      ÜBERLAPP-BEARBEITET(super)
ELSE  RETURN ∅
```

Die Prozedur ÜBERLAPP-BEARBEITET sorgt dafür, daß die Überlappung super als bearbeitet vermerkt wird.
Die Prozedur TEST-WINKLER liefert true, falls die Überlappung super vom Typ "Winkler" ist, d.h. das dazugehörige kritische Paar nicht bearbeitet werden muß. Ansonsten liefert sie false. Die Methode der Prozedur lehnt sich direkt an die Definition von "Winkler-Überlappung" an:
Zunächst wird zusätzlich zu den beiden Eingaberegeln eine dritte Regel $l_3 \rightarrow r_3$ ausgewählt, die zur Bildung der Überlappungen $(l_1 \rightarrow r_1, \sigma_1, v, l_3 \rightarrow r_3)$ und $(l_3 \rightarrow r_3, \sigma_2, w, l_2 \rightarrow r_2)$ herangezogen wird. Regel $l_3 \rightarrow r_3$ wird der Menge der bereits überlappten Regeln (MR, siehe Kapitel 3) des bisher generierten Regelsystems entnommen. Für jede Position w wird für den Fall, daß w Anfangswort von u ist, der Versuch unternommen, die zweite Überlappung $(l_3 \rightarrow r_3, \sigma_2, w, l_2 \rightarrow r_2)$ zu bilden. Existiert eine solche Überlappung und gibt es auch die Überlappung $(l_1 \rightarrow r_1, \sigma_1, v, l_3 \rightarrow r_3)$ mit $u = w.v$, so bleibt noch zu testen, ob die Bedingungen $\sigma_1(l_3) \leq \sigma(l_2)/w$ und $\sigma_2(l_2) \leq \sigma(l_2)$ erfüllt werden.

Beispiel: Sei $\mathfrak{R}$ ein noethersches Regelsystem mit

g(h(x))	→	b	(R1)
f(g(h(y)))	→	c	(R2)
h(i(i(i(a))))	→	a	(R3)

Zwischen R1 und R2 bzw. R3 und R1 gibt es Überlappungen, die nicht vom Typ Winkler sind. Es entstehen die Regeln

$f(b) \rightarrow c$ (R4)
$g(a) \rightarrow b$ (R5)

Die Überlappung $(l_3 \rightarrow r_3, \{\sigma_1(y) = i(i(i(a)))\}, 11, l_2 \rightarrow r_2)$ ist vom Typ Winkler: Es gibt zwei Überlappungen

$(l_1 \rightarrow r_1, \{\sigma(y) = x\}, 1, l_2 \rightarrow r_2)$ und
$(l_3 \rightarrow r_3, \{\sigma_2(x) = i(i(i(a)))\}, 1, l_1 \rightarrow r_1)$

mit
- $u = 11, v = 1, w = 1 \quad \rightsquigarrow \quad u = v.w$
- $\sigma_2(l_1) = g(h(i(i(i(a))))), \sigma_1(l_2)/w = g(h(i(i(i(a)))))) \quad \rightsquigarrow \quad \sigma_2(l_1) \leq \sigma_1(l_2)/w$
- $\sigma(l_2) = f(g(h(x))), \sigma_1(l_2) = f(g(h(i(i(i(a)))))) \quad \rightsquigarrow \quad \sigma(l_2) \leq \sigma_1(l_2)$
- $\sigma(l_2)$ (führte zu R4) und $\sigma_2(l_1)$ (führte zu R5) sind bereits bearbeitet.

Außer dem vorgestellten Algorithmus, der auf dem Verfahren von Winkler basiert, wurde auch eine erweiterte Version[1] implementiert. Der zweite vorzustellende Algorithmus baut auf dem ersten auf. Die grundlegende Idee läßt sich wie folgt veranschaulichen:

Betrachten wir eine Überlappung s, für die alle Bedingungen des Winkler-Kriteriums bis auf die Frage nach der Bearbeitung der beiden zusätzlichen Superpositionen zutreffen. Mit Hilfe der benötigten dritten Regel ergibt sich Bild 1.

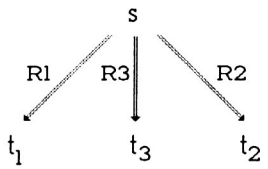


Bild 1

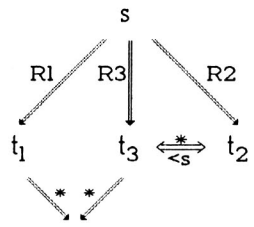


Bild 2

Die Überlappung zwischen R2 und R3 sei bearbeitet. Daraus folgt, daß t_3, t_2 unterhalb von s verbunden sind. Eine Aussage über die Superposition zwischen R1 und R3 ist noch nicht getroffen worden. Erzwingt man nun die Konfluenz von t_1, t_3 (Bild 2) durch Hinzunahme einer neuen Regel, die aus dem kritischen Paar zwischen R1 und R3 gewonnen wird, so braucht nach Winklers Lemma die Überlappung s nicht mehr weiter untersucht werden. In einer solchen Situation wird in dieser erweiterten Fassung des originalen Algorithmus' sofort das kritische Paar zu der Überlappung zwischen R1 und R3 gebildet. Analog wird für den Fall, daß die Superposition zwischen R1 und R3 bearbeitet ist und die Überlappung zwischen R3 und R2 unbetrachtet vorliegt, sofort das kritische Paar zu der Überlappung von R3 und R2 erzeugt.
Die Frage, ob Überlappungen bereits bearbeitet sind, ist sehr wesentlich und leider auch zeitintensiv. Es müssen nämlich Mittel zur Verfügung gestellt werden, mit deren Hilfe sich

---

[1] vgl. EQ-CP-WINKLER-ERW, Kapitel 9

die im Verlauf des Algorithmus' gebildeten Superpositionen gemerkt werden können. In [So88] werden zwei Möglichkeiten beschrieben. Die erste, verwirklichte Lösung baut auf AVL-Bäumen auf, während die zweite ihre Information direkt aus dem Stand der Rechnung abliest.

Der erweiterte Algorithmus ist nicht so stark von der Reihenfolge der zu bearbeitenden Superpositionen abhängig wie der Original-Algorithmus. Details zu dieser Strategie entnehme man bitte [So88].

4.4 Verfahren von Küchlin

Dem von Küchlin entwickelten Kriterium[1] liegt prinzipiell dieselbe Idee zugrunde wie dem Kriterium von Winkler. Mit Hilfe der Menge der bereits betrachteten Überlappungen wird entschieden, ob die aktuelle Superposition bearbeitet werden muß. Im Gegensatz zu Winkler stellt Küchlins Verfahren einen verallgemeinerten Ansatz dar. Es läßt sich folgendes Lemma formulieren: Ein noethersches Regelsystem ist genau dann konfluent, wenn alle kritischen Paare, deren zugehörige Überlappungen nicht vom Typ "Küchlin" sind, konfluent sind.

Eine Überlappung $(l_1 \to r_1, \sigma, u, l_2 \to r_2)$ heißt "Küchlin-Überlappung" (oder vom Typ "Küchlin"), falls es eine Position $w \in O(\sigma(l_2))$, eine Regel $l_3 \to r_3$ und eine Substitution τ mit $\tau(l_3) = \sigma(l_2)/w$ gibt, so daß entweder a) oder b) gilt:

a) $w \in O(l_2)$ und die Überlappung $(l_3 \to r_3, \sigma_1, w, l_2 \to r_2)$ ist bereits bearbeitet worden und es gilt eine der folgenden fünf Bedingungen:

   a.1) $u \mid w$, d.h. u (bzw. w) ist kein Teilwort von w (bzw. u)
   a.2) $\exists p_1$: $w = u.p_1$, $p_1 \in O(l_1)$ und $(l_3 \to r_3, \sigma_2, p_1, l_1 \to r_1)$ wurde bereits bearbeitet.
   a.3) $\exists p_1$: $w = u.p_1$, $p_1 \notin O(l_1)$.
   a.4) $\exists p_1$: $u = w.p_1$, $p_1 \in O(l_3)$ und $(l_1 \to r_1, \sigma_2, p_1, l_3 \to r_3)$ wurde bereits bearbeitet.
   a.5) $\exists p_1$: $u = w.p_1$, $p_1 \notin O(l_3)$.

b) $w \notin O(l_2)$ und es gilt eine der drei folgenden Bedingungen:
   b.1) $u \mid w$
   b.2) $\exists p_1$: $w = u.p_1$, $p_1 \in O(l_1)$ und $(l_3 \to r_3, \sigma_2, p_1, l_1 \to r_1)$ wurde bereits bearbeitet.
   b.3) $\exists p_1$: $w = u.p_1$, $p_1 \notin O(l_1)$.

Entsprechend Küchlins Lemma wird die Prozedur KRIT-PAAR um einen Test erweitert, der angibt, ob eine Überlappung vom Typ "Küchlin" ist. Ist dies der Fall, so liefert die Prozedur kein kritisches Paar zurück, da es nicht betrachtet werden muß.

---

[1] vgl. EQ-CP-KÜCHLIN, Kapitel 9

```
PROC  KRIT-KÜCHLIN(l1 → r1, l2 → r2, u)
IF    es gibt eine Überlappung super = (l1 → r1, σ, u, l2 → r2)
THEN  t1 := σ(r2)
      t2 := σ(l2)[u ← σ(r1)]
      IF   t1 = t2
      THEN RETURN ∅
      ELSE IF   TEST-KÜCHLIN(super) = true
           THEN RETURN ∅
           ELSE RETURN (t1,t2)
      ÜBERLAPP-BEARBEITET(super)
ELSE  RETURN ∅
```

Die Prozedur TEST-KÜCHLIN liefert true, falls die Überlappung super vom Typ "Küchlin" ist, d.h. das dazugehörige kritische Paar nicht bearbeitet werden muß. Ansonsten liefert sie false.
Die Methode der Prozedur TEST-KÜCHLIN basiert auf der Definition von "Küchlin-Überlappung". Zunächst wird zusätzlich zu den Eingaberegeln eine dritte Regel (R3) gesucht, die auf die aktuelle Überlappung super anwendbar ist. R3 wird aus der Menge MR gewählt, da im Verlauf der darauf folgenden Testserie die Frage gestellt wird, ob bestimmte Superpositionen zwischen R3 und $l_2 \to r_2$ (bzw. R3 und $l_1 \to r_1$) bereits bearbeitet wurden. Dies ist nie der Fall, falls R3 noch gänzlich unbetrachtet vorliegt.
Solange die bool'sche Ausgabevariable test der Prozedur den Wert false hat, wird für jede Position reduk-pos des Terms $\sigma(l_2)$ geprüft, ob R3 an dieser Stelle auf den Term anwendbar ist. Ist dies der Fall, so nimmt die Variable test den Wert true an, falls entweder reduk-pos nicht in der Menge $O(l_2)$ enthalten ist oder die Superposition $(l_3 \to r_3, \sigma_1, \text{reduk-pos}, l_2 \to r_2)$ bereits bearbeitet worden ist - und zusätzlich eine der drei Bedingungen erfüllt ist:

- u | reduk-pos
- $\exists p$: reduk-pos = u.p und entweder $p \notin O(l_3)$ oder die Überlappung $(l_3 \to r_3, \sigma_1, p, l_1 \to r_1)$ wurde bereits bearbeitet
- $\exists p$: u = reduk-pos.p und entweder $p \notin O(l_1)$ oder die Überlappung $(l_1 \to r_1, \sigma_1, p, l_3 \to r_3)$ wurde bereits bearbeitet

Ist reduk-pos in der Menge $O(l_2)$ enthalten, so müssen zusätzlich die Bedingungen a.4) und a.5) (auf Seite 26) überprüft werden. Behält die Variable test den Wert false und sind alle Positionen von $\sigma(l_2)$ betrachtet worden, beginnt die Testserie mit einer neuen Regel $R3 \in MR$. Sind alle Regeln aus MR gewählt worden, ohne daß test den Wert true erhalten hat, so liefert die Prozedur TEST-KÜCHLIN das Ergebnis false. Hat die Variable test dagegen einmal den Wert true angenommen, so stoppt die Prozedur sofort mit dem Wert true.

Die Prozedur ÜBERLAPP-BEARBEITET sorgt dafür, daß die Überlappung super als bearbeitet vermerkt wird.

Beispiel: Das Verfahren von Küchlin greift immer, wenn es sich um eine Superposition vom Typ Winkler handelt. Für das Regelsystem des Beispiels in 4.3 werden exakt die gleichen kritischen Paare generiert und bearbeitet.
Dieses Beispiel soll nur andeuten, daß die Verfahren von Küchlin und Winkler sehr ähnlich sind. Die Küchlin'sche Methode ist jedoch i.a. mächtiger.

Abschließend wird eine Erweiterung des Küchlin'schen Verfahrens vorgestellt, die ebenfalls implementiert wurde. Der Algorithmus[1] baut ebenfalls auf der Definition von "Küchlin-Überlappung" auf. Eine Abweichung vom Original-Algorithmus ergibt sich für folgende Fälle:

Sei s eine Superposition zwischen Rl und R2. Es gibt eine dritte Regel (R3), die auf s anwendbar ist. Dann liegt die in Bild 1 beschriebene Situation vor.

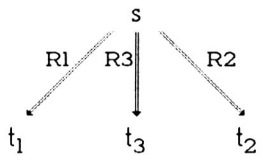


Bild 1

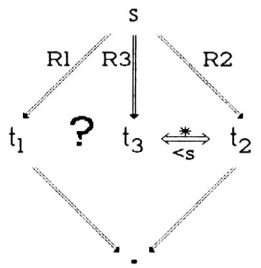


Bild 2

s sei keine "Küchlin-Überlappung", da die entsprechende Überlappung zwischen Rl und R3 nicht bearbeitet wurde. In diesem Fall bricht der Original-Algorithmus ab und erzwingt die Konfluenz von t_1 und t_2 (Bild 2). Sind t_2 und t_3 nicht verbunden unterhalb von s, so erfolgt - im Unterschied zu dem erweiterten Algorithmus von Winkler - die Bearbeitung der Überlappung zwischen R3 und R2 analog zu der von R3 und Rl.

Im Gegensatz dazu garantiert diese Erweiterung des Algorithmus' die *direkte* Bearbeitung von der Überlappung zwischen Rl und R3 (s_l_3). Auf diese Weise wird s zu einer Superposition vom Typ "Küchlin" und muß nach Küchlins Lemma nicht mehr betrachtet werden. Auf die Frage wie s_l_3 weiterverarbeitet wird, muß jetzt noch eingegangen werden. Zunächst wird versucht, s_l_3 als "Küchlin-Überlappung" nachzuweisen. Ist dies möglich, so handelt es sich auch bei s um eine Superposition vom Typ "Küchlin" (Bild 3).

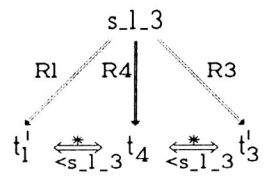


Bild 3

[1] vgl. EQ-CP-KÜCHLIN-ERW, Kapitel 9

Handelt es sich bei s_1_3 nicht um eine "Küchlin-Überlappung", da keine dritte Regel (R4) existiert, die auf s_1_3 anwendbar ist, so wird die Konfluenz von t_1' und t_3' durch Aufnahme der Regel $\hat{t}_1' \rightarrow \hat{t}_3'$ (bzw. $\hat{t}_3' \rightarrow \hat{t}_1'$, $\hat{t}_i$ Normalform zu t_i) erzwungen (Bild 4).

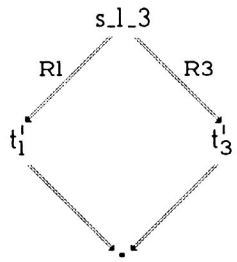


Bild 4

Details dieser Strategie (und über deren Implementierung) sowie aller anderen vorgestellten Verfahren können [So88] entnommen werden.

5 Reduktionsstrategien

Während der Vervollständigung von Regelsystemen ist es sehr oft erforderlich, eine Normalform zu einem gegebenen Term zu bestimmen. Es ist deshalb wichtig, effiziente Strategien zu entwickeln. Eine Reduktionsstrategie führt auf einem Term t, in deterministischer Weise, eine Reihe von Reduktionsschritten aus, solange, bis keine weitere Reduktion mehr möglich ist. Stoppt das Verfahren, so ist der durch endlich viele Reduktionen entstandene Term eine Normalform.
Ist das zugrundegelegte Termersetzungssystem nicht eindeutig terminierend, so ist diese Normalformbildung im allgemeinen ebenfalls nicht eindeutig. Verschiedene Reduktionsstrategien werden dann auch unterschiedliche Normalformen liefern. Die Implementierung berücksichtigt die Existenz mehrerer Normalformen jedoch nicht. Jede Strategie liefert genau eine Normalform, die für die weitere Berechnung Gültigkeit hat. Deshalb ist das Ziel einer Reduktionsstrategie, schnell eine Normalform zu bestimmen und nicht, eine bestimmte Normalform zu finden. Ein nicht-deterministischer Algorithmus zur Bestimmung einer Normalform kann folgendermaßen aussehen. Sei t der Eingabeterm:

WHILE $\exists l \Rightarrow r\ ,\ u \in O(t):\quad t/u = \sigma(l)$
DO $t := t[u \leftarrow \sigma(r)]$

Damit dieser Algorithmus deterministisch arbeitet, muß die Wahl des Teilterms t/u und der Regel $l \Rightarrow r$ genau festgelegt werden:

- Wahl der Stelle u: Die verwendeten Reduktionsstrategien unterscheiden sich nur durch die Reihenfolge, in der die Stellen eines Terms ausgewählt werden. Man unterscheidet hier im wesentlichen zwei Fälle, Reduktion der kleinsten und Reduktion der größten Teilterme.

- Wahl der Regel: Der Algorithmus wählt aus dem aktuellen Regelsystem die kürzeste (hinsichtlich der Anzahl der in der Regel enthaltenen Funktionssymbole und Variablen) Regel aus.

In den folgenden Abschnitten werden zunächst vier Reduktionsstrategien vorgestellt:

LEFTMOST-INNERMOST	(Küchlin)
BOTTOM-UP	(Gallier & Book)
LEFTMOST-OUTERMOST	(Küchlin)
TOP-DOWN	

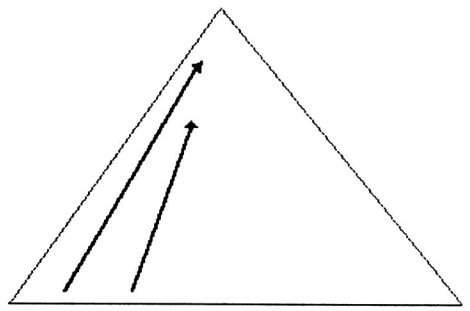
Die ersten beiden Strategien arbeiten einen Term von innen nach außen, die beiden anderen Strategien von außen nach innen ab. Innerhalb der beiden Klassen arbeitet jeweils eine Strategie den Term im depth-first-Verfahren, die andere im breadth-first-Verfahren ab.
Wie diese Strategien durch Markierung irreduzibler Teilterme verbessert werden können, zeigt der Abschnitt 5.5.

Außer diesen Strategien wurde eine triviale, 'nicht-deterministische' Strategie[1] implementiert. Sie arbeitet die Menge der Stellen eines Terms - eine Liste - sukzessive von vorne nach hinten ab.

Damit der Vervollständigungsalgorithmus korrekt arbeitet, ist es u.a. unbedingt erforderlich, daß die verwendete Reduktionsstrategie zulässig ist, d.h. terminierend und vollständig. Eine Strategie ist terminierend, falls die Anwendung von Regeln auf einen Term terminiert. Eine Reduktionsstrategie ist vollständig, falls beim Stop des Verfahrens der hergeleitete Term irreduzibel ist. Alle hier vorgestellten Strategien haben diese Eigenschaften.

Die Reduktionsstrategien werden durch ihre jeweiligen Ideen und anhand einprägsamer graphischer Darstellungen beschrieben. Ferner werden ihre Vor- und Nachteile aufgezeigt, und ihre Arbeitsweisen an einem Beispiel deutlich gemacht. Ist ein Term gegeben, so ist auch - wie bereits bekannt - in eindeutiger Weise der dazugehörige (Syntax-) Baum festgelegt. Deshalb werden im weiteren die Begriffe Term und Baum gleichbedeutend verwendet.

5.1 Leftmost - Innermost[2]



Der zu reduzierende Term wird im Postorderverfahren (depth-first) durchlaufen. Zu jedem Zeitpunkt gilt: wird ein Knoten bearbeitet, so sind alle seine Nachfolger und links von ihm liegenden Knoten bearbeitet und somit irreduzibel.

\+ Ein wesentlicher Pluspunkt (im Gegensatz zu den später behandelten Strategien Leftmost-Outermost und Top-down) ist es, daß kein Knoten überprüft wird, der schon einmal als irreduzibel erkannt wurde.

\- Kann ein Knoten reduziert werden, so werden alle Knoten dieses Teilbaums entfernt. Alle diese Knoten wurden aber bereits bearbeitet. Diese gewonnene Information geht also verloren.

Beispiel: Als Regelsystem sei gegeben

$\mathfrak{R}$:	g(x,b)	→	a	(R1)
	a	→	b	(R2)
	f(g(x,a),b)	→	x	(R3)
	h(f(x,a))	→	f(x,a)	(R4)

Der zu reduzierende Term t hat folgende Gestalt: h(f(g(g(x,a),a),a)).

---

[1] vgl. EQ-REDUCE, Kapitel 9

[2] vgl. EQ-REDUCE-INNERMOST, Kapitel 9

t

$\Downarrow$ R2 auf t/1112 anwenden

t_1 = h(f(g(**g(x,b)**,a),a))

$\Downarrow$ R_1 auf t_1/111 anwenden

t_2 = h(f(g(**a**,a),a))

$\Downarrow$ R2 auf t_2/111 anwenden

t_3 = h(f(g(b,**a**),a))

$\Downarrow$ R2 auf t_3/112 anwenden

t_4 = h(f(**g(b,b)**,a))

$\Downarrow$ R1 auf t_4/11 anwenden

t_5 = h(f(**a**,a))

$\Downarrow$ R2 auf t_5/11 anwenden

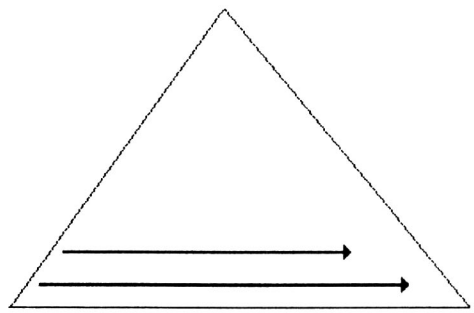
t_6 = h(f(b,**a**))

$\Downarrow$ R2 auf t_6/12 anwenden

$\hat{t}$ = h(f(b,b))

5.2 Bottom-up[1]

Der Bottom-up-Algorithmus arbeitet wie die Leftmost-Innermoststrategie den Baum von den Blättern zur Wurzel hin ab. Jedoch wird der Baum nicht im Depth-first-Verfahren bearbeitet, sondern parallel von unten nach oben durchlaufen. Dabei werden jeweils alle Teilterme gesucht, die möglicherweise reduzierbar sind, aber selbst keine reduzierbaren Teilterme besitzen. Anschließend werden diese reduziert und das Verfahren beginnt von vorne, an der Stelle wo reduziert wurde.



\+ Es wird kein Knoten zweimal bearbeitet (siehe 5.1). Bottom-up ist geeignet für parallele Bearbeitung.

\- Kann reduziert werden, so gehen Teilbäume verloren, die schon vollständig bearbeitet wurden (siehe 5.1).

---

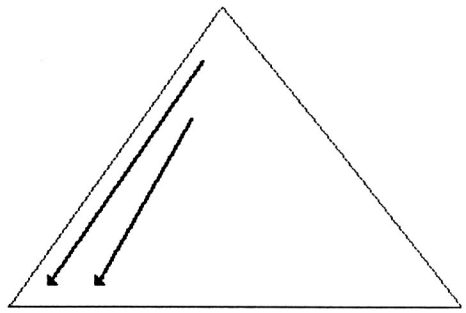
[1] vgl. EQ-REDUCE-BOTTOM-UP

Beispiel: Es sei das Regelsystem $\mathfrak{R}$ und der zu reduzierende Term t aus 5.1 gegeben.

t
$\Downarrow$ R2 auf t/1112 anwenden
t_1 = h(f(g(g(x,b),**a**),a))
$\Downarrow$ R2 auf t_1/112 anwenden
t_2 = h(f(g(g(x,b),b),**a**))
$\Downarrow$ R2 auf t_2/12 anwenden
t_3 = h(f(g(**g(x,b)**,b),b))
$\Downarrow$ R1 auf t_3/111 anwenden
t_4 = h(f(g(**a**,b),b))
$\Downarrow$ R2 auf t_4/111 anwenden
t_5 = h(f(**g(b,b)**,b))
$\Downarrow$ R1 auf t_5/11 anwenden
t_6 = h(f(**a**,b))
$\Downarrow$ R2 auf t_6/11 anwenden
$\hat{t}$ = h(f(b,b))

5.3 Leftmost-Outermost[1]

Der zu reduzierende Term wird bei der Wurzel beginnend im Preorder-Verfahren durchlaufen. Kann ein Knoten reduziert werden, beginnt das Verfahren von vorne mit dem neuen Term. Zu jedem Zeitpunkt gilt: Wird ein Knoten bearbeitet, so sind alle seine Vorgänger, sowie die links von ihm liegenden Knoten bereits bearbeitet, also irreduzibel.



Der Algorithmus versucht immer wieder, den Term an der Wurzel zu reduzieren. Jedesmal, wenn eine Reduktion den Term verändert hat, ist es möglich, daß er vollständig reduzierbar ist. Deshalb wird das Verfahren mit dem neuen Term noch einmal gestartet. Wurde der Term ganz durchlaufen, ohne daß eine Reduktion vorgenommen wurde, so wurden alle Knoten besucht und als irreduzibel erkannt, d.h. es liegt nun eine Normalform vor.

\+ Wird ein Knoten reduziert, so sind alle seine Teilbäume verloren. Diese wurden jedoch noch nicht bearbeitet, deshalb wurden hier keine überflüssigen Reduktionsversuche durchgeführt.

\- Da nach jeder Reduktion der Term von vorne abgearbeitet wird, werden viele Knoten getestet, die schon einmal bearbeitet und als irreduzibel erkannt wurden.

---

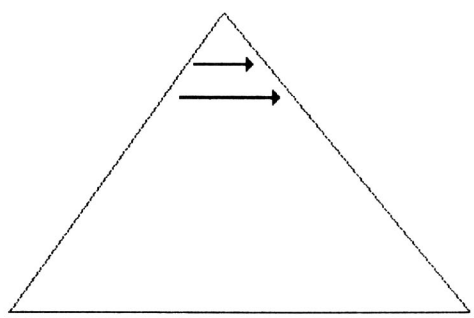
[1] vgl. EQ-REDUCE-OUTERMOST, Kapitel 9

Beispiel: Es sei das Regelsystem $\mathfrak{R}$ und der zu reduzierende Term t aus 5.1 gegeben.

t		
$\Downarrow$	R4 auf t	anwenden
t_1	= f(g(g(x,**a**),a),a)	
$\Downarrow$	R2 auf t_1/112	anwenden
t_2	= f(g(**g(x,b)**,a),a)	
$\Downarrow$	R1 auf t_2/11	anwenden
t_3	= f(g(**a**,a),a)	
$\Downarrow$	R2 auf t_3/11	anwenden
t_4	= f(g(b,**a**),a)	
$\Downarrow$	R2 auf t_4/12	anwenden
t_5	= f(**g(b,b)**,a)	
$\Downarrow$	R1 auf t_5/1	anwenden
t_6	= f(**a**,a)	
$\Downarrow$	R2 auf t_6/1	anwenden
t_7	= f(b,**a**)	
$\Downarrow$	R2 auf t_7/2	anwenden
$\hat{t}$	= f(b,b))	

5.4 Top-down[1]

Ziel dieser Strategie ist es, möglichst große Teilterme des zu reduzierenden Terms zuerst zu untersuchen.Wie die Leftmost-outermost Strategie arbeitet auch dieses Verfahren den Baum bei der Wurzel beginnend ab. Danach werden alle direkten Nachfolger nach dem breadth-first-Prinzip abgearbeitet. Zu jedem Zeitpunkt gilt: Wird ein Knoten bearbeitet, so sind alle seine Vorgänger, sowie alle links von ihm liegenden Brüder irreduzibel.



\+ Eine Reduktion findet nur an Knoten statt, deren Nachfolger noch nicht bearbeitet wurden.

\- Nach einer Reduktion werden Knoten bearbeitet, die bereits zu einem früheren Zeitpunkt als irreduzibel erkannt wurden.

---

[1] vgl. EQ-REDUCE-TOP-DOWN, Kapitel 9

Beispiel: Es sei wieder das Regelsystem $\mathfrak{R}$ und der zu reduzierende Term t aus 5.1 gegeben.

$$
\begin{array}{ll}
t & \\
\Downarrow & \text{R4 auf t anwenden} \\
t_1 & = f(g(g(x,a),a),\mathbf{a}) \\
\Downarrow & \text{R2 auf } t_1/2 \text{ anwenden} \\
t_2 & = \mathbf{f(g(g(x,a),a),b)} \\
\Downarrow & \text{R3 auf } t_2 \text{ anwenden} \\
t_3 & = g(x,\mathbf{a}) \\
\Downarrow & \text{R2 auf } t_3/2 \text{ anwenden} \\
t_4 & = \mathbf{g(x,b)} \\
\Downarrow & \text{R1 auf } t_4 \text{ anwenden} \\
t_5 & = \mathbf{a} \\
\Downarrow & \text{R2 auf } t_5 \text{ anwenden} \\
\hat{t} & = b
\end{array}
$$

5.5 Markierung irreduzibler Teilterme

In den Reduktionsstrategien Leftmost-outermost und Top-down werden oft Knoten bearbeitet, die schon einmal untersucht und als irreduzibel erkannt wurden. Wenn ein Knoten als irreduzibel erkannt und seine Teilbäume nicht verändert wurden, so bleibt auch ein erneuter Reduktionsversuch dieses Knotens erfolglos. Deshalb ist es sinnvoll, Knoten, die bereits bearbeitet wurden und irreduzibel sind, zu markieren. Die vier vorgestellten Reduktionsstrategien werden nun dahingehend verbessert, daß alle Knoten, die einmal als irreduzibel erkannt wurden, markiert werden und daß ein Knoten, der markiert ist, unbearbeitet bleibt. Variable sind, da sie immer irreduzibel sind, markiert. Wenn ein Knoten markiert ist, so können seine Nachfolger trotzdem reduzierbar sein, d.h. aus der Tatsache, daß ein Knoten markiert ist, folgt zwar, daß der dazugehörige Baum nicht vollständig (an der Stelle ε) reduzierbar ist, er kann aber trotzdem reduzierbar sein. Wird ein Nachfolger eines markierten Knotens reduziert, so muß diese Marke wieder entfernt werden, da er möglicherweise reduzierbar wird.
Die Markierung wirkt sich nicht nur vorteilhaft bei bereits bearbeiteten Knoten aus, sondern auch bei den, während einer Reduktion, "einsubstituierten" Knoten. Einige dieser Knoten können unter bestimmten Umständen markiert werden. Welche Knoten unter welchen Voraussetzungen markiert werden können, hängt von der jeweiligen Reduktionsstrategie ab.

Leftmost-Innermost mit expliziter Markierung[1]

Der Depth-first Algorithmus Leftmost-innermost bringt es mit sich, daß nur Knoten markiert werden, deren Nachfolger alle markiert sind (da alle Nachfolger bereits bearbeitet wurden).

---

[1] vgl. EQ-REDUCE-EX-INNERMOST, Kapitel 9

Es gilt: Ist ein Knoten k markiert, so ist der dazugehörige Teilbaum irreduzibel. Da bei dieser Strategie kein Knoten besucht wird, der schon einmal bearbeitet wurde, bringt die Markierung nur bei der Bearbeitung des an der Stelle k *neu* eingegliederten Terms σ(r) eine Verbesserung bezüglich der ohne Marken arbeitenden Leftmost-innermost Strategie. Während dort nämlich σ(r) ganz abgearbeitet wird, bleiben hier die einsubstituierten, markierten Terme σ(x) unbearbeitet. Alle σ(x) sind Teilterme von t/k, also des Terms, der gerade ersetzt wird. Da dieser Term aber bereits ganz bearbeitet wurde und alle Nachfolger markiert und somit irreduzibel sind, sind alle σ(x) markiert. Keine Verbesserung ergibt sich dann, wenn der Term nur mit Regeln reduziert werden kann, deren rechte Seite ein Grundterm ist, oder wenn alle σ(x) Variable sind.

Bottom-up mit expliziter Markierung[1]

Bei dieser Strategie lassen sich dieselben Merkmale feststellen wie bei Leftmost-innermost mit expliziter Markierung: Wenn reduziert werden kann, so sind die in der rechten Seite einer Regel einsubstituierten Terme σ(x) bereits bearbeitet, also markiert. Ein erneuter Reduktionsversuch ist überflüssig.

Leftmost-Outermost mit expliziter Markierung[2]

Die Leftmost-outermost Strategie arbeitet den Baum von der Wurzel zu den Blättern hin ab. Dabei werden Teilbäume markiert, die zwar nicht an der Stelle ε reduzierbar, aber doch reduzierbar sein können. Es ist deshalb zu beachten, daß bei einer Reduktion eines Knotens k die Marken aller direkten Vorgänger wieder entfernt werden. Denn wird in einem Term, der an ε irreduzibel ist, ein Teilterm verändert, so ist es möglich, daß der Term anschließend an der Wurzel reduziert werden kann. Wenn ein Knoten reduziert werden kann, so werden alle links von ihm liegenden Knoten nicht mehr bearbeitet, da sie markiert sind und bleiben.

Top-down mit expliziter Markierung[3]

Die Top-down Strategie durchläuft, wie die Leftmost-outermost Strategie, den Baum von der Wurzel zu den Blättern. Deshalb ist es auch hier erforderlich, die Marken aller direkten Vorgänger eines reduzierten Knotens zu entfernen. Wird der Term an einem Knoten k reduziert, so bearbeitet die Top-down Strategie ohne Markierung den ganzen Term noch einmal. Die Strategie mit Markierung besucht, außer den noch nicht bearbeiteten Knoten, nur die direkten Vorgänger von k.

---

[1] vgl. EQ-REDUCE-EX-BOTTOM-UP, Kapitel 9

[2] vgl. EQ-REDUCE-EX-OUTERMOST, Kapitel 9

[3] vgl. EQ-REDUCE-EX-TOP-DOWN, Kapitel 9

6 Termordnungen

Die Terminierung von Termersetzungssystemen mit Variablen ist im allgemeinen unentscheidbar (sie entspricht dem allgemeinen Halteproblem für Turingmaschinen). Es gibt jedoch Verfahren, mit denen diese Eigenschaft für bestimmte Regelsysteme nachgewiesen werden kann. Im folgenden werden einige solcher Methoden vorgestellt. Diese Methoden haben eine Gemeinsamkeit: es sind Simplifikationsordnungen, sie besitzen also die Teiltermeigenschaft und sind kompatibel mit der Termstruktur.
Ferner basieren alle implementierten Termordnungen auf einer sogenannten Vorordnung. Eine Vorordnung (oder Präzedenz) ist eine partiell geordnete Menge $(\mathfrak{F}, \triangleright)$, bestehend aus der Menge $\mathfrak{F}$ der Funktionssymbole und einer, auf den Elementen aus $\mathfrak{F}$ definierten, irreflexiven und transitiven Relation $\triangleright$. Im folgenden werden wir eine Vorordnung immer mit dem Symbol $\triangleright$ kennzeichnen.
Ein weiteres gemeinsames Merkmal ist die Art und Weise, Teilterme zu vergleichen. Da Operatoren Terme als Argumente besitzen, liegt es nahe, eine Ordnung auf Tupel von Termen zu definieren. Sei $\succ$ eine Ordnung, s_i und t_i Terme. Es gilt

$$
\begin{array}{lll}
(s_1,s_2,\ldots,s_m) & \succ^{lex} & (t_1,t_2,\ldots,t_n) \\
\text{falls} & \text{entweder} & m > 0 \quad \wedge \quad n = 0 \\
& \text{oder} & s_1 \succ t_1 \\
& \text{oder} & s_1 = t_1 \quad \wedge \quad (s_2,\ldots,s_m) \quad \succ^{lex} \quad (t_2,\ldots,t_n)
\end{array}
$$

ist die lexikographische Erweiterung von $\succ$.
Ist keine Reihenfolge auf den Elementen dieser Tupel vorgeschrieben, so handelt es sich bei den Strukturen um sogenannte Multimengen (engl. Multisets). Multimengen sind Mengen, in denen identische Elemente mehrfach auftreten können. Die Erweiterung einer Ordnung $\succ$ auf Multimengen (M_1 bzw. M_2) von Termen ist definiert durch

$$
\begin{array}{lll}
M_1 & \gg & M_2 \\
\text{gdw} & & \cdot \; M_1 \neq M_2 \quad \wedge \\
& & \cdot \; (\forall y \in M_2 \backslash M_1)\, (\exists x \in M_1 \backslash M_2) \quad x \succ y .
\end{array}
$$

Die Multimengendifferenz wird durch das Symbol $\backslash$ dargestellt.

Diese beiden Methoden, mehrere Terme mit mehreren Termen zu vergleichen, können formal zusammengefaßt werden, indem der Begriff des Status' eingeführt wird. Jedem Operator f wird ein Status $\tau(f)$ zugewiesen, der die Reihenfolge festlegt, in der die Argumente (direkten Teilterme) von f verglichen werden. D.h., der Status ist eine Funktion, die die Menge der Funktionssymbole in die Menge {mult, left, right} abbildet. Infolgedessen kann ein Operator einen der folgenden Stati haben:

- mult (die Argumente werden als Multimengen verglichen)
- left (lexikographischer Vergleich von links nach rechts)
- right (die Argumente werden lexikographisch von rechts nach links verglichen).

Fast alle in COMTES implementierten Termordnungen verfügen über eine solche Statusfunktion.

Außerdem definieren alle ein und dieselbe Kongruenz $\sim$. Sie ist abhängig von $\mathfrak{F}$ und τ:

$$
\begin{aligned}
& f(s_1,\ldots,s_m) \sim g(t_1,\ldots,t_n) \\
& \text{gdw} \quad f = g \ \wedge \ m = n \ \wedge \\
& \qquad \cdot\ \tau(f) = \text{mult} \ \wedge \ (\exists \pi)\,(\forall i \in [1,n])\ \ s_i \sim t_{\pi(i)} \\
& \qquad \cdot\ \tau(f) \neq \text{mult} \ \wedge \ (\forall i \in [1,n])\ \ s_i \sim t_i \,.
\end{aligned}
$$

Um die Lesbarkeit der einzelnen Definitionen der Ordnungen zu erleichtern, machen wir Gebrauch von zwei Konzepten:

- Der Index $\tau(f)$ von $>_{ord,\tau(f)}$ beschreibt die Erweiterung von $>_{ord}$ bzgl. des Status' des Operators f:

$$
\begin{aligned}
& (s_1,\ldots,s_m) \ >_{ord,\tau(f)} \ (t_1,\ldots,t_n) \\
& \text{gdw} \quad \tau(f) = \text{mult} \ \wedge \ \{s_1,\ldots,s_m\} \gg_{ord} \{t_1,\ldots,t_n\} \\
& \text{oder} \quad \tau(f) = \text{left} \ \wedge \ (s_1,\ldots,s_m) >^{lex}_{ord} (t_1,\ldots,t_n) \\
& \text{oder} \quad \tau(f) = \text{right} \ \wedge \ (s_m,\ldots,s_1) >^{lex}_{ord} (t_n,\ldots,t_1) \,.
\end{aligned}
$$

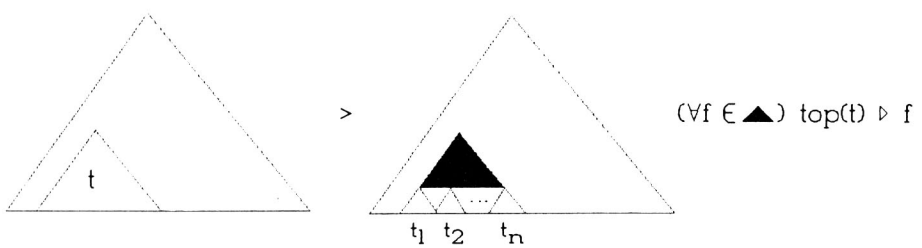
- Die lexikographische Ausführung von Bedingungen wird dargestellt durch Bindestriche, d.h.

$$
\begin{aligned}
& s > t \\
& \text{gdw} \quad - \ s >_1 t \\
& \qquad\quad - \ s >_2 t
\end{aligned}
$$

steht für $s > t$ gdw $s >_1 t$ oder $[s =_1 t \ \wedge \ s >_2 t]$.

6.1 Pfadordnungen

Die bekannteste aller Pfadordnungen ist die rekursive Pfadordnung[1] von Nachum Dershowitz, RPOS genannt. Der Vergleich zweier Terme basiert auf folgender Idee: Ein Term wird kleiner (bzgl. RPOS), falls ein Teilterm t durch eine Anzahl kleinerer Terme $t_1,\ldots,t_n$ ersetzt wurde. Die t_i wurden zu einem Term in irgendeiner Weise zusammengefügt unter Verwendung von Operatoren, die kleiner sind als das führende Funktionssymbol des zu ersetzenden Terms t:



[1] vgl. TR-RPO, Kapitel 9

Die Methode ist abhängig von den führenden Funktionssymbolen der zu vergleichenden Terme. Das Verhältnis dieser beiden Operatoren (und deren Status) ist verantwortlich für das Entfernen des einen oder beider Operatoren. Wurde so irgendwann einer der beiden Terme vollständig reduziert (zum "leeren" Term), so ist der andere der größere:

$s >_{RPOS} t$ gdw

i) $top(s) \rhd top(t) \wedge \{s\} \gg_{RPOS} args(t)$

ii) $top(s) = top(t) \wedge \tau(top(s)) = mult \wedge args(s) \gg_{RPOS} args(t)$

iii) $top(s) = top(t) \wedge \tau(top(s)) \neq mult \wedge args(s) >_{RPOS,\tau(top(s))} args(t) \wedge \{s\} \gg_{RPOS} args(t)$

iv) $args(s) \gg_{RPOS} \{t\}$

Die Relation $s \geq_{RPOS} t$ bedeutet, daß entweder $s >_{RPOS} t$ oder $s \sim t$.

Beispiel: Wir werden nachweisen, daß der Term $s = x*(y+z)$ größer ist (bzgl. RPOS) als der Term $t = x*y + x*z$. Als Voraussetzungen verwenden wir die totale Vorordnung $* \rhd +$ und die Statusfunktion $\tau(+) = \tau(*) = left$. Da $* \rhd +$, muß gelten, daß $\{s\} \gg_{RPOS} args(t)$. D.h., $\{s\} \gg_{RPOS} \{x*y, x*z\}$: s ist größer als x*y, da die führenden Funktionssymbole entfernt werden müssen und man unter Ausnutzung der Teiltermeigenschft der RPOS zeigen kann, daß $(x,y+z) >_{RPOS,left} (x,y) \wedge \{s\} \gg_{RPOS} \{x,y\}$. $s >_{RPOS} x*z$ kann analog dazu gezeigt werden.

Außer dieser Pfadordnung wurden noch zwei weitere implementiert. Ihre Definitionen verlangen gewisse Kenntnisse über Pfade.
Ein Pfad eines Terms ist eine Folge von Termen, beginnend mit dem Term selbst, gefolgt von dem Pfad einer seiner Argumente:

- $path_{\varepsilon}(\triangle) = \triangle$ falls $\triangle$ eine Konstante oder eine Variable ist
- $path_{i.u}(f(t_1,...,t_n)) = f(t_1,...,t_n) ; path_u(t_i)$ falls u eine Endstelle von t_i ist

Ferner bezeichnet $path(\{t_1,...,t_n\}) = \{path_u(t_i) \mid i \in [1,n]$, u Endstelle von $t_i\}$ die Multimenge aller Pfade der angegebenen Terme $t_1,...,t_n$. Ein Pfad wird im folgenden mit eckigen Klammern umschlossen. Die Menge set(p) eines Pfades p besteht aus allen Termen, die in p vorkommen: $set([t_1;...;t_n]) = \{t_1,...,t_n\}$. Auf einem Pfad $p = [t_1;t_2;...;t_n]$ sind ferner folgende zwei Operationen definiert:

- $sub(p,t_i) = [t_{i+1};...;t_n]$ ist der Pfad der echten Teilterme bzgl. t_i
- $sup(p,t_i) = [t_1;...;t_{i-1}]$ ist der Pfad der echten Superterme bzgl. t_i.

Sei $t = x*y + x*z$ ein Term, dann ist $path_{21}(t) = [t;x*z;x]$, $path(\{t\}) = \{path_{11}(t), path_{12}(t), path_{21}(t), path_{22}(t)\}$ und $sup(path_{21}(t),x) = [t;x*z]$.

Die Pfadordnung von David Plaisted[1], PSO (path of subterms ordering) genannt, entstand zeitlich noch vor der RPOS. Sie vergleicht zwei Terme, indem deren Pfade gegenübergestellt werden. Eine leicht modifizierte Variante (äquivalent zum Original) von Michael Rusinowitch folgt:

$s >_{PSO} t$
gdw $path(\{s\}) \gg_{PO} path(\{t\})$
mit $p >_{PO} q$
gdw $set(p) \gg_T set(q)$
mit $s >_T t$
gdw - $top(s) \triangleright top(t)$
- $path(args(s)) \gg_{PO} path(args(t))$

Die Äquivalenz zweier Terme bzgl. PSO ist eine spezielle Art von $\sim$, da die PSO über keine Statusfunktion verfügt. Die PSO vergleicht also alle Elemente bzgl. mult-Status. Infolgedessen sind zwei Terme äquivalent, falls sie permutativ kongruent sind.

Beispiel (siehe Beispiel RPOS): Seien $s = x*(y+z)$ und $t = x*y + x*z$ Terme. Ferner $* \triangleright +$. Im folgenden wird nachgewiesen, daß $s >_{PSO} t$. Deshalb ist zu zeigen, daß $path(\{s\}) = \{[s; x], [s; y+z; y], [s; y+z; z]\} \gg_{PO} \{[t; x*y; x], [t; x*y; y], [t; x*z; x], [t; x*z; z]\}$, d.h. für jeden Pfad in t muß es einen größeren (bzgl. PO) in s geben.

i) $[s; x] >_{PO} [t; x*y; x]$
gdw $\{s, x\} \gg_T \{t, x*y, x\}$
gdw $\{s\} \gg_T \{t, x*y\}$

- $s >_T t$, da $top(s) = * \triangleright + = top(t)$
- $s >_T x*y$
  gdw $path(\{x, y+z\}) \gg_{PO} path(\{x, y\})$
  gdw $path(\{y+z\}) \gg_{PO} path(\{y\})$: Dies gilt, da y ein echter Teilterm von y+z ist.

ii) $[s; x] >_{PO} [t; x*z; x]$,
$[s; y+z; y] >_{PO} [t; x*y; y]$,
$[s; y+z; z] >_{PO} [t; x*z, z]$:
Diese Aussagen können analog zu i) gezeigt werden.

Ähnlich der PSO vergleicht die Pfadordnung mit Status von Deepak Kapur, Paliath Narendran und G. Sivakumar[2] (KNSS genannt) die Pfade zweier Terme:

---

[1] vgl. TR-PSO, Kapitel 9
[2] vgl. TR-KNS, Kapitel 9

$s >_{KNSS} t$
gdw $path(\{s\}) \gg_{LK} path(\{t\})$
mit $p >_{LK} q$
gdw $(\forall t' \in q)\ (\exists s' \in p)\ s' >_{LT} t'$
mit $p\ \ s >_{LT} t \in q$
gdw i) $top(s) \triangleright top(t)$
ii) $top(s) = top(t) \wedge \tau(top(s)) = mult \wedge$
- $sub(p,s) >_{LK} sub(q,t)$
- $path(args(s)) \gg_{LK} path(args(t))$
- $sup(p,s) >_{LK} sup(q,t)$

iii) $top(s) = top(t) \wedge \tau(top(s)) \neq mult \wedge$
- $args(s) >_{KNSS,\tau(top(s))} args(t) \wedge \{s\} \gg_{KNSS} args(t)$
- $sup(p,s) >_{LK} sup(q,t)$

Die Kongruenz, die durch diese Ordnung induziert wird, ist identisch mit der Kongruenz $\sim$.

Beispiel: Die grundlegenden Vergleichsoperationen der KNSS sind denen der PSO sehr ähnlich. Deshalb verweisen wir an dieser Stelle auf das Beispiel über die PSO.

6.2 Dekompositionsordnungen

Die Beziehung zwischen Dekompositions- und Pfadordnungen ist durch die Definition einer Dekomposition festgelegt:

Eine Dekomposition ist ein ungeordneter Pfad.

Formal: Ist p ein Pfad, so ist set(p) die zugehörige Pfad-Dekomposition $dec_u(t)$, d.h. $dec_u(t) = set(path_u(t))$. Ein Element (also ein Term) einer Pfad-Dekomposition wird auch elementare Dekomposition genannt. Wie auch bei Pfaden, kennzeichnet die Dekomposition $dec(\{t_1,....,t_n\}) = \{dec_u(t_i) \mid i \in [1,n],$ u Endstelle von $t_i\}$ die Multimenge aller Pfad-Dekompositionen der Terme $t_1,....,t_n$. Die Operationen sub und sup auf Pfaden gelten für Pfad-Dekompositionen in analoger Weise.
Sei $t = x*y + x*z$ ein Term. Dann ist $dec_{21}(t) = \{t, x*z, x\}$, $dec(\{t\}) = \{\{t, x*y, x\}, \{t, x*y, y\}, \{t, x*z, x\}, \{t, x*z, z\}\}$ und $sub(dec_{21}(t), x) = \emptyset$, $sup(dec_{21}(t), x*z) = \{t\}$.

Die erste vorzustellende rekursive Dekompositionsordnung mit Status (RDOS) stellt eine Weiterentwicklung der RPOS dar. Ein wesentlicher Unterschied zur RPOS ist die Tatsache, daß die RDOS den Vergleich abbricht, sobald unvergleichbare (bzgl. $\triangleright$) Operatoren vorliegen. Ein Term s ist größer als ein Term t (bzgl. RDOS), falls die Dekomposition von s größer ist als diejenige von t. Die Ordnung auf diesen Multimengen ($\gg \gg_{LD}$) ist eine Erweiterung der Basis-Ordnung auf Termen ($>_{LD}$) für Multimengen von Multimengen.

$s >_{RDOS} t$
gdw $dec(\{s\}) \gg \gg_{LD} dec(\{t\})$
mit $dec_u(s) \quad s' >_{LD} t' \in dec_v(t)$
gdw i) $top(s') \triangleright top(t')$
ii) $top(s') = top(t') \wedge \tau(top(s')) = mult \wedge$
- $sub(dec_u(s), s') \gg_{LD} sub(dec_v(t), t')$
- $args(s') \gg_{RDOS} args(t')$
iii) $top(s') = top(t') \wedge \tau(top(s')) \neq mult \wedge$
$args(s') >_{RDOS,\tau(top(s'))} args(t') \wedge \{s'\} \gg_{RDOS} args(t')$

Terme sind äquivalent bzgl. RDOS ($s =_{RDOS} t$), falls sie in der Kongruenzbeziehung $\sim$ stehen ($s \sim t$).

Beispiel: (siehe Beispiel RPOS): Wir werden nachweisen, daß $s = x*(y+z)$ größer ist als $t = x*y + x*z$. Dazu setzen wir voraus, daß $* \triangleright +$ und $\tau(+) = \tau(*) = left$.

$s >_{RDOS} t$
gdw $dec(\{s\}) = \{dec_1(s), dec_{21}(s), dec_{22}(s)\} \gg \gg_{LD} \{dec_{11}(t), dec_{12}(t), dec_{21}(t), dec_{22}(t)\} = dec(\{t\})$

i) $dec_1(s) = \{s, x\} \gg_{LD} \{t, x*y, x\} = dec_{11}(t)$
gdw $\{s\} \gg_{LD} \{t, x*y\}$:
- $s >_{LD} t$
da $top(s) = * \triangleright + = top(t)$
- $s >_{LD} x*y$
gdw $(x, y+z) >_{RDOS,left} (x, y) \wedge \{s\} \gg_{RDOS} \{x, y\}$:
Diese Aussagen sind gültig, da y ein echter Teilterm von y+z ist und x, y echte Teilterme von s sind.

ii) $dec_1(s) = \{s, x\} \gg_{LD} \{t, x*z, x\} = dec_{21}(t)$,
$dec_{21}(s) = \{s, y+z, y\} \gg_{LD} \{t, x*y, y\} = dec_{12}(t)$,
$dec_{22}(s) = \{s, y+z, z\} \gg_{LD} \{t, x*z, z\} = dec_{22}(t)$:
kann in analoger Weise zu i) gezeigt werden.

Diese Ordnung (RDOS) wurde nicht implementiert, da sie die schwächste der (hier vorgestellten) Dekompositionsordnungen ist. Das Prinzip von Dekompositionsordnungen i.a. ist an ihr jedoch sehr einfach zu demonstrieren.

Eine andere Dekompositionsordnung (PSD) basiert auf dem Verfahren der PSO. Die PSO ist eine extrem rekursive Ordnung, die *drei* Basisordnungen ($>_{PO}$, $>_T$ und $\triangleright$) benötigt. Die PSD ist äquivalent zur PSO und stellt dagegen eine wesentlich einfachere Methode dar. Die PSD besitzt noch einen weiteren Vorteil: Die Integration einer Statusfunktion ist problemloser. Aus diesen Gründen haben wir die PSDS[1] (PSD mit Status) implementiert:

$$
\begin{array}{l}
s >_{PSDS} t \\
\text{gdw} \quad dec(\{s\}) \gg\gg_{LP} dec(\{t\}) \\
\qquad \text{mit} \quad s >_{LP} t \\
\qquad\qquad \text{gdw} \quad \text{i)} \quad top(s) \triangleright top(t) \\
\qquad\qquad\qquad\quad \text{ii)} \quad top(s) = top(t) \;\wedge\; \tau(top(s)) = mult \;\wedge \\
\qquad\qquad\qquad\qquad\quad dec(args(s)) \gg\gg_{LP} dec(args(t)) \\
\qquad\qquad\qquad\quad \text{iii)} \quad top(s) = top(t) \;\wedge\; \tau(top(s)) \neq mult \;\wedge \\
\qquad\qquad\qquad\qquad\quad args(s) >_{PSDS,\tau(top(s))} args(t) \;\wedge\; \{s\} \gg_{PSDS} args(t)
\end{array}
$$

$s =_{PSDS} t$ gdw $s \sim t$.

Beispiel: Der Vergleich der Terme x*(y+z) und x*y + x*z bzgl. der PSDS ist identisch (LD durch LP ersetzen) zu dem der RDOS (siehe Seite 42).

Gemäß der Definition der PSDS werden in ii) *alle* direkten Teilterme von s und t verglichen: $dec(args(s)) \gg\gg_{LP} dec(args(t))$. Man könnte somit diese Vorgehensweise als Breitensuche bezeichnen. Die nun folgende rekursive Dekompositionsordnung[2] (IRDS) wählt sich an dieser Stelle zunächst nur *ein* Argument aus (Tiefensuche).

$$
\begin{array}{l}
s >_{IRDS} t \\
\text{gdw} \quad dec(\{s\}) \gg\gg_{EL} dec(\{t\}) \\
\qquad \text{mit} \quad dec_u(s) \quad s' >_{EL} t' \in dec_v(t) \\
\qquad\qquad \text{gdw} \quad \text{i)} \quad top(s') \triangleright top(t') \\
\qquad\qquad\qquad\quad \text{ii)} \quad top(s') = top(t') \;\wedge\; \tau(top(s')) = mult \;\wedge \\
\qquad\qquad\qquad\qquad\quad - \; sub(dec_u(s),s') \gg_{EL} sub(dec_v(t),t') \\
\qquad\qquad\qquad\qquad\quad - \; dec(args(s')) \gg\gg_{EL} dec(args(t')) \\
\qquad\qquad\qquad\quad \text{iii)} \quad top(s') = top(t') \;\wedge\; \tau(top(s')) \neq mult \;\wedge \\
\qquad\qquad\qquad\qquad\quad args(s') >_{IRDS,\tau(top(s'))} args(t') \;\wedge\; \{s'\} \gg_{IRDS} args(t')
\end{array}
$$

Die Relation $s \geq_{IRDS} t$ gilt, falls entweder $s >_{IRDS} t$ oder $s \sim t$.

---

[1] vgl. TR-PSDS, Kapitel 9

[2] vgl. TR-IRDS, Kapitel 9

Beispiel: Der Vergleich der Terme x*(y+z) und x*y + x*z bzgl. der IRDS ist ebenfalls identisch (LD durch EL ersetzen) zu dem der RDOS (siehe Seite 42).

6.3 Knuth-Bendix Ordnung

Eine wohlfundierte Menge $(S , >_S)$ kann benutzt werden, um die Terminierung eines Termersetzungssystems zu zeigen. Zur Konstruktion einer Ordnung wählt man solch eine Menge und eine sogenannte Terminierungsfunktion, die Γ in S abbildet. S kann z.B. die Termalgebra selbst sein: Die Pfad- und Dekompositionsordnungen unterliegen diesem Prinzip. Die Ordnung von Knuth und Bendix (KBO) basiert auf der wohlfundierten Menge $(\mathbb{N} , >)$, d.h. jedem Funktionssymbol wird eine natürliche (oder reelle) Zahl (sein Gewicht) zugewiesen. Ein Term wird in $\mathbb{N}$ abgebildet, indem die Gewichte aller in ihm auftretenden Symbole addiert werden. Der Vergleich zweier Terme geschieht über den Vergleich der Gewichte und über den lexikographischen Vergleich der Argumente. Analog zu den Pfad- und Dekompositionsordnungen ist es auch bei der KBO möglich, eine Statusfunktion zu integrieren. Um diese Ordnung[1] (KBOS) zu beschreiben, benötigen wir noch einige zusätzliche Notationen.

Die Anzahl der Vorkommen einer Variablen x in einem Term t wird durch $\pi_x(t)$ abgekürzt. Wir weisen jedem Operator $f \in \mathfrak{F}$ eine nicht negative ganze Zahl $\varphi(f)$ - das Gewicht von f - und jeder Variablen eine einheitliche positive ganze Zahl φ_0 zu:

- $\varphi(c) \geq \varphi_0$ falls c eine Konstante ist,
- $\varphi(f) > 0$ falls f nur ein Argument hat.

Diese Gewichtsfunktion wird auf Terme erweitert, indem für jeden Term $t = f(t_1,...,t_n)$

$$\varphi(t) = \varphi(f) + \Sigma \varphi(t_i) .$$

$s >_{KBOS} t$
gdw $(\forall x \in \mathfrak{V})\ \pi_x(s) \geq \pi_x(t) \quad \wedge$
- $\varphi(s) > \varphi(t)$
- $top(s) \triangleright top(t)$
- $args(s) >_{KBOS,\tau(top(s))} args(t)$

Zwei Terme sind äquivalent bzgl. KBOS, falls sie in der Relation $\sim$ stehen.

Es ist möglich, diese Definition zu verfeinern, indem *ein* einstelliges Funktionssymbol mit dem Gewicht 0 zugelassen wird. Dann muß jedoch gewährleistet sein, daß alle anderen Operatoren kleiner sind bzgl. $\triangleright$.

---

[1] vgl. TR-KBO, Kapitel 9

Beispiel: Seien s = x*((-y)*y) und t = (-y*y)*x zwei Terme, $\tau(*)$ = mult und * ▷ -. Ferner sei folgende Gewichtsfunktion gegeben:

symbol	x , y	-	*
φ	1	1	0

$s >_{KBOS} t$, da $\varphi(s) = \varphi(t) = 4$, top(s) = top(t) = *
und $\{x, (-y)*y\} \gg_{KBOS} \{-y*y, x\}$
da $(-y)*y >_{KBOS} -y*y$ (Gewichte sind identisch und * ▷ -).

6.4 Polynomordnung

In 6.3 wurde deutlich gemacht, daß sich eine noethersche Partialordnung in zwei Teile zerlegen läßt: eine Terminierungsfunktion φ, die Terme in eine Menge S abbildet und eine noethersche "Standardordnung" $>_s$ auf S. Wir wählen jetzt $(S , >_s)$ mit S = P(ℕ), der Menge der Polynome mit ganzzahligen Koeffizienten, und $>_s$ = >.
Die Terminierungsfunktion φ ordnet jedem Funktionssymbol f ein streng monoton wachsendes Polynom φ(f) - seine Interpretation - über P(ℕ) zu:

- $P_0(\mathbb{N}) = \{a \mid a \in \mathbb{N}\}$ ist die Menge der konstanten Polynome.
- $p \in P_n(\mathbb{N})$ gdw $P = \sum_{k_1,\ldots,k_n} a_{k_1\ldots k_n} x_1^{k_1} \cdot \ldots \cdot x_n^{k_n}$ mit $a_{k_1\ldots k_n} \in \mathbb{N}$.
- Sei f ein Funktionssymbol mit n Argumenten:
  $\varphi(f) = p \in P_n(\mathbb{N})$ mit $(\forall i \in [1,n])\ (\exists a_{k_1\ldots k_n}$ von p) $a_{k_1\ldots k_n} \neq 0 \quad \wedge \quad k_i \neq 0$
- $\varphi(x) = X$ falls x eine Variable ist.
- $\varphi(f(t_1,\ldots,t_n)) = \varphi(f)(\varphi(t_1),\ldots,\varphi(t_n))$.

Der Vergleich zweier Terme[1] geschieht mittels des Vergleichs ihrer entsprechenden Polynome:

$s >_{POL} t$
gdw $\varphi(s) \sqsupset \varphi(t)$
mit $p \sqsupset q$
gdw $(\exists M \subseteq \mathbb{N})\ (\forall x_i \in M)\ p(x_1,\ldots,x_n) > q(x_1,\ldots,x_n)$

[1] vgl. TR-POL, Kapitel 9

Der Vergleich zweier Terme bzgl. der Polynomordnung reduziert sich auf den Test, ob ein Polynom größer ist als 0. Dieser Test ist i.a. unentscheidbar. Hinreichende Kriterien wurden von Ahlem Ben Cherifa und Pierre Lescanne entwickelt. Ihre Überlegungen basieren darauf, daß ein Polynom über $\mathbb{N}$ mit nur positiven Koeffizienten größer als 0 ist. Um zu zeigen, daß ein Polynom p positiv ist, versucht man also es in eine solche Form zu überführen. D.h., es soll eine Kette $p = p_0 \geq p_1 \geq \ldots \geq p_n$ erzeugt werden, wobei p_n nur noch positive Koeffizienten enthält. Bei jeder Umformung, einem Schritt von p_i nach p_{i+1}, soll ein negatives Monom mit einem positiven Monom verrechnet werden:

WHILE es gibt einen negativen Koeffizienten
DO **IF** $(\exists a_{k_1 \ldots k_n} > 0)\ (\exists a_{m_1 \ldots m_n} < 0)\ (\forall i \in [1,n])\ k_i \geq m_i$
THEN CHANGE $(a_{k_1 \ldots k_n}, a_{m_1 \ldots m_n})$
ELSE **RETURN FAIL**
RETURN POSITIVE

Ahlem Ben Cherifa und Pierre Lescanne geben zwei mögliche change-Funktionen an. Für die eine Methode schränken sie die Menge $M \subseteq \mathbb{N}$ ein auf $M = \{x \mid x > 1\}$ und das andere Verfahren verwendet $M = \{x \mid x > 2\}$. Implementiert wurde ein allgemeines Verfahren, wobei der Benutzer die Untergrenze für die Variablen selbst angibt.

Der Übergang von Polynom p_i zum Polynom p_{i+1} geschieht dann folgendermaßen (u sei die oben erwähnte Untergrenze):

$$p_i = \ldots - k * x_1^{k_1} \ldots x_n^{k_n} \ldots + g * x_1^{g_1} \ldots x_n^{g_n}$$
mit $k,g > 0$ und $(\forall i \in [1,n])\ g_i \geq k_i$

$$p_{i+1} = \ldots - (k \dot{-} g * u^{\Sigma(g_i - k_i)})\ x_1^{k_1} \ldots x_n^{k_n} \ldots + (g \dot{-} k * u^{\Sigma(k_i - g_i)})\ x_1^{g_1} \ldots x_n^{g_n}$$

Beispiel: Die Regel $s = x*(y+z) \rightarrow x*y + x*z = t$ (siehe Beispiele in 6.1 und 6.2) soll als terminierend nachgewiesen werden. Dazu verwenden wir die Polynomordnung mit folgenden Interpretationen:

$$\varphi(+)(x,y) = x+y$$
$$\varphi(*)(x,y) = xy^2$$

Damit erhält man

$$\varphi(s) = x(y+z)^2 = xy^2 + 2xyz + xz^2$$
$$\varphi(t) = xy^2 + xz^2$$

und $\varphi(s) \sqsupset \varphi(t)$, da $(\forall x,y,z > 0)\ 2xyz > 0$.

7 Spezielle Parameter

In diesem Kapitel werden besondere Verfahren zur Vermeidung eines Abbruchs des Vervollständigungsalgorithmus' vorgestellt.

7.1 Interreduktion

Ein Regelsystem $\mathfrak{R}$ heißt interreduziert (oder normiert), falls für jede Regel $l \to r \in \mathfrak{R}$ gilt:

- l ist irreduzibel bzgl. $\mathfrak{R}\backslash\{l \to r\}$
- r ist irreduzibel bzgl. $\mathfrak{R}$.

Die Interreduktion erfüllt im wesentlichen zwei Aufgaben:

- Das Regelsystem wird immer so klein wie möglich gehalten.

- Die Einsetzbarkeit der Termordnung wird gesteigert:
  fak(s(x)) → s(x)*fak(p(s(x)))
  fak(0) → s(0)
  p(s(x)) → x
  Dieses Regelsystem zur Berechnung der Fakultät einer natürlichen Zahl kann mittels einer Simplifikationsordnung nicht als terminierend nachgewiesen werden. Die Terminierung des interreduzierten Systems
  fak(s(x)) → s(x)*fak(x)
  fak(0) → s(0)
  p(s(x)) → x
  kann durch eine RPOS mit fak ▷ * und fak ▷ s garantiert werden.

Jedes Regelsystem läßt sich in ein äquivalentes interreduziertes Regelsystem transformieren, indem die linke und rechte Seite aller Regeln auf Normalform gebracht werden.
Das zu interreduzierende Regelsystem wird unterteilt in ein Teilsystem $\mathfrak{R}$, das bereits interreduziert ist, und die restlichen Regeln, von denen nicht bekannt ist, ob sie diese Eigenschaft haben. Im Verlauf der Vervollständigung sind die "nicht interreduzierten" Regeln die kritischen Paare $\mathfrak{P}$, also Gleichungen. Ausgehend von diesen beiden Mengen wird jeweils eine Gleichung aus $\mathfrak{P}$ entfernt und nachdem ihre linke und rechte Seite auf Normalform bzgl. $\mathfrak{R}$ gebracht wurde, wird die neuentstandene Regel $l \to r$ zu $\mathfrak{R}$ hinzugenommen. Das neue System $\mathfrak{R} \cup \{l \to r\}$ ist nun möglicherweise nicht mehr normiert. Deshalb muß überprüft werden, ob mit der neuen Regel eine der ursprünglichen Regeln reduziert werden kann. Ist die linke Seite einer Regel aus $\mathfrak{R}$ reduzierbar, so wird sie aus $\mathfrak{R}$ entfernt und in $\mathfrak{P}$ aufgenommen. Ist die rechte Seite einer Regel aus $\mathfrak{R}\backslash\{l \to r\}$ reduzierbar, so wird eine Normalform bzgl. $\mathfrak{R} \cup \{l \to r\}$ gebildet:

WHILE $\mathfrak{P} \neq \emptyset$
DO $\mathfrak{P} := \mathfrak{P} \setminus \{(t_1, t_2)\}$
$\hat{t}_1$:= Normalform von t_1 bzgl. $\mathfrak{R}$
$\hat{t}_2$:= Normalform von t_2 bzgl. $\mathfrak{R}$
Sei $l \rightarrow r$ die aus $(\hat{t}_1, \hat{t}_2)$ entstehende Regel, sonst Abbruch.
FOR $l_i \rightarrow r_i \in \mathfrak{R}$
DO **IF** $(\exists u, \sigma)\ l_i/u = \sigma(l)$
THEN $\mathfrak{R} := \mathfrak{R} \setminus \{l_i \rightarrow r_i\}$
$\mathfrak{P} := \mathfrak{P} \cup \{(l_i[u \leftarrow \sigma(r)], r_i)\}$
FOR $l_i \rightarrow r_i \in \mathfrak{R}$
DO **IF** $(\exists u, \sigma)\ r_i/u = \sigma(l)$
THEN Bilde Normalform von r_i
$\mathfrak{R} := \mathfrak{R} \cup \{l \rightarrow r\}$

Die Interreduktion[1] ist ein optionaler Parameter, der vom Benutzer zu Beginn eines Vervollständigungslaufs gesetzt wird.

7.2 Postponing

Wenn ein kritisches Paar gebildet wurde, so werden Normalformen der beiden Seiten bestimmt und diese mit der vorgegebenen Ordnung verglichen. Dabei kann es passieren, daß die beiden Terme unvergleichbar sind, weil es keine totale Ordnung auf Termen mit Variablen gibt. Wenn ein unvergleichbares Paar gefunden wurde, so sieht der Knuth-Bendix Algorithmus einen Abbruch des Verfahrens vor. Dabei könnte es sein, daß zu einem späteren Zeitpunkt eine der beiden Seiten (mit einer neuen Regel) weiter reduziert werden könnte, und die beiden Seiten vergleichbar werden.
Um sich diese Möglichkeit zu eröffnen, können unvergleichbare Paare zunächst zurückgestellt[2] (engl. postpone) und zu einem späteren Zeitpunkt wieder verarbeitet werden, d.h. die beiden Seiten eines unvergleichbaren Paares werden auf Normalform gebracht und mit der vorgegebenen Ordnung verglichen. Können die beiden Seiten verglichen werden, so wird die neue Regel in das bestehende System integriert. Sind die beiden Seiten immer noch unvergleichbar, so wird das Paar weiterhin zurückgestellt. Das Zurückstellen unvergleichbarer Paare verletzt die Korrektheit des Vervollständigungsalgorithmus' nicht. Weil diese Paare in regelmäßigen Abständen wieder betrachtet werden, kommt dies der Tatsache nahe, daß die entsprechenden kritischen Paare erst später gebildet wurden. Können zu irgendeinem Zeitpunkt keine kritischen Paare mehr gebildet werden, weil bereits alle Regeln miteinander überlappt wurden, und es existieren noch unvergleichbare Paare, die nicht weiter reduziert werden können, so muß der Algorithmus abbrechen. In diesem Fall konnte mit der vorgegebenen Ordnung kein eindeutig terminierendes Regelsystem aus dem eingegebenen Regelsystem erzeugt werden.

---

[1] vgl. KB-NORMALIZE, Kapitel 9

[2] vgl. KB-POSTPONE, Kapitel 9

Die Möglichkeit des Postponing wird dem Benutzer zur Verfügung gestellt. Wurde ein unvergleichbares kritisches Paar generiert, so kann es der Benutzer - neben anderen Aktionen - zurückstellen lassen.

7.3 Einführung neuer Operatoren

Als Ausgangssituation sei diejenige von 7.2 gegeben: ein, mit der aktuellen Termordnung, unvergleichbares kritisches Paar (t_1,t_2). Die Einführung eines neuen Operators $f \notin \mathfrak{F}$ verursacht die Generierung zweier Regeln:

$$t_1 \rightarrow f(x_1,....,x_n)$$
$$t_2 \rightarrow f(x_1,....,x_n)$$

wobei $\{x_1,....,x_n\}$ die Schnittmenge der Variablenmengen von t_1 und t_2 ist: $V(t_1) \cap V(t_2)$.
Die prinzipielle Idee, auf der diese Methode basiert, ist die folgende: Die beiden Terme des kritischen Paares liegen in einer Äquivalenzklasse. Diese Äquivalenzklasse wird nun um einen *neuen* Term erweitert, und die unvergleichbaren Paare werden darauf gebildet.
Der Trick des Einführens neuer Operatoren wird oft angewendet, wenn $V(t_1) \neq V(t_1) \cap V(t_2) \neq V(t_2)$.

Beispiel: siehe Kapitel 8, Seite 59/60

7.4 Erweiterung der Vorordnung

Wurde ein unvergleichbares Paar generiert, so erscheint das folgende Menü auf dem Bildschirm:

suggestions	
l	. . .
	:
i-1	. . .
i	- HELP -
i+1	- SHOW ORDERING -
i+2	ORIENT PAIR
i+3	POSTPONE PAIR
i+4	EXTEND PRECEDENCE
i+5	INTRODUCE NEW OPERATOR
i+6	- ABORT -

Wählt der Benutzer "EXTEND PRECEDENCE", so hat er die Möglichkeit, die Vorordnung 'beliebig' zu erweitern. Erweitern heißt, daß alle bestehenden Relationen zwischen Operatoren unverändert bleiben müssen. Es dürfen nur zusätzliche Relationen eingeführt werden, die zu keinen Zyklen führen.

Wählt der Benutzer "ORIENT PAIR", so kann er - ohne irgendwelche Daten der aktuellen Ordnung zu ändern - das unvergleichbare Paar zu einer beliebigen Regel machen. Vorsicht: Die Terminierung ist dann natürlich nicht mehr garantiert.

Außer diesen Alternativen bietet das System (falls dies überhaupt möglich ist) selbst einige Vorschläge zur Erweiterung der Vorordnung an. Rechts neben den Nummern 1 bis i-1 erscheinen Informationen folgender Bauart:

$$f \triangleright g : \quad Term2 > Term1.$$

Diese Ausgabe bedeutet, daß Term2 größer ist als Term1, falls die Vorordnung um die Relation $f \triangleright g$ erweitert werden würde. Wählt der Benutzer diese Möglichkeit aus, so wird die Vorordnung automatisch um diese Relation verfeinert und aus dem vorher unvergleichbaren kritischen Paar wird eine Regel erzeugt.
COMTES überprüft jedoch nur, ob eine - um *eine* Relation erweiterte - Vorordnung das Paar vergleichbar machen würde.

8 Sitzungsbeispiel

Anhand eines Protokolls eines Vervollständigungsprozesses soll die Benutzeroberfläche des COMTES-Systems demonstriert werden. Benutzereingaben sind **fett** gedruckt. Diesen Eingaben geht immer ein Menü mit verschiedenen, numerierten Alternativen voraus. Der Benutzer kann entweder die Nummer oder ein eindeutiges Präfix der jeweiligen Alternative wählen (siehe z.B. Seite 53).
Erläuternde Kommentare haben wir in *kursiver* Schrift in das Protokoll eingestreut.

Es wird folgendes Gleichungssystem (von Miki Hermann) vervollständigt:

$$\begin{aligned} f(f(x,y),z) &= f(x,f(y,z)) \\ f(g(x,y),x) &= y \\ f(x,h(x,y)) &= y. \end{aligned}$$

Die Operatoren haben folgende Bedeutung:

$$\begin{aligned} f(x,y) &= x * y \\ g(x,y) &= y * x^{-1} \\ h(x,y) &= x^{-1} * y. \end{aligned}$$

Da der Vervollständigungsprozess für dieses Beispiel sehr viele kritische Paare (und Regeln) erzeugt, wurden nur die wichtigsten Informationen hier aufgeführt. Es wurde jeweils vermerkt, zu welchem Zeitpunkt das System mehr Ausgabe produziert als sie hier angegeben ist.

> **(comtes)**

The german research group

PROGRESS

of the university at Kaiserslautern
presents

COMTES

Version 2.2 - 28/02/1989
A test environment for completing term rewriting systems

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxx
x    input / output      x
x------------------------x
x  1 --HELP--            x
x  2 SPECIFICATION       x
x  3 INTERN FILE         x
x  4 OUTPUT FILES        x
x  5 STATISTIC FILE      x
x  6 --CONTINUE--        x
xxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Type any symbol  -> 1
You have chosen  - HELP -
```

```
+----------------------------------------+
|        HELP : input / output           |
+----------------------------------------+
```

COMTES can be viewed as a parametrized completion algorithm that is particularly suited for efficiency experiments. The kernel of the system is a slightly modified version of Huet's method. Various strategies for generating critical pairs, reducing terms and for proving the termination of rules are integrated.

Most of the input and output data are contained in files allied to directories. The defaults of these directories are the following ones :

- The specification file is contained in ~system/inout/specs.
- Its internal representation is contained in ~system/inout/intern.
- The output files are contained in ~system/inout/comtes.
- Some chronometries are contained in ~system/inout/statistic/trace.

These values can be changed by choosing the appropriate menu slot (SPECIFICATION INTERN-FILE OUTPUT-FILES STATISTIC-FILES) and by typing the desired path!

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxx
x    input / output      x
x------------------------x
x  1 --HELP--            x
x  2 SPECIFICATION       x
x  3 INTERN FILE         x
x  4 OUTPUT FILES        x
x  5 STATISTIC FILE      x
x  6 --CONTINUE--        x
xxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Type any symbol  -> 6
You have chosen  - CONTINUE -
```

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxx
x       choice             x
x--------------------------x
x  1 --HELP--              x
x  2 COMPLETION            x
x  3 REDUCTION             x
x  4 COMPARISON            x
x  5 UNDO                  x
x  6 --QUIT--              x
xxxxxxxxxxxxxxxxxxxxxxxxxxxx
Type any symbol  -> 1
You have chosen  -HELP -
```

```
+-----------------------------------------+
|              HELP : choice              |
+-----------------------------------------+
```

COMTES provides three independent tools:

- A tool for the completion of a term rewriting system. The input must be given on file.
- A tool for computing a normal form of a term w.r.t. a term rewriting system. The system must be given on file whereas the term have to be typed by hand. The normal form will be computed relative to a special reduction strategy chosen by the user.
- A tool for comparing a set of equations (given on file) or two terms (by hand) w.r.t. an ordering. You can choose one of seven different orderings.

press RETURN to continue

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxx
x       choice             x
x--------------------------x
x  1 --HELP--              x
x  2 COMPLETION            x
x  3 REDUCTION             x
x  4 COMPARISON            x
x  5 --QUIT--              x
xxxxxxxxxxxxxxxxxxxxxxxxxxxx
Type any symbol  -> compl
You have chosen COMPLETION


Filename : example

Internal representation of the file does not exist!
Parsing ...
```

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
x       strategies            x
x-----------------------------x
x    1 --HELP--               x
x    2 NOTHING                x
x    3 INNER                  x
x    4 EX-INNER               x
x    5 OUTER                  x
x    6 EX-OUTER               x
x    7 TOP-DOWN               x
x    8 EX-TOP-DOWN            x
x    9 BOTTOM-UP              x
x   10 EX-BOTTOM-UP           x
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Type any symbol  ->  4
You have chosen EX-INNER


xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
x                orderings                            x
x-----------------------------------------------------x
x  1 --HELP--                                         x
x  2 DERSHOWITZ                                       x
x  3 KAPUR/NARENDRAN/SIVAKUMAR                        x
x  4 KNUTH/BENDIX                                     x
x  5 LANKFORD/MANNA/NESS                              x
x  6 PLAISTED                                         x
x  7 PLAISTED-DECOMPOSITION                           x
x  8 RUSINOWITCH-DECOMPOSITION                        x
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Type any symbol  ->  2
You have chosen DERSHOWITZ

xxxxxxxxxxxxxxxxxxxxxxxxxx
x     precedence        x
x-----------------------x
x  1 --HELP--           x
x  2 SET PREC           x
x  3 --CONTINUE-        x
xxxxxxxxxxxxxxxxxxxxxxxxxx
Type any symbol  ->  2
You have chosen SET PREC
```

```
xxxxxxxxxxxxxxxxx
x  f-symbols  x
x-------------x
x  1 F        x
x  2 G        x
x  3 H        x
xxxxxxxxxxxxxxxxx
SEQUENCE  ->  f g
SEQUENCE  ->  f h
SEQUENCE  ->
```

```
+-------------------------------------------+
|              PRECEDENCE                   |
+-------------------------------------------+
x  F > G H
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
xxxxxxxxxxxxxxxxxxxxxxxxxx
x         status          x
x-------------------------x
x  1 --HELP--             x
x  2 SET STATUS           x
x  3 --CONTINUE--         x
xxxxxxxxxxxxxxxxxxxxxxxxxx
Type any symbol  ->  2
You have chosen  SET STATUS
```

```
xxxxxxxxxxxxxxxxx
x  f-symbols  x
x-------------x
x  1 F        x
x  2 G        x
x  3 H        x
xxxxxxxxxxxxxxxxx
Left-to-right-status : f h
Right-to-left-status :
```

```
+-------------------------------------------+
|              STATUS                       |
+-------------------------------------------+
x  F   LEFT
x  G   MULTISET
x  H   LEFT
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
x          criteria                  x
x------------------------------------x
x  1 --HELP--                        x
x  2 SINGLE STEP                     x
x  3 KAPUR/NARENDRAN                 x
x  4 KUECHLIN                        x
x  5 KUECHLIN EXTENDED               x
x  6 WINKLER                         x
x  7 WINKLER EXTENDED                x
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Type any symbol  -> 2
You have chosen SINGLE STEP


xxxxxxxxxxxxxxxxxxxxxxxx
x  interreduction      x
x----------------------x
x  1 --HELP--          x
x  2 NO                x
x  3 YES               x
xxxxxxxxxxxxxxxxxxxxxxxx
Type any symbol  -> 3
You have chosen YES


+++ NEW RULE +++
(EXAMPLE 1): F(F(X,Y),Z)  ->  F(X,F(Y,Z))


+++ NEW RULE +++
(EXAMPLE 2): F(G(X,Y),X)  ->  Y


+++ NEW RULE +++
(EXAMPLE 3): F(X,H(X,Y))  ->  Y


+--------------------------------------+
|          initial rulesystem          |
+--------------------------------------+
(EXAMPLE 1): F(F(X,Y),Z)  ->  F(X,F(Y,Z))
(EXAMPLE 2): F(G(X,Y),X)  ->  Y
(EXAMPLE 3): F(X,H(X,Y))  ->  Y
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
press RETURN to continue
```

COMPUTING CRITICAL PAIRS WITH
(EXAMPLE 1): F(F(X,Y),Z) → F(X,F(Y,Z))

COMPUTING CRITICAL PAIRS WITH
(EXAMPLE 2): F(G(X,Y),X) → Y

+++ NEW RULE +++
EXAMPLE 1, EXAMPLE 2 ⟹ (EXAMPLE 4): F(G(Z,X),F(Z,Y)) → F(X,Y)

Die Angabe 'EXAMPLE 1, EXAMPLE 2' bezieht sich auf die Regeln, aus denen diese neue Regel (mit Nummer 'EXAMPLE 4') entstanden ist. Regeln aus dem initialen System haben dementsprechend nur eine Identifizierung (siehe auch 9.1).

COMPUTING CRITICAL PAIRS WITH
(EXAMPLE 3): F(X,H(X,Y)) → Y

+++ NEW RULE +++
EXAMPLE 3, EXAMPLE 1 ⟹ (EXAMPLE 5): F(X,F(Y,H(F(X,Y),Z))) → Z

+++ NEW RULE +++
EXAMPLE 1, EXAMPLE 3 ⟹ (EXAMPLE 6): F(Z,F(H(Z,X),Y)) → F(X,Y)

COMPUTING CRITICAL PAIRS WITH
EXAMPLE 1, EXAMPLE 2 ⟹ (EXAMPLE 4): F(G(Z,X),F(Z,Y)) → F(X,Y)

+++ NEW RULE +++
EXAMPLE 4, EXAMPLE 1 ⟹ (EXAMPLE 7): F(G(F(Y,Z),X),F(Y,F(Z,U))) → F(X,U)

+++ NEW RULE +++
EXAMPLE 4, EXAMPLE 2 ⟹ (EXAMPLE 8): F(G(G(X,Z),Y),Z) → F(Y,X)

+++ NEW RULE +++
EXAMPLE 4, EXAMPLE 3 ⟹ (EXAMPLE 9): F(G(X,Y),Z) → F(Y,H(X,Z))

--- Delete Rule ---
(EXAMPLE 2): F(G(X,Y),X) → Y

--- Delete Rule ---
EXAMPLE 1, EXAMPLE 2 ⟹(EXAMPLE 4): F(G(Z,X),F(Z,Y)) → F(X,Y)

An dieser Stelle wurden einige kritische Paare und Regeln generiert, die nicht aufgeführt sind!

¤¤¤ COMPUTING CRITICAL PAIRS WITH ¤¤¤
EXAMPLE 4, EXAMPLE 2 ⟹ (EXAMPLE 11): F(Y,H(G(X,Z),Z)) → F(Y,X)

```
+-----------------------------------------+
|          uncomparable pair              |
+-----------------------------------------+
Term1 : "F(X,F(Y,Z))"
Term2 : "F(X,H(G(Y,U),F(U,Z)))"
I have no suggestions!!
The arity of a new operator is "3".

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
x            suggestions                    x
x-------------------------------------------x
x  1 --HELP--                               x
x  2 --SHOW ORDERING--                      x
x  3 ORIENT PAIR                            x
x  4 POSTPONE PAIR                          x
x  5 EXTEND PRECEDENCE                      x
x  6 INTRODUCE NEW OPERATOR                 x
x  7 --ABORT--                              x
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Type any symbol → po
You have chosen POSTPONE PAIR
```

An dieser Stelle wurden einige kritische Paare und Regeln generiert, die nicht aufgeführt sind!

```
+-----------------------------------------+
|          uncomparable pair              |
+-----------------------------------------+
Term1 : "H(X,X)"
Term2 : "G(Y,Y)"
I have no suggestions!!
There are variables in both terms
which don't occur in the other term !!
The arity of a new operator is "0".

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
x            suggestions                    x
x-------------------------------------------x
x  1 --HELP--                               x
x  2 --SHOW ORDERING--                      x
x  3 ORIENT PAIR                            x
x  4 POSTPONE PAIR                          x
x  5 EXTEND PRECEDENCE                      x
x  6 INTRODUCE NEW OPERATOR                 x
x  7 --ABORT--                              x
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Type any symbol  -> 1
You have chosen  -HELP-
```

```
+-----------------------------------------+
|      HELP : uncomparable pair           |
+-----------------------------------------+
```

COMTES possesses some special operations for handling incomparable pairs :

- Extending the precedence of the ordering : Either the system suggests some suitable possibilities (if existing) or the user may extend the precedence.
- Orienting a pair by hand.
- Postponing a pair.
- Introducing a new operator.

In order to extend the precedence you must have some informations about the actual ordering (choose SHOW ORDERING). ABORT forces a break of the comparison of this pair.

```
+-----------------------------------------+
|          uncomparable pair              |
+-----------------------------------------+
Term1 : "H(X,X)"
Term2 : "G(Y,Y)"
I have no suggestions!!
There are variables in both terms
which don't occur in the other term !!
The arity of a new operator is "0".
```

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
x              suggestions                        x
x-------------------------------------------------x
x  1 --HELP--                                     x
x  2 --SHOW ORDERING--                            x
x  3 ORIENT PAIR                                  x
x  4 POSTPONE PAIR                                x
x  5 EXTEND PRECEDENCE                            x
x  6 INTRODUCE NEW OPERATOR                       x
x  7 --ABORT--                                    x
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Type any symbol  -> 6
You have chosen INTRODUCE NEW OPERATOR

Type the name of the new operator  -> e
INTRODUCE NEW OPERATOR:   E

NEW EQUATIONS :
EXAMPLE 16, EXAMPLE 3 ==>( ): H(X,X)  ->  E
EXAMPLE 16, EXAMPLE 3 ==>( ): G(Y,Y)  ->  E
```

Die Nummern der zukünftigen Regeln (==> ( )) sind noch nicht eingetragen, da es sich noch um Gleichungen handelt (siehe nächste Seite).

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxx
x     precedence             x
x----------------------------x
x  1 --HELP--                x
x  2 SET PREC                x
x  3 --CONTINUE--            x
xxxxxxxxxxxxxxxxxxxxxxxxxxxx
Type any symbol  -> 5
ERROR : Try again  -> 2
You have chosen SET PREC

xxxxxxxxxxxxxxxxxx
x  f-symbols     x
x----------------x
x  1 F           x
x  2 G           x
x  3 H           x
x  4 E           x
xxxxxxxxxxxxxxxxxx
SEQUENCE  -> h e
SEQUENCE  -> g e
SEQUENCE  ->
```

```
+-----------------------------------------------+
|                  PRECEDENCE                   |
+-----------------------------------------------+
x  F > E G H
x  G > E
x  H > E
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Nach der Festlegung einer Vorordnungserweiterung wird vom Benutzer üblicherweise der Status des neuen Operators gefordert. Da es sich aber hier um eine Konstante handelt, ist der Status irrelevant, d.h. er wird automatisch auf 'multiset' gesetzt. Das gleiche Verfahren wird auch bei einstelligen Funktionssymbolen (und bei der Initialisierung des Status' zu Beginn eines Laufs) angewendet.

+++ NEW RULE +++
EXAMPLE 16, EXAMPLE 3 $\Longrightarrow$ (EXAMPLE 23): H(X,X) $\rightarrow$ E

--- Delete Rule ---
(EXAMPLE 13): F(Y,H(X,X)) $\rightarrow$ Y

--- Delete Rule ---
EXAMPLE 13, EXAMPLE 9 $\Longrightarrow$ (EXAMPLE 16): F(X,H(Y,H(Z,Z))) $\rightarrow$ G(Y,X)

+++ NEW RULE +++
EXAMPLE 16, EXAMPLE 3 $\Longrightarrow$ (EXAMPLE 24): G(Y,Y) $\rightarrow$ E

+++ NEW RULE +++
EXAMPLE 13, EXAMPLE 9 $\Longrightarrow$ (EXAMPLE 25): F(X,H(Y,E)) $\rightarrow$ G(Y,X)

+++ NEW RULE +++
(EXAMPLE 26): F(Y,E) $\rightarrow$ Y

```
+-----------------------------------------+
|  checking uncomparable pairs            |
+-----------------------------------------+
```

EXAMPLE 1, EXAMPLE 11 $\Longrightarrow$ F(X,F(Y,Z)) = F(X,H(G(Y,U),F(U,Z)))

An dieser Stelle wurden einige kritische Paare und Regeln generiert, die nicht aufgeführt sind!

```
+--------------------------------------------+
|          uncomparable pair                 |
+--------------------------------------------+
Term1 : "H(X,E)"
Term2 : "G(X,E)"
The arity of a new operator is "1".

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
x           suggestions                      x
x--------------------------------------------x
x  1 H > G : TERM1 > TERM2                   x
x  2 G > H : TERM2 > TERM1                   x
x  3 --HELP--                                x
x  4 --SHOW ORDERING--                       x
x  5 ORIENT PAIR                             x
x  6 POSTPONE PAIR                           x
x  7 EXTEND PRECEDENCE                       x
x  8 INTRODUCE NEW OPERATOR                  x
x  9 --ABORT--                               x
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Type any symbol  -> 4
You have chosen  - SHOW ORDERING -


xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
x        I N F O R M A T I O N                      x
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
x ORDERING :  DERSHOWITZ
+----------------------------------------------+
|              PRECEDENCE                      |
+----------------------------------------------+
x  F >  E G H
x  G >  E
x  H >  E
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

+--------------------------------------------+
|                STATUS                      |
+--------------------------------------------+
x  F   LEFT
x  G   MULTISET
x  H   LEFT
x  E   MULTISET
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
+----------------------------------------+
|          uncomparable pair             |
+----------------------------------------+
```

Term1 : "H(X,E)"
Term2 : "G(X,E)"
The arity of a new operator is "1".

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
x            suggestions                     x
x------------------------------------------x
x  1 H > G : TERM1 > TERM2                   x
x  2 G > H : TERM2 > TERM1                   x
x  3 --HELP--                               x
x  4 --SHOW ORDERING--                      x
x  5 ORIENT PAIR                            x
x  6 POSTPONE PAIR                          x
x  7 EXTEND PRECEDENCE                      x
x  8 INTRODUCE NEW OPERATOR                 x
x  9 --ABORT--                              x
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Type any symbol → **8**
You have chosen INTRODUCE NEW OPERATOR

Type the name of the new operator → **i**
INTRODUCE NEW OPERATOR: I

NEW EQUATIONS :
EXAMPLE 25, EXAMPLE 28 ⟹ ( ): H(X,E) → I(X)
EXAMPLE 25, EXAMPLE 28 ⟹ ( ): G(X,E) → I(X)

```
xxxxxxxxxxxxxxxxxxxxxxxxxx
x     precedence          x
x-------------------------x
x  1 --HELP--             x
x  2 SET PREC             x
x  3 --CONTINUE--         x
xxxxxxxxxxxxxxxxxxxxxxxxxx
```

Type any symbol → **2**
You have chosen SET PREC

```
xxxxxxxxxxxxxxxxxx
x  f-symbols  x
x-------------x
x  1  F       x
x  2  G       x
x  3  H       x
x  4  E       x
x  5  I       x
xxxxxxxxxxxxxxxxxx
SEQUENCE  ->  g (h i)
SEQUENCE  ->
```

```
+-------------------------------------------+
|              PRECEDENCE                   |
+-------------------------------------------+
x  H >I E
x  G >H I E
x  I >H E
x  F >I H E G
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
+++ NEW RULE +++
EXAMPLE 25, EXAMPLE 28 ==> (EXAMPLE 37): H(X,E)  ->  I(X)


--- Delete Rule ---
EXAMPLE 13, EXAMPLE 9 ==> (EXAMPLE 25): F(X,H(Y,E))  ->  G(Y,X)


+++ NEW RULE +++
EXAMPLE 25, EXAMPLE 28 ==> (EXAMPLE 38): G(X,E)  ->  I(X)
```

An dieser Stelle wurden einige kritische Paare und Regeln generiert, die nicht aufgeführt sind!

```
+-------------------------------------------+
|          uncomparable pair                |
+-------------------------------------------+
Term1 : "G(I(H(Y,G(Z,U))),X)"
Term2 : "G(Z,G(I(H(Y,U)),X))"
I have no suggestions!!
The arity of a new operator is "4".
```

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
x                  suggestions                      x
x-------------------------------------------------x
x  1 --HELP--                                       x
x  2 --SHOW ORDERING--                              x
x  3 ORIENT PAIR                                    x
x  4 POSTPONE PAIR                                  x
x  5 EXTEND PRECEDENCE                              x
x  6 INTRODUCE NEW OPERATOR                         x
x  7 --ABORT--                                      x
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Type any symbol  -> po
You have chosen POSTPONE PAIR
```

Hier wurden einige kritische Paare und Regeln generiert, die nicht aufgeführt sind! Ferner wurden diverse unvergleichbare Paare zurückgestellt!

i

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
x                   interrupt                       x
x-------------------------------------------------x
x  1 --HELP--                                       x
x  2 SHOW RULES                                     x
x  3 SHOW INCOMPARABLE PAIRS                        x
x  4 SHOW ORDERING                                  x
x  5 SAVE                                           x
x  6 --ABORT--                                      x
x  7 --CONTINUE--                                   x
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Type any symbol  -> 2
You have chosen SHOW RULES
```

```
+-----------------------------------------+
|               rulesystem                |
+-----------------------------------------+
EXAMPLE 16, EXAMPLE 3 ==> (EXAMPLE 23): H(X,X)  ->  E
EXAMPLE 28, EXAMPLE 3 ==> (EXAMPLE 30): H(E,X)  ->  X
EXAMPLE 25, EXAMPLE 28 ==> (EXAMPLE 37): H(X,E)  ->  I(X)
EXAMPLE 37, EXAMPLE 23 ==> (EXAMPLE 40): I(E)  ->  E
EXAMPLE 31, EXAMPLE 37 ==> (EXAMPLE 47): I(I(X))  ->  X
EXAMPLE 39, EXAMPLE 47 ==> (EXAMPLE 49): F(X,Y)  ->  H(I(X),Y)
EXAMPLE 43, EXAMPLE 47 ==> (EXAMPLE 58): G(X,Y)  ->  H(I(Y),I(X))
EXAMPLE 41, EXAMPLE 3 ==> (EXAMPLE 44): H(Y,H(I(Y),X))  ->  X
```

(EXAMPLE 64): H(I(X),H(X,Y)) → Y
EXAMPLE 31, EXAMPLE 43 ⟹ (EXAMPLE 48): H(H(Y,I(X)),I(Y)) → X
EXAMPLE 41, EXAMPLE 22 ⟹ (EXAMPLE 66): H(I(H(Y,Z)),I(X)) → H(Y,H(I(Z),I(X)))
EXAMPLE 28, EXAMPLE 11 ⟹ (EXAMPLE 67): H(H(I(Y),I(X)),Y) → X
EXAMPLE 3, EXAMPLE 1 ⟹ (EXAMPLE 62): H(I(X),H(I(Y),H(H(I(X),Y),Z))) → Z
EXAMPLE 9, EXAMPLE 3 ⟹ (EXAMPLE 63): H(I(Z),H(Y,H(H(I(Z),I(Y)),X))) → X
EXAMPLE 14, EXAMPLE 3 ⟹ (EXAMPLE 61): H(I(X),H(Z,H(I(U),H(H(Z,U),Y)))) → H(I(X),Y)
EXAMPLE 4, EXAMPLE 1 ⟹ (EXAMPLE 59): H(I(X),H(H(I(Y),Z),H(I(Y),H(I(Z),U)))) → H(I(X),U)
EXAMPLE 12, EXAMPLE 9 ⟹ (EXAMPLE 60): H(I(X),H(H(I(Y),I(Z)),H(I(Y),H(Z,U)))) → H(I(X),U)

Die Regel (EXAMPLE 64): H(I(X),H(X,Y)) → Y besitzt keine Verweise auf die Regeln, aus denen sie entstanden ist (siehe Seite 57). Dies ist ein Indiz dafür, daß es sich um eine reduzierte Regel des initialen Systems handelt.

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
x                interrupt                             x
x------------------------------------------------------x
x  1 --HELP--                                          x
x  2 SHOW RULES                                        x
x  3 SHOW INCOMPARABLE PAIRS                           x
x  4 SHOW ORDERING                                     x
x  5 SAVE                                              x
x  6 --ABORT--                                         x
x  7 --CONTINUE--                                      x
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Type any symbol → 3
You have chosen SHOW INCOMPARABLE PAIRS
```

```
+-----------------------------------------+
|          incomparable pairs             |
+-----------------------------------------+
```

EXAMPLE 1, EXAMPLE 11 ⟹ F(X,F(Y,Z)) = F(X,H(G(Y,U),F(U,Z)))
EXAMPLE 22, EXAMPLE 9 ⟹ G(I(H(Y,G(Z,U))),X) = G(Z,G(I(H(Y,U)),X))
EXAMPLE 39, EXAMPLE 9 ⟹ G(I(H(Y,I(Z))),X) = G(Z,G(Y,X))
EXAMPLE 16, EXAMPLE 1 ⟹ G(I(G(Z,Y)),X) = G(Z,G(I(Y),X))
EXAMPLE 4, EXAMPLE 3 ⟹ G(I(Z),G(X,Y)) = G(I(H(X,Z)),Y)
G(I(Z),G(I(Y),X)) = G(I(G(I(Z),Y)),X)

Die letzte Gleichung besitzt keinen Hinweis auf die Regeln, aus denen sie generiert wurde, da es eine initiale, reduzierte Regel ist.

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
x              interrupt                        x
x-----------------------------------------------x
x  1 --HELP--                                   x
x  2 SHOW RULES                                 x
x  3 SHOW INCOMPARABLE PAIRS                    x
x  4 SHOW ORDERING                              x
x  5 SAVE                                       x
x  6 --ABORT--                                  x
x  7 --CONTINUE--                               x
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Type any symbol  ->  4
You have chosen SHOW ORDERING
```

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
x        I N F O R M A T I O N        x
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
x  ORDERING :  DERSHOWITZ
+---------------------------------------------+
|                 PRECEDENCE                  |
+---------------------------------------------+
x  H > I E
x  G > H I E
x  I > H E
x  F > I H E G
+---------------------------------------------+
|                   STATUS                    |
+---------------------------------------------+
x  F  LEFT
x  G  MULTISET
x  H  LEFT
x  E  MULTISET
x  I  MULTISET
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
x              interrupt                             x
x-------------------------------------------------x
x  1 --HELP--                                        x
x  2 SHOW RULES                                      x
x  3 SHOW INCOMPARABLE PAIRS                         x
x  4 SHOW ORDERING                                   x
x  5 SAVE                                            x
x  6 --ABORT--                                       x
x  7 --CONTINUE--                                    x
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Type any symbol  -> 5
You have chosen SAVE


Information has been written on file :
~/system/inout/comtes/example.out

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
x              interrupt                             x
x-------------------------------------------------x
x  1 --HELP--                                        x
x  2 SHOW RULES                                      x
x  3 SHOW INCOMPARABLE PAIRS                         x
x  4 SHOW ORDERING                                   x
x  5 SAVE                                            x
x  6 --ABORT--                                       x
x  7 --CONTINUE--                                    x
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Type any symbol  -> 7
You have chosen -CONTINUE-
```

An dieser Stelle wurden einige kritische Paare und Regeln generiert, die nicht aufgeführt sind!

```
+-----------------------------------------+
|  checking uncomparable pairs            |
+-----------------------------------------+
EXAMPLE 1, EXAMPLE 11 ⟹ F(X,F(Y,Z)) = F(X,H(G(Y,U),F(U,Z)))
EXAMPLE 22, EXAMPLE 9 ⟹ G(I(H(Y,G(Z,U))),X) = G(Z,G(I(H(Y,U)),X))
EXAMPLE 39, EXAMPLE 9 ⟹ G(I(H(Y,I(Z))),X) = G(Z,G(Y,X))
EXAMPLE 16, EXAMPLE 1 ⟹ G(I(G(Z,Y)),X) = G(Z,G(I(Y),X))
EXAMPLE 4, EXAMPLE 3 ⟹ G(I(Z),G(X,Y)) = G(I(H(X,Z)),Y)
G(I(Z),G(I(Y),X)) = G(I(G(I(Z),Y)),X)
ALL PAIRS REDUCED!!!
```

An dieser Stelle wurden einige kritische Paare und Regeln generiert, die nicht aufgeführt sind!

```
+--------------------------------------+
|          initial rulesystem          |
+--------------------------------------+
(EXAMPLE 1): F(F(X,Y),Z)   ->   F(X,F(Y,Z))
(EXAMPLE 2): F(G(X,Y),X)   ->   Y
(EXAMPLE 3): F(X,H(X,Y))   ->   Y

+--------------------------------------+
|         canonical rulesystem         |
+--------------------------------------+
EXAMPLE 16, EXAMPLE 3 ==> (EXAMPLE 23): H(X,X)   ->   E
EXAMPLE 28, EXAMPLE 3 ==> (EXAMPLE 30): H(E,X)   ->   X
EXAMPLE 25, EXAMPLE 28 ==> (EXAMPLE 37): H(X,E)   ->   I(X)
EXAMPLE 37, EXAMPLE 23 ==> (EXAMPLE 40): I(E)   ->   E
EXAMPLE 31, EXAMPLE 37 ==> (EXAMPLE 47): I(I(X))   ->   X
EXAMPLE 39, EXAMPLE 47 ==> (EXAMPLE 49): F(X,Y)   ->   H(I(X),Y)
EXAMPLE 43, EXAMPLE 47 ==> (EXAMPLE 58): G(X,Y)   ->   H(I(Y),I(X))
EXAMPLE 41, EXAMPLE 3 ==> (EXAMPLE 44): H(Y,H(I(Y),X))   ->   X
(EXAMPLE 64): H(I(X),H(X,Y))   ->   Y
EXAMPLE 68, EXAMPLE 64 ==> (EXAMPLE 75): I(H(Y,X))   ->   H(X,Y)
EXAMPLE 74, EXAMPLE 47 ==> (EXAMPLE 79): H(H(X,Y),Z)   ->   H(Y,H(I(X),Z))


xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
x          I N F O R M A T I O N          x
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
x  ORDERING :  DERSHOWITZ
+----------------------------------------------+
|                 PRECEDENCE                   |
+----------------------------------------------+
x  H > I E
x  G > H I E
x  I > H E
x  F > I H E G
+----------------------------------------------+
|                  STATUS                      |
+----------------------------------------------+
x  F   LEFT
x  G   MULTISET
x  H   LEFT
x  E   MULTISET
x  I   MULTISET
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
+-----------------------------------------+
|              STATISTIC                  |
+-----------------------------------------+
× Total run time used                : 73.93
× Number of rules generated          : 80
× Number of critical pairs generated : 153
× ---------------------------------------------------
× Time spent in normalization        : 58.12   (79%)
× Time spent in unification          : 0.431   (1%)
× Number of unifiers generated       : 227
× Time spent in reduction            : 20.645  (28%)
× Number of reduction steps          : 446
× Time spent in ordering             : 0.786   (1%)
××××××××××××××××××××××××××××××××××
press RETURN to continue


          ××××××××××××××××××××××××××××
          ×   input / output         ×
          ×--------------------------×
          × 1 --HELP--               ×
          × 2 SPECIFICATION          ×
          × 3 INTERN FILE            ×
          × 4 OUTPUT FILES           ×
          × 5 STATISTIC FILE         ×
          × 6 --CONTINUE--           ×
          ××××××××××××××××××××××××××××
          Type any symbol → 6
          You have chosen —CONTINUE—


          ××××××××××××××××××××××××××
          ×       choice             ×
          ×--------------------------×
          × 1 --HELP--               ×
          × 2 COMPLETION             ×
          × 3 REDUCTION              ×
          × 4 COMPARISON             ×
          × 5 --QUIT--               ×
          ××××××××××××××××××××××××××
          Type any symbol → 5
          You have chosen —QUIT—


          GOOD BYE
```

9 Implementierung

In diesem Kapitel werden wichtige Teile der Implementierung etwas genauer beschrieben. Ein wesentlicher Aspekt sind z.B. die benutzten Datenstrukturen (siehe 9.1).
Dazu gehört aber ebenfalls die Aufzählung der wichtigsten COMTES-Funktionen, nach ihrer Zugehörigkeit zu einzelnen Modulen getrennt und in alphabetischer Reihenfolge. Die Bindung einer Funktion zu einem Modul wird durch Zusatz eines Präfixes deutlich gemacht: z.B. gehört EQ=<name der Funktion> zum Modul EQUATION.
Ein Entwurf einer neuen Vervollständigungsumgebung könnte einige der bestehenden Funktionen enthalten. Aus diesem Grund wird für jede beschriebene Funktion von COMTES die Menge der von ihr abhängigen Funktionen aufgelistet.
Eine detaillierte on-line Beschreibung aller Funktionen findet man unter support/modules.

9.1 Datenstrukturen

COMTES ist in LISP geschrieben. Daher ist es naheliegend, die Datenstrukturen intern als Listen festzulegen. Die Elemente einer Liste sind oft Atome, die kleinsten syntaktischen Einheiten von LISP:

<atom> := <literal> | <nat> | <nat>.<nat>

Terme

Ein Term ist entweder eine Variable oder eine Liste, bestehend aus einem Funktionssymbol und mehreren (≥ 0) Termen:

<term> := <var> |
(<function> <term>*)

wobei sowohl eine Variable als auch ein Funktionssymbol als LISP-Atom implementiert ist:

<var> := <atom>
<function> := <atom>

Bemerkung: Eine Konstante ist eine Liste mit einem Atom, d.h. (a) ist eine Konstante und a ist eine Variable.

Zur expliziten Markierung irreduzibler Terme (siehe Reduktionsstrategien, 5.5) benötigen wir eine Datenstruktur 'markierte Terme':

```
<lterm>  :=  <var> |
             ( (<function> <mark>)  <lterm>* )
```

Die zusätzliche Marke

```
<mark>  :=  + | -
```

gibt an, ob ein Term markiert (+) - also irreduzibel ist - oder reduzierbar (-) ist.

Gleichungen und Regeln

Eine Gleichung wird durch ihre beiden Seiten (Terme) und eine Identifizierung dargestellt:

```
<equation>  :=  ( (<term> <term>)  <label> )
```

Die eindeutige Kennzeichnung einer Gleichung ist eine Numerierung folgender Art:

```
<label>  :=  (<ident>  [ (<ident> <ident>) ])
<ident>  :=  <atom> <nat>
```

Der Identifier <ident> besteht aus dem Namen der Spezifikation und einer natürlichen Zahl, der fortlaufenden Numerierung.

Handelt es sich um eine initiale Gleichung, so wird nur der Identifier <ident> angegeben: z.B. mit (gruppe 1) wird die als erste eingegebene Gleichung gekennzeichnet. Ist die Gleichung ein kritisches Paar, so charakterisieren die beiden zusätzlichen Identifiers die beiden Regeln, aus denen es entstanden ist: z.B. (gruppe 4 (gruppe 2 gruppe 3)).

Die interne Datenstruktur für eine Regel ist identisch mit der einer Gleichung. Sie wird nur als orientierte Gleichung (erster Term entspricht linker Seite, zweiter Term der rechten Seite) interpretiert.

Substitutionen und Stellen

Eine Substitution besteht aus einer Liste von Paaren der Form

```
<subst>  :=  ( (<var> <term>)* )
```

Eine markierte Substitution unterscheidet sich von der allgemeinen Substitution dadurch, daß anstelle von Termen markierte Terme verwendet werden:

```
<lsubst>  :=  ( (<var> <lterm>)* )
```

Eine Stelle eines Terms ist als Liste natürlicher Zahlen implementiert:

<pos> := (<nat>*)

Ordnungen

Eine Ordnung hängt üblicherweise von mehreren Parametern ab. Diese werden, neben der Identifizierung (Name der Ordnung), als Liste verwaltet:

<ordering> := (<atom> <prec> <stat> <weight>)

Das erste Element der Liste enthält den Namen des Entwicklers:

dershowitz	:	RPOS
kapur/narendran/sivakumar	:	KNSS
knuth/bendix	:	KBOS
lankford/manna/ness	:	POL
plaisted	:	PSO
plaisted-decomposition	:	PSDS
rusinowitch-decomposition	:	IRDS

Die Vorordnung besteht aus einer Liste von Listen:

<prec> := ((<function>*)*)

Jede Teilliste enthält eine Folge von Funktionssymbolen: $(f_1 ... f_n)$ bedeutet, daß f_1 größer ist als alle restlichen Symbole, d.h. $f_1 \triangleright f_2,...,f_n$.

Die Statusfunktion ist als Liste von Paaren implementiert:

<stat> := ((<function> multiset | left | right)*)

Für jedes Funktionssymbol existiert ein Paar, bestehend aus dem Funktionssymbol selbst und seinem Status.

Ist die gewählte Ordnung diejenige von Donald E. Knuth und Peter B. Bendix, so ist die dazugehörige Gewichtsfunktion als <weight> vorhanden:

<weight> := ((<function> <nat>)*)

Wie auch beim Status ist hier eine Liste von Paaren "Funktionssymbol und sein Gewicht" implementiert.

Hat der Benutzer sich für die Polynomordnung entschieden, so hat <weight> die folgende Form:

```
<weight>  :=  ( ( ( (<function> <var>*) <inter>)*) <nat>)
<inter>   :=  ('*' <inter>+) | ('+' <inter>+) | <arg>
<arg>     :=  <nat> | <var>
```

Die Interpretation besteht aus einer Liste mit zwei Elementen:

- die eigentliche Interpretation:
  Funktionenpattern ($f(x_1,....,x_n)$) und das entsprechende Polynom <inter> (aufgebaut aus Ausdrücken mittels $x_1,....,x_n$, Addition, Multiplikation und Konstanten.

- die untere Grenze der Werte für Variable (siehe 6.4).

Pfade sind die Basis für die Pfadordnungen von David Plaisted und Deepak Kapur et. al.:

```
<path>  :=  (<term>*)
```

Ein Pfad ist somit eine Folge von Termen.

Dekompositionsordnungen zerlegen Terme. Eine solche Zerlegung wurde folgendermaßen implementiert (siehe auch 6.2):

```
<dec>  :=  (<pdec>*)
```

Eine Dekomposition ist eine Liste von Pfaddekompositionen (für jedes Blatt). Eine Pfaddekomposition ist wiederum eine Liste von sogenannten elementaren Dekompositionen:

```
<pdec> :=  (<edec>*)
<edec> :=  (<atom>      Topsymbol des Terms
            <term>      Argument des Terms
            (<term>*)   Restliche Argumente
            (<dec>*)    Dekompositionen der restlichen Argumente
            (<dec>*)    Dekompositionen aller Argumente
            (<dec>*)    Dekompositionen aller Argumente
            <dec>)      Pointer auf die vollständige Dekomposition des Terms
```

9.2 Modul Equation

EQ-CP-KAPUR

Ein-/Ausgabe :

```
<equation>  <equation>  <pos>  ⇒  (<equation>)
```

Diese Funktion wird benötigt von :

KB:KB

EQ-CP-KÜCHLIN

Ein-/Ausgabe :

<equation> <equation> <pos> → (<equation>)

Diese Funktion wird benötigt von :

KB-KB

EQ-CP-KÜCHLIN-ERW

Ein-/Ausgabe :

<equation> <equation> <pos> → (<equation> +)

Diese Funktion wird benötigt von :

KB-KB

EQ-CP-ONE-LIST

Ein-/Ausgabe :

<equation> <equation> <pos> → (<equation>)

Diese Funktion wird benötigt von :

KB-KB

EQ-CP-WINKLER

Ein-/Ausgabe :

<equation> <equation> <pos> → (<equation>)

Diese Funktion wird benötigt von :

KB-KB

EQ-CP-WINKLER-ERW

Ein-/Ausgabe :

<equation> <equation> <pos> → (<equation>)

Diese Funktion wird benötigt von :

KB-KB

EQ-REDUCE

Ein-/Ausgabe :

<term> (<equation>*) → <term>

Diese Funktion wird benötigt von :
KB:KB , KB:SINGLE , COMTES , EQ:REDUCTION , KB:REDUCE-UNCOMP , KB:NORMALIZE-OHNE-INTERRED , EQ:NORMALFORM , KB:NORMALIZE , EQ:SPLIT-LHS1 , EQ:NORMAL , KB:SIMPLIFY-RIGHT

EQ-REDUCE-BOTTOM-UP

Ein-/Ausgabe :

<term> (<equation>*) → <term>

Diese Funktion wird benötigt von :
KB:KB , KB:SINGLE , COMTES , EQ:REDUCTION , KB:SIMPLIFY-RIGHT , KB:REDUCE-UNCOMP , KB:NORMALIZE-OHNE-INTERRED , KB:NORMALIZE , EQ:REDUCE-BOTTOM-UP , EQ:NORMALFORM

EQ-REDUCE-EX-BOTTOM-UP

Ein-/Ausgabe :

<lterm> (<equation>*) → <lterm>

Diese Funktion wird benötigt von :
KB:KB , KB:SINGLE , COMTES , EQ:REDUCTION , KB:SIMPLIFY-RIGHT , KB:REDUCE-UNCOMP , KB:NORMALIZE-OHNE-INTERRED , KB:NORMALIZE , EQ:REDUCE-EX-BOTTOM-UP , EQ:NORMALFORM

EQ-REDUCE-EX-INNERMOST

Ein-/Ausgabe :

<lterm> (<equation>*) → <lterm>

Diese Funktion wird benötigt von :
KB:KB , KB:SINGLE , COMTES , EQ:REDUCTION , KB:SIMPLIFY-RIGHT , KB:REDUCE-UNCOMP , KB:NORMALIZE-OHNE-INTERRED , KB:NORMALIZE , EQ:REDUCE-EX-INNERMOST , EQ:NORMALFORM

EQ-REDUCE-EX-OUTERMOST

Ein-/Ausgabe :

<lterm> (<equation>*) → <lterm>

Diese Funktion wird benötigt von :
KB=KB , KB=SINGLE , COMTES , EQ=REDUCTION , KB=SIMPLIFY-RIGHT , KB=REDUCE-UNCOMP , KB=NORMALIZE-OHNE-INTERRED , KB=NORMALIZE , EQ=NORMALFORM

EQ-REDUCE-EX-TOP-DOWN

Ein-/Ausgabe :

<lterm> (<equation>*) → <lterm>

Diese Funktion wird benötigt von :
KB=KB , KB=SINGLE , COMTES , EQ=REDUCTION , KB=SIMPLIFY-RIGHT , KB=REDUCE-UNCOMP , KB=NORMALIZE-OHNE-INTERRED , KB=NORMALIZE , EQ=NORMALFORM

EQ-REDUCE-INNERMOST

Ein-/Ausgabe :

<term> (<equation>*) → <term>

Diese Funktion wird benötigt von :
KB=KB , KB=SINGLE , COMTES , EQ=REDUCTION , KB=SIMPLIFY-RIGHT , KB=REDUCE-UNCOMP , KB=NORMALIZE-OHNE-INTERRED , KB=NORMALIZE , EQ=REDUCE-INNERMOST , EQ=NORMALFORM

EQ-REDUCE-OUTERMOST

Ein-/Ausgabe :

<term> (<equation>*) → <term>

Diese Funktion wird benötigt von :
KB=KB , KB=SINGLE , COMTES , EQ=REDUCTION , KB=SIMPLIFY-RIGHT , KB=REDUCE-UNCOMP , KB=NORMALIZE-OHNE-INTERRED , KB=NORMALIZE , EQ=NORMALFORM

EQ-REDUCE-TOP-DOWN

Ein-/Ausgabe :

<term> (<equation>*) → <lterm>

Diese Funktion wird benötigt von :
KB:KB , KB:SINGLE , COMTES , EQ:REDUCTION , KB:SIMPLIFY-RIGHT , KB:REDUCE-UNCOMP , KB:NORMALIZE-OHNE-INTERRED , KB:NORMALIZE , EQ:NORMALFORM

9.3 Modul Knuth-Bendix

KB-NORMALIZE

Ein-/Ausgabe :

(<equation>*) (<equation>*) (<equation>*) <ordering>
→ (<equation>*) (<equation>*) (<equation>*) <ordering>

Diese Funktion wird benötigt von :
KB:INIT-GLOBAL-VARS , KB:KB

KB-NORMALIZE-OHNE-INTERRED

Ein-/Ausgabe :

(<equation>*) (<equation>*) (<equation>*) <ordering>
→ (<equation>*) (<equation>*) (<equation>*) <ordering>

Diese Funktion wird benötigt von :
KB:KB

KB-POSTPONE

Ein-/Ausgabe :

<equation> (<equation>*) → (<equation>*)

Diese Funktion wird benötigt von :
KB:NORMALIZE-OHNE-INTERRED , KB:NORMALIZE

KB-SINGLE

Ein-/Ausgabe :

(<equation>*) <ordering> → (<equation>*) (<equation>*) <ordering>

Diese Funktion wird benötigt von :
COMTES , KB:KB

9.4 Modul Termination

TR-IRDS

Ein-/Ausgabe :

<term> <term> <prec> <stat> → t | nil

Diese Funktion wird benötigt von :
KB:REDUCE-UNCOMP , KB:NORMALIZE-OHNE-INTERRED , KB:NORMALIZE , TR:COMPARISON , TR:COMP-FILE , TR:COMP , COMTES , KB:KB , KB:SINGLE , TR:SHOW-UNCOMPARABLES-COMP , TR:SHOW-UNCOMPARABLES , TR:DIRECT-UNCOMP , TR:DIRECT-COMP , TR:DIRECT , TR:DIRECT-HELP

TR-KBO

Ein-/Ausgabe :

<term> <term> <prec> <stat> <weight> → t | nil

Diese Funktion wird benötigt von :
KB:REDUCE-UNCOMP , KB:NORMALIZE-OHNE-INTERRED , KB:NORMALIZE , TR:COMPARISON , TR:COMP-FILE , TR:COMP , COMTES , KB:KB , KB:SINGLE , TR:SHOW-UNCOMPARABLES-COMP , TR:SHOW-UNCOMPARABLES , TR:DIRECT-UNCOMP , TR:DIRECT-COMP , TR:DIRECT , TR:KBO , TR:KBO-LEX , TR:DIRECT-HELP

TR-KNS

Ein-/Ausgabe :

<term> <term> <prec> <stat> <weight> → t | nil

Diese Funktion wird benötigt von :
KB:REDUCE-UNCOMP , KB:NORMALIZE-OHNE-INTERRED , KB:NORMALIZE , TR:COMPARISON , TR:COMP-FILE , TR:COMP , COMTES , KB:KB , KB:SINGLE , TR:SHOW-UNCOMPARABLES-COMP , TR:SHOW-UNCOMPARABLES , TR:DIRECT-UNCOMP , TR:DIRECT-COMP , TR:DIRECT , TR:KNS-PATH , TR:KNS-LEX , TR:DIRECT-HELP

TR-POL

Ein-/Ausgabe :

<term> <term> <weight> ⇒ t | nil

Diese Funktion wird benötigt von :
KB:REDUCE-UNCOMP , KB:NORMALIZE-OHNE-INTERRED , KB:NORMALIZE , TR:COMPARISON , TR:COMP-FILE , TR:COMP , COMTES , KB:KB , KB:SINGLE , TR:SHOW-UNCOMPARABLES-COMP , TR:SHOW-UNCOMPARABLES , TR:DIRECT-UNCOMP , TR:DIRECT-COMP , TR:DIRECT , TR:DIRECT-HELP

TR-PSDS

Ein-/Ausgabe :

<term> <term> <prec> <stat> ⇒ t | nil

Diese Funktion wird benötigt von :
KB:REDUCE-UNCOMP , KB:NORMALIZE-OHNE-INTERRED , KB:NORMALIZE , TR:COMPARISON , TR:COMP-FILE , TR:COMP , COMTES , KB:KB , KB:SINGLE , TR:SHOW-UNCOMPARABLES-COMP , TR:SHOW-UNCOMPARABLES , TR:DIRECT-UNCOMP , TR:DIRECT-COMP , TR:DIRECT , TR:DIRECT-HELP

TR-PSO

Ein-/Ausgabe :

<term> <term> <prec> <stat> ⇒ t | nil

Diese Funktion wird benötigt von :
KB:REDUCE-UNCOMP , KB:NORMALIZE-OHNE-INTERRED , KB:NORMALIZE , TR:COMPARISON , TR:COMP-FILE , TR:COMP , COMTES , KB:KB , KB:SINGLE , TR:SHOW-UNCOMPARABLES-COMP , TR:SHOW-UNCOMPARABLES , TR:DIRECT-UNCOMP , TR:DIRECT-COMP , TR:DIRECT , TR:DIRECT-HELP

TR-RPO

Ein-/Ausgabe :

<term> <term> <prec> <stat> ⇒ t | nil

Diese Funktion wird benötigt von :
KB:REDUCE-UNCOMP , KB:NORMALIZE-OHNE-INTERRED , KB:NORMALIZE , TR:COMPARISON , TR:COMP-FILE , TR:COMP , COMTES , KB:KB , KB:SINGLE , TR:SHOW-UNCOMPARABLES-COMP , TR:SHOW-UNCOMPARABLES , TR:DIRECT-UNCOMP , TR:DIRECT-COMP , TR:DIRECT , TR:RPO-LEX , TR:RPO , TR:DIRECT-HELP

9.5 Modul Variables

ALL-RULES	:	Aktuelles Regelsystem
AUTO	:	Bool'sche Variable für eine automatische Vervollständigung (noch nicht implementiert, d.h. *AUTO* = nil)
COMTES-COMTES-DIR	:	Ausgabe-Directory
COMTES-INTERN-DIR	:	Directory für "geparste" Dateien
COMTES-SPEC-DIR	:	Spezifikations-Directory
COMTES-STATISTIC-FILE	:	Datei, wo die Statistik abgelegt wird
CP-GLEICH-NACH	:	Anzahl der kritischen Paare, deren Normalformen bzgl. des aktuellen Regelsystems syntaktisch gleich sind
CP-GLEICH-ZAEHLER	:	Anzahl der trivialen kritischen Paare (syntaktisch gleich)
CP-KRIT-GREIFT	:	Anzahl der kritischen Paare, die nicht bearbeitet werden müssen bei Subconnectedness-Kriterien (EQ=CP-KAPUR, EQ=CP-KÜCHLIN, EQ=CP-KÜCHLIN-ERW, EQ=CP-WINKLER, EQ=CP-WINKLER-ERW)
CRIT-GEN	:	Anzahl der kritischen Paare
KRITERIUM	:	Verfahren zur kritischen Paarbildung (entsprechend ihrer Funktionen)
NEXT-RULE-NO	:	Nummer der Regel, die als nächstes generiert wird
NORMALIZE	:	Gibt an, ob interreduziert (KB=NORMALIZE) wird oder nicht (KB=NORMALIZE-OHNE-INTERRED)
OPS	:	Enthält die Signatur der aktuellen Spezifikation
RED	:	Anzahl der Reduktionsschritte
RULES-GEN	:	Anzahl der generierten Regeln
SPEC-NAME	:	Name der Spezifikation
STRATEGIE	:	Name der Reduktionsstrategie
TIME-NORM	:	Zeit für die Interreduktion
TIME-ORD	:	Zeit für das Vergleichen von Termpaaren
TIME-RED	:	Zeit für die Reduktion
TIME-UNIF	:	Zeit für die Unifikation
TOTAL-TIME	:	Gesamtzeit
TREE	:	AVL-Baum zur Speicherung der bearbeiteten Überlappungen
UEBERLAPPTE-REGELN	:	Diejenigen Regeln, mit denen bereits alle kritischen Paare gebildet wurden
UNIF	:	Anzahl der generierten Unifikatoren
VARIABLE	:	Gewicht der Variablen (bei Verwendung der KBOS)

10 Literatur

Nachfolgend sind die Kürzel derjenigen Literaturstellen fett gedruckt, die sich unmittelbar auf die Implementierung beziehen.

[ABGM86] J. Avenhaus, B. Benninghofen, R. Göbel, K. Madlener
TRSPEC: A Term Rewriting Based System for Algebraic Specifications
Proceedings of the 8th International Conference on Automated Deduction, LNCS 230, Oxford, England, July 1986, pp. 665-667

[AGGMS87] J. Avenhaus, R. Göbel, B. Gramlich, K. Madlener, J. Steinbach
TRSPEC: A Term Rewriting Based System for Algebraic Specifications
Proceedings of the 1st International Workshop on Conditional Term Rewriting Systems, LNCS 308, Orsay (Paris), France, July 8 - 10, 1987, pp. 245-248

[AHM89] S. Anantharaman, J. Hsiang, J. Mzali
SbReve2: A Term Rewriting Laboratory with (AC-) Unfailing Completion
Proceedings of 3rd RTA'89, Chapel Hill, North Carolina, USA, LNCS 355, April 1989, N. Dershowitz (Ed.), pp. 533-537

[Ai85] H. Ait-Kaci
An algorithm for finding a minimal recursive path ordering
R.A.I.R.O. Theoretical Informatics, Vol. 19, No. 4, 1985, pp. 359-382

[Al87] E. Altendorf
Terminationskriterien für Termersetzungssysteme
Studienarbeit, Technische Universität Berlin, Oktober 1987

[Al89] E. Altendorf
Termination von Termersetzungssystemen: Theoretische Untersuchung und Implementierung von KBO, RPO und KNS
Diplomarbeit, Technische Universität Berlin, 1989

[AM89] J. Avenhaus, K. Madlener
Term Rewriting and Equational Reasoning
In: Formal Techniques in Artificial Intelligence, A Source Book, R.B. Banerji (ed.), Academic Press, 1989

[AMS89] J. Avenhaus, K. Madlener, J. Steinbach
COMTES - An Experimental Environment for the Completion of Term Rewriting Systems
Proc. of 3rd RTA '89, Chapel Hill, North Carolina, USA, LNCS 355, April 1989, N. Dershowitz (Ed.), pp. 542-546

[Av84a] J. Avenhaus
On Congruences and Normal Forms Definable by Term Rewriting Systems
Interner Bericht 116/84, Fachbereich Informatik, Universität Kaiserslautern, 1984

[Av84b] J. Avenhaus
On the Termination of the Knuth-Bendix Completion Algorithm
Interner Bericht 120/84, Fachbereich Informatik, Universität Kaiserslautern, 1984

[Av86a] J. Avenhaus
On the Descriptive Power of Term Rewriting Systems
Journal of Symbolic Computation 2, 1986, pp. 109-122

[Av86b] J. Avenhaus
Vorlesungsmitschrift zur Vorlesung Reduktionssysteme
Universität Kaiserslautern, 1986

[Ba81] G. Bauer
Zur Darstellung von Monoiden durch konfluente Regelsysteme
Dissertation, Fachbereich Informatik, Universität Kaiserslautern, W. Germany, Februar 1981

[BCCKSV89] M. Bidoit, F. Capy, C. Choppy, S. Kaplan, F. Schlienger, F. Voisin
ASSPEGIQUE: An Integrated Specification Environment
Proc. of 3rd RTA '89, Chapel Hill, North Carolina, USA, LNCS 355, April 1989, N. Dershowitz (Ed.), p. 547

[BD88] L. Bachmair, N. Dershowitz
Critical Pair Criteria for Completion
Journal of Symbolic Computation 6, August 1988, pp. 1-18

[BDP87] L. Bachmair, N. Dershowitz, D.A. Plaisted
Completion Without Failure
CREAS '87

[Be86] A. Ben Cherifa
Preuves de terminaison de systemes de reecriture - un outil fonde sur les interpretations polynomiales
These de doctorat, Universite de Nancy I, Octobre 1986

[BL86] A. Ben Cherifa, P. Lescanne
An Actual Implementation of a Procedure That Mechanically Proves Termination of Rewriting Systems Based on Inequalities Between Polynomial Interpretations
8th CADE, Oxford, England, LNCS 230, July 1986, pp. 42-51

[BL87] A. Ben Cherifa, P. Lescanne
Termination of Rewriting Systems by Polynomial Interpretations and its Implementation
Centre de Recherche en Informatique de Nancy, France, 1987

[Bu85] B. Buchberger
Basic features and development of the critical-pair/completion procedure
Proceedings of the 1st International Conference on Rewriting Techniques and Applications, LNCS 202, Dijon, France, May 1985, pp. 1-45

[Ch84] G. Choque
How to compute a complete set of minimal incrementations with the recursive decomposition ordering?
Internal Report 84-R-056, Centre de recherche en informatique de Nancy, France, 1984

[Ch89] J. Christian
Fast Knuth-Bendix Completion: Summary
Proc. of 3rd RTA '89, Chapel Hill, North Carolina, USA, LNCS 355, April 1989, N. Dershowitz (Ed.), pp. 551-555

[Da88] M. Dauchet
Termination of Rewriting is Undecidable in the One-Rule Case
MFCS, Carlsbad, CSSR, August 1988

[De79] N. Dershowitz
A note on simplification orderings
Information Processing Letters, Vol. 9, No. 5, November 1979, pp. 212-215

[De80] N. Dershowitz
On representing ordinals up to Γ_0
Unpublished note, Department of Computer Science, University of Illinois, Urbana, Illinois, 1980

[De82] N. Dershowitz
Orderings for term rewriting systems
Journal of Theoretical Computer Science, Vol. 17, No. 3, March 1982, pp. 279-301

[De83] N. Dershowitz
Well-founded orderings
Technical Report ATR-83(8478)-3, Information Sciences Research Office, The Aerospace Corporation, El Segundo, California, May 1983

[De87] N. Dershowitz
Termination of Rewriting
Journal of Symbolic Computation 3, 1987, pp. 69-116

[DF85] D. Detlefs, R. Forgaard
A procedure for automatically proving the termination of a set of rewrite rules
Proceedings of the 1st International Conference on Rewriting Techniques and Applications, LNCS 202, Dijon, France, May 1985, pp. 255-270

[DJ89] N. Dershowitz, J.-P. Jouannaud
Rewriting Systems
Handbook of Theoretical Computer Science, eds.: A. Meyer, M. Nivat, M. Peterson, D. Perrin, (to appear 1989)

[DM79] N. Dershowitz, Z. Manna
Proving termination with multiset orderings
Communications of the Association for Computing Machinery, Vol. 22, No. 8, August 1979, pp. 465-476

[DMT86] N. Dershowitz, L. Marcus, A. Tarlecki
Existence, Uniqueness, and Construction of Rewrite Systems
SIAM J. Computing, Vol. 17, No. 4, August 1988, pp. 629-639

[Fa88] C. Falter
Erweiterbare Benutzerschnittstelle für das TRSPEC-System
Projektarbeit, Fachbereich Informatik, Universität Kaiserslautern, WS 88/89

[Fe88] R. Fettig
Dynamische Multisetordnungen für Grundterme
Projektarbeit, Fachbereich Informatik, Universität Kaiserslautern, W. Germany, Mai 1988

[FG83] R. Forgaard, J. V. Guttag
REVE: A term rewriting system generator with failure-resistant Knuth-Bendix
Proceedings of an NSF Workshop on the rewrite rule laboratory, September 1983 (ed. Guttag, Kapur, Musser) General-Electrics-Rep. No. 84GEN008, April 1984

[FGJM84] K. Futatsugi, J.A. Goguen, J.-P. Jouannaud, J. Meseguer
Principles of OBJ2
CRIN Report 84-R-066, Nancy, France, 1984, also in Proc. 12th ACM Symp. on Principles of Programming Languages, New Orleans, 1985, pp. 52-66

[Fo84] R. Forgaard
A program for generating and analyzing term rewriting systems
Master's Thesis, Massachusetts Institute of Technology, Laboratory for Computer Science, MIT/LCS/TR-343, September 1984

[GB85] J. Gallier, R. V. Book
Reductions in tree replacement systems
Theoretical Computer Science 37, 1985

[Ge89] O. Geupel
Overlap Closures and Termination of Term Rewriting Systems
MIP-8922, Fakultät für Mathematik und Informatik, Universität Passau, Juli 1989

[GG87] H. Ganzinger, R. Giegerich
A Note on Termination in Combinations of Heterogeneous Term Rewriting Systems
EATCS 31, 1987

[GKKMMW88] J. Goguen, C. Kirchner, H. Kirchner, A. Megrelis, J. Meseguer, T. Winkler
An Introduction to OBJ 3
Centre de Recherche en Informatique de Nancy, 88-R-001

[Gn87] I. Gnaedig
Knuth-Bendix Procedure and Non Deterministic Behavior - An Example -
EATCS 32, 1987, pp. 86-92

[HKK89] M. Hermann, C. Kirchner, H. Kirchner
Implementation of Term Rewriting Systems
Draft, Nancy, France, 1989

[HL78] G. Huet, D.S. Lankford
On the uniform halting problem for term rewriting systems
Rapport Laboria 283, IRIA, Paris, INRIA Rocquencourt, France, March 1978

[HM88] J. Hsiang, J. Mzali
SbReve Users Guide
Technical Report, LRI, 1988

[HO80] G. Huet, D. Oppen
Equations and rewrite rules: A survey
Formal languages: Perspectives and open problems (R. Book, ed.), Academic Press, New York, 1980, pp. 349-405

[HP86] M. Hermann, I. Privara
On Nontermination of Knuth-Bendix Algorithm
13th ICALP, Rennes, July 1986, LNCS 226, pp. 146-156

[HS76] E. Horowitz, S. Sahni
Fundamentals of Data Structures
Pitmann 1976, pp. 442-456

[Hu80] G. Huet
Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems
Journal of the Association for Computing Machinery, Vol. 27, No. 4, October 1980, pp. 797-821

[Hu81] G. Huet
A complete proof of correctness of the Knuth-Bendix completion algorithm
Journal of Computer and System Sciences 23, 1981

[JL82] J.-P. Jouannaud, P. Lescanne
On multiset orderings
Information Processing letters 15(2), September 1982, pp. 57-63

[JL87] J.-P. Jouannaud, P. Lescanne
Rewriting systems
Technology and Science of Informatics, Vol. 6, No. 3, 1987, pp. 181-199

[JLR82] J.-P. Jouannaud, P. Lescanne, F. Reinig
Recursive decomposition ordering
I.F.I.P. Working Conference on Formal Description of Programming Concepts II (D. Bjørner, ed.), North Holland, Garmisch Partenkirchen, W. Germany, 1982, pp. 331-348

[Jo87] J.-P. Jouannaud, P. Lescanne
Proving Completeness of Completion Procedures
RTA '87, pp. 1-32

[KB70] D. E. Knuth, P. B. Bendix
Simple word problems in universal algebras
Computational problems in abstract algebra (J. Leech, ed.), Pergamon Press, 1970, pp. 263-297

[KL80] S. Kamin, J.-J. Lévy
Attempts for generalizing the recursive path orderings
Unpublished manuscript, Department of Computer Science, University of Illinois, Urbana, Illinois, February 1980

[KMN84] D. Kapur, D. R. Musser, P. Narendran
Only Prime Superpositions Need to be Considered in the Knuth-Bendix Completion procedure
Unpublished report, General Electric Research and Development, Schenectady, NY, December 1984, also in JSC 4, 1988, pp. 19-36

[KN85] M. S. Krishnamoorthy, P. Narendran
Note on recursive path ordering
Theoretical Computer Science 40, 1985, pp. 323-328

[KNS85] D. Kapur, P. Narendran, G. Sivakumar
A path ordering for proving termination of term rewriting systems
Proceedings of the 10th Colloquium on Trees in Algebra and Programming, LNCS 185, 1985, pp. 173-187

[KS84] D. Kapur, G. Sivakumar
Architecture of and experiments with RRL, a rewrite rule laboratory
Proceedings of a NSF Workshop on the rewrite rule laboratory, September 1983 (ed. Guttag, Kapur, Musser) General-Electrics-Rep. No. 84GEN008, April 1984

[Ku82] W. Küchlin
Some reduction strategies for algebraic term rewriting
SIGSAM Bulletin 16, 4, November 1982

[Ku85] W. Küchlin
A Confluence Criterion Based on the Generalized Newman Lemma
Proceedings of the European Conference of Computer Algebra, Linz, 1985, pp. 390-399

[KZ88] D. Kapur, H. Zhang
RRL: A Rewrite Rule Laboratory

[KZ89] D. Kapur, H. Zhang
An Overview of Rewrite Rule Laboratory (RRL)
Proceedings of 3rd RTA '89, Chapel Hill, North Carolina, USA, LNCS 355, April 1989, N. Dershowitz (Ed.), pp. 559-563

[La77] D. S. Lankford
Some approaches to equality for computational logic: A survey and assessment
Report ATP-36, Automatic Theorem Proving Project, Departments of Mathematics and Computer Science, University of Texas, Austin, Texas 78712, Spring 1977

[Le81a] P. Lescanne
Two implementations of the recursive path ordering on monadic terms
Proceedings of the 19th Allerton Conference on Communication, Control and Computing, Allerton House, Monticello, Illinois, September 1981, pp. 634-643

[Le81b] P. Lescanne
Decomposition ordering as a tool to prove the termination of rewriting systems
Proceedings of the 7th International Joint Conference on Artificial Intelligence, Vancouver, Canada, August 1981, pp. 548-550

[Le82] P. Lescanne
Some properties of decomposition ordering, a simplification ordering to prove termination of rewriting systems
R.A.I.R.O. Theoretical Informatics, Vo. 16, No. 4, 1982, pp. 331-347

[Le83a] P. Lescanne
How to prove termination? An approach to the implementation of a new recursive decomposition ordering
Proceedings of an NSF Workshop on the Rewrite Rule Laboratory (Guttag, Kapur, Musser, eds.), General Electric Research and Development Center Report 84GEN008, September 1983, pp. 109-121

[Le83b] P. Lescanne
Computer experiments with the REVE term rewriting system generator
Internal Report 83-R-037, Centre de recherche en informatique de Nancy, France, 1983

[Le84] P. Lescanne
Uniform termination of term rewriting systems - the recursive decomposition ordering with status
Proceedings of the 9th Colloquium on Trees in Algebra and Programming (B. Courcelle, ed), Cambridge University Press, Bordeaux, France, 1984, pp. 182-194

[Le86a] P. Lescanne
Divergence of the Knuth-Bendix completion procedure and termination orderings
Bulletin of the European Association for Theoretical Computer Science, No. 30, 1986, pp. 80-83

[Le86b] J.J. Levy
Computation and Deduction
Course Notes 7, Unpublished Manuscript, Lecture Notes given at Pittsburgh in 1986

[Le87a] P. Lescanne
REVE - a Rewrite Rule Laboratory
4th STACS '87, Passau, FRG, LNCS 247, Februay 1987, pp. 482-483

[Le87b] P. Lescanne
On the recursive decomposition ordering with lexicographical status and other related orderings
preprint, December 1987

[Le88] P. Lescanne
Completion Procedures as Transition Rules + Control
CRIN, France, 1988

[LS77] R. J. Lipton, L. Synder
On the halting of tree replacement systems
Proceedings of a Conference on Theoretical Computer Science, Waterloo, Ontario, August 1977, pp. 43-46

[Ma86a] U. Martin
Extension Functions for Multiset Orderings
Dept. of Computer Science, University of Manchester, February 1986, also in IPL 26 '87/88, pp. 181-186

[Ma86b] U. Martin
Multiset Orderings
Technical Report Series UMCS-86-5-1, Dept. of Computer Science, University of Manchester, 1986

[Ma86c] U. Martin
Doing Algebra with Reve
Technical Report Series UMCS-86-10-4, Dept. of Computer Science, University of Manchester, 1986

[Ma87] U. Martin
How to choose the weights in the Knuth-Bendix ordering
Proceedings of the 2nd International Conference on Rewriting Techniques and Applications, LNCS 256, Bordeaux, France, May 25 - 27, 1987, pp. 42-53

[Me83] Y. Metivier
About the Rewriting Systems Produced by the Knuth-Bendix Completion Algorithm
IPL 16, 1983, pp. 31-34

[MN70] Z. Manna, S. Ness
On the termination of Markov algorithms
Proceedings of the 3rd Hawaii International Conference on System Sciences, Honolulu, Hawaii, 1970, pp. 789-792

[MS86] J. Müller, J. Steinbach
Topologische Multisetordnungen
Proceedings of the 10th German Workshop and 2nd Austrian Conference on Artificial Intelligence, Ottenstein, Austria, Informatik Fachberichte 124, September 1986, pp. 254-264

[Ok86] M. Okada
Ackermann's ordering and its relationship with ordering systems in term rewriting theory

[Pa89] D. Paul
Implementierung von Polynomordnungen
Projektarbeit, Universität Kaiserslautern, 1989

[Pe81] A. Pettorossi
Comparing and putting together recursive path ordering, simplification orderings and non-ascending property for termination proofs of term rewriting systems
Proceedings of the 8th EATCS International Colloquium on Automata, Languages and Programming, LNCS 115, Acre, Israel, July 1981, pp. 432-447

[Pl78a] D. A. Plaisted
A recursively defined ordering for proving termination of term rewriting systems
Report UIUCDCS-R-78-943, Department of Computer Science, University of Illinois, Urbana, IL, September 1978

[Pl78b] D. A. Plaisted
Well-founded orderings for proving termination of systems of rewrite rules
Report UIUCDCS-R-78-932, Department of Computer Science, University of Illinois, Urbana, IL, July 1978

[Pl85] D. A. Plaisted
The undecidability of self-embedding for term rewriting systems
Information Processing Letters 20, February 1985, pp. 61-64

[Re81] F. Reinig
Les ordres de décomposition: un outil incrémental pour prouver la terminaison finie de systèmes de réécriture de termes
Thèse présentée pour l'obtention du grade de docteur de 3ème cycle en informatique, Université de Nancy I & Centre de recherche en informatique de Nancy, France, Octobre 1981

[Re89] U.S. Reddy
Rewriting Techniques for Program Synthesis
Proc. of 3rd RTA '89, Chapel Hill, North Carolina, USA, LNCS 355, April 1989, N. Dershowitz (Ed.), pp. 388-403

[RJ81] F. Reinig, J.-P. Jouannaud
Decomposition orderings, a new family of recursive simplification orderings
Report CRIN 81-R-040, Centre de recherche en informatique de Nancy, France, June 1981

[Ro71] J. A. Robinson
Computational Logic: The unification computation
in: Machine Intelligence 6, B. Meltzer & D. Michie (eds.), Edinburgh Univ. Press, Scotland, 1971, pp. 63-72

[Ro88] J. Rouyer
Preuves de Terminaison de Systemes de Reecriture Fondees sur les Interpretations Polynomiales - Une Methode Basee sur le Theoreme de Sturm
France, Juin 1988

[Ru87] M. Rusinowitch
Path of subterms ordering and recursive decomposition ordering revisited
Journal of Symbolic Computation 3 (1 & 2), 1987, pp. 117-131

[Sa84] K. Sakai
An Ordering Method for Term Rewriting Systems
Technical Report TR-062, ICOT Research Center, Tokyo, Japan, April 1984

[Sa87] A. Sattler-Klein
Minimierung der kritischen Paare im Vervollständigungssystem COSY
Diplomarbeit, Universität Kaiserslautern, November 1987

[Si87] C. C. Sims
Verifying nilpotence
Journal of Symbolic Computation 3, 1987, pp. 231-247

[So88] I. Sonntag
Implementierung und Vergleich von Subconnectedness Kriterien
Projektarbeit, Universität Kaiserslautern, 1988

[St83] M. E. Stickel
A note on leftmost-innermost reduction
SIGSAM Bulletin 17, 1983

[St86] J. Steinbach
Ordnungen für Termersetzungssysteme
Diplomarbeit, Universität Kaiserslautern, Juni 1986

[St88] J. Steinbach
Comparison of simplification orderings
SEKI REPORT SR-88-02, Artificial Intelligence Laboratories, Department of Computer Science, University of Kaiserslautern, W. Germany, February 1988

[St88] J. Steinbach
Term orderings with status
SEKI REPORT SR-88-12, Universität Kaiserslautern, 1988

[St89a] J. Steinbach
Extensions and Comparison of simplification Orderings
Proceedings of 3rd RTA, Chapel Hill, USA, April 1989, LNCS 355, pp. 434-448

[St89b] J. Steinbach
Comparing on strings: Iterated syllable ordering & recursive path ordering
SEKI-Report, University of Kaiserslautern, 1989

[SW84] J. Steinbach, E. Wagner
Praktikum 'Effiziente Algorithmen': Thema 'Unifikation'
Universität Kaiserslautern, SS 1984

[Wa86] E. Wagner
Strategien für den Knuth-Bendix Algorithmus
Diplomarbeit, Universität Kaiserslautern, Juni 1986

[Wi84] F. Winkler
The Church Rosser Property in Computer Algebra and Special Theorem Proving: An Investigation of Critical Pair Completion algorithms
Dissertation der Johannes-Kepler-Universität, Linz, 1984

[Wi85] F. Winkler
Reducing the Complexity of the Knuth-Bendix Completion Algorithm: A "Unification" of Different Approaches
Eurocal 1985, pp. 378-389

[Wi88] D. Wissmann
Applying rewriting techniques to groups with power-commutation-presentations
Proceedings of the 1988 International Symposium on Symbolic and Algebraic Computation, Rome, July 4 - 8, 1988

[WS84] E. Wagner, J. Steinbach
Implementierung einer Grundversion des Knuth-Bendix Algorithmus
Projektarbeit, Universität Kaiserslautern, Sommer 1984

[Ze89] M. Zehnter
Theorieverträgliche Ordnungen - Eine Übersicht
Projektarbeit, Universität Kaiserslautern, 1989

[ZK89] H. Zhang, D. Kapur
Consider Only General Superpositions in Completion Procedures
Proc. of 3rd RTA '89, Chapel Hill, North Carolina, USA, LNCS 355, April 1989, N. Dershowitz (Ed.), pp. 513-527

11 Index