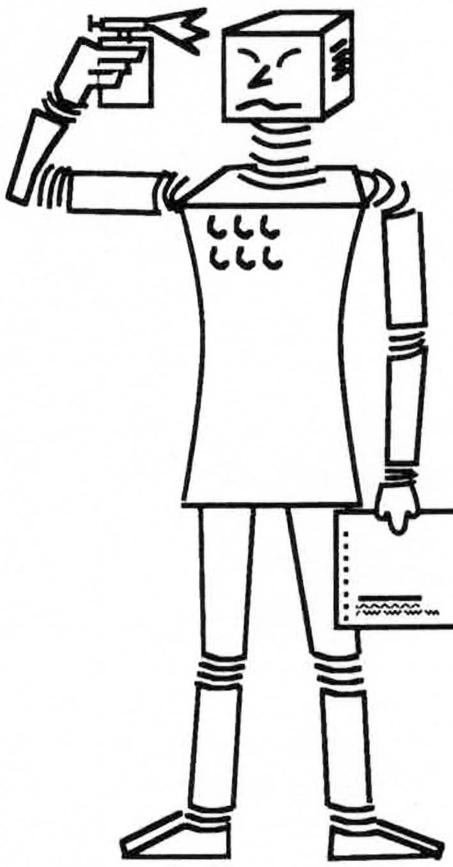
Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern

SEKI - REPORT



Ext. Abstracts of the 1st German Workshop Term Rewriting: Theory and Applications

Jürgen Müller, Harald Ganzinger (Eds.)
SEKI Report SR-89-02

Preface

The subgroups 1.2.1 Deduction Systems and 2.1.3 Implementation of Programming Languages of the Gesellschaft für Informatik arranged their first workshop "**Term Rewriting: Theory and Application**" from March 6 to March 8, 1989 at the University of Kaiserslautern.
The aim of the workshop was to bring together theorists and practitioners on Term rewriting systems, completion algorithms, narrowing approaches, completion theorem provers etc. from the German speaking countries. 43 participants from Germany and Austria have got an interesting overview of the actual rewrite activities within 27 talks. The talks are separated into blocks of:

- conditional term rewriting
- inductive proofs via completion
- functional programming and rewrite systems
- applications of rewriting techniques
- special strategies
- theoretics within equational theories.

There also was a system demonstration and poster section where 10 systems were presented.
The workshop has shown how broad the applications of term rewriting techniques have became. Special fields as program specification and verification, automated translation of languages, code generators, automated theorem proving, logical-functional programming languages and simulation of parallel processes were considered.

This report will give a brief overview on the topics of the workshop. The extended abstracts are [almost] arranged in groups of their special fields.

March 1989

Jürgen Müller [University of Kaiserslautern]
Harald Ganzinger [University of Dortmund]

Compilation von Termersetzungssystemen

Dietmar Wolz
Paul Boehm

Technische Universität Berlin
Fachbereich Informatik (20)
Institut für Software und Theoretische Informatik
Franklinstraße 28/29, Sekr. FR 6-1
D-1000 Berlin 10

Berlin, Januar 1989

Abstract

Algebraische Spezifikationssprachen gewinnen in den letzten Jahren in der Forschung immer mehr an Bedeutung. Sie ermöglichen die formale Beschreibung von Datentypen und Softwaresystemen unabhängig von ihrer konkreten Repräsentation und ohne Beziehung zu speziellen Eigenschaften einer Programmiersprache oder eines Betriebssystems. Ihre mathematisch klar definierte Semantik machen sie unabhängig von technologischen Veränderungen und zu einer zuverlässigen Grundlage der Dokumentation und Implementierung von Softwaresystemen.

Die Compilation wird als ein Verfahren zur effizienten Auswertung von ausführbaren Gleichungs-Spezifikationen mit initialer Semantik vorgestellt. Dieses Verfahren verbessert die Verwendbarkeit dieser Spezifikationen als Prototyp der spezifizierten Software.

Termersetzung wird dazu verwendet, einen Repräsentanten der durch die Spezifikation definierten Kongruenzklasse eines Terms zu bestimmen. Die Ersetzungsregeln müssen bestimmten Kriterien genügen, damit die Termination des Verfahrens und die Eindeutigkeit des Resultats sichergestellt sind.

Bei der Compilation wird ein Termersetzungssystem in mehreren Phasen in eine Menge von Entscheidungsbäumen, in eine Menge von Funktionsdefinitionen, in den Code einer abstrakten Maschine, und schließlich in ausführbaren Maschinencode transformiert.

Ersetzungsrelationen auf partiellen Termen und Eigenschaften verschiedener Ableitungsstrategien werden vorgestellt. Ein Termersetzungs-Algorithmus mit "lazy"-Strategie wird angegeben, seine Korrektheit und Termination wird bewiesen.

Der Algorithmus wird, entsprechend den einzelnen Compilationsphasen, schrittweise konkretisiert. Die Korrektheit der Transformationen wird diskutiert.

Die abstrakte Termersetzungs-Maschine LATERM wird definiert. Diese Maschine ermöglicht Ableitungen bez. der "lazy"-Strategie mit zwei unterschiedlichen Verfahren und kann zusätzlich innermost-Ableitungen effizient durchführen.

Why We Need More Interaction In Inductive Theorem Proving

Ralf-Detlef Kutsche
TU Berlin, FB Informatik, FR 6-2
Franklinstr. 28/29, D-1000 Berlin 10

1. Introduction

Based on the work of Musser [Mus80], Goguen [Gog80], Huet & Hullot [HH82], Jouannaud & Kounalis [JK86] and others, we have shown in Hofbauer & Kutsche [HK88] how to prove inductive theorems over a term rewriting system under certain conditions by just applying rewrite steps between distinguished pairs of terms. Our results allow to weaken the premises of both (ground) confluence and termination : Only critical pairs of the underlying system *on* the rules to be proved have to be considered. Secondly we split the set of those rules into two parts, one of which has to fulfill certain termination requirements, but the other does not. Furthermore we allow the use of (already proved) lemmata, splitted in the same manner.

Here I want to stress the pragmatical aspects of proving inductive theorems, based on some experience in applying our results to a few examples, cf. again [HK88]. My emphasis lies on pointing out severe difficulties, arising if one wants to proceed fully automatically, rather than giving a solution of them. My demand is for human-oriented interactive systems, to integrate the users expertise and intuition of a theorem and its possible proof with the speed of a computer system to perform necessary but tedious computations.

2. Basic theory

We assume the reader to be familiar with basic notations for term rewriting systems, as given e.g. in [Huet80] and many others, and just recall a few definitions and our main theorem from [HK88].

Definitions

Let R be a TRS. The set of equations $\mathrm{ITh}(R) := \{\ s{=}t \mid s\sigma \xleftrightarrow{*}_{R} t\sigma$ for all ground substitutions $\sigma\ \}$ is called *inductive theory* of R . (For convenience we often speak of a *set of rules* to be in the inductive theory of a TRS, always meaning the *associated set of equations.*)

A term t is called *ground-reducible* (or: *quasi-reducible*) *under* R iff $t\sigma$ is R-reducible for all ground substitutions σ , a TRS R' is *ground-reducible under* R iff the left-hand side of every rule in R' is ground-reducible under R .

Theorem 3 (from Hofbauer & Kutsche 1988)

Let R, I, E, A, L be TRS's such that

(1) $A \cup L \subseteq \mathrm{ITh}(R)$ *("lemmata")*

(2) $R \cup I \cup A$ is terminating

(3) $I \cup E$ is ground reducible under R

(4) for every critical pair <c,p> of R on I :

$$\begin{tikzcd} c \arrow[d, "R\cup I\cup A"', "*"] & & p \arrow[d, "*"', "R\cup I\cup A"] \\ \bullet \arrow[r, "="', "E"] & \bullet \arrow[r, leftrightarrow, "*", "L"'] & \bullet \end{tikzcd}$$

(5) for every critical pair <c,p> of R on E :

$$\begin{tikzcd} c \arrow[d, "R\cup I\cup A"', "*"] & & \\ \bullet \arrow[r, "="', "E"] & \bullet \arrow[r, leftrightarrow, "*", "L"'] & p \end{tikzcd}$$

Then $I \cup E \subseteq \mathrm{ITh}(R)$. *("inductive theorems")*

The proof of this theorem is found in [HK88]. Meanwhile, we know that again some weakening of the premises is possible : In (4) resp. (5) only those critical pairs with rules $R' \subseteq R$ have to be considered, which are needed for the assertion of ground reducibility. Furthermore, also in (4) resp. (5) only certain occurencies of I- (E- resp.) left hand sides have to be taken into account for critical pairs, namely if there are *inductive complete positions*, i.e. occurencies that guarantee ground reducibility under R again. Of course, reducing once more the number of critical pairs seems helpful to succeed in inductive proving. Nevertheless, basic problems remain in the process, as we show in our case study.

3. A case study : Proving properties of binomial coefficients

We assume a standard specification ARITH of natural numbers arithmetic, defining addition, multiplication and powers, recursively on the first argument, based on 0 and successor. Applying two special cases of theorem 3, we can easily prove lemmata like (a1') $x+0 \rightarrow x$, (a2') $x+s(y) \rightarrow s(x+y)$, the commuted versions of our original rules, as well as associativity and commutativity (for addition), to be in the inductive theory of ARITH.

Now we define binomial coefficients by addition, in order to avoid fractions arising from the factorial definition :

$$\text{BIN}: \quad (b1)\ \binom{0}{s(k)} \rightarrow 0\ , \quad (b2)\ \binom{n}{0} \rightarrow s(0)\ , \quad (b3)\ \binom{s(n)}{s(k)} \rightarrow \binom{n}{s(k)} + \binom{n}{k}\ .$$

We further include the sum of binomial coefficients with fixed n into our definition, writing $\overset{k}{\Sigma}_n$ for $\sum_{j=0}^{k}\binom{n}{j}$:

$$\text{SUM}: \quad (s1)\ \overset{0}{\Sigma}_n \rightarrow \binom{n}{0}, \quad (s2)\ \overset{s(k)}{\Sigma}_n \rightarrow \binom{n}{s(k)} + \overset{k}{\Sigma}_n\ .$$

Our goal is to prove the theorem $\sum_{j=0}^{n}\binom{n}{j} = 2^n$, formalized as rule : (*) $\overset{n}{\Sigma}_n \rightarrow s(s(0))^n$.

Let $R := \text{ARITH} \cup \text{BIN} \cup \text{SUM}$. We prove a few elementary properties of binomial coefficients first, for later use as lemmata. Start with

(b4) $\binom{n}{n+s(m)} \rightarrow 0$, which easily turns out to be in ITh(R) , using $A := \{a2'r\}$, i.e. (a2') in reverse direction. (Check for termination !) However, we have to separate this proof from the following of (b5) and (b6) ,

(b5) $\binom{n}{n} \rightarrow s(0)$, (b6) $\binom{n}{s(n)} \rightarrow 0$, because (b6) requires some assistance by (a1') and (a2') *and* their reverses, together with (b4), which on its part needs several reduction steps with ARITH and (a2'r). One succeeds in choosing L = {a1', a2', b4}, which allows to apply (a1') and (a2') in either direction, mixed with one required (b4)-step.

Now we can proceed with some inductive properties of binomial sums, simultaneously proving

(s3) $\overset{k}{\Sigma}_0 \rightarrow s(0)$, (s4) $\binom{n}{k} + \overset{k}{\Sigma}_{s(n)} \rightarrow \overset{k}{\Sigma}_n + \overset{k}{\Sigma}_n$, and (*) , which is our goal.

With L = {b4, b5, b6, a1', a2', ass, comm} we choose a non-terminating auxiliary set again, furthermore $E = \emptyset$ and A = {b4, b5, b6, ass} , and apply theorem 3 . Another problem occurs during the reduction of some critical pairs : One can happen to fail with that proof if one proceeds too far reducing with $R \cup I \cup A$. So it is necessary to check for L -equivalence during intermediate steps as well .

4. Discussion of problems and consequences

As we could see in our example, several decisions have to be made during the proof process. We sketch a few of them :

(1) When do we have to prove theorems simultaneously, when hierarchically ?

(2) How do we group them into the sets I and E, how the lemmata into A and L ?

(3) How far can we simply apply reduction, and when do we have to check for L -equivalence ?

(4) How do we find appropriate lemmata ?

(5) How do we find sufficiently strong induction hypotheses ?

Of course, one can hope to automatize parts of those problems by improvements in theory : E.g. further reduction of the number of critical pairs might help in some cases ; likewise the use of unfailing Knuth-Bendix, automatic generation of termination orderings, etc. Also clever heuristics to obtain ideas for lemmata from the Knuth-Bendix process can be useful, as well as clever heuristics to generalize induction hypotheses. The author is aware that lots of work into these directions has been or is currently being done.

Nevertheless, there are lots of well-known reasons (e.g. no recursive enumerability, high complexity) to believe that there will be little chance to write *automatic* induction procedures for a wider class of inductive theorems. From our examples I could learn about the high vulnerability of a possibly automatized proof procedure. It often needs human interaction, or the process will fail.

So I demand a different way of looking on the things : as in software development a bunch of helpful tools exists, I want to have *proof development tools* which highly interact with the human expert upon the area of proofs to be given. The system should provide the user with useful information of a running proof in easy readible (or better: visible) form, especially the book-keeping about the proof up to now. It should allow manual interaction at any point, and it should be able to assist the users intuition by concretely computed suggestions, say for induction hypotheses, lemmata structuring and others. Since both of them, computers and humans, have certain strong capabilities (most of them complementary), but their weaknesses in other aspects too, it seems to me most promising for such a hardly accessible area like inductive proofs, to establish a methodology of information exchange, and to build systems based on interaction rather than fully automatized black boxes.

Literature

[Gog80] Goguen, J. *How to prove algebraic inductive hypotheses without induction, with applications to the correctness of data type implementation*, Lecture Notes in Comput. Sci., Vol.87, Springer-Verlag (1980), pp.356-73.

[Huet80] Huet, G. *Confluent reductions: abstract properties and applications to term rewriting systems*, J.ACM, Vol.27 (1980), pp.797-821.

[HH82] Huet, G.; Hullot, J.M. *Proofs by induction in equational theories with constructors*, J. Comp. and Syst. Sci., Vol.25 (1982), pp.239-66.

[HK88] Hofbauer, D.; Kutsche, R.-D. *Proving Inductive Theorems Based on Term Rewriting Systems*, Proc. 1st Int. Workshop on Alg. and Logic Programming (Gaußig, DDR), Akademie-Verlag, 1988. Revised version : Tech. Report 88-27, TU Berlin.

[JK86] Jouannaud, J.P.; Kounalis, E. *Automatic proofs by induction in theories without constructors*, CNRS Rapport de Recherche No.295 (1986). Preliminary version in Proc. 1st LICS (1986).

[Mus80] Musser, D.R. *On proving inductive properties of abstract data types*, Proc. 7th ACM Symp. on Principles of prog. languages (1980), pp.154-62.

Equational Proofs by Induction using Second Order Variables

Dieter Hofbauer

Technische Universität Berlin, Fachbereich Informatik, FR 6-2
D-1000 Berlin 10 , Franklinstr. 28/29

Extended Abstract
March 1989

Introduction

The inductive theory of a set of (first order) equations, i.e. the set of equations valid in the initial model, is not easily accessible by proof theoretic means. "Inductionless induction" refers to a class of approaches which reduce this problem to well developed concepts for term rewriting systems such as (ground) confluence, temination and ground reducibility (cf. Jouannaud & Kounalis [JK86]). If such methods are formulated not aiming at the use of completion procedures (Hofbauer & Kutsche [HK88]) , it turns out that some conditions posed on the underlying set of equations can be weakend.

The following example shows, how an extension of the language by second order variables enables inductive proofs which would have failed otherwise. Equations with second order variables are used as a finite description of an infinite set of first order equations.

Example

Let the term rewriting system (TRS) R consist of the rules

$0 \leq 0 \to \text{true}$	$0 \leq s(y) \to \text{true}$
$s(x) \leq 0 \to \text{false}$	$s(x) \leq s(y) \to x \leq y$
$\text{if true then } x \text{ else } y \to x$	$\text{if false then } x \text{ else } y \to y$

Suppose we want to prove the inductive theorem

$$\text{if } x \leq y \text{ then } f(x, y) \text{ else } f(y, x) = \text{if } y \leq x \text{ then } f(y, x) \text{ else } f(x, y) \qquad (*)$$

(where f is a binary operation symbol). Replacing the equality symbol by "$\to$" we get a nonterminating rewrite rule. We thus could try to use the following

Theorem [HK88]

Let R and E be TRS's such that R is terminating, every left hand side of a rule in E is ground reducible under R, and for every critical pair < c, p > of R on E we have

$$c \xrightarrow{*}_R \xrightarrow{=}_E \xleftrightarrow{*}_R p \ .$$

Then E (if read as a set of equations) is in the inductive theory of R .

Using a completion approach to generate an appropriate set E would yield the infinite set of equations

$$\text{if } x \le y \text{ then } f(s^i(x), s^i(y)) \text{ else } f(s^i(y), s^i(x)) \ =$$
$$\text{if } y \le x \text{ then } f(s^i(y), s^i(x)) \text{ else } f(s^i(x), s^i(y)) \qquad (i \in \mathbb{N})$$

But after a closer look at R and at equation (*) one could guess that even

$$\text{if } x \le y \text{ then } t[x, y] \text{ else } t[y, x] \ = \ \text{if } y \le x \text{ then } t[y, x] \text{ else } t[x, y]$$

holds as an inductive theorem for all "context terms" t . This is a proper generalization of (*) . It can easily be written as an equation with a binary second order variable F :

$$\text{if } x \le y \text{ then } F(x, y) \text{ else } F(y, x) \ = \ \text{if } y \le x \text{ then } F(y, x) \text{ else } F(x, y) \qquad (**)$$

Trying now to use the above theorem to prove (**) as an inductive consequence of R (meaning that every first order instance of (**) is an inductive theorem of R) we are considering critical pairs of R on the left hand side of (**). The most interesting superposition is the term $\text{if } s(x) \le s(y) \text{ then } F(s(x), s(y)) \text{ else } F(s(y), s(x))$ which yields the critical pair < c, p > where

$$c \ = \ \text{if } x \le y \text{ then } F(s(x), s(y)) \text{ else } F(s(y), s(x)) \qquad \text{and}$$
$$p \ \rightarrow_R \ \text{if } y \le x \text{ then } F(s(x), s(y)) \text{ else } F(s(y), s(x)) \quad := p' \ .$$

The substitution $\sigma = \begin{bmatrix} F \\ \lambda vw.\ F(s(v),s(w)) \end{bmatrix}$ is a match of c by the left hand side of (**) . We thus can apply the rewrite rule that corresponds to (**) and reduce c to p' , joining the critical pair as required. The other possible critical pairs[1] as well as the other premises in the above theorem cause no problems.

Conclusion

There are several possibilities in order to cope with situations where infinitely many (first order) equations (or rules) occur, e.g. during a completion like process :

- Introducing new operation symbols and extending the set of rules accordingly (see M. Hermann [Her88]), even generating such an extension automatically (K.-P. Jantke & M. Thomas [JT87])

[1] Note that it suffices to look at critical pairs arising from superposition of R on the subterm $x \le y$ in (**) .

- Using the concept of metarules and metavariables of H. Kirchner [Kir87]

The use of second order variables however seems to be most natural in a number of examples (cf. B. Gramlich [Gra88], who applies results on unifiability of second order terms to analyse the divergence behaviour of completion). Even though there are some severe difficulties in developing this approach. Unification of second order terms is undecidable in general (W. Goldfarb [Gol81]) and most general unifiers need not to exist (cf. G. Huet [Hu75]). In this context one would benefit from results on syntactically restricted subclasses of second order terms with decidable unification problem.

References

[Gol81] Goldfarb, W. *The undecidability of the second-order unification problem,* TCS 13 (1981)

[Gra88] Gramlich, B. *Unification of term schemes - theory and applications,* SEKI-Report SR-88-18, Universität Kaiserslautern (1988).

[JK86] Jouannaud, J.P. and Kounalis, E. *Automatic proofs by induction in theories without constructors*, CNRS Rapport de Recherche No.295 (1986). Preliminary version in Proc. 1st LICS (1986).

[Her88] Hermann, M. *Vademecum of divergent term rewriting systems*, Research Report CRIN 88-R-082, Nancy (1988).

[HK88] Hofbauer, D. and Kutsche, R.-D. *Proving inductive theorems based on term rewriting systems*, Proc. Int. Workshop on Algebraic and Logic Programming, Eds. J. Grabowski, P. Lescanne and W. Wechler, Akademie Verlag Berlin (1988).

[Hu75] Huet, G. *A unification algorithm for typed λ- calculus* TCS 1 (1975).

[JT87] Jantke, K.-P. and Thomas, M. *A note on inductive inference for solving divergence in Knuth-Bendix-completion*, unpublished (1987).

[Kir87] Kirchner, H. *Schematization of infinite sets of rewrite rules. Applications to the divergence of completion processes*, Proc. 2nd RTA, LNCS 256 (1987).

Inductive Theorem Proving Using Refined Unfailing Completion Techniques

Bernhard Gramlich

Universität Kaiserlautern
Fachbereich Informatik
Postfach 3049
D-6750 Kaiserslautern
E-mail: gramlich@uklirb.uucp

Extended Abstract

We present a brief overview on completion based inductive theorem proving techniques, point out the key concepts for the underlying "proof by consistency" - paradigm and try to get an abstract description of what is necessary for an algorithmic realization of such methods.

In particular, we give several versions of proof orderings, which - under certain conditions - are well-suited for that purpose. Together with corresponding notions of (inductive) covering sets (cf. [Ba88]) we get abstract "positive" and "negative" characterizations of inductive validity. This leads to a better understanding of various sufficient operational characterizations of inductive validity in a static sense (cf. [JoKo86], [Kü87], [HoKu88]). It provides a straightforward generalization of an inductive validity criterion of [Kü87] to the case where some of the equational conjectures may not be orientable. To be a little bit more precise concerning the "positive" and "negative" approach, let us assume that we have given an equational theory presented by a ground convergent, i.e. terminating and ground confluent term rewriting system R, and a set C of (equational) inductive conjectures of R. Then, proving inductive validity

of C from a positive point of view amounts to replacing all C-steps by R-steps in every ground proof using R union C. In the negative case the aim consists in transforming the potentially inconsistent set C into a provably inconsistent one by deducing appropiate (inductive) consequences from R union C (cf. [Ba88]). Inductive validity of C is guaranteed if potential inconsisteny is impossible due to verifiable consistency. In both cases appropiate proof orderings assure that the necessary proof transformation process turns into a terminating proof simplification process.

Furthermore we consider several refinements and optimizations of completion based inductive theorem proving techniques. In particular, sufficient criteria for being a covering set including restrictions of critical pairs (cf. [Gö85], [Fr86], [Kü87], [Ba88]) and the usage of non-equational inductive knowledge (cf. [HuHo80], [Pa84]) are discussed.

Moreover a couple of lemma generation methods are briefly summarized, most of which are known from classical inductive theorem proving using induction schemes (cf. [BoMo79]). Techniques of save generalization (cf. [Gr85], [JaTh88]) are particularly interesting, since they provide means for syntactic generalizations, i.e. simplifications, of conjectures without loosing semantic equivalence. To be more precise, an equation s' = t' is a save generalization of s = t, if s = t is an instance of s' = t' and s = t, s' = t' are equivalent concerning inductive validity w.r.t. R.

Finally we present the main features and characteristics of UNICOM, an inductive theorem prover with refined unfailing completion techniques and built on top of TRSPEC, a term rewriting based system for algebraic specifications (cf. [AvBeGöMa86], [Sc88]).

References

[AvBeGöMa86] Avenhaus, J., Benninghofen, B., Göbel, R., Madlener, K.: TRSPEC: A Term Rewriting Based System for Algebraic Specifications, Proc. 8th CADE, Oxford,England, 1986

[Ba88] Bachmair, L.: Proof by Consistency in Equational Theories, Proc. of LICS, 1988

[BaDeHs86] Bachmair, L., Dershowitz, N., Hsiang,J.: Orderings for equational proofs, Proc. of Symb. Logic in Computer Science, Cambridge, MA, 1986

[BoMo79] Boyer, R., Moore, J.: A Computational Logic, Academic Press, 1979

[Fr86] Fribourg, L.: A strong restriction of the inductive completion procedure, Proc. 13th ICALP, Rennes, France, 1986

[Go80] Goguen, J.A.: How to prove algebraic inductive hypotheses without induction, Proc. of 5th CADE, ed. W. Bibel and R. Kowalski, LNCS 87, 1980

[Gö85] Göbel, R.: A Specialized Knuth-Bendix Algorithm for Inductive Proofs, Proc. Combinatorial Algorithms in Algebraic Structures, Universität Kaiserslautern, 1985

[Gö88] Göbel, R.: A Completion Procedure for Generating Ground Confluent Term Rewriting Systems, Dissertation, FB Informatik, Universität Kaiserslautern, Feb. 1988

[Gr85] Gramlich, B.: Zum Beweisen durch Reduktion und Induktion, Diplomarbeit, Fakultät für Informatik I, Universität Karlsruhe, 1985

[HsRu86] Hsing, J., Rusinowitch, M.: On word problems in equational theories, Tech. Rep. 86/29, Dept. of Computer Science, SUNY at Stony Brook, USA

[HoKu88] Hofbauer, D., Kutsche, R.-D.: Proving inductive theorems based on term rewriting systems, Techn. Report 88-12, TU Berlin, 1988, see also Proc. of an International Workshop on Algebraic and Logic Programming, Gaussig, GDR, Nov. 1988

[HuHu82] Huet, G., Hullot, J.: Proofs by Induction in Equational Theories with Constructors, JACM 25(2), 1982

[JaTh88] Jantke, K.P., Thomas, M.: Inductive Inference for Solving Divergence in Knuth-Bendix Completion, Tech. Rep. 88/R6, Dept. of Computer Science, University of Glasgow, 1988

[JoKo86] Jouannaud, J.-P., Kounalis, E.: Automatic proofs by induction in equational theories without constructors, Proc. Symb. Logic in Computer Science, Boston, Massachusetts, 1986

[Ka87] Kapur, D.: Proof by consistency, Artificial Intelligence 31, 1987

[KaNaZh86] Kapur, D., Narendran, P., Zhang, H.: Proof by induction using test sets, Proc. of 8th CADE, ed. J. Siekmann, LNCS 230, 1986

[Ki84] Kirchner, H.: A general inductive completion algorithm and application to abstract data types, Proc. of 7th CADE, LNCS 170, 1984

[Kü87] Küchlin, W.: Inductive Completion by Ground Proof Transformation, Proc. CREAS, Lakeway, Texas, 1987

[Mu80] Musser, D.: On proving inductive properties of abstract data types, Proc. 7th ACM Symp. on Principles of Programming Languages, Las Vegas, Nevada, USA, 1980

[Pa84] Paul, E.: Proof by induction in equational theories with relations between constructors, Proc. of 9th CAAP, ed. B. Courcelle, Cambridge Univ. Press, 1984

[Pl85] Plaisted, D.A.: Semantic confluence tests and completion methods, Information and Control 65, 1985

[Sc88] Scherer, R.: UNICOM: Ein verfeinerter Rewrite-basierter Beweiser für induktive Theoreme, Diplomarbeit, FB Informatik, Universität Kaiserslautern, 1988

[To86] Toyama, Y.: How to prove equivalence of term rewriting systems without induction, LNCS 230, 1986

An equational approach to sorts for logic programming

Ulrich Furbach

March 1989

Forschungsgruppe Intellektik
Institut für Informatik, TU München
Postfach 202420, D 8000 München 2

Many-sorted logic programming is discussed in the literature in various directions. There is work aiming at an improvement of Prolog by augmenting it with sorts in order to relieve the task of software engineering. Other approaches are aiming at bringing in "is-a"-structures to close the gap between programming and knowledge representation.

In this short note I want to report an observation we made during the work on combining logic and functional languages:
If one chooses equations, e.g. term rewriting systems as a functional language, it is very easy to use this functional part of the combined language to define and to handle sorts and "is-a"-taxonomies. As in (Ait-Kaci, Nasr 86) all this can be done within the unification procedure, but we do not have to give any sophisticated semantics. The semantics of the sorted language is exactly the same as the semantics of the combined language.

It is important to note that the method proposed in this paper is best suited within a framework, in which the decision for the use of a combined logic and functional language has been made nevertheless. In other words: if one uses a term-rewrite system as a functional language within logic our mechanism of bringing in sorts can be seen as a very cheap byproduct. At least it gives a formal semantic of order-sorted logic programming which can be run on a Hornclause interpreter.

Equational logic programs

We will use in the following equational logic programs, i.e. pairs (R, P) where R is a rewrite system and P is a set of Hornclauses. We assume that P does not contain the 2-ary predicate symbol '=', i.e. there is no equality built-in in P. The following is a simple example:

R: $0 + X = X.$
$s(X) + Y = s(X + Y).$

P: $nodes(nil, 0).$
$nodes(t(L, N, R), (NL + NR) + s(0)) :-$
$nodes(L, NL),$
$nodes(R, NR).$

Following an approach from (Hölldobler 88) we assume a complete unification procedure U_R, which recursively enumerates a complete set of R-unifiers, i.e. unifiers for the theory defined by R. Then we can use SLDE-resolution to interpret equational logic programs; SLDE-resolution is simply like SLD-resolution with the exception, that is uses U_R to interpret the P-part of a program (R, P). The following strong completeness result is the base for the interpretation of equational logic programs.

Theorem 1 (Hölldobler) *Let (R, P) be an equational logic program and U_R a complete unification procedure for R. For every correct answer substitution σ for (R, P) and goal statement D there exists a computed answer substitution θ for $P \cup D$ wrt to SLDE-resolution with arbitrary selection function, such that $\theta \leq_R \sigma$.*

For the implementation of the extended unification procedure one can use the paramodulation rule which is proven complete in this context in (Furbach et al 88).

Sorts

To introduce sorts we simply assume that a subset $S$of the function symbols is used as sort symbols and that a partial order $\prec$ over S is given. For the ease of notation we write functional applications with sort symbols η as $t : \eta$ instead of $\eta(t)$.
Furthermore we agree on stating the fact that a term t has sort η by defining the equation $t : \eta = t$ in R, abbreviating this as $t : \eta$, too. Finally we use equations of the form $x : \xi : \eta = x : \xi$ to express that $\xi \prec \eta$ holds.

Using these conventions we can give a sorted version of the previous example:

R :

$$0 : nat$$
$$s(x) : nat = (x : nat)$$
$$0 + x : nat = x$$
$$s(x : nat) + y : nat = s(x + y)$$

P :

$nodes(nil, 0).$
$nodes(maketree(L, N : nat, R), (NL + NR) + s(0)) : -$
$nodes(L, NL),$
$nodes(R, NR).$

The following theorem states, that our mechanism 'really works'.

Theorem 2 *If η and ξ are two different sort-symbols and t is a term, then $t : \eta$ is a logical consequence of $t : \xi$, $\xi \prec \eta$ and R for a given sorted equational program (R, T).*

A typical example for the use of 'is-a'-hierarchies is the following simple program, which can be used to answer questions like $? - eats(Tom, Jerry)$.

$R:$

$cats \prec carnivorous$
$omnivorous \prec carnivorous$
$omnivorous \prec herbivorous$
$mice \prec herbivorous$
$humans \prec omnivorous$
$gorillas \prec omnivorous$
$Tom : cats$
$Jerry : mice$

$P:$

$eats(x : carnivorous, y : herbivorous).$
$eats(x : herbivorous, y : plants).$

We have implemented this approach as a part of the ALPES-Prolog environment within ESPRIT P973. We have given up occur-check, as Prolog did, to maintain execution time in acceptable limits. In this prototype we have choosen a method which generates in a pre-processing step a pure Prolog-program for a given equational logic program (R, P). For this the term-rewrite system R is transformed into a set of Prolog-clauses which is the extended unification procedure.

For a more efficient implementation one could use the framework for SLDE-paramodulation from (Furbach et al. 89), which allows to seperate the inferences which deal with sorts from the rest of the term rewrite system. Then the handling of sorts can be done by a special purpose unification procedure.

References

Ait-Kaci,H., Nasr, R. LOGIN: A Logic Programming Language with Built-in Inheritance, Journal of Logic Programming, 1986.

Conrad, T., Furbach, U. Sorts are nothing but Functions. An Equational Approach to sorts for Logic Programming. FKI-89-88, TU München 1988.

Furbach, U., Hölldobler, Schreiber, J. Horn Equality Theories and Paramodulation, to appear in: Journal of Automated Reasoning, 1988.

Furbach, U., Hölldobler, Schreiber, J. SLDE-Paramodulation, this volume 1989.

Hölldobler, S. On the Foundations of Equational Logic Programming. Dissertation UniBw München 1988, to appear: Springer Lecture Notes in AI.

SLDE-Paramodulation

Ulrich Furbach [1], Steffen Hölldobler [2], Joachim Schreiber [3]

Recent interest in combined functional and logic programming languages has led to numerous proposals for the integration of equational and logic languages based on linear paramodulation. In order to increase efficiency various restrictions for paramodulation such as narrowing were proposed. However, the applicability of narrowing is limited, since narrowing is only complete if the equational theory is confluent and terminating. For example, if we want to compute with sets via the operation "$\cup$" then, of course, we have to specify that "$\cup$" is associative, commutative, and idempotent. Unfortunately, such an equational theory is no longer terminating. On the other hand, unification under associativity, commutativity, and idempotence is finitary and a type conformal unification algorithm exists. Should we not build such a theory-unification algorithm into the narrowing procedure? Though Jouannaud et al. (1983) showed that narrowing modulo equality is sound and complete if the theory in consideration is confluent, coherent, and terminating modulo equality, many questions are still open. Is narrowing modulo equality also independent of a selection function? Can we generalize these results to conditional equational theories? Can we restrict the application of narrowing modulo equality to socalled basic occurrences?

In Furbach et al. (1989) and Hölldobler (1989) we proved the strong completeness of linear paramodulation for Horn equational theories and demonstrated how various conditions imposed one-by-one on the equational theory restrict the search space and allow to apply special forms of paramodulation. The goal of this paper is to show that this framework can be generalized to SLDE-paramodulation, i.e paramodulation modulo equality, in much the same way as SLD-resolution can be generalized to SLDE-resolution.

Throughout this paper we consider equational programs, i.e. Horn clauses with the only one predicate symbol $\doteq$ written infix. Since we intend to build parts of the theory into the unification algorithm we partition an equational program into the parts E and EP. E will be

[1] Forschungsgruppe Intellektik, Institut für Informatik, TU München, Postfach 202420, D-8000 München, E-mail: uli@tumki.tu-muenchen.de

[2] Fachgruppe Intellektik, Fachbereich Informatik, TH Darmstadt, Alexanderstr. 10, D-6100 Darmstadt, E-mail: xiisshoe@ddathd21.bitnet

[3] Fakultät für Informatik, Universität der Bundeswehr München, Werner-Heisenberg-Weg 39, D-8014 Neubiberg, E-mail: i21bjfs@ubw-m.uucp

Theorem:

If $[Q]$ is an element of the least fixpoint of $T_{E,EP}$, then there exists a refutation of $EP \cup \{\Leftarrow Q\}$ with respect to SLDE-i-paramodulation and SLDE-reflection.

Lifting this result yields a completeness result for our calculus. In analogy to (Hölldobler 1989) we can proof a switching lemma and, thus, we obtain the strong completeness of SLDE-i-paramodulation and SLDE-reflection.

Theorem:

Let R be a computation rule. For each correct answer substitution σ for E,EP and $\Leftarrow F$ there exists an R-computed answer substitution θ obtained by a refutation of $EP \cup \{\Leftarrow F\}$ with respect to SLDE-i-paramodulation and SLDE-reflection such that θ is not less than σ modulo E.

We proceed as usual and impose conditions on our equational program in order to restrict the search space. If the equational program is confluent modulo E then program clauses may be used only in one direction without loosing completeness. The next step is to look for conditions such that we have to apply paramodulation only upon non-variable subterms. In the case where $E=\emptyset$ these conditions were the restriction to term rewriting systems and to normalized answer substitutions. However, if $E\neq\emptyset$ then we must also require that E is regular and E-normal form preserving, i.e. whenever a term s is in normal form with respect to a term rewriting system and $s=_E t$, then t is also in normal form. By splitting goal clauses into a skeleton and an environment part we obtain a strong completeness result for basic SLDE-narrowing and SLDE-reflection.

There are other interesting aspects: Our work demonstrates how special unification algorithms can be combined with universal unification procedures based on paramodulation or special forms of it. Furthermore, applying the proof technique developed in (Hölldobler 1989) allows to view unification problems under a special equational theory as constraints and to force or delay the solution of these constraints according to an overall strategy. Finally, since SLDE-paramodulation is sound and strongly complete, combining it with the lazy resolution rule yields a sound and strongly complete set of inference rules for equational logic programs.

References

U. Furbach, S. Hölldobler, J. Schreiber: Horn Equality Theories and Paramodulation. To appear in: Journal of Automated Reasoning: 1989

S. Hölldobler: Foundations of Equational Logic Programming. Dissertation; to appear in: Lecture Notes in Artifical Intelligence: 1989

J. P. Jouannaud, C. Kirchner, H. Kirchner: Incremental Construction of Unification Algorithms in Equational Theories. Proceedings of the 10th ICALP, LNCS 154, 361-373: 1983

J. W. Lloyd: Foundations of Logic Programming. Springer: 1984

J. Siekmann: Universal Unification. Proc. ECAI, 365-400: 1986

handled by the unification algorithm used by the paramodulation rule, which is applied upon clauses from EP. For example, E may contain the axioms of associativity, commutativity, and idempotence for a set operator, which can be used within EP, or the equations defining order-sorted operators for a typed program EP.

An equational program E,EP admits a least Herbrand E-model over the Herbrand E-universe, i.e. the quotient of the Herbrand universe modulo the finest congruence generated by E.

In the sequel we assume that whenever $l \doteq r \Leftarrow F$ is an element of EP then $r \doteq l \Leftarrow F$ is also an element of EP, where F denotes a set of equations; this is a technicality allowing us to apply paramodulation only from left-to-right.

As an intermediate step towards the intended completeness result for SLDE-paramodulation we give a fixpoint characterization of the least Herbrand E-model using the function

$\mathbf{T}_{\mathbf{E,EP}}(I) = \{\ [t \doteq t]\ \mid t \text{ is a ground term}\}$
$\qquad \cup \{\ [Q]\ \mid$ there exists an occurrence π in Q and a ground instance $l \doteq r \Leftarrow F$ of a clause in EP such that $[Q|\pi|] = [l]$ and $\{[F]\} \cup \{[Q|\pi \leftarrow r|]\} \subseteq I\ \}$,

where [Q] denotes the congruence class containing the equation Q, $[F] = \{[Q] | Q \in F\}$, and $Q|\pi|$ denotes the subterm of Q at π. Along the lines of (Furbach et al. 1989) we can show that $T_{E,EP}$ admits a least fixpoint which is equal to the least Herbrand E-model.

Turning to the proof theoretic aspects we assume the reader to be familiar with basic notions from logic programming and universal unification (for a thorough treatment see e.g. Lloyd 1984 and Siekmann 1986). In the sequel we suppose to have a correct and complete E-unification procedure UP_E for the equational theory E at our disposal.

Let $\Leftarrow F \cup \{Q\}$ be a goal clause, Q be the selected equation, $l \doteq r \Leftarrow F'$ be a new variant of a program clause, and π be an occurrence of Q. If $Q|\pi|$ and l are E-unifiable with $\sigma \in UP_E(Q|\pi|,l)$, then $\Leftarrow \sigma(F \cup F' \cup \{Q|\pi \leftarrow r|\})$ is called **SLDE-paramodulant.**

Instead of adding the axiom of reflexivity to our equational program we use the following inference rule to terminate paramodulation proofs successfully. Let $\Leftarrow F \cup \{s \doteq t\}$ be a goal clause and $s \doteq t$ be the selected equation. If s and t are E-unifiable with $\sigma \in UP_E(s,t)$, then $\Leftarrow \sigma F$ is an **SLDE-reflectant.**

We demonstrated in (Furbach et al. 1989) that linear paramodulation is only complete if we add the functional reflexive axioms to the equational program. The same effect can be achieved if we allow to instantiate goal clauses before performing a paramodulation step. We call such a step an **SLDE-i-paramodulation** step.

By relating applications of $T_{E,EP}$ with SLDE-i-paramodulation resp. SLDE-reflection steps in the usual way we obtain

Narrowing as Operational Semantics for Logic-Functional Programming

Alexander Bockmayr
Sonderforschungsbereich 314 "Künstliche Intelligenz - Wissensbasierte Systeme"
Institut für Logik, Komplexität und Deduktionssysteme
Universität Karlsruhe, Postfach 6980, D-7500 Karlsruhe 1, F. R. Germany
e-mail: bockmayr@ira.uka.de

Extended Abstract

During the last years there has been an increasing interest in the combination of logic and (first order) functional programming. While some authors in a more pragmatic approach propose a synthesis of existing programming languages like Prolog and Lisp, others develop the idea of logic-functional programming on a more theoretical level.

It has turned out that term rewriting and narrowing in conditional equational theories provide a nice theoretical framework for the integration of logic and functional programming. In this approach a logic-functional program is a set of conditional equations. As conditional rewrite rules these equations may be employed for the simplification or evaluation of terms ("functional programming"), whereas in the conditional narrowing process, which can be seen as a generalization of Prolog's SLD-resolution, the same equations are used for the solution of goals or equations ("logic programming").

From a theoretical point of view, narrowing provides a complete unification procedure for any equational theory that can be defined by a canonical term rewriting system (without extravariables in the conditional case). For practical applications however, narrowing in its original form is much too inefficient.

For functions inductively defined over some set of constructors C – these are typical functional programs – the narrowing algorithm enumerates the whole constructor term algebra T(C,X). Moreover there are serious inefficiencies in this enumeration process: the same substitutions are generated in many different ways. This means that the narrowing algorithm behaves worser than a trivial generate-and-test algorithm.

In order to improve this poor behaviour many optimizations have been proposed. Usually they restrict the set of occurrences at which a narrowing step is performed (basic narrowing, innermost narrowing, outermost narrowing, selection narrowing) and normalize the goal after each narrowing step (normal narrowing). But even the most sophisticated narrowing procedure is inadequate to solve for example a system of linear equations. However, such equations occur very often in practical applications.

It is therefore necessary to incorporate special theories and their unification algorithms into the general narrowing process. This can be done using narrowing modulo an equality theory E. Building-in equality theories may reduce the search space of the narrowing algorithm dramatically. The main difficulty is that the E-unification algorithms must be able to deal with additional free function symbols.

A logic-functional programming language without built-in theories will not meet the requirements of practical applications.

References

[1] A. Bockmayr: Conditional Rewriting and Narrowing as a Theoretical Framework for Logic-Functional Programming. Int. Ber. 10/86. Universität Karlsruhe, Fakultät für Informatik, 1986

[2] A. Bockmayr: Narrowing with Inductively Defined Functions. Int. Ber. 25/86. Universität Karlsruhe, Fakultät für Informatik, 1986

[3] A. Bockmayr: A Note on a Canonical Theory with Undecidable Unification and Matching Problem. J. Autom. Reas. 3 (1987), 379-381

[4] A. Bockmayr: Narrowing with Built-in Theories. Proc. Int. Workshop on Logic and Algebraic Programming, Gaußig, GDR, Nov. 88. Akademie-Verlag Berlin, 1988

On Goal and Term Reduction Calculi for Conditional Rewriting

Peter Padawitz
Universität Passau
Postfach 2540
D-8390 Passau

[BK86] and [DOS88] classify approaches to conditional rewriting with regard to the way the conditions are evaluated. This suggests starting out from a notion of *goal* reduction and deriving from that the notion of conditional *term* reduction. (Goals are sets of atomic formulas; $\emptyset$ denotes the empty goal; for further basic notions used here, cf. [Pad88], Sect. 2.)

Given a set R of conditional equations of the form $u \equiv u' \Leftarrow \vartheta$ (with goal ϑ), let us first follow [Kap84] and define the goal reduction relation recursively (as the limit of approximating relations $\vdash_{rR,i}$):

$$
\begin{array}{lll}
\gamma \vdash_{rR} \gamma' & \Leftrightarrow_{def} & \exists\, i \geq 0 \text{ s.t. } \gamma \vdash_{rR,i} \gamma' \\
\gamma \vdash_{rR,0} \gamma' & \Leftrightarrow_{def} & \exists \textit{ normal atom } p : \gamma = \gamma' \cup \{p\} \text{ or} \\
 & & \exists\, u \equiv u' \in R, \delta, x, f : \gamma = \delta[u[f]/x],\ \gamma' = \delta[u'[f]/x] \\
\gamma \vdash_{rR,i+1} \gamma' & \Leftrightarrow_{def} & \gamma \vdash_{rR,i} \gamma' \text{ or} \\
 & & \exists\, u \equiv u' \Leftarrow \vartheta \in R, \delta, x, f : \gamma = \delta[u[f]/x],\ \gamma' = \delta[u'[f]/x],\ \vartheta[f] \vdash_{rR,i}^{*} \emptyset.
\end{array}
$$

If the only predicate symbols of the underlying specification are equality predicates, the set NA of normal atoms is usually chosen as the set of *reflexive* equations $t \equiv t$. But even in the equational case, other definitions of normal atoms make sense as well. For instance, the *normal-join systems* of [DOS88] correspond to the restriction of NA to equations $t \equiv t$ where t is *irreducible* w.r.t. R. Reducibility, however, refers to the *term* reduction relation $\longrightarrow_{rR}$, which can be derived from the goal reduction relation $\vdash_{rR}$:

$$t \longrightarrow_{rR} t' \quad \Leftrightarrow_{def} \quad \{t \equiv x\} \vdash_{rR} \{t' \equiv x\}.$$

Another definition of normal atoms admits non-reflexive equations or even non-equational atoms, which belong to some *base* theory (cf. [Pad88], Sect. 7).

In the book just cited we have defined goal reduction non-recursively, with the help of two inference rules:

Goal Reduction Rule
$$\frac{\delta[u[f]/x]}{\delta[u'[f]/x] \cup \vartheta[f]} \qquad u \equiv u' \Leftarrow \vartheta \in R$$

Reflection Rule
$$\frac{\gamma \cup \{p\}}{\gamma} \qquad \text{p normal}$$

Let us denote the corresponding inference relation by $\vdash_R$. (Since inference relations are *per se* transitive, we need not write $\vdash_R^*$.) Based on this notion, the *term* reduction relation is defined as follows:

$$t \longrightarrow_R t' \quad \Leftrightarrow_{def} \quad \exists\, u \equiv u' \Leftarrow \vartheta \in R, c, x, f : t = c[u[f]/x],\ t' = c[u'[f]/x],\ \vartheta[f] \vdash_R \emptyset.$$

With respect to *successful* derivation sequences, both notions of goal reduction coincide:

Proposition 1: $\quad \gamma \vdash_R \emptyset \;$ iff $\; \gamma \vdash_{rR}^{*} \emptyset.$

Following [BDH86], we say that *γ has a rewrite proof* iff $\gamma \vdash_R \emptyset$. An immediate consequence of Prop. 1 is

Proposition 2: $t \longrightarrow_R t'$ iff $t \longrightarrow_{rR} t'$.

We remind of the fact that goal and term reduction depend on the choice of normal atoms. In the sequel, let us suppose that these atoms are exactly the reflexive equations (see above). Then goal reduction and term reduction are related to each other as follows:

Proposition 3: $\{t \equiv t'\} \vdash_R \emptyset$ iff $t \longrightarrow_R^* u$ and $t' \longrightarrow_R^* u$ for some u.

So one may switch between the recursive and the non-recursive view on goal reduction. However, given that R is finite, it is easy to decide whether $\vdash_R$ is applicable to γ, while the reducibility of γ w.r.t. $\vdash_{rR}$ is in general undecidable. [Kap87], [JW86], [DOS88] overcome this problem by requiring R to be *simplifying, reductive* or *decreasing,* respectively. All these notions include the existence of a Noetherian term ordering > that contains $\longrightarrow_R$. The notions differ with respect to further conditions on >. The question remains whether these conditions are necessary. Let us approach it from another side. The actual purpose of goal reduction is to use it for proving (unconditional) theorems. Of course, $\vdash_R$ is *sound*:

Proposition 4: $\gamma \vdash_R \emptyset$ implies $Mod(R) \models \gamma$.

The *completeness* of $\vdash_R$ agrees with the *Church-Rosser property* of R: R is called **Church-Rosser** if

$$Mod(R) \models \gamma \text{ implies } \gamma \vdash_R \emptyset. \qquad \text{(CR)}$$

In the unconditional case, it is well-known that the Church-Rosser property of R is equivalent to the *confluence* of $\longrightarrow_R^*$. This result remains valid for conditional equations, although the proof must proceed somewhat differently. One cannot conclude from *$Mod(R) \models t \equiv t'$* that there is a sequence of reductions of the form

$$t = t_1{}^*\!\longleftarrow u_1 \longrightarrow^* t_2{}^*\!\longleftarrow u_2 \longrightarrow^* t_3 \ldots t_{n-1}{}^*\!\longleftarrow u_{n-1} \longrightarrow^* t_n = t'$$

(and then derive *$t \equiv t' \vdash_R \emptyset$*, i.e., by Prop. 3, *$t \longrightarrow_R^* u$* and *$t' \longrightarrow_R^* u$*, by induction on n). Instead, one may induce on the length of a shortest *paramodulating* derivation from $t \equiv t'$ to $\emptyset$ whose existence follows from *$Mod(R) \models t \equiv t'$* (cf. [Pad88], Thm. 5.3.5).

So we have to look for decidable criteria for the confluence of $\longrightarrow_R^*$. In the unconditional case, the *Knuth-Bendix theorem* tells us that the confluence test can be reduced to finitely many *critical pairs,* provided that $\longrightarrow_R$ is Noetherian. Other confluence criteria avoid *this* assumption, but require that R be *non-ambiguous* (cf. [Hue80], [BK86]). However, *all* criteria rely on the assumption that the infinite set of *independent* reductions need not be considered because they are confluent in any case. Unfortunately, this does not hold in the conditional case, as the following argument shows.

Given $t \equiv t' \Leftarrow u \equiv u' \in R$ and a reduction $f \longrightarrow_R g$ such that $u[f] \equiv u'[f] \vdash_R \emptyset$, one obtains the independent reductions

$$c[t[f]/x] \longrightarrow_R c[t'[f]/x] \quad \text{and} \quad c[t[f]/x] \longrightarrow_R c[t[g]/x]. \qquad (1)$$

Of course, $c[t'[f]/x]$ can be reduced into $c[t'[g]/x]$. However, $c[t[g]/x]$ need not be reducible into this term because $u[f] \equiv u'[f] \vdash_R \emptyset$ need not imply $u[g] \equiv u'[g] \vdash_R \emptyset$.

On the other hand, by Prop. 3, $u[f] \equiv u'[f] \vdash_R \emptyset$ is equivalent to:

$$u[f] \longrightarrow_R^* v \quad \text{and} \quad u'[f] \longrightarrow_R^* v \quad \text{for some } v.$$

Thus we have the "branchings"

$$u[f] \rightarrow_R^* v,\ u[f] \rightarrow_R u[g] \qquad (2)$$

and

$$u'[f] \rightarrow_R^* v,\ u'[f] \rightarrow_R u'[g]. \qquad (3)$$

If there would be a Noetherian term ordering > such that c[t[f]/x] is greater than u[f] and u'[f] w.r.t. >, we could apply the induction hypothesis and deduce that (2) and (3) can be made confluent, i.e.,

$$v \rightarrow_R^* v',\ u[g] \rightarrow_R^* v',\ v \rightarrow_R^* v'',\ u'[g] \rightarrow_R^* v''$$

for some v',v". If we could further assume that u[f] > v, then we could apply the induction hypothesis once more and infer that the two reductions starting from v can be made confluent as well, i.e.,

$$v' \rightarrow_R^* v_o \text{ and } v'' \rightarrow_R^* v_o$$

for some v_o. Putting together these reductions, we would obtain

$$u[g] \rightarrow_R^* v_o \text{ and } u'[g] \rightarrow_R^* v_o$$

and thus, by Prop. 3, $u[g] \equiv u'[g] \vdash_R \emptyset$, so that (1) can be made confluent.

We conclude that the confluence of independent reductions is guaranteed only if some Noetherian term ordering > satisfies the following property:

(A) For all $t \equiv t' \Leftarrow \vartheta \in R$ with, say, $\vartheta = \{u_1 \equiv u_1', \dots, u_n \equiv u_n'\}$, substitutions f, terms c, $x \in var(c)$ and $1 \le i \le n$, $\vartheta[f] \vdash_R \emptyset$ implies $c[t[f]/x] \gg \{c[t'[f]/x], u_i[f], u_i'[f]\}$ (where $\gg$ is the multiset extension of >).

Only the further requirement that c[t[f]/x] be greater than $u_i[f]$ and $u_i'[f]$ even if $\vartheta[f]$ has no rewrite proof ensures that reducibility w.r.t. R is decidable (cf. the proof of [DOS88], Prop. 4). This strengthening of (A) reads precisely as follows:

(B) For all $t \equiv t' \Leftarrow \vartheta \in R$ with, say, $\vartheta = \{u_1 \equiv u_1', \dots, u_n \equiv u_n'\}$, substitutions f, terms c, $x \in var(c)$ and $1 \le i \le n$, $c[t[f]/x] \gg \{u_i[f], u_i'[f]\}$ and $\vartheta[f] \vdash_R \emptyset$ implies $c[t[f]/x] > c[t'[f]/x]$.

(B) *and* (CR) ensure that the equational theory of Mod(R) is decidable. For weakening (B), one has to change the definition of goal and term reduction. So far, two modifications have been proposed.

In [Pad88], we have restricted the set of normal atoms to those reflexive equations $t \equiv t$ where t is irreducible w.r.t. R. Under this assumption, the above argument on independent reductions proceeds differently: v becomes irreducible and thus v, v' and v" are all the same so that we only need *c[t[f]/x] > {u[f],u'[f]}* for concluding that $u[g] \equiv u'[g]$ has a rewrite proof. Indeed, the reference to > can be avoided completely because the induction step can now be justified by the fact that the shortest proof of (1) is longer than the shortest proofs of (2) and (3). This is essential for establishing the *strong-confluence* criterion [Pad88], Thm. 9.6.1, which does not presuppose any Noetherian term ordering (and which generalizes [Hue80], Lemma 3.3, to conditional equations). The restriction of normal atoms to irreducible equations, however, entails that (CR) can be guaranteed only for *normalizable* equations, i.e., for equations $t \equiv t'$ such that $t \rightarrow_R u$ and $t' \rightarrow_R u'$ for some irreducible terms u,u'. (The details of this approach are given in [Pad88], Sect. 7).

A second approach for weakening (B) stems from the concept of *unfailing completion* (cf. [HR87]). It starts out from the observation that goal and term reduction need only be defined on *ground,* i.e. variable-free, goals and terms, respectively. This holds true because soundness and completeness of the cut calculus w.r.t. Mod(R) (cf. [Pad88], Thm.

4.2.2) immediately imply:

$$Mod(R) \models \gamma \quad \text{iff} \quad Mod(R) \models \gamma'$$

where γ' is γ with all variables be replaced by different Skolem constants. The term ordering $>$ is now built into the definitions of goal and term reduction: the Goal Reduction Rule becomes:

$$\frac{\delta[t/x]}{\delta[t'/x] \cup \vartheta[f]} \qquad \begin{array}{l} u \equiv u' \Leftarrow \vartheta \text{ or } u' \equiv u \Leftarrow \vartheta \ \in R \\ t = c[u[f]/y] > c[u'[f]/y] = t' \\ t \text{ has no proper superterm in } \delta[t/x] \end{array}$$

Accordingly, the term reduction relation is now defined as follows:

$$t \longrightarrow_R t' \quad \Leftrightarrow_{def} \quad \exists\, u \equiv u' \Leftarrow \vartheta \text{ or } u' \equiv u \Leftarrow \vartheta \in R,\ c,x,f : t = c[u[f]/x] > c[u'[f]/x] = t',\ \vartheta[f] \vdash_R \emptyset.$$

The conditions on $>$ are that $>$ is Noetherian on ground terms (including Skolem constants) and

(C) for all $t \equiv t' \Leftarrow \vartheta \in R$ with, say, $\vartheta = \{u_1 \equiv u_1', \ldots, u_n \equiv u_n'\}$, ground substitutions f, terms c, $x \in var(c)$ and $1 \le i \le n$, $c[t[f]/x] >> \{u_i[f], u_i'[f]\}$ and $\vartheta[f] \vdash_R \emptyset$ implies $c[t[f]/x] > c[t'[f]/x]$ or $c[t'[f]/x] > c[t[f]/x]$.

(C) yields both that (CR) is equivalent to the confluence of $\longrightarrow_R^*$ (for ground goals and terms and w.r.t. the new definitions of $\vdash_R$ and $\longrightarrow_R$) and that the equational theory of Mod(R) is decidable.

This approach shifts the descent property from a condition on R to a feature of rewrite proofs via R. It admits generalizations to non-equational and inductive theories that will be worked out in a forthcoming paper.

References

[BDH86] L. Bachmair, N. Dershowitz, J. Hsiang, Orderings for Equational Proofs, Proc. IEEE Symp. Logic in Computer Science (1986) 346-357

[BK86] J.A. Bergstra, J.W. Klop, Conditional Rewrite Rules: Confluence and Termination, J. Comp. and Syst. Sciences 32 (1986) 323-362

[DOS88] N. Dershowitz, M. Okada, G. Sivakumar, Canonical Conditional Rewrite Systems, Proc. CADE '88, Springer LNCS 310 (1988) 538-549

[HR87] J. Hsiang, M. Rusinowitch, On Word Problems in Equational Theories, Proc. ICALP '87, Springer LNCS 267 (1987) 54-71

[Hue80] G. Huet, Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems, Journal ACM 27 (1980) 797-821

[JW86] J.-P. Jouannaud, B. Waldmann, Reductive Conditional Term Rewriting Systems, Proc. Conf. Formal Description of Programming Concepts III, North-Holland (1986) 223-244

[Kap84] S. Kaplan, Conditional Rewrite Rules, Theoretical Computer Science 33 (1984) 175-194

[Kap87] S. Kaplan, Simplifying Conditional Term Rewriting Systems: Unification, Termination and Confluence, J. Symbolic Computation 4 (1987) 295-334

[Pad88] P. Padawitz, Computing in Horn Clause Theories, EATCS Monographs on Theor. Comp. Sci. 16, Springer (1988)

An algorithm for testing sufficient completeness of a simple class of conditional specifications

Stephan Uhrig
FB 10 – Informatik
Universität des Saarlandes
6600 Saarbrücken 11

e – mail: stuh@sbsvax.informatik.uni – saarland.dbp.de

The following is an extended abstract of [RU88], which has been written by Jean – Luc Rémy and me at CRIN in Nancy (France). We assume familiarity with the basic notions of term rewriting.

1.Introduction: Sufficient completeness of equational specifications is a well – known prerequisite for proof by consistency. We present a method for testing sufficient completeness of a simple class of conditional specifications which can be looked upon as special reductive conditional term rewriting systems (CTRSs for short). Such a CTRS consists of a finite set of rules $p::l \rightarrow r$ over a signature Σ, Σ being devided in a set C of constructors and a set D of defined operators. p is a conjunction of boolean terms (called *literals*) and l a *rooted* term, i.e. a term $t = f(t_1,\dots,t_n)$, where $f \in D$ and all subterms are constructor – terms (t is also called *f – rooted*). Former algorithms (as [Zh84]) made use of inductively defined *test sets*: For every term in such a set there must be at least one rule, that can be applied. The method, which is described here, uses a different strategy, consisting in two steps: the test of *half – spannedness* and the test of *well – spannedness.*

2.Half – spannedness: Let (Σ, R) be a CTRS and $f \in D$. f is said to be *half – spanned,* if for every ground – term $t = f(u_1,\dots,u_n)$ (u_i being a constructor term for every i, $1 \leqq i \leqq n$) there exists a rule $p::l \rightarrow r$ and a ground – substitution σ, such that $\sigma(l) = t$. (Σ, R) is called half – spanned, if every $f \in D$ is half – spanned. Let now G(t) denote the set of all ground – instances of term t (wrt C) and $G(\{t_1,\dots,t_n\})$ the set $G(t_1) \cup \dots \cup G(_n)$. The method for testing half – spannedness is an extension to the multi – sorted case of a method presented in [Th84] and [LLT87]. Its theoretical background is shortly presented here: The method

makes use of complements of terms, a complement of a term $t \in T_C$ (T_C = set of all constructor ground-terms) being a finite set T of terms, such that $T_C = G(t) \cup^* G(T)$ ($\cup^*$ denoting the disjoint union). For linear terms this complement can easily be constructed (see [Th84], [LLT87], [RU88]). These notions are extended to linear substitutions in such a way, that:

Theorem1: Every rooted term t and every linear substitution s, such that domain of s equals the set of variables of t, verify:
$G(t) = G(\{r(t) | \ r \in \text{complement of } s\}) \cup^* G(s(t))$ ○

Let now a semi-linear term t be a term, such that every variable of sort s in t, where there exists only a finite set of ground-terms for s, appears exactly once in t. The main theorem is the following:

Theorem2: Assume A is an unambiguous set of linear rooted terms (the terms in A share no ground-instances), B is a set of rooted terms, such that the set of all f-rooted terms in B is semi-linear for every $f \in D$, that appears as root in B. V(A) and V(B) are the variables that appear respectively in A and B and $X = X_A \cup^* X_B$ is a partition of the variables, such that $V(A) \subseteq X_A$ and $V(B) \subseteq X_B$, used for the construction of the complements in A and B respectively. Then B *covers* A (that means $G(A) \subseteq G(B)$) iff one of the following assertions is true:

- $A = \emptyset$;
- There exists a substitution σ that unifies a in A and b in B, i.e. $\sigma(a) = \sigma(b)$, such that $\sigma(a)$ is linear, and: $(B - \{b\}) \cup \{r(b) | r \in \text{compl. of } \sigma'_b\}$ covers $(A - \{a\}) \cup \{r(a) | \ r \in \text{compl. of } \sigma'_a\}$, where σ'_a (resp. σ'_b) denotes the restriction of σ to $Var(a) \cap Dom(\sigma)$ (resp. $Var(b) \cap Dom(\sigma)$), such that all variables of the image of σ'_a (resp. σ'_b) have been isomorphically renamed by variables of $X_A - V(A)$ (resp. $X_B - V(B)$). ○

The algorithm itself is now an implementation of the above result, where for every $f \in D$ A is initialized by $\{f(x_1,\ldots,x_n)\}$ and B by the set of all f-rooted lefthand-sides of (Σ, R).

3.Well-spannedness: Considering literals as boolean atoms, we form for every lefthand-side in R the disjunction of the preconditions of all rules with this lefthand-side. Then we test, if this disjunction is logically equal to true. This can be done by some propositional calculus ([RU88]) or using the rewriting system given in [Hs85]. The system (Σ, R) is called *well-spanned*, if this test is successfull for all lefthand-sides in R and if (Σ, R) is half-spanned. If (Σ, R) is a well-spanned, reductive, left-rooted and semi-linear CTRS, then it is sufficiently complete wrt C.

4.Conclusion: The algorithm, developped according to the above mentioned results applies to a wide class of CTRSs. It is more efficient for this class as algorithms based on the construction of inductive test sets and we think that further investigation in the method could lead to the solution of some of the problems inherent to the "test set"-method (see f. ex. [Zh84]).

BIBLIOGRAPHY

[Hs85] : J. Hsiang: Refutational theorem proving using term rewriting systems; Artificial Intelligence, 25, pp. 255-300, 1985

[LLT87] : A. Lazrek, P. Lescanne, J.J. Thiel: Proving inductive equalities, algorithms and implementation; Rapports de Recherche No. 682, INRIA, Juin 1987

[RU88] : J.L. Rémy, St. Uhrig: An algorithm for testing sufficient completeness of a simple class of conditional specifications; CRIN Report No. 88-R-155, CRIN Nancy, 1988

[Th84] : J.J. Thiel: Stop losing sleep over incomplete data type specifications; Proc. of the eleventh ACM Conference on programming languages, Salt Lake City, (Utah, USA), 1984

[Zh84] : H. Zhang: REVEUR4: Etude et mise en oeuvre de la reecriture conditionnelle, these de 3ieme cycle, Universite de Nancy 1, 1984

Vervollständigung von ordnungssortierten bedingten Gleichungen

Harald Ganzinger

Universität Dortmund
Abteilung Informatik
Postfach 50 05 00

D - 4600 Dortmund 50

Abstract

Order-sorted specifications can be transformed into equivalent many-sorted ones by using injections to implement subsort relations.

In this paper we improve a result of Goguen, Jouannaud, and Meseguer about the relation between order-sorted and many-sorted rewriting.

We then apply recent techniques in completion of many-sorted conditional equations to systems obtained from translating order-sorted conditional equations.

Emphasis will be on ways to overcome some of the problems with non-sort-decreasing rules.

Vervollständigung von Horn Klauseln Programmen mit eingeschränkten Anwendbarkeitseigenschaften*

Hubert Bertling
Fachbereich Informatik, Universität Dortmund
D-4600 Dortmund 50, W. Germany
uucp, bitnet: hubert@unidoi5

18. März 1989

1 Einleitung

Dies ist ein sehr kurz gehaltener Versuch eines Überblicks über eine laufende Arbeit zur Vervollständigung von Horn-Klausel Programmen an der Universität Dortmund. Da diese Arbeiten noch nicht abgeschlossen sind, werden in diesem Überblick die Ansatzpunkte der Arbeit thesenartig vorgestellt und einige Ideen und Konzepte an einfachen Beispielen nicht vollständig formal eingeführt.

Seit mehreren Jahren arbeitet die Dortmunder Gruppe des ESPRIT-Projekts PROSPECTRA an der Entwicklung und Implementierung von Konzepten der "Knuth-Bendix"-Vervollständigung von bedingten Gleichungssystemen. Während dieser Zeit veröffentlichte Arbeiten sind in der Literaturliste angegeben.

Im Rückblick erweisen sich als wichtigste Ideen dieser Arbeiten das Konzept der sog. "nicht-operationalen" Gleichungen vgl. etwa [Gan88a], welches jüngst verallgemeinert wurde auf das Konzept der "anwendungs-eingeschränkten" Gleichungen [BG88]. Letzteres wiederum erlaubt verallgemeinerte Konzepte bedingter Termersetzung, vgl. etwa das Konzept der "quasi-reduktiven" Ersetzungsregel, ebenfalls in [BG88]. Mit Hilfe dieser Konzepte kann ein Standardproblem bedingter Termersetzung, nämlich daß bedingte Gleichungen mit "Extra"-Variablen in den Bedingungen nicht zugelassen werden können, behandelt werden. Beispielsweise kann der Vervollständigungsprozeß so gesteuert werden, daß im finalen System von Gleichungen Gleichungen mit "Extra"-Variablen eliminiert werden können. Die Transitivitätsdefinition

$$(x < y) \doteq true, (y < z) \doteq true \Rightarrow (x < z) \doteq true$$

ist ein Beispiel einer Gleichung mit einer "Extra"-Variablen, nämlich der Variablen y.

In unserer aktuellen Arbeit versuchen wir eine Zusammenschau der bisherigen Ergebnisse unter einer neuen Sicht von "Knuth-Bendix"-Vervollständigung, die allgemeinere Anwendungseinschränkungen von Gleichungen berücksichtigt. Diese kann auf den allgemeinen Horn-Klausel Fall erweitert werden.

*Überlegungen zu dieser Arbeit entstanden im Rahmen des ESPRIT-Projekts PROSPECTRA, ref#390.

2 Skizze des Ansatzes

Der Ansatz kann durch folgende Thesen grob skizziert werden:

1. Wie im unbedingten Gleichungsfall ist der grundsätzliche Mechanismus der Vervollständigung der, "kritische Konseqenzen (Paare)" (Gleichungen bzw. Klauseln) zur betrachteten Klauselnmengemenge hinzuzufügen und zu überprüfen, ob sie simplifiziert bzw. eliminiert werden können.

2. Simplifikation bzw. Elimination gehen jedoch für den bedingten Gleichungsfall wie für den allgemeinen Horn-Klausel Fall über die Simplifikation bzw. Elimination durch Termersetzung, wie sie fuer den unbedingten Gleichungsfall üblich ist, hinaus. Dieser allgemeinere Simplifizier- und Eliminierbarkeitsbegriff von Klauseln ist im allgemeinen nicht entscheidbar. Ein konkretes Vervollständigungssystem wird daher möglichst viele entscheidbare Fälle von Simplifizier- und Eliminierbarbarkeit untersuchen. Der allgemeinere Simplifizier- und Eliminierbarkeitsbegriff von Klauseln ist desweiteren abhängig von den betrachteten Anwendungseinschränkungen. Je restriktiver die Anwendungseinschraenkungen desto schwächer die Simplifizier- und Eliminierbarkeitsvoraussetzungen.

3. Die Simplifizier- Eliminierbarkeit von Klauseln im finalen System von Klauseln ist durch geeignete Addition "zusätzlicher kritischer Konsequenzen" beeinflußbar. D.h. falls gewisse "kritische Konsequenzen" während der Vervollständigung addiert werden, kann die Eliminierbarkeit gewisser Klauseln im finalen System garantiert werden. Diese Beobachtung wird angewendet bei der Behandlung der oben erwähnten nicht-operationalen Gleichungen.

4. In Verallgemeinerung des vorigen Punktes können gewisse Anwendungen einer Klausel im finalen System von Klauseln "überflüssig" gemacht werden durch Addition gewisser "kritische Konsequenzen" während der Vervollständigung. Andere Anwendungen derselben Klausel werden im finalen System von Klauseln nicht "überflüssig" sein.

5. Es können wie im umbedingten Gleichungsfall eine failing und eine unfailing Vervollständigungsvariante unterschieden werden.

6. In der unfailing Variante bedeutet die Vervollständigung nichts anderes als die Wiederherstellung der Vollständigkeit linearen Beweisens, die möglicherweise durch die Einschränkung auf reduktives Refutationsbeweisen verloren wurde.

7. Die Argumentationstechnik für unserer Aussagen ist die Bachmair'sche Beweistransformationstechnik zusammen mit den noetherschen Beweisordnungen.

3 Begriffe erläutert am Beispiel

Einen Eindruck der oben gemachten Aussagen möge das folgende Beispiel geben. Wir betrachten Refutationsbeweise mit der Resolutionsregel als einzigen Inferenzregel. Desweiteren wird nur der Grundfall betrachtet (keine Variablen). Der allgemeine Fall ist sehr viel komplizierter, folgt aber derselben Idee. Wir werden einige für den allgemeineren Fall konzipierten Begriffe nur eingeschränkt am Beispiel einführen.

$p(t_1, t_2, \ldots t_n)$ heißt ein *Atom*, falls p ein Prädikatensymbol und $t_1, t_2, \ldots t_n$ Terme über einer betrachteten Signatur sind. Eine *Klausel* ist ein Paar von Mengen von Atomen,

geschrieben als $\Gamma \Rightarrow \Delta$. Klausel $\Gamma \Rightarrow \Delta$, die höchstens ein Atom in der Konklusion Δ besitzen, heißen *Horn-Klauseln*. Horn-Klauseln mit nicht leerer Konklusion heißen auch *Programm-Klauseln*, während Horn-Klauseln mit leerer Konklusion *Ziel-Klauseln* heißen. Für unser Beispiel betrachten wir nur die Resolutionsregel:

$$\textit{Resolution}: \quad \frac{\Gamma_1 \Rightarrow A \quad B, \Gamma_2 \Rightarrow \Delta_2}{\Gamma_1\sigma, \Gamma_2\sigma \Rightarrow \Delta_2\sigma}$$

wobei σ ein mgu von A and B ist. Wir sagen, die Klausel $\Gamma_1 \Rightarrow A$ wird ***angewandt*** auf die Klausel $B, \Gamma_2 \Rightarrow \Delta_2$. Sei $>$ eine Reduktionsordnung auf Atomen mit zusätzlichen hier nicht näher spezifizierten Einschränkungen der Vergleichbarkeit von Gleichheitsatomen und anderen Atomen. Die Anwendung einer Klausel $A_1, \ldots, A_n \Rightarrow B$ unter Substitution σ heißt ***reduktiv***, falls $B\sigma > A_i\sigma$, $1 \leq i \leq n$. Eine Klausel $A_1, \ldots, A_n \Rightarrow B$ heißt ***reduktiv***, falls $B > A_i$, $1 \leq i \leq n$. Reduktive Klauseln sind in jeder Anwendung reduktiv. Reduktive Anwendungen von Klauseln wirken zielreduzierend.

Beispiel 3.1 *Gegeben das Horn-Klausel Programm* $N = \{A, B \Rightarrow C, \ \Rightarrow A, \ \Rightarrow B\}$ *mit* $C > B > A$.

$$(P): \quad \dfrac{\Rightarrow B \qquad \dfrac{\Rightarrow A \qquad \dfrac{A, B \Rightarrow C \quad C \Rightarrow}{A, B \Rightarrow}}{B \Rightarrow}}{\Rightarrow}$$

(P) beweist, daß C ein log. Konsequenz in N ist. (P) ist reduktiv, da nur reduktive Klauseln angewandt werden. Jeder Schritt ist *linear*, d.h. die angewandte Klausel ist Element von N, und zielreduzierend: $C \Rightarrow \ \gg A, B \Rightarrow \gg B \Rightarrow \ \gg \ \Rightarrow$, wobei $\gg$ die Multisetfortsetzung von $>$ ist.

Bezeichne $\mathcal{R}$ die Anwendungseischränkung: "$A, B \Rightarrow C$ sei nicht-operational", d.h. $A, B \Rightarrow C$ soll nicht angewandt werden müssen. Dann ist N nicht ***vollständig bzgl.*** $\mathcal{R}$, da es keine lineare Alternative zur Anwendung von $A, B \Rightarrow C$ für den Beweis von C gibt. Durch Addition der ***kritischen Konsequenz*** $A \Rightarrow C$ oder $B \Rightarrow C$ oder beider Konsequenzen erlangen wir Vollständigkeit bzgl. $\mathcal{R}$. Sowohl $A \Rightarrow C$ wie auch $B \Rightarrow C$ sind Resolventen zwischen den Programmklauseln $\Rightarrow B$ und $A, B \Rightarrow C$ bzw. zwischen den Programmklauseln $\Rightarrow A$ und $A, B \Rightarrow C$. Sowohl die Menge $N_1 = (N - \{A, B \Rightarrow C\}) \cup \{A \Rightarrow C\}$ wie auch die Menge $N_2 = (N - \{A, B \Rightarrow C\}) \cup \{B \Rightarrow C\}$ ist *vollständig bzgl. R*. D.h in N_1 wie auch in N_2 ist dieselbe Menge von Zielen linear beweisbar wie in N, jedoch ohne $A, B \Rightarrow C$ anwenden zu müssen. In beiden möglichen finalen Systemen N_1 oder N_2 konnte deshalb die nicht-operationale Klausel $A, B \Rightarrow C$ eliminiert werden. Beispielsweise ist in $N_1 \Rightarrow C$ linear beweisbar durch (P_1) und in N_2 durch (P_2)

$$(P_1): \quad \dfrac{\Rightarrow A \qquad \dfrac{A \Rightarrow C \quad C \Rightarrow}{A \Rightarrow}}{\Rightarrow} \qquad\qquad (P_2): \quad \dfrac{\Rightarrow B \qquad \dfrac{B \Rightarrow C \quad C \Rightarrow}{B \Rightarrow}}{\Rightarrow}$$

Wir beobachten, daß es nicht notwendig ist, alle Konsequenzen zu addieren, um Vollständigkeit zu erlangen. Wir bezeichnen daher $\{A \Rightarrow C\}$ bzw. $\{B \Rightarrow C\}$ bzw. $\{A \Rightarrow C, \ B \Rightarrow C\}$ als *faire Selektionen von Konsequenzen von* N.

4 Anwendungseinschränkungen

Hinter jeder Art von "Knuth-Bendix"-Vervollstüdigung steht die Einschränkung beliebiger Beweise auf reduktive Beweise. (Genauer: nur Refutationsbeweise ohne Ermittlung von Lösungssubstitutionen für Ziele werden auf Reduktivität eingeschränkt.) Diese Einschränkung bestimmt noch keine eindeutigen Beweisformen, siehe obiges Beispiel: alle drei Beweise $(P), (P_1), (P_2)$ sind reduktiv. Über Einschränkungen der Anwendung von Horn-Klauseln werden weitere Einschränkungen reduktiver Beweise definiert. "Einschränkung der Anwendung von Horn-Klauseln" bedeutet die Menge der Substitutionen, unter der die Klausel in Refutationsbeweisen angewandt werden darf, einzuschränken. Dabei ist zu beachten:

1. Nur Horn-Klauseln $\Gamma \Rightarrow \Delta$ mit nicht leerer Bedingung Γ dürfen über die Reduktivität hinaus eingeschränkt werden (unfailing Variante), nur reduktive Horn-Klauseln $\Gamma \Rightarrow \Delta$ mit nicht leerer Bedingung Γ dürfen weiter eingeschränkt werden (failing Variante).

2. Termersetzung mit Ersetzungsregeln im konventionellen Sinn kann als spezielle Einschränkung aufgefaßt werden: "Nur Klauseln, die unter allen Substitutionen reduktiv sind, dürfen angewandt werden."

Eigenschaften fairer Selektionen

1. Eine Selektion von Konsequenzen ist eine Teilmenge der Menge aller Resolventen/Paramodulanten zwischen Programmklauseln N.

2. Eine faire Selektion garantiert die Transformierbarkeit bzgl. $\mathcal{R}$ unzulässiger reduktiver Beweise in zulässige reduktive und lineare Beweise. Der Begriff der fairen Selektion wird über diese Eigenschaft definiert. Unter Transformierbarkeit wird hier die Anwendbarkeit einer Beweistransformationsregel aus einer fest gegebenen Menge solcher Regeln verstanden. Beispiel einer solchen Beweistransformationsregel (Schema) ist die folgende:

$$\cfrac{\cfrac{\Rightarrow A \qquad \cfrac{A,\Gamma \Rightarrow C \quad C \Rightarrow}{A,\Gamma \Rightarrow}}{\Gamma \Rightarrow}}{\dots} \quad\Longrightarrow\quad \cfrac{\cfrac{\cfrac{\Rightarrow A \quad A,\Gamma \Rightarrow C}{\Gamma \Rightarrow C} \qquad C \Rightarrow}{\Gamma \Rightarrow}}{\dots}$$

   Beweise auf der rechten Seite der Transformationsregel zerstören möglicherweise die Linearität. Dies ist dann der Fall, wenn weder $\Gamma \Rightarrow C$ in N ist noch es einen alternativen linearen $\mathcal{R}$-eingeschränkten Beweis für den Subbeweis

$$\cfrac{\cfrac{\Gamma \Rightarrow C \quad C \Rightarrow}{\Gamma \Rightarrow}}{\dots}$$

   gibt. Offensichtlich hängt die Fairness einer Selektion von der Anwendungseinschränkung $\mathcal{R}$ ab.

3. Die Menge aller möglichen Resolventen/Paramodulanten zwischen Programmklauseln in Nicht-Variablen-Positionen ist eine faire Selektion von Konsequenzen. Diese ist endlich, vergleichbar der endlichen Mengen kritischer Paare in der konventionellen Vervollständigung.

4. Sei S eine faire Selektion. Dann ist auch jede Obermenge $S \cup S'$ eine faire Selektion.

5. In der Praxis wird man Selektionen wählen, deren Fairneßeigenschaft entscheidbar ist. Sie werden im allgemeinen nicht minimal sein.

5 Kriterium für Vollständigkeit

Failing Variante: $\mathcal{R}$ ist Einschränkung auf Anwendung (quasi)-reduktiver Klauseln. Ein Klauselprogramm N ist vollständig bzgl. der Anwendungseinschränkung $\mathcal{R}$ und der durch $>$ induzierten Beweisordnung $>_{\mathcal{P}}$, falls

1. die Teilmenge der nicht reduktiven unbedingten Klauseln $U \subseteq N$ von $N - U$ subsumiert wird bzgl. $>_{\mathcal{P}}$ und R.

2. $S_{\mathcal{R}}(N)$ eine faire Selektion von Konsequenzen in N für $\mathcal{R}$ ist und

3. $S_{\mathcal{R}}(N)$ von N subsumiert wird bzgl. $>_{\mathcal{P}}$ und R.

Im Beispiel wird die faire Selektion $\{A \Rightarrow C\}$ von der finalen Klauselmenge $\{A \Rightarrow C, \Rightarrow A, \Rightarrow B,\}$ subsumiert bzgl. $>_{\mathcal{P}}$ und $\mathcal{R}$. Aufgrund der Anwendungseinschränkung $\mathcal{R}$ und der Fairneß der Selektion $S_{\mathcal{R}}(N)$ können wir schließen, daß auch $A, B \Rightarrow C$ von N subsumiert wird.bzgl. $>_{\mathcal{P}}$ und R. Der Subsumptionsbegriff ist abgestützt auf der zugrundeliegenden Beweisordnung, die wiederum von der zugrundeliegenden Reduktionsordnung induziert wird. Darüberhinaus ist er abhängig von der betrachteten Anwendungseinschränkung $\mathcal{R}$. Es soll hier nicht näher darauf eingegangen werden.

6 Abstrakte Vervollständigung

Die von uns angestrebte Vervollständigungstechnik läßt sich durch die folgenden abtrakten Inferenzregeln beschreiben.

Abstrakte Addition:

$$\frac{N}{N \cup \{\Gamma \Rightarrow \Delta\}} \quad \text{falls } \Gamma \Rightarrow \Delta \in S_{\mathcal{R}}(N)$$

Abstrakte Elimination:

$$\frac{N \cup \{\Gamma \Rightarrow \Delta\}}{N} \quad \textit{falls } \Gamma \Rightarrow \Delta \textit{ subsumiert wird von } N \textit{ bzgl. } >_{\mathcal{P}} \textit{ und } \mathcal{R}$$

Abstrakte Simplifikation:

$$\frac{N \cup \{\Gamma_1 \Rightarrow \Delta_1\}}{N \cup \{\Gamma_2 \Rightarrow \Delta_2\}} \quad \textit{falls } \Gamma_1 \Rightarrow \Delta_1 \textit{ subsumiert wird von } N \cup \{\Gamma_2 \Rightarrow \Delta_2\} \textit{ bzgl } >_{\mathcal{P}} \textit{ und } \mathcal{R}$$

$S_{\mathcal{R}}(N)$ ist eine Teilmenge aller Paramodulanten/Resolventen in N. In konkreten Vervollständigungssytemen sind diese abstrakten Regeln durch Mengen konkreter Regeln für jede abstrakte Regel ersetzt. Chancen für erfolgreiche Vervollständigungen können durch die Wahl möglichst kleiner fairer Selektionen wie auch durch eine "mächtige" Menge von Simplifikations- und Eliminationsregeln, die die Simplifizier- und Eliminierbarkeit in möglichst vielen Fällen aufdecken, verbessert werden.

7 Veröffentlichungen

[BGS88] Bertling, H., Ganzinger, H. and Schäfers, R.: CEC: A system for conditional equational completion. User Manual Version 1.4, PROSPECTRA-Report M.1.3-R-7.0, U. Dortmund, 1988.

[Gan87] Ganzinger, H.: A Completion procedure for conditional equations. Report 234, U. Dortmund, 1987, also in: Proc. 1st Int'l Workshop on Conditional Term Rewriting, Orsay, 1987, Springer LNCS 308, 1988, 62–83 (revised version to appear in J. Symb. Computation).

[Gan88a] Ganzinger, H.: Completion with History-Dependent Complexities for Generated Equations. In Sannella, Tarlecki (eds.): Recent Trends in Data Type Specifications. Springer LNCS 332, 1988, 73–91.

[Gan88b] Ganzinger, H.: Order-Sorted Completion: The Many-Sorted Way. Report 274, FB Informatik, Univ. Dortmund, 1988. Extended abstract to appear in Proc. TAPSOFT(CAAP) '89, Barcelona.

[BG88] Bertling, H., Ganzinger, H.: Completion-time optimization of rewrite-time goal solving. PROSPECTRA-Bericht M1.3-R-12.0, 1988. (Erscheint in Proc. 3rd Int. Conf. on Rewriting Techniques and Applications, Chapel Hill, 1989).

Unification in Monoidal Theories

WERNER NUTT

Deutsches Forschungsinstitut für Künstliche Intelligenz (DFKI),
6750 Kaiserslautern, West Germany

1. Introduction

We introduce a class of equational theories by which we generalize several well-known theories for which unification problems have been studied. Among them are the theories of commutaive monoids (AC), commutative idempotent monoids (ACI), and abelian groups (AG). These theories have the common characteristic that unification algorithms for them basically consist in solving some kind of linear equation system.

The same is true of monoidal theories. Every monoidal theory determines canonically a semiring, an algebraic structure that can be thought of as a generalized ring. Then every unification problem can be translated into a linear equation system and vice versa.

Having established this correspondence between unification and linear algebra, we are able to characterize the unification type (unitary, finitary, infinitary, nullary) of monoidal theories in algebraic terms. For instance, an application of Hilbert's Basis Theorem gives a sufficient criterion for a monoidal theory to be unitary.

Monoidal theories can be characterized in categroical terms: The category consisting of finitely generated algebras as objects and homomorphisms as arrows is semi-additive. On the other hand, if for an equational theory this category is semi-additive, then by a signature transformation the theory can be turned into a monoidal theory. Thus monoidal theories cover the same subject as the commutative theories defined in (Baader 1989).

In the sequel, we give a short account on the basic defintions and results concerning monoidal theories. A detailed presentation is given in (Nutt 1989).

2. Monoidal Theories and Semirings

Terms, substitutions, equational theories, algebras and other basic notions of unification theory are defined as usual (Kirchner 1989).

An equational theory $\mathcal{E}$ is *monoidal* if its signature Σ consists of a constant 0, a binary symbol $+$, and a finite number of unary symbols, such that $+$ is associative and commutative, 0 is the identity for $+$, and every unary symbol h is a homomorphism for $+$ and 0, i.e. $\mathcal{E}$ contains the equalities $h(x+y) \doteq h(x) + h(y)$ and $h(0) \doteq 0$.

Obviously, the theories of commutaive monoids (AC), commutative idempotent monoids (ACI), and abelian groups (AG) are monoidal. Generally, monoidal theories describe varieties of abelian monoids with homomorphisms.

A *semiring* is a tuple $(\mathcal{S}, +, 0, \cdot, 1)$ such that $(\mathcal{S}, +, 0)$ is an abelian monoid, $(\mathcal{S}, \cdot, 1)$ is a monoid, and all $\alpha,\ \beta,\ \gamma \in \mathcal{S}$ satisfy the equalities $(\alpha + \beta) \cdot \gamma = \alpha \cdot \gamma + \beta \cdot \gamma$, $\alpha \cdot (\beta + \gamma) = \alpha \cdot \beta + \alpha \cdot \gamma$, and $0 \cdot \alpha = \alpha \cdot 0 = 0$

We call the binary operations $+$ and $\cdot$ the addition and the multiplication, respectively, of the semiring. A semiring is *commutative* if its multiplication is commutative. Semirings are different from rings in that they need not be groups with respect to addition.

Examples. The set $\mathbf{N}$ of natural numbers with usual addition and multiplication is a semiring. Every ring is a semiring. In particular, the integers $\mathbf{Z}$ with usual addition and multiplication form a semiring. The set $\{0,1\}$ becomes a semiring $\mathcal{S}_{\mathrm{ACI}}$ if we define $1{+}1 := 1$ and extend addition and multiplication as required by the axioms for a semiring.

Analogously to fields, for every semiring $\mathcal{S}$ we can define $\mathcal{S}$-modules as generalized vector spaces and linear mappings between $\mathcal{S}$-modules. Especially, the cartesian product $\mathcal{S}^n$ becomes an $\mathcal{S}$-module if addition and scalar multiplication are defined pointwise.

As usual, a linear map $\sigma: \mathcal{S}^m \to \mathcal{S}^n$ between free $\mathcal{S}$-modules can be described by an $n \times m$-matrix C_σ with entries from $\mathcal{S}$, and every such matrix defines a linear mapping. The *transpose* σ^t of σ is the linear mapping corresponding to the transpose C_σ^t of the matrix C_σ.

Every monoidal theory $\mathcal{E}$ defines a semiring $\mathcal{S}_\mathcal{E}$ as follows: Let 1 be a variable symbol. Then the carrier of $\mathcal{S}_\mathcal{E}$ is $\mathcal{F}_\mathcal{E}(1)$, the free $\mathcal{E}$-algebra over $\{1\}$, addition and zero are inherited from $\mathcal{F}_\mathcal{E}(1)$, the unit is $\overline{1}$, i.e. the $\mathcal{E}$-equivalence-classe 1, and multiplication of two $\mathcal{E}$-classes of terms $\overline{s},\overline{t}$ is defined as $\overline{s} \cdot \overline{t} := \overline{\langle 1 \leftarrow t \rangle s}$, i.e. the product is obtained by replacing all occurrences of 1 in s with t.

The semiring $\mathcal{S}_\mathcal{E}$ mirrors properties of $\mathcal{E}$. A monoidal theory is a *theory of groups* if for some term t it contains the equation $x + t \doteq 0$. Intuitively, this means that there exist inverse elements for the addition. A monoidal theory is a *theory with commuting homomorphisms* if for all h, $h' \in \mathcal{H}$ it contains the equation $h(h'(x)) \doteq h'(h(x))$.

THEOREM 2.1.
1. *$\mathcal{S}_\mathcal{E}$ is a ring if and only if $\mathcal{E}$ is a theory of groups.*
2. *$\mathcal{S}_\mathcal{E}$ is commutative if and only if $\mathcal{E}$ is a theory with commuting homomorphisms.*

Next we show that it is just a matter of perspective whether one views an algebraic structure as an $\mathcal{E}$-algebra or as an $\mathcal{S}_\mathcal{E}$-module. Let A be an $\mathcal{E}$-algebra and $a \in A$. Evaluation in a is defined as the unique homomorphism $\varepsilon_a: \mathcal{T}_\Sigma(1) \to A$ from Σ-terms ov er 1 to A satisfying $\varepsilon_a(1,a) = a$. Then A can be turned into an $\mathcal{S}_\mathcal{E}$-module by defining the scalar multiplication as $\overline{s}a := \varepsilon_a(s)$ for $\overline{s} \in \mathcal{S}_\mathcal{E}$ and $a \in A$. On the other hand, every $\mathcal{S}_\mathcal{E}$-module M can be turned into an $\mathcal{E}$-algebra by interpreting every unary function symbol h as the function $h^M(m) := \overline{h(1)} \cdot m$ for $m \in M$.

Switching from $\mathcal{E}$-algebras to $\mathcal{S}_\mathcal{E}$-modules and backwards turns homomorphisms into linear mappings and linear mappings into homomorphisms. In particular, the free algebra on n generators $\mathcal{F}_\mathcal{E}(x_1, \ldots, x_n)$ viewed as a module is isomorphic to the module $\mathcal{S}^n$, and vice versa.

3. Unification Problems in Monoidal Theories

Our view of unification is slightly more abstract than the usual one. An $\mathcal{E}$-unification problem is given by two homomorphisms (i.e. substitutions) $\sigma,\tau: \mathcal{F}_\mathcal{E}(X) \to \mathcal{F}_\mathcal{E}(Y)$ between finitely generated free $\mathcal{E}$-algebras. A unifier of σ and τ is a homomorphism $\delta: \mathcal{F}_\mathcal{E}(Y) \to \mathcal{F}_\mathcal{E}(Z)$ such that $\delta\sigma = \delta\tau$.

Now the instance relation on homomorphisms ("δ is more general than η"), complete and minimal complete sets of unifiers as well as most general unifiers can be defined as usual.

When we are considering a monoidal theory $\mathcal{E}$, we can view free algebras as modules $\mathcal{S}_{\mathcal{E}}^{n}$ and homomorphisms as linear mappings. Therefore we can treat unification problems in the framework of linear algebra over semirings.

Let σ, $\tau{:}\, \mathcal{S}_{\mathcal{E}}^{l} \to \mathcal{S}_{\mathcal{E}}^{m}$ be linear. The *kernel* of σ and τ is the set $ker(\sigma, \tau) := \{a \in \mathcal{S}_{\mathcal{E}}^{l} \mid \sigma(a) = \tau(a)\}$. The kernel of σ and τ is a submodule of $\mathcal{S}_{\mathcal{E}}^{l}$. The *image* of σ is the set $im\,\sigma := \{b \in \mathcal{S}_{\mathcal{E}}^{m} \mid \exists a \in \mathcal{S}_{\mathcal{E}}^{l}.\sigma(a) = b\}$. The image of σ is a submodule of $\mathcal{S}_{\mathcal{E}}^{m}$.

The next theorems are the basic results on monoidal theories. They relate unification properties to algebraic properties. The first theorem characterizes the instance relation.

THEOREM 3.1. *Let $\delta{:}\, \mathcal{S}_{\mathcal{E}}^{m} \to \mathcal{S}_{\mathcal{E}}^{n}$ and $\eta{:}\, \mathcal{S}_{\mathcal{E}}^{m} \to \mathcal{S}_{\mathcal{E}}^{k}$ be linear mappings. Then δ is more general than η if and only if $im\,\eta^{t} \subseteq im\,\delta^{t}$.*

Unifiers can be characterized in terms of images and kernels.

THEOREM 3.2. *Let $\sigma,\tau{:}\, \mathcal{S}_{\mathcal{E}}^{l} \to \mathcal{S}_{\mathcal{E}}^{m}$ and $\delta{:}\, \mathcal{S}_{\mathcal{E}}^{m} \to \mathcal{S}_{\mathcal{E}}^{n}$ be linear mappings. Then the following equivalences hold:*

1. *δ is a unifier of σ and τ $\iff$ $im\,\delta^{t} \subseteq ker(\sigma^{t}, \tau^{t})$*
2. *δ is a most general unifier of σ and τ $\iff$ $im\,\delta^{t} = ker(\sigma^{t}, \tau^{t})$.*

The type of a unification problem depends, loosely speaking, on the size of the kernel of the two linear mappings.

THEOREM 3.3. *Let $\sigma,\tau{:}\, \mathcal{S}_{\mathcal{E}}^{l} \to \mathcal{S}_{\mathcal{E}}^{m}$ be linear mappings.*

1. *There exists a most general unifier of σ and τ if and only if $ker(\sigma^{t}, \tau^{t})$ is finitely generated.*
2. *For every unifier η of σ and τ there exists a more general unifier δ if and only if $ker(\sigma^{t}, \tau^{t})$ is not finitely generated.*

Since a unification problem in a monoidal theory is either of type unary or of type finitary, the same is true of the whole theory.

COROLLARY 3.4. *(1-0-Alternative) A monoidal theory is either of unification type 1 or of type 0.*

A sufficient criterion for a monoidal theory to be unitary follows from Theorem 3.3.

COROLLARY 3.5. *Let $\mathcal{E}$ be a monoidal theory. If $\mathcal{F}_{\mathcal{E}}(1)$ is finite, then $\mathcal{E}$ is of unification type 1.*

References

Baader, F. (1989). "Unification in Commutative Theories", in Kirchner, C. (ed.), *Special Issue on Unification, Journal of Symbolic Computation.*

Kirchner, C. (1989), editor. *Special Issue on Unification, Journal of Symbolic Computation.*

Nutt, W. (1989). "Unification in Monoidal Theories", *SEKI Report*, Universität Kaiserslautern, forthcoming.

Rewrite Systems for Semigroup Varieties

Franz Baader
IMMD 1, Universität Erlangen-Nürnberg
Martensstraße 3, D-8520 Erlangen (West Germany)
E-mail: baader@informatik.uni-erlangen.de

Extended Abstract

There are two possible ways of using finite canonical rewrite systems to solve word problems for semigroup varieties. On the one hand we may consider term rewrite systems (TRS) which realize associativity by rules. On the other hand we may reduce modulo associativity, i.e. consider words instead of terms. In this case we have word rewrite systems (WRS).

Advantages of TRS over WRS are:

1) For a finite, terminating TRS the confluence property is decidable. One has to check only finitely many critical pairs. Since associative unification is infinitary there may be infinitely many critical pairs obtained by superposition of two rules of a WRS.

2) The Knuth-Bendix Algorithm may be used to complete a given TRS.

Example 1. Consider RB = $\{ x\cdot(y\cdot z) = (x\cdot y)\cdot z,\ x\cdot(y\cdot x) = x \}$.

The TRS R = $\{ (x\cdot y)\cdot z \to x\cdot(y\cdot z),\ x\cdot(y\cdot x) \to x \}$ terminates but is not confluent since the terms $x\cdot z$ and $x\cdot(y\cdot z)$ are R-irreducible and RB-equivalent.

The Knuth-Bendix Algorithm yields the canonical system

$$S = \{ (x\cdot y)\cdot z \to x\cdot z,\ x\cdot(y\cdot z) \to x\cdot z,\ x\cdot x \to x \}$$

for RB.

But there are semigroup theories which have finite canonical word rewrite systems and do not have finite canonical term rewrite systems.

Example 2. Consider LR = $\{ x\cdot(y\cdot z) = (x\cdot y)\cdot z,\ x\cdot x = x,\ x\cdot(y\cdot x) = x\cdot y \}$.

R = $\{ xx \to x,\ xyx \to xy \}$ is a finite canonical WRS for LR but there is no finite canonical TRS for LR. The Knuth-Bendix Algorithm generates the following infinite canonical TRS for LR:

$$\{ \quad (x\cdot y)\cdot z \to x\cdot(y\cdot z),\ x\cdot x \to x,$$
$$x_1\cdot(x_2\cdot(\ldots(x_{n-1}\cdot(x_n\cdot x_1))\ldots)) \to x_1\cdot(x_2\cdot(\ldots(x_{n-1}\cdot x_n)\ldots)),$$
$$x_1\cdot(x_2\cdot(\ldots(x_{n-1}\cdot(x_1\cdot x_n))\ldots)) \to x_1\cdot(x_2\cdot(\ldots(x_{n-1}\cdot x_n)\ldots));\ n \geq 2 \quad \}$$

We may now ask whether there is a semigroup variety with decidable word problem but without finite canonical WRS (TRS). The lattice of all varieties of idempotent semigroups yields countably many natural examples of that kind.

Theorem

1) There are countably many varieties of idempotent semigroups and they all have decidable word problem (Birjukov (1970), Fennemore (1971), Gerhard (1970)).

2) There are only three varieties of idempotent semigroups with finite canonical TRS and nine varieties of idempotent semigroups with finite canonical WRS (Baader (1989)).

The proof of 2) for WRS is rather involved. It requires a thorough knowledge of the solution of the word problem for varieties of idempotent semigroups. The proof of 2) for TRS uses the fact that canonical term rewrite systems for regular semigroup theories are of a very specific form. Regular means that the variables occurring on the left side or right side of an identity are the same. Thus LR is regular but RB is not regular.

Lemma

Let E be a regular semigroup theory and let R be a canonical TRS for E. Then there is a reduction chain $(x\cdot y)\cdot z \xrightarrow{+}_R x\cdot(y\cdot z)$ or $x\cdot(y\cdot z) \xrightarrow{+}_R (x\cdot y)\cdot z$. If, in addition, R is reduced then (modulo variable renaming) $(x\cdot y)\cdot z \to x\cdot(y\cdot z) \in R \cup R^{-1}$.

The reduced canonical system S for RB shows that the condition "E regular" is necessary. Now the following may be proved, using the fact that for all n the identity

$$x_1\cdot(x_2\cdot(\ldots(x_{n-1}\cdot(x_n\cdot(x_1\cdot(x_2\cdot(\ldots(x_{n-1}\cdot x_n)\ldots)))))\ldots)) = x_1\cdot(x_2\cdot(\ldots(x_{n-1}\cdot x_n)\ldots))$$

is valid in any idempotent semigroup.

Proposition

Let E be a regular semigroup theory defining a variety of idempotent semigroups. Then there does not exist a finite canonical TRS for E.

References

Baader, F. (1989). Unifikation und Reduktionssysteme für Halbgruppenvarietäten. Ph.D. Dissertation, Universität Erlangen-Nürnberg.

Birjukov, A.P. (1970). Varieties of Idempotent Semigroups. *Algebra i Logika* **9**. English translation in *Algebra and Logic* **9** (1970).

Fennemore, C.F. (1971). All Varieties of Bands I, II. *Math. Nachr.* **48**.

Gerhard, J.A. (1970). The Lattice of Equational Classes of Idempotent Semigroups. *J. Algebra* **15**.

RESTRICTIONS OF CONGRUENCES GENERATED BY FINITE CANONICAL STRING-REWRITING SYSTEMS

- Extended Abstract -

Friedrich Otto

Department of Computer Science, State University of New York,
Albany, N.Y. 12222, U.S.A.

currently visiting at

Fachbereich Informatik, Universität Kaiserslautern,
6750 Kaiserslautern
E-Mail Adresse: Otto @ uklirb.uucp

Let $\mathfrak{M}$ be a monoid that is given through a finite string-rewriting system S on alphabet Σ. One way to attempt to solve the word problem for $\mathfrak{M}$ consists in trying to determine a finite canonical string-rewriting system R on Σ that is equivalent to S by way of completion. However, even if the word problem for $\mathfrak{M}$ is decidable, there may not exist a finite canonical system R that is equivalent to S as exemplified by the system S = {[aba,bab]} [2]. In some cases introducing additional letters as abbreviations for certain words from Σ^* will help to overcome this difficulty, but this does not always work, either. In fact, there exist finitely presented monoids with decidable word problem that cannot be presented by any finite canonical string-rewriting system, no matter which finite set of generators we use [4]. On the other hand, every finitely generated monoid with a decidable word problem can be embedded into a monoid that is presented by a finite string-rewriting system which is canonical on the embedded monoid [1]. Here we are concerned with a decision problem that is closely related to this embedding theorem, and which we call the PROBLEM OF RESTRICTION:

INSTANCE: A finite string-rewriting system R_1 on alphabet Σ_1 such that the word problem for R_1 is decidable, and a finite string-rewriting system R_2 on alphabet $\Sigma_2 \supsetneq \Sigma_1$.

QUESTION: Is $\xleftrightarrow{*}_{R_1} = \xleftrightarrow{*}_{R_2}|_{\Sigma_1^* \times \Sigma_1^*}$, i.e., is the congruence $\xleftrightarrow{*}_{R_1}$ the restriction of the congruence $\xleftrightarrow{*}_{R_2}$ to Σ_1^* ?

The following results have been obtained.

Theorem 1. The following restricted version of the PROBLEM OF RESTRICTION is undecidable in general:

INSTANCE: A finite, length-reducing, and confluent string-rewriting system R_2 on alphabet Σ_2, and a subalphabet $\Sigma_1 \subsetneq \Sigma_2$.

QUESTION: Is $\xleftrightarrow{*}_{R_2}|_{\Sigma_1^* \times \Sigma_1^*} = id_{\Sigma_1^*}$?

Here a string-rewriting system R is called **length-reducing** if $|l| > |r|$ holds for each rule $(l \longrightarrow r)$ of R. It is called **monadic** if it is length-reducing, and also $r \in \Sigma \cup \{e\}$ holds for each rule $(l \longrightarrow r)$ of R.

Theorem 2. The following restricted version of the PROBLEM OF RESTRICTION is undecidable in general:

INSTANCE: A finite monadic string-rewriting system R_2 on alphabet Σ_2 such that the word problem for R_2 is decidable, and a subalphabet $\Sigma_1 \subsetneq \Sigma_2$.

QUESTION: Is $\xleftrightarrow{*}_{R_2}|_{\Sigma_1^* \times \Sigma_1^*} = id_{\Sigma_1^*}$?

Theorem 3. The PROBLEM OF RESTRICTION is decidable, if R_2 is being restricted to finite, monadic, and confluent string-rewriting systems.

The combinatorial restrictions of Theorem 3 can be relaxed somewhat if algebraic restrictions are placed on the monoids presented.

Theorem 4. The following variant of the PROBLEM OF RESTRICTION is decidable:

INSTANCE: A finite string-rewriting system R_1 on alphabet Σ_1 such that the word problem for R_1 is decidable, and the monoid $\mathfrak{M}_1 := \Sigma_1^*/\xleftrightarrow{*}_{R_1}$ is a group, and a finite monadic string-rewriting system R_2 on alphabet $\Sigma_2 \supsetneq \Sigma_1$ such that R_2 is confluent on $[e]_{R_2}$.

QUESTION: Is $\xleftrightarrow{*}_{R_1} = \xleftrightarrow{*}_{R_2}|_{\Sigma_1^* \times \Sigma_1^*}$?

Additional algebraic restrictions allow to even consider non-monadic string-rewriting systems R_2.

Theorem 5. The following variant of the PROBLEM OF RESTRICTION is decidable:

INSTANCE: A finite Noetherian string-rewriting system R_1 on alphabet Σ_1 such that R_1 is confluent on $[e]_{R_1}$, and such that the monoid $\mathfrak{M}_1 := \Sigma_1^*/\overset{*}{\longleftrightarrow}_{R_1}$ is a group, and a finite, weight-reducing, and confluent string-rewriting system R_2 on alphabet $\Sigma_2 \supsetneq \Sigma_1$ such that the monoid $\mathfrak{M}_2 := \Sigma_2^* /\overset{*}{\longleftrightarrow}_{R_2}$ is a group.

QUESTION: Is $\overset{*}{\longleftrightarrow}_{R_1} = \overset{*}{\longleftrightarrow}_{R_2}|_{\Sigma_1^* \times \Sigma_1^*}$?

The proofs of Theorems 1 to 3 can be found in [3].

References

1. **G. Bauer**; Zur Darstellung von Monoiden durch konfluente Regelsysteme; Ph.D. dissertation, Fachbereich Informatik, Universität Kaiserslautern, 1981.

2. **D. Kapur, P. Narendran**; A finite Thue system with decidable word problem and without equivalent finite canonical system; Theoretical Computer Science 35 (1985), 337-344.

3. **F. Otto**; Restrictions of congruences generated by finite canonical string-rewriting systems; Proceedings of RTA-89, to appear.

4. **C. Squier**; Word problems and a homological finiteness condition for monoids; Journal of Pure and Applied Algebra 49 (1987), 201-217.

Deciding Equality under Generalized Associative and Commutative Axioms

J. Avenhaus
I. Frobenius

Universität Kaiserslautern, FB Informatik
D-6750 Kaiserslautern

1. Introduction

Deciding equality is the following problem
INPUT: E, a set of equational axioms, and two terms s,t.
QUESTION: Does $s =_E t$ hold ?

The rewriting approach to this problem is first to transform E into a convergent term rewriting system R. If this is successful then E-equality can be tested by rewriting: $s =_E t$ iff $\hat{s} \equiv \hat{t}$ where $\hat{s}$ and $\hat{t}$ are the unique R-normal forms of s and t.
Unfortunately, the Knuth-Bendix completion procedure - which transforms E into R - may fail because i) it produces an infinite number of rules or ii) it encounters equations that cannot be ordered by the reduction order in use. We are interested in tools to overcome these problems for special axiom systems expressing generalized associativity and generalized commutativity. Both schematas of axioms allow only finite congruence classes, so the E-equality is decidable. But preprocessing E into R will increase the efficiency. Such problems appear when one uses globally finite rewriting systems instead of rewriting modulo a congruence, see [Göbel 1987].

2. Generalized associativity

Intuitively, by an equation expressing generalized associativity we mean equations of the form

$$f((f(x_1,x_2,x_3), x_4, x_5) = f(x_1, f(x_2,x_3,x_4), x_5) \quad \text{or}$$

$$g(g(x_1,x_2), g(x_3,x_4)) = g(x_1, g(x_2, g(x_3,x_4))) .$$

One characteristic of such an equation s = t is that s and t, expressed as trees, have the same frontier. Another characteristic is that they are length-equal.

Definition: Let T(F,V) be the set of terms built with operators in F and variables in V, let $s,t \in T(F,V)$
a) s,t are **leaf-equal** if frontier(s) ≡ frontier(t)
b) s,t are **length-equal** if $|s|_F = |t|_F$ and $|s|_x$ for all $x \in V$.

The key observation of our approach is

Lemma 1: The Knuth-Bendix completion procedure started with a leaf- (resp. a length-) equal set of equations produces only leaf- (resp. length-) equal rules and equations.

So, to avoid abortion of the Knuth-Bendix procedure when started with a set of leaf-equal equations, one has to construct a reduction ordering which is total on leaf-equal terms. To do so, we start with a total precedence $\cdot\!>$ on F and denote by $\#(s)$ the multiset of leaves of term s (where s is represented as a tree).

Definition: Length order $>_L$

$s \equiv f(s_1,\dots,s_m) >_L t \equiv g(t_1,\dots,t_n)$

iff lexicographically (α) $\#(s) \not\supseteq \#(t)$ or (β) $|s| > |t|$

or (γ) $f \cdot\!> g$ or (δ) $(s_1,\dots,s_n) >_{L,lex} (t_1,\dots,t_n)$

Lemma 2: $>_L$ is a reduction order and total on leaf-equal terms.

This together with Lemma 1 gives an unfailing Knuth-Bendix completion procedure for leaf-equal equational systems E as input. It can be used as an decision procedure for E-equality if E is leaf- and length-equal. For, in this case one can produce during completion the rules in increasing length and stop for the problem "$s =_E t$?" as soon as all rules $l \longrightarrow r$ with length $|l| \leq |s|$ are produced.

Theorem: The Knuth-Bendix completion procedure working with a length order does not abort for leaf-equal equational systems E as input. It gives a decision procedure for E-equality if E is both leaf- and length-equal.

3. Generalized Commutativity

By a permutation equation we mean an equation of the form

$$f(x_1,\dots,x_n) = f(x_{\pi(1)},\dots,x_{\pi(n)})$$

where π is a permutation. We write $f(\bar{x}) = f(\pi(\bar{x}))$ in this case.
The problem now is to solve E-equality "$s =_E t$?" for

$$E = \{f(\bar{x}) = f(\pi_i(\bar{x}));\ i = 1,\dots,k\}\ .$$

Of course, no permutation equation can be oriented by a reduction order. So one may start the unfailing Knuth-Bendix procedure. But it will generate too much equations, namely O(n !) equations in many cases.

We notice

a) applying $f(\bar{x}) = f(\pi(\bar{x}))$ to $f(t_1,\dots,t_n)$ results in the transformation of $\bar{t} = t_1,\dots,t_n$ into $\pi(\bar{t})$ or $\pi^{-1}(\bar{t})$.

b) $f(x_1,\dots,x_n) =_E f(x_{\pi(1)},\dots,x_{\pi(n)})$ iff $\pi \in G$, where $G = <\pi_1,\dots,\pi_k>$ is the subgroup of the full permutation group $\mathfrak{S}_n$ generated by $\pi_1,\dots,\pi_k$.

c) $f(t_1,\dots,t_n) =_E f(s_1,\dots,s_n)$ iff for some $\pi \in G$ $t_i =_E s_{\pi(i)}$ for $i = 1,\dots,n$.

There are polynomial time algorithms to transform $U = \{\pi_1,...,\pi_k\}$ into $V = \{\sigma_1,...,\sigma_m\}$ such that $G = <U> = <V>$ and the problem "$\pi \in G$?" can be solved in time $O(n)$, see [Fürst et al 1980]. This algorithm can be used to order sequences $a = a_1,...,a_n$ with $a_i \neq a_j$ for $i \neq j$ modulo G in polynomial time, i.e. to compute the lexicographical minimal $\pi(a) = a_{\pi(1)}...a_{\pi(n)}$ among all $\pi \in G$.

The permutations σ_i in the transformed set V of generators for G can directly be translated into a set of rewrite rules which rewrite the term $f(a_1,.....,a_n)$ into the minimal term $f(a_{\pi(1)},...,a_{\pi(n)})$ in its E-congruence class.

In general one can solve the problem "$t \equiv f(t_1,...,t_n) =_E s \equiv f(s_1,...,s_n)$?" by sorting both terms bottom up modulo G. Then $s =_E t$ iff the sorted forms are identical. Unfortunately, to do so we need sorting modulo G also if some element a_i in the sequence $a \equiv a_1,...,a_n$ appear more than once, and in this case the above mentioned sorting algorithm becomes non-deterministic. Even worse, this generalized problem of sorting modulo G is NP-hard, see [Babai and Luks 1983]. But nevertheless, this approach of sorting modulo G leads in general to more efficient algorithms then computing all permutations in G. And this basically is what the unfailing Knuth-Bendix procedure does. For example, from V one can see whether $G = \mathfrak{S}_n$. In this case sorting modulo G is the normal sorting and can be done in time $O(n \cdot \log n)$.

References:

[R. Göbel]: Ground Confluence, RTA-Conference 1987, LNCS 256, pp. 156-167.

[M. Fürst, J. Hopcroft, E. Luks]: Polynomial time algorithms for permutation groups, FOCS-Conference 1980, pp. 36-41.

[L. Babai, E. Luks]: Canonical labeling of graphs, STOC-Conference 1983, pp. 171-183.

Rewriting in abelian monoids as an appropriate means for the simulation of Petri nets [1]

Harald - Reto Fonio

GMD, Institut für Systemtechnik, 5205 S^t Augustin; email fonio@gmdzi.uucp

Among the specification tools for the design of distributed systems Petri nets are getting more and more important. One of the reasons for this fact is that the constructive characteristics of the theory of Petri nets can be understood also as a graphical paradigm and hence be applied as an advantageous support for the intuition when specifying parallel processes. For the analysis of the total behaviour of distributed systems it is necessary to describe the local characteristics and the parts of these systems and to relate them with each other. It is characteristical for these systems that processes can be executed concurrently or be started in conflict with each other. But it is also of some importance that the data to be processed as well as the processes can be specified locally. But then the localisation of partial aspects in distributed systems requires to work out the interdependencies of the system parts using them as characteristics for the scheduling of these system parts. All these aspects are covered in a natural way by Petri nets, for example by high level nets.

Working with high level nets means to specify both the data and the processes which manipulate the data. The data being described for example initially by abstract data types, this suggests to combine the expressive power of Petri nets with the theory of abstract data types. First valuable steps into this direction have been shown up in /Kra 89/ and in /Sch 89/. But albeith Petri nets are to be understood 'quite easily' it was surprisingly difficult to fit Petri nets into such an algebraic context which were able to reflect the cited aspects of locality in its own light. An appropriate description of the data space had to be found which provides the states space over which the actions of the Petri nets could be defined. One of the reasons for these efforts was of course to open the specification of distributed systems towards the methodology of abstract data types and to use the term rewriting methods being developed in the ADT - context for the analysis and for the simulation of these systems. In this context it was a useful hint that Petri nets themselves are already substitution systems.

A satisfactory solution of this problems has been given by Meseguer and Montanari in a brilliant paper (/MeMo 88/). Also Kaplan has already worked into this direction (/Ka 87/). Independently of these authors I have introduced abelian additive monoids allowing cancellation, shortly AC - monoids, for spanning up the states space. The cancellation property together with the addition operation allows for a unique algebraic representation of the substitution mechanism within these monoids, a point where associative and commutative term rewriting techniques come in quite naturally. Combinations of cancelling and adding items within terms of such monoids are nothing else

1 This work was performed in the frame of the ESPRIT Project 125 - GRASPIN.

than rewrite rules for these structures. Thus the cancellation/addition combinations form the algebraic link between the rewrite rules mentioned above and the firing rules of Petri nets. This provides the bridge for the algebraic simulation of Petri nets by associative commutative term rewriting within freely generated abelian structures.

AC - monoids, which are decomposable into direct sums, allow to describe local aspects using the injections and projections. Here we think for example of sequential and parallel composition of local steps. AC - monoids can be generated freely over sets of interesting data where the latter can be presented for example by initial abstract data types. Essentially the free construction guarantees the invariance, the logical independence of the description of the data with respect to the substitution steps in the underlying monoid.

Now an abelian monoid freely generated over some set is nothing else than the class of multisets (bags) defined with respect to that set, and there is a uniquely defined injective embedding of this set into the monoid generated over it. It can be specified by a parameterized specification using the embeddings as copy operations and interpreting the set of the data as the actualization of the parameter. In this context the cancellation/addition combinations appear as rules for such specifications. This opens the way towards specifying Petri nets and their behaviours by ADT's.

It remains to single out the class of Petri nets which we are going to treat: the high level nets. This will be done in three conceptional steps. We remind of Kowalski's paradigm "ALGORITHM = LOGIC + CONTROL" which must be a guide line for the specification of a distributed system,too. Thus we speak on one hand about the states as objects within a state space spanned up by the data and on the other hand about the actions on these states as state manipulating actions leaving the data invariant.

The Petri net realization begins with a triple $N = (S, T, F)$ of sets S of S-elements, T of T-elements and F of directed arcs connecting only S- and T-elements such that these sets obey the Petri net axioms (/BeFe 86/). There is a well defined 'projection' $\varphi : F \longrightarrow S$. A resource R is associated to the net N which is a family $R = \{M_s\}_{s\in S}$ of at most countable sets of "token types". Markings of the net N are functions $\mu : S \longrightarrow \bigcup_{s\in S} \mathcal{M}(M_s)$ [1] such that $\mu(s) \in \mathcal{M}(M_s)$ for any $s \in S$.

Labels of the net N are functions $\lambda : F \longrightarrow \bigcup_{f\in F} \mathcal{M}(M_{\varphi(f)})$ such that $\lambda(f) \in \mathcal{M}(M_{\varphi(f)})$ for any $f \in F$. A capacity K on the net N is a function $K : \bigcup_{s\in S} \{s\} \times M_s \longrightarrow \overline{\mathbb{N}}$ [2] such that for any $s \in S$ $K(s,x) \in \{0, \infty\}$ for all but finitely many $x \in M_s$; we call a token type $x \in M_s$ K-finite if $K(s,x) < \infty$ and nontrivial if $0 < K(s,x)$. A marking μ is K-admissible if for any $s \in S$ and $x \in M_s$ $\#(x,\mu(s)) \leq K(s,x)$. This closes the first step of the definition.

The K-complementation of the net N is a net $N_K := (S_K, T, F_K)$ which contains the net N, where $S_K \setminus S$ consists of all those complements $\underline{s}$ of $s \in S$ such that M_s allows for nontrivial K-finite token types x, and $F_K \setminus F$ contains for

1 $\mathcal{M}(A)$ denotes for any set A the class of finite multisets over A, and $\overset{n}{\underset{i=1}{+}} n_i.a_i$ for naturals n_i and $a_i \in A$ denotes the multiset containing each a_i n_i-times. For any multiset m and $x \in m$ $\#(x,m)$ is the number of occurrences of x in m.

2 $\overline{\mathbb{N}}$ denotes the naturals together with ∞. $f(x) = \infty$ for some function $f : A \longrightarrow \overline{\mathbb{N}}$ means that f is undefined on x.

any $t \in T$ all those $(\underline{s},t)$ (resp. $(t,\underline{s})$) for which $(t,s) \in F$ (resp. $(s,t) \in F$) . The resource R is complemented to $R_K := \{M_s\}_{s \in S_K}$ where $M_{\underline{s}}$ consists of the nontrivial K-finite token types of M_s . The label λ is complemented to λ_K by extending λ by $\lambda_K((\underline{s},t)) := \lambda((t,s))$ and $\lambda_K((t,\underline{s})) := \lambda((s,t))$ for $\underline{s} \in S_K \setminus S$. K-admissible markings μ are complemented to K-admissible markings μ_K by extending μ by $\mu_K(\underline{s}) := (\sum^{+}_{x \in M_{\underline{s}}} K(\underline{s},x).x) \setminus \mu(s)$ for $\underline{s} \in S_K \setminus S$ (the sums involved are finite). Given a K-admissible marking μ or the complemented marking μ_K we define for any T-element t a firing rule $t : \mu \longrightarrow \mu'$ or $t_K : \mu_K \longrightarrow \mu'_K$ by extending the corresponding definitions from place/transition nets (/BeFe 86/) to our case. We say that a T-element t or the corresponding firing rule of t has concession in a K-admissible marking μ if for all $s \in S_K$ and $a \in F_K$ with $a = (s,t)$ the relation $\lambda_K(a) \subseteq \mu_K(s)$ holds. We apply firing rules only in the case of concession. The reason for K-complementations is to avoid dealing explicitly with the postconditions in the definition of concession. Thus the second step of the definition is performed, and in the sequal we deal only with K-complete nets in the sense of the definition above.

The markings stand for the states; they are generated over the resources distributed over the S-elements. The state changing actions are realized by the firing rules. Clearly they do not interfere with any internal structure on the sets M_s . Thus Petri net specifications meet Kowalski's paradigm.

The token types represent the data which are processed by applying the firing rules of the Petri net. For the third step leading to the definition of high level nets we assume SPEC = (Σ, EQ) to be a computable specification for a signature Σ = (SO, OP) over sets SO of sorts and OP of operation symbols as well as a set EQ of equations together with a SO-sorted family $X = \{X_{so}\}_{so \in SO}$ of sets X_{so} of so-sorted variables. We say that the specification SPEC is computable if it allows for a convergent set RU of rewrite rules reflecting all the equations generated by the set EQ. We split the resource $R = \{M_s\}_{s \in S}$ into two resources $R_G = \{M_{G,s}\}_{s \in S}$ and $R_X = \{M_{X,s}\}_{s \in S}$ where the members $M_{G,s}$ (resp. $M_{X,s}$) are sets of ground terms $T_{\Sigma,so}$ (resp. terms with/without variables $T_\Sigma(X)_{so}$ for the same S-element s and the same sort so !) over sorts so of interest in Σ . We redefine moreover markings over ground terms as $\mu : S \longrightarrow \bigcup_{s \in S} \mathcal{M}(M_{G,s})$ as well labels over all the terms as $\lambda : F \longrightarrow \bigcup_{f \in F} \mathcal{M}(M_{X,\varphi(f)})$. But then the concession checks require equation solving via suitable ground substitutions, and the applications of the firing rules are done with respect to the ground substitutions determined occasionally in the concession checks. It turns out again that the firing rules leave the description of the data invariant.

The result of these three steps are S-sorted families of multisets or abelian monoids. Taking finally the coproduct over these families we get the total abelian monoids providing the algebraic states spaces for the distributed systems.

References :

/BeFe/ E. Best, C. Fernandez : "Notions and Terminology on Petri Net Theory - revised version"; Petri Net Newsletter 23, april 1986.

/Ka 87/ S. Kaplan : "Rewriting with a nondeterministic choice operator" ; Paris, Univers. Paris - Sud, TR METEOR/t4/LRI/3-6 (1987).

/Kra 89/ B. Kraemer : "Concepts, Syntax and Semantics of SEGRAS : A Specification Language for Distributed Systems" ; to appear as GMD - Bericht, GMD - Birlinghoven, St. Augustin, FRG, 1989.

/MeMo 88/ J. Meseguer, H. Montanari : "Petri Nets Are Monoids" ; SRI International, Menlo Park CA94025, USA, 1989.

/Sch 89/ H. W. Schmidt : "Specification and Correct Implementation of Non - Sequential Systems Combining Abstract Data Types and Petri Nets"; GMD - Bericht 176, GMD - Birlinghoven, St. Augustin, FRG, 1989.

Workshop Termersetzung: Grundlagen und Anwendungen , Kaiserlautern, March 1989

Knuth–Bendix Procedure and Buchberger Algorithm — A Synthesis [*)]

Franz Winkler
Institut für Mathematik and
Research Institute for Symbolic Computation
Johannes Kepler Universität Linz, Austria

Abstract

The Knuth–Bendix procedure for the completion of a rewrite rule system and the Buchberger algorithm for computing a Gröbner basis of a polynomial ideal are very similar in two respects: they both start with an arbitrary specification of an algebraic structure (axioms for an equational theory and a basis for a polynomial ideal, respectively) which is transformed to a very special specification of this algebraic structure (a complete rewrite rule system and a Gröbner basis of the polynomial ideal, respectively). This special specification allows to decide many problems concerning the given algebraic structure. Moreover, both algorithms achieve their goals by employing the same basic concepts: formation of critical pairs and completion.

Although the two methods are obviously related, the exact nature of this relation remains to be clarified. Based on previous work we show how the Knuth–Bendix procedure and the Buchberger algorithm can be seen as special cases of a more general completion procedure.

1. Introduction

The Buchberger algorithm BU has been introduced by B. Buchberger in 1965 [Bu 65], [Bu 85a] and it solves the following problem:

given a finite set F of multivarate polynomials over a field, construct a finite set F' of multivariate polynomials such that $\equiv_F = \equiv_{F'}$ and $\rightarrow_{F'}$ is Church–Rosser.

Here, for a set F of polynomials, $\equiv_F$ is the ideal congruence modulo the ideal generated by F (i.e. $f \equiv_F g \iff f - g \in \text{ideal}(F)$) and $\rightarrow_F$ is a certain Noetherian reduction relation on polynomials induced by F [Bu 85a] with the property that $\leftrightarrow_F^*$ (the reflexive–symmetric–transitive closure of $\rightarrow_F$) is equal to $\equiv_F$. If $F' = \text{BU}(F)$, then the Church–Rosser property guarantees, that for arbitrary polynomials f, g the congruence $f \equiv_F g$ can be decided by computing normal forms of f and g modulo $\rightarrow_{F'}$ and checking for syntactic equality. A basis F' with this property is usually called a *Gröbner basis* [Bu 85a].

*) Work reported herein has been supported by the *Fonds zur Förderung der wissenschaftlichen Forschung* under project Nr. P6763.

Such a Gröbner basis can be computed by the *Buchberger algorithm* BU in the following way:

```
$F' \leftarrow$ BU($F$);
[$F$ and $F'$ are finite sets of multivariate polynomials over a field.
$\equiv_F = \equiv_{F'}$ and $\rightarrow_{F'}$ is Church–Rosser.]
$F' \leftarrow F$;
while not all critical pairs of $F'$ are considered do
(*)  choose a critical pair $(p_1, p_2)$ of $F'$;
     reduce $(p_1, p_2)$ to normal forms $(q_1, q_2)$ modulo $\rightarrow_{F'}$;
(**) if $q_1 \neq q_2$ then $F' \leftarrow F' \cup \{q_1 - q_2\}$ endif
endwhile ■
```

The two basic strategies of the algorithm are the formation of critical pairs in (*) and the successive completion step (**). A critical pair of F' is constructed in the following way: choose two different polynomials f, g in F'; reduce the least common multiple of the leading terms of f and g by f, getting p_1, and by g, getting p_2; then (p_1, p_2) is a critical pair of F'. Instead of reducing (p_1, p_2) to normal forms (q_1, q_2) and checking for syntactic equality, one could reduce $p_1 - p_2$ and check for equality to 0. The polynomial $p_1 - p_2$ is usually called the S-polynomial of f and g [Bu 85a]. Buchberger has shown [Bu 65], [Bu 85a] that this algorithm terminates for all inputs and computes a Gröbner basis for ideal(F).

The same basic strategies have been used independently by D.E. Knuth and P.B. Bendix [KB 67] in the context of an equational theory T over an algebra $\mathbb{T}$ of first-order terms. The Knuth–Bendix procedure solves the following problem:

given a finite set E of equations between first-order terms, construct a finite set E' of equations such that $\equiv_E = \equiv_{E'}$ and $\rightarrow_{E'}$ is Church–Rosser and Noetherian.

Here, for a set E of first-order equations [HO 80], $\equiv_E$ is the equational theory generated by E, i.e. the set of all equations $s = t$ which can be derived from E, $E \vdash s = t$ [BL 83]. $\rightarrow_E$ is the reduction relation on terms induced by E viewed as a system of rewrite rules with $\equiv_E = \leftrightarrow_E^*$. Again, the Church–Rosser property guarantees that $s \equiv_E t$ can be decided by reducing s and t to normal forms modulo $\rightarrow_{E'}$ and checking for syntactic equality. A finite set of equations E, viewed as a system of rewrite rules, such that $\rightarrow_E$ is Church–Rosser and Noetherian is called a *canonical* rewrite rule system.

The *Knuth–Bendix procedure* KB attempts to compute a canonical rewrite rule system in the following way:

```
$E' \leftarrow$ KB($E$);
[$E$ and $E'$ are finite sets of equations of first-order terms which can be viewed
as Noetherian rewrite rule systems.
$\equiv_E = \equiv_{E'}$ and $\rightarrow_{E'}$ is Church–Rosser.]
$E' \leftarrow E$;
while not all critical pairs of $E'$ are considered do
     choose a critical pair $(c_1, c_2)$ of $E'$;
     reduce $(c_1, c_2)$ to normal forms $(d_1, d_2)$ modulo $\rightarrow_{E'}$;
     if $d_1 \neq d_2$ then
```

```
        if $\rightarrow_{E'\cup\{d_1=d_2\}}$ is Noetherian then $E' \leftarrow E' \cup \{d_1 = d_2\}$
        elsif $\rightarrow_{E'\cup\{d_2=d_1\}}$ is Noetherian then $E' \leftarrow E' \cup \{d_2 = d_1\}$
        else exit with failure
    endif
endwhile ■
```

For the notion of a critical pair we refer to [BL 83]. We say that an equation $s = t$ can be viewed as a rewrite rule $s \rightarrow t$ if every variable occurring in t also occurs in s. A set of equations $E = \{s_1 = t_1, \dots, s_n = t_n\}$ can be viewed as a rewrite rule system $\{s_1 \rightarrow t_1, \dots, s_n \rightarrow t_n\}$ if every equation $s_i = t_i$ in it can be viewed as a rewrite rule $s_i \rightarrow t_i$. In contrast to the Buchberger algorithm there are situations in which the Knuth–Bendix procedure may terminate with **failure** or run forever.

Certain types of equations cannot be handled by the Knuth–Bendix procedure: a commutativity axiom immediately destroys the Noetherianity of the reduction, and an associativity axiom together with other equations can cause the procedure to run indefinitely. Peterson and Stickel [PS 81] have proposed to keep such equations in an equational theory T (the equations in T are not viewed as rewrite rules) and do all the computations in KB modulo this equational theory T, i.e. not terms t in $\mathbb{T}$ are reduced but equivalence classes $[t]_T$ in $\mathbb{T}_{/T}$. This approach works whenever a complete unification algorithm modulo this theory T exists. For technical reasons the equational theory E has to be modified so that it becomes T–compatible. For theories T consisting of commutativity and associativity axioms this is a straightforward process.

The striking similarity between the Buchberger algorithm and the Knuth–Bendix procedure have been observed in [Lo 81], [BL 83], [Bu 85b]. Llopis de Trias [Ll 83] and Kandri-Rody and Kapur [KK 83] have made first attempts to clarify the relationship between the two methods. The problem with [Ll 83] is that is does not deal adequately with the arithmetic on the coefficients of the polynomials in the Buchberger algorithm. In [KK 83] the useful idea of separating simplification of coefficients from reduction of polynomials is introduced. The problem with [KK 83] is that it does not really show that the two methods can be viewed as special cases of a general procedure, but that the correctness proofs can be arranged in similar ways. Le Chenandec [Le 86] gives a completion algorithm for commutative polynomials over rings generated by a finite set G of generators. His method does not apply to the case where the base coefficients belong to a field, since fields cannot be described equationally. In [Wi 84] various ideas of these papers together with [Hu 80] have been used for demonstrating the exact nature of the relationship between BU and KB.

2. Theoretical results

In the following we suppose that M is an arbitrary set, $\rightarrow$ a Noetherian relation on M, and $\Rightarrow$ a Noetherian confluent relation on M. By x, y, z, u, v, w we denote elements of M. $\leftarrow$, $\leftrightarrow$, $\rightarrow^+$, $\rightarrow^*$ are the inverse, the symmetric closure, the transitive closure, and the reflexive–transitive closure of $\rightarrow$, respectively.

Def.: $\rightarrow$ is *confluent modulo* $\Rightarrow$ iff for all x, y, x', y' such that $x' \leftarrow^* x \Leftrightarrow^* y \rightarrow^* y'$ there are

x'', y'' such that $x' \rightarrow^* x'' \Leftrightarrow^* y'' \leftarrow^* y'$ (i.e., since $\Rightarrow$ is confluent, $x' \rightarrow^* x'' \Downarrow^* y'' \leftarrow^* y'$). ■

Lemma 2.1: Let $\rightarrow$ be confluent modulo $\Rightarrow$. Then for all x, y, z such that $y(\Leftarrow \cup \leftarrow)^* x \rightarrow^* z$ and y is irreducible modulo $\rightarrow \cup \Rightarrow$ and z is irreducible modulo $\rightarrow$, we have $z \Rightarrow^* y$. ■

Later on we will separate the reduction $\rightarrow$ of polynomials in the Gröbner basis algorithm from the simplification $\Rightarrow$ of the coefficients in the polynomials. What we ultimately want to achieve is that the combination $\rightarrow \cup \Rightarrow$ is a confluent relation. As the following theorem shows, it is enough to guarantee confluence of $\rightarrow$ modulo $\Rightarrow$.

Theorem 2.2: If $\rightarrow$ is confluent modulo $\Rightarrow$, then $\rightarrow \cup \Rightarrow$ is confluent.

Proof: Let $\rightarrow$ be confluent modulo $\Rightarrow$. Suppose x, y, z are such that $y(\Leftarrow \cup \leftarrow)^* x(\rightarrow \cup \Rightarrow)^* z$. Let x' be a normal form of x modulo $\rightarrow$ and y', z' be normal forms of y, z modulo $\rightarrow \cup \Rightarrow$, respectively. Then by Lemma 2.1 $y' \Leftarrow^* x' \Rightarrow^* z'$. Since y' and z' are also in normal form modulo $\Rightarrow$ and $\Rightarrow$ is confluent, we have $y' = z'$. (See Figure 2.1) ■

$$\begin{tikzcd}
 & & x \arrow[dll, "\Leftarrow \cup \leftarrow"', "*"] \arrow[d, "*"] \arrow[drr, "\rightarrow \cup \Rightarrow", "*"'] & & \\
y \arrow[dd, "\rightarrow \cup \Rightarrow"', "*"] & & x' \arrow[ddll, Rightarrow, "*"] \arrow[ddrr, Rightarrow, "*"'] & & z \arrow[dd, "\rightarrow \cup \Rightarrow", "*"'] \\
 & \text{Lemma 2.1} & & \text{Lemma 2.1} & \\
y' & & = & & z'
\end{tikzcd}$$

Figure 2.1

It is essential for an effective completion procedure that the confluence property of the reduction relation under consideration can be checked locally. This program can also be carried out for the notion of confluence modulo $\Rightarrow$.

Def.: $\rightarrow$ is *locally confluent modulo* $\Rightarrow$ iff
(L1) for all x, y, z with $y \leftarrow x \rightarrow z$ there are y', z' such that $y \rightarrow^* y' \Downarrow^* z' \leftarrow^* z$ and
(L2) for all x, y, z with $z \leftarrow x \Leftrightarrow y$ there are y', z' such that $z \rightarrow^* z' \Downarrow^* y' \leftarrow^* y$. ■

Def.: $\Rightarrow$ is *orthogonal to* $\rightarrow$ iff
(O1) for all x, y, y' with $x \Rightarrow y \rightarrow^+ y'$ there are x'', y'' such that $x \rightarrow^+ x'' \Downarrow^* y'' \leftarrow^* y'$ and
(O2) for all x, y, x' with $x' \leftarrow^+ x \Rightarrow y$ there are x'', y'' such that $x' \rightarrow^* x'' \Downarrow^* y'' \leftarrow^* y$. ■

With these definitions we get the following theorem.

Theorem 2.3: Let $\rightarrow \cup \Rightarrow$ be Noetherian, and $\Rightarrow$ orthogonal to $\rightarrow$. Then $\rightarrow$ is confluent modulo $\Rightarrow$ if and only if $\rightarrow$ is locally confluent modulo $\Rightarrow$. ■

We are especially interested in the case where the reduction relation $\rightarrow$ is induced by a rewrite rule system R, i.e. $\rightarrow := \rightarrow_R$, on a set of terms modulo an associative–commutative theory.

Theorem 2.4: Let T be an equational theory over the term algebra $\mathbb{T}$, R a T–compatible rewrite rule system, $\Rightarrow$ a Noetherian confluent relation on $\mathbb{T}_{/T}$ which is stable and com-

patible (i.e. if $[s]_T \Rightarrow [t]_T$, σ a substitution, p an occurrence in u, then $[\sigma(s)]_T \Rightarrow [\sigma(t)]_T$ and $[u[p \leftarrow s]]_T \Rightarrow [u[p \leftarrow t]]_T$) such that $\rightarrow_R \cup \Rightarrow$ is Noetherian and $\Rightarrow$ is orthogonal to $\rightarrow_R$.
Then $\rightarrow_R$ is confluent modulo $\Rightarrow$ if and only if for all critical pairs $([s]_T, [t]_T)$ of R modulo T there are $[s']_T, [t']_T$ such that $[s]_T \rightarrow_R^* [s']_T \Downarrow^* [t']_T \leftarrow_R^* [t]_T$. ■

Theorem 2.4 immediately leads to the following general completion procedure:

```
$R' \leftarrow$ COMPLETE($R, T, \Rightarrow$);
[$R$ is a finite Noetherian rewrite rule system over the term algebra $\mathbb{T}$,
$T$ an equational theory for which there exists a complete unification algorithm,
$\Rightarrow$ a Noetherian confluent stable and compatible relation over $\mathbb{T}_{/T}$,
such that $\rightarrow_R \cup \Rightarrow$ is Noetherian and $\Rightarrow$ is orthogonal to $\rightarrow_R$.
$R'$ is a finite Noetherian rewrite rule system such that
$(\rightarrow_R \cup \Rightarrow)^* = (\rightarrow_{R'} \cup \Rightarrow)^*$ and $\rightarrow_{R'}$ is confluent modulo $\Rightarrow$.]
$R' \leftarrow T$-compatible extension of $R$;
while not all critical pairs of $R'$ have been considered do
    choose a critical pair $(c_1, c_2)$ of $R'$;
    reduce $(c_1, c_2)$ to normal forms $(d_1, d_2)$ modulo $\rightarrow_{R'} \cup \Rightarrow$;
    if $d_1 \neq d_2$ then
        if   terms $s, t$ can be constructed such that $d_1$ and $d_2$
             have a common successor modulo $\rightarrow_{R' \cup \{s \rightarrow t\}} \cup \Rightarrow$ and
             $\rightarrow_{R' \cup \{s \rightarrow t\}} \cup \Rightarrow$ is Noetherian
        then $R' \leftarrow T$-compatibel extension of $R' \cup \{s \rightarrow t\}$
        else exit with failure
        endif
    endif
endwhile   ■
```

3. A common ancestor to BU and KB

The procedure COMPLETE can be specialized both to the Knuth–Bendix procedure and to the Buchberger algorithm. We get KB from COMPLETE by letting $\Rightarrow$ be the identity and $T = \emptyset$.

It is a little bit more complicated to specialize COMPLETE to BU. We have to meet the following requirements:

(C1) give an injective mapping from the polynomial ring $K[x_1, \ldots, x_n]$ into some term algebra $\mathbb{T}$ modulo an equational theory T,
(C2) give a simplification relation $\Rightarrow$ on $\mathbb{T}_{/T}$,
(C3) construct a rewrite rule system R for a given basis F of a polynomial ideal

such that

(P1) $\rightarrow_R \cup \Rightarrow$ simulates $\rightarrow_F$, i.e. every reduction step modulo $\rightarrow_F$ can be considered as a series of reduction steps modulo $\rightarrow_R \cup \Rightarrow$,
(P2) there exists a finite complete unification algorithm for T,

(P3) R is a finite Noetherian rewrite rule system,
(P4) $\Rightarrow$ is a Noetherian confluent stable and compatible relation over $\mathbb{T}_{/T}$,
(P5) $\rightarrow_R \cup \Rightarrow$ is Noetherian,
(P6) $\Rightarrow$ is orthogonal to $\rightarrow_R$.

<u>Ad (C1)</u>: The term algebra $\mathbb{T}$ contains the binary function symbols $\oplus, \odot$, the unary function symbol $\ominus$, the constants $X_1, \ldots, X_n$ and a for every $a \in K$, and the denumerable set of variables $V = \{x_0, x_1, \ldots\}$ (for convenience we denote the first variables by $x, y, z, w, \ldots$, similarly for the constants X_i). As the equational theory T we choose the associative–commutative theory of $\oplus$ and $\odot$, i.e. a basis for T is

$$\{x \oplus y = y \oplus x,\ (x \oplus y) \oplus z = x \oplus (y \oplus z),\ x \odot y = y \odot x,\ (x \odot y) \odot z = x \odot (y \odot z)\}.$$

A nonzero polynomial $f = \sum_{i=1}^{m} a_i X_1^{e_{i1}} \cdots X_n^{e_{in}}$ is mapped onto the equivalence class of $s_1 \oplus (s_2 \oplus \cdots \oplus (s_{m-1} \oplus s_m) \cdots)$ modulo T, where s_i is the obvious description of $a_i X_1^{e_{i1}} \cdots X_n^{e_{in}}$ in $\mathbb{T}$. The zero polynomial is mapped onto the constant 0. This mapping is called *term*. We let $\odot$ have higher precedence than $\oplus$, so that we can omit parentheses because of the associativity of the operators. So, for instance, the polynomial $3x^2y^2 - 2x^2y + 4x - 5 \in \mathbb{Q}[x, y]$ is mapped onto the equivalence class $[3 \odot X \odot X \odot Y \odot Y \oplus (-2) \odot X \odot X \odot Y \oplus 4 \odot X \oplus (-5)]_T$. *term* is an injective mapping from $K[x_1, \ldots, x_n]$ onto $\mathbb{T}_{/T}$.

<u>Ad (C2)</u>: The simplification relation $\Rightarrow$ on $\mathbb{T}_{/T}$ is defined in such a way that it simulates the operations involving the constants of the coefficient field K.

$$[s]_T \Rightarrow [t]_T \ :\Longleftrightarrow \text{ there are } s' \equiv_T s, t' \equiv_T t, \text{ such that}$$
$$t' = s'[p \leftarrow u] \text{ for some occurrence } p \text{ in } s' \text{ and } s'_{/p} \hookrightarrow u,$$

where for coefficients $a_1, a_2 \in K$ and terms $s \in \mathbb{T}$:

$a_1 \odot a_2 \hookrightarrow a_1 \cdot a_2$ $\qquad$ $a_1 \oplus a_2 \hookrightarrow a_1 + a_2$
$\ominus a_1 \hookrightarrow -a_1$ $\qquad$ $\ominus(a_1 \odot s) \hookrightarrow (-a_1) \odot s$
$a_1 \odot s \oplus a_2 \odot s \hookrightarrow (a_1 + a_2) \odot s$ $\qquad$ $0 \oplus a_1 \odot s \hookrightarrow a_1 \odot s$
$0 \odot s \hookrightarrow 0$

The relation $\Rightarrow$ is well–defined on $\mathbb{T}_{/T}$.

<u>Ad (C3)</u>: We start with the rules of the canonical rewrite rule system for the ring structure modulo the AC–theory T which are not already incorporated in $\Rightarrow$, i.e.

$x \odot (y \oplus z) \rightarrow (x \odot y) \oplus (x \odot z)$ $\qquad$ $\ominus(\ominus x) \rightarrow x$
$\ominus(x \oplus y) \rightarrow (\ominus x) \oplus (\ominus y)$ $\qquad$ $x \odot (\ominus y) \rightarrow \ominus(x \odot y)$

We call this rewrite rule system R_r.

For every polynomial f in the ideal basis F we include the following rule in the rewrite sytem R_F:

$$term(lt(f)) \rightarrow \ term(red(f)),$$

where $lt(f)$ is the leading term of f and $red(f)$ is the reductum of f.

We let the rewrite rule system R be the union of R_r and R_F. ■

This completes the simulation (C1) — (C3). Now it can be shown that (P1) — (P6) hold [Wi 84]. We illustrate this simulation of BU by the following example.

Example: We consider the ideal basis

$$F = \{\underbrace{x^2y - x^2 + 2xy}_{f_1},\ \underbrace{y^2 - y + 1}_{f_2}\} \subseteq \mathbb{Q}[x,y].$$

The power products are ordered according to the graduated lexicographic ordering. First there is only one critical pair of F, namely the one resulting from the reduction of x^2y^2 modulo f_1 and f_2, respectively.

$$\begin{tikzcd}[row sep=small] & x^2y - 2xy^2 \arrow[r, "f_1"] & -2xy^2 + x^2 - 2xy \arrow[r, "f_2"] & x^2 - 4xy + 2x \\ x^2y^2 \arrow[ur, "f_1"] \arrow[dr, "f_2"'] & & & \\ & x^2y - x^2 \arrow[r, "f_1"] & -2xy & \end{tikzcd}$$

So we add $f_3 = x^2 - 2xy + 2x$ to the ideal basis and proceed. All the other critical pairs lead to common successors, so $\{f_1, f_2, f_3\}$ is a Gröbner basis for the ideal.

The rewrite rule system corresponding to F is

$$R_F = \{(1): \underbrace{X \odot X \odot Y}_{s_1} \rightarrow \underbrace{X \odot X \oplus (-2) \odot X \odot Y}_{t_1},\quad (2): \underbrace{Y \odot Y}_{s_2} \rightarrow \underbrace{Y \oplus (-1)}_{t_2}\}.$$

Applying the procedure COMPLETE to $R = R_\tau \cup R_F$, we first have to construct a T-compatible extension $R^e$of R. Because T is an AC-theory, this means adding a new rule $u \circ s \rightarrow u \circ t$ (u a new variable) for every rule $s \rightarrow t$ with outermost operator $\circ \in \{\oplus, \odot\}$.

$$R' = R^e = R_\tau \cup R^e_\tau \cup R_F \cup$$
$$\underbrace{\{\underbrace{u \odot X \odot X \odot Y}_{s^e_1} \rightarrow \underbrace{u \odot (X \odot X \oplus (-2) \odot X \odot Y)}_{t^e_1}, \underbrace{v \odot Y \odot Y}_{s^e_2} \rightarrow \underbrace{v \odot (Y \oplus (-1))}_{t^e_2}\}}_{R^e_F}.$$

The only interesting critical pair results from unifying s^e_1 and s^e_2 by the unifier $\sigma = \{u \leftarrow Y,\ v \leftarrow X \odot X\}$. For brevity, we will omit the operator $\odot$ from now on.

$$\begin{tikzcd}[column sep=tiny] & {[\sigma(s^e_1)]_T = [XXYY]_T = [\sigma(s^e_2)]_T} \arrow[dl, "(1^e)"'] \arrow[dr, "(2^e)"] & \\ \begin{array}{l} [Y(XX \oplus (-2)XY)]_T \rightarrow_{R_\tau} \\ [XXY \oplus (-2)XYY]_T \rightarrow_{(1)} \\ [XX \oplus (-2)XY \oplus (-2)XYY]_T \rightarrow_{(2)} \\ [XX \oplus (-2)XY \oplus (-2)XY \oplus (-2)(-1)X]_T \Rightarrow^* \\ [XX \oplus (-4)XY \oplus 2X]_T \end{array} & & \begin{array}{l} [XX(Y \oplus (-1))]_T \rightarrow_{R_\tau} \\ [XXY \oplus (-1)XX]_T \rightarrow_{(1)} \\ [XX \oplus (-2)XY \oplus (-1)XX]_T \Rightarrow^* \\ [(-2)XY]_T \end{array} \end{tikzcd}$$

We add the new rule $(3): XX \rightarrow 2XY \oplus (-2)X$ to R' in order to guarantee a common successor of the two normal forms of $[XXYY]_T$ modulo $\rightarrow_{R'} \cup \Rightarrow$. We also have to add the extended rule, so that R' remains T-compatible.

All the other critical pairs of R' have common successors. So $\rightarrow'_R$ with $R' = R_\tau \cup R^e_\tau \cup \{(1),(2),(3)\} \cup \{(1^e),(2^e),(3^e)\}$ is confluent modulo $\Rightarrow$. ∎

We want to point out that we do not claim or intend to be able to improve the efficiency of BU or KB by such a simulation. However, we think that the general completion

procedure COMPLETE might help to understand the intricate relationship between two important algorithmic concepts for constructing canonical rewrite systems.

References

[Bu 65] B. Buchberger: *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal.* Dissertation, Univ. Innsbruck, Austria (1965).

[Bu 85a] B. Buchberger: "Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory". In: *Multidimensional Systems Theory*, N.K. Bose (ed.), 184–232, D. Reidel Publ. Comp. (1985).

[Bu 85b] B. Buchberger: "Basic Features and Development of the Critical-Pair/Completion Procedure", *Rewriting Techniques and Applications*, J.-P. Jouannaud (ed.), Springer Lecture Notes in Comp. Sci. 202, 1–45 (1985).

[BL 83] B. Buchberger, R. Loos: "Algebraic Simplification", in: *Computer Algebra — Symbolic and Algebraic Computation, 2nd ed.*, Buchberger, Collins, Loos (eds.), Springer–Verlag, 11–44 (1983).

[Hu 80] G.P. Huet: "Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems", *J.ACM* **27/4**, 797–821 (1980).

[HO 80] G.P. Huet, D.C. Oppen: "Equations and Rewrite Rules — A Survey", in: *Formal Language Theory*, R.V. Book (ed.), Academic Press, 349–405 (1980).

[KK 83] A. Kandri-Rody, D. Kapur: "On Relationship between Buchberger's Grobner Basis Algorithm and the Knuth–Bendix Completion Procedure", General Electric Technical Report No. 83CRD286, Schenectady, New York (1983).

[KB 67] D.E. Knuth, P.B. Bendix: "Simple Word Problems in Universal Algebra", *Proc. of the Conf. on Computational Problems in Abstract Algebra*, Oxford, 1967, J. Leech (ed.), Pergamon Press (1970).

[Le 86] P. Le Chenandec: *Canonical Forms in Finitely Presented Algebras*, Pitman, London (1986).

[Ll 83] R. Llopis de Trias: "Canonical Forms for Residue Classes of Polynomial Ideals and Term Rewriting Systems", Techn. Rep., Univ. Autonoma de Madrid, Division de Matematicas (1983).

[Lo 81] R. Loos: "Term Reduction Systems and Algebraic Algorithms", *Proc. 5th GI Workshop on Artif. Intell.*, Bad Honnef, Springer–Verlag, Informatik Fachberichte 47, 214–234 (1981).

[PS 81] G.E. Peterson, M.E. Stickel: "Complete Sets of Reductions for Some Equational Theories", *J.ACM* **28/2**, 233–264 (1981).

[Wi 84] F. Winkler: *The Church-Rosser Property in Computer Algebra and Special Theorem Proving: An Investigation of Critical-Pair/Completion Algorithms.* Dissertation, Univ. Linz (1984).

Critical Pairs of Reduction Schemes

Werner Th. WOLFF
Lochhamer Str. 39
D-8032 LOCHHAM (W.Germany)

By generalizing the Knuth-Bendix procedure to reduction modulo a compatible equational theory we will come across reduction schemes quite naturally. A reduction scheme stands for a recursively enumerable set of reduction rules which are summarized in a rule with condition. When superponing such schemes their conditions get only partially instantiated by unification, so that we obtain terms connected with a hypothesis. Such terms reduce w.r.t. "normal" rules by passing along their hypothesis, and w.r.t. schemes by means of a consistency checker. To prove the confluence of a system with schemes, we must also look at critical pairs of "terms under hypothesis".

1. Theory

Consider a noetherian relation of reduction $\longrightarrow$ and a equivalence relation $\sim$.

1.1. Def: $\longrightarrow$ is *confluent modulo* $\sim$:iff $x_1 \longleftarrow^* x \sim y \longrightarrow^* y_1$ implies $x_1 \rightarrow\leftarrow y_1$ (mod $\sim$), where $x_1 \rightarrow\leftarrow y_1$ (mod $\sim$) :iff exist x', y' s.t.: $x_1 \longrightarrow^* x' \sim y' \longleftarrow^* y_1$. □

1.2. Def: $\longrightarrow$ is *locally confluent modulo* $\sim$:iff

(a) $y \longleftarrow x \longrightarrow z$ implies $y \rightarrow\leftarrow z$ (mod $\sim$) and

(b) $y \sim x \longrightarrow z$ implies $y \rightarrow\leftarrow z$ (mod $\sim$). □

We have the following result (cf. [1], Lemma 7):

1.3. Theorem: $\longrightarrow$ is confluent modulo $\sim$ iff it is locally confluent modulo $\sim$. □

Let T be the set of terms over function symbols in F and variables in X. To deal with term rewriting, we introduce

1.4. <u>Def:</u> $\sim$ is a *<u>compatible simplification</u>* :iff $\sim$ is an equivalence relation s.t.: for all s, t, t_1, t_2 ε T

(a) s $\sim$ t implies $\sigma(s) \sim \sigma(t)$ for any substitution σ, and

(b) $t_1 \sim t_2$ implies $s[u \leftarrow t_1] \sim s[u \leftarrow t_2]$ for any tree node u in s (subterm replacement). □

For a term rewriting system R that generates the noetherian relation $\longrightarrow_R$ on T, and a simplification $\sim$ which is compatible with the terms of T we get (cf.[3], Satz 7.4):

1.5. <u>Theorem:</u> If $\longrightarrow_R$ and $\sim$ fulfill condition (b) of Def.1.2, then: $\longrightarrow_R$ is locally confluent modulo $\sim$ iff for all critical pairs (p, q) of R: p $\rightarrow\leftarrow_R$ q (mod $\sim$). □

Cf. [2], p.12, for orthogonality of two relations of reduction which gives a restriction similar to (b) of 1.2.

2. <u>Implementation</u>

To handle this kind of confluence mechanically, I implemented the extension of the Knuth-Bendix procedure to reduction schemes and terms under hypothesis in the simple case of conditions which involve an ordering >. Let $Th_>$ be the theory of > and Subst = $\{\sigma:X \longrightarrow T \mid dom(\sigma) \text{ finite}\} \cup \{\perp\}$.

2.1. <u>Def:</u> a $\longrightarrow$ b :- P is a *<u>reduction scheme</u>* :iff a$\longrightarrow$b ε R and P ε $Th_>$, while its application to terms is defined by

s *<u>reduces to</u>* t *<u>w.r.t. a $\longrightarrow$ b :- P</u>* :iff there exists redex u in s w.r.t. a $\longrightarrow$ b s.t.: $t = s[u \leftarrow \sigma(b)]$ and

prove(σ, P) $\neq \perp$, where σ is the match of s/u and a.

prove(σ:Subst, A:$Th_>$):Subst can be defined by the rules

prove($\perp$, A) $\longrightarrow \perp$, prove(σ, <u>true</u>) $\longrightarrow \sigma$,

prove(σ, $t_1 = t_2$) $\longrightarrow$ unif(σ, t_1, t_2),

prove(σ, $t_1 > t_2$) $\longrightarrow$ σ :- $\sigma(t_1) > \sigma(t_2)$,

prove(σ, [&]) $\longrightarrow$ σ (empty conjunction),

prove(σ, [& A B]) $\longrightarrow$ prove(prove(σ,A), [& B]). □

2.2. Def: A *term under hypothesis* is a pair (t, H) ε T x Th>.

(s,H) *reduces to* (t,H') *w.r.t. a $\longrightarrow$ b :- P* :iff there exists redex u in s s.t.: t = s[u $\leftarrow$ match(s/u,a)(b)] and H' = prove_consistency([& P match(s/u,a)(H)]) $\neq \perp$. □

The consistency checker for Th> is based on the principles:

X > X resp. [& X>Y Y>X] are inconsistent, return $\perp$.

If [& X>Y Y>Z] is consistent, then so is [& X>Y Y>Z X>Z].

Finally, to superpone reduction schemes we must have

2.3. Def: (p,H) and (q,H) are a *critical pair* w.r.t. R :iff exist $a_1 \longrightarrow b_1$:-P_1, $a_2 \longrightarrow b_2$:-P_2 ε R, node u in a_1, and μ = unif(id,a_1/u,a_2) $\neq \perp$ s.t.: (i) p = $\mu(a_1)[u \leftarrow \mu(b_2)]$ and q = $\mu(b_1)$ form a critical pair,

(ii) H = prove_consistency([& $\mu(P_1)$ $\mu(P_2)$]) $\neq \perp$. □

If one (or both) rules are reductions we may assume the corresponding condition P to be true, so that the task of proving consistency becomes trivial.

2.4. Def: A critical pair (p,H), (q,H) is *confluent* :iff

(p,H) $\longrightarrow^*$ (p',H_1), (q,H) $\longrightarrow^*$ (q',H_2), and

prove_consistency([& H_1 H_2]) $\neq \perp$. □

References:

1. **HUET**, G.: Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems, in: 18th IEEE Symposium on Foundations of Computer Science (1977), pp. 30 - 45

2. **KANDRI-RODY**, A.; **KAPUR**, D.: On Relationship between Buchberger's Gröbner Basis Algorithm and the Knuth-Bendix Completion Procedure, General Electric Technical Information Series, Schenectady 1983

3. **WOLFF**, W.Th.: Ein Vervollständigungsalgorithmus für Termersetzungssysteme und Polynomidealbasen, Diplomarbeit an der Fakultät für Mathematik der Uni München, 1988

Constructor-Based Rewriting and Narrowing without a Confluence Condition

Extended Abstract

Heinrich Hussmann
Universität Passau
Fakultät für Mathematik und Informatik
Postfach 2540
D-8390 Passau
EUNet: hussmann@unipas.uucp

1. Introduction

It is quite usual for work on the theory of term rewriting to presuppose (or to test) the confluence of the rewrite relation under consideration. But, under certain circumstances, term rewriting can be quite interesting even without this precondition. In particular, results for such systems are valuable for modelling nondeterministic computations. Besides that, the question may be of theoretical interest which results depend essentially on the confluence condition and which do not.

The work presented here came out of a thesis on the extension of algebraic specifications to nondeterminism ([Hussmann 88]). Based on the semantic framework of [Nipkow 86] and [Hesselink 88], non-confluent term rewriting can be seen as a specification language and a calculus for specifications of nondeterministic data types.

Below we skectch a model-theoretic semantics for non-confluent term rewriting systems and give correctness and (weakened) completeness results for rewriting as a calculus. It turns out that the "narrowing" method ([Hullot 80]) can be easily transferred to the case of non-confluent rewriting.

The results presuppose a special shape of the rewriting rules which is sometimes called "constructor-based". A large number of specifications occurring in practice are subsumed by this type of rules. In contrast to other approaches, no further preconditions (termination, constructor-completeness) are made.

2. Constructor-Based Rewrite Systems

For our purposes, the standard notion of term rewriting systems can be loosened somewhat by *omitting all variable restrictions*. For instance, a rule

$\text{some} \rightarrow x$

where x is a variable and some a constant, will be considered as a correct rewrite rule.

We assume a signature $\Sigma = <S, F>$ to be given where a subset $C \subseteq F$ of *constructor* function symbols is designated. $T_\Sigma(X)$ means the set of terms over Σ and a variable set X, $T_C(X)$ is the subset of*constructor terms* . A term rewrite system R is called *constructor-based* iff for all rules $l \rightarrow r$ in R the left hand side is of the special form

$l = f(t_1, \ldots, t_n)$ where $f \in F \setminus C$ and all $t_i \in T_C(X)$.

3. Model-theoretic Semantics

The notion of a *multialgebra* generalizes algebras to the case of *set-valued* operations, i.e. an operation yields an element of the powerset of the corresponding carrier as its result. If the rank of $f \in F$ is $f: s_1 \times \ldots \times s_n \rightarrow s$, then

$f^A: s_1{}^A \times \ldots \times s_n{}^A \rightarrow \wp(s^A)$.

A subset $C^A \subseteq F^A$ of constructor operations exists also in A:

$C^A = \{f \in F^A \mid f \in C\}$;

and all constructor operations are allowed only to have singleton sets as results:

if $c^A \in C^A$ then $\mid c^A(a_1, \ldots, a_n) \mid = 1$.

This means, the operations within an algebra as defined above may be *partial* and *non-deterministic*, constructor operations are forced to be deterministic and total.

A *valuation* β of the variables in a multialgebra A is defined as usual. The *interpretation* of a term $t \in T_\Sigma(X)$ of sort s in a multialgebra A under a valuation β is defined by additive extension: $I^A[x](\beta) = \{\beta(x)\}$ (if x is a variable), $I^A[f(t_1, \ldots, t_n)](\beta) = \{a \in f^A(a_1, \ldots, a_n) \mid a_i \in I^A[t_i](\beta)\}$ (if $f \in F$).

A term rewrite rule $l \rightarrow r$ is called*valid* in a multialgebra A (denoted by $A \models l \rightarrow r$) iff

$\forall$ valuations β: $I^A[l](\beta) \supseteq I^A[r](\beta)$.

A is called a *model* of the rewrite system R iff all rules in R are valid in A. We write $R \models l \rightarrow r$ iff $A \models l \rightarrow r$ for all models A of R.

4. Rewriting without a Confluence Condition

The usual notion of rewriting is generalized to *constructor-based rewriting* by

$t1 \rightarrow_C t2$ iff there is an occurrence u in t1, a rule $l \rightarrow r$ in R and a *constructor* substitution σ such that $t1/u = \sigma l$, $t2 = \sigma(t1[u \leftarrow r])$.

(A constructor substitution means a substitution assigning only constructor terms to variables.)

In a constructor-based rewrite-system R we have for all terms t1, t2$\in T_\Sigma(X)$ the correctness result:

$$t1 \to_C^* t2 \;\Rightarrow\; R \models t1 \to t2.$$

We have also a completeness result under the restriction that t2 has to be a constructor term, i.e. for all t1$\in T_\Sigma(X)$, t2$\in T_C(X)$:

$$R \models t1 \to t2 \;\Rightarrow\; t1 \to_C^* t2.$$

5. Narrowing without a Confluence Condition

The "narrowing" technique can be adapted to constructor-based rewriting in a quite straightforward way:

$t1 \text{-}N\!\!\to_\sigma t2$ iff there is a non-variable occurrence u in t1, a rule $l \to r$ in R and a *constructor* substitution σ such that σ is the most general unifier of t1/u and l, t2 = σ(t1[u$\leftarrow$r]).

This kind of narrowing corresponds to constructor-based rewriting in the following sense:

Let R be a constructor-based rewrite system, Q = [t1 $\to$ t2] a given query in R. Then for a *constructor* substitution σ, σ is a solution of Q (i.e. $\sigma t1 \to_C^* \sigma t2$) if and only if there are constructor substitutions λ, σ' and a term t2' such that $\sigma = \lambda\sigma'$, $t1 \text{-}N\!\!\to_{\sigma'}^* t2'$ and λ is a unifier of σ' t2 and t2'.

References:

[Hesselink 88]
W. H. Hesselink, A mathematical approach to nondeterminism in data types, *ACM Transactions on Programming Languages and System* s**10** (1988) 87-117

[Hullot 80]
J. M. Hullot, Canonical forms and unification, in: W. Bibel, R. Kowalski (eds.), *5th Conference on Automated Deduction*, Lecture Notes in Computer Science **87** (Springer, Berlin, 1980) 318-334.

[Hussmann 88]
H. Hussmann, Nondeterministic algebraic specifications (in German). Ph. D. thesis, University of Passau, 1988.

[Nipkow 86]
T. Nipkow, Nondeterministic data types: Models and implementations, *Acta Informatica* **22**(1986) 629-661.

A Preference Ordering on AC-Terms

ALBRECHT FORTENBACHER

IBM Wissenschaftliches Zentrum,
Tiergartenstr. 15, D-6900 Heidelberg.

Introduction: The efficiency of algorithms on AC-terms (classes of terms modulo associativity and commutativity) depends heavily on the underlying data structure. After discussing commonly used implementations of AC-terms by terms and by canonical forms, we present a partial ordering on AC-terms, the AC preference ordering, and demonstrate how it can be used to implement AC-terms efficiently.

Basic algebraic properties: Given a signature $\Sigma = (S, F, \tau)$, the term algebra $\mathcal{T}_\Sigma(\mathcal{V})$ is the free Σ-algebra over the set of variables $\mathcal{V}$. Therefore substitutions, endomorphisms on $\mathcal{T}_\Sigma(\mathcal{V})$, are uniquely determined by their restriction to $\mathcal{V}$. Any set of equations $\mathcal{E} \subseteq \mathcal{T}_\Sigma(\mathcal{V})^2$ induces a congruence relation $\approx$ with $\mathcal{E} \subseteq \approx \subseteq \mathcal{T}_\Sigma(\mathcal{V})^2$, the equational theory of $\mathcal{E}$. We get a Σ-algebra $\mathcal{T}_\Sigma(\mathcal{V})_{/\approx}$ which, in the case of collaps free theories, forms the free algebra over $\mathcal{V}$ in the $\mathcal{E}$-variety. In this case, endomorphisms (substitutions) of the term algebra can be extended in a natural way to endomorphisms on $\mathcal{T}_\Sigma(\mathcal{V})_{/\approx}$, which we also call substitutions. The AC-theory, consisting of associativity *and* commutativity laws for some operators, is a permutative theory, therefore the quotient algebra $\mathcal{T}_\Sigma(\mathcal{V})_{AC}$, the AC-term algebra, consists of finite classes of terms which are called AC-terms.

Implementation of AC-terms: An AC-operator which may be regarded as having arbitrary arity (exploiting associativity) permits permutation of its argument AC-terms (exploiting associativity and commutativity). Thus, an AC-term, which is constructed by an AC-operator, can be represented by a multiset of AC-terms. The implementation of this multiset affects the efficiency of algorithms on $\mathcal{T}_\Sigma(\mathcal{V})$. We shall demonstrate this by a closer look at two basic algorithms, namely comparison of AC-terms (congruence classes) and substitution. Usually the multisets are implemented as lists (i.e. an AC-term is represented by any term

of its congruence class). For example the congruence class of the term $*(y,*(*(2,x),-(y)))$ has a representation $(*\ y\ 2\ x\ (-\ y))$. Substitution can be performed easily by substituting the term which represents the congruence class, but comparison is costly: two terms have to be tested for "equality under permutation". To improve the latter algorithm, AC-terms could be implemented by canonical forms, e.g. by an ordered list. This yields the following representation: $(*\ 2\ x\ y\ (-\ y))$. Now comparison of AC-terms can be reduced to comparison of terms, which is very fast, but substitution is costly: the canonical form is not preserved in general. A way out of this dilemma could be an ordering which is "stable under substitution", i.e. $t_1\sigma \prec t_2\sigma$ follows from $t_1 \prec t_2$ for all AC-terms t_1, t_2 and all substitutions σ.

The AC preference ordering: First it has to be remarked that an ordering on AC-terms which is stable under substitution cannot be total. Our goal is to find a partial ordering which, besides being efficient, is "as large as possible". This can be achieved by extending a total ordering $(\leq, F)$ on the operator symbols to the partial AC preference ordering $(\preceq, \mathcal{T}_\Sigma(\mathcal{V})_{AC})$ on AC-terms in the following way:

1. $f < g$:
   $f(\ldots) < g(\ldots)$
2. f is not an AC-operator:
   $f(t_1, \ldots, t_n) \preceq f(t'_1, \ldots, t'_n) \Leftrightarrow (t_1, \ldots, t_n) \preceq\!\!\!\prec_{lex} (t'_1, \ldots, t'_n)$
3. f is an AC-operator, $t_1 = f(\ldots)$, $t_2 = f(\ldots)$:
   $t_1 \preceq t_2 \Leftrightarrow \mathcal{M}_f(t_1) \preceq\!\!\!\prec_{mult} \mathcal{M}_f(t_2)$,

where the multiset of an AC-term t is defined by

$$\mathcal{M}_f(t) = \begin{cases} \mathcal{M}_f(t') + \mathcal{M}_f(t''), \text{ if } t = f(t', t'') \\ \{\, t \,\}, \text{ otherwise.} \end{cases}$$

The orderings $\preceq\!\!\!\prec_{lex}$ and $\preceq\!\!\!\prec_{mult}$ are recursively defined as the lexicographic resp. multiset extension of $(\preceq, \mathcal{T}_\Sigma(\mathcal{V})_{AC})$. Variables are excluded from ordering. $(\preceq, \mathcal{T}_\Sigma(\mathcal{V})_{AC})$ is stable under substitution and, as a preference ordering, it is very efficient (compared to the recursive path ordering, for example). Furthermore, it can be shown that $\preceq$ cannot be extended without loosing stability under substitution or getting complicated to compute.

Implementation using the AC preference ordering: Like in the case of canonical forms, where the ordered list reflects the total ordering on AC-terms, we can use the partial AC preference ordering to construct a

data structure for AC-terms. For example, the multiset $\mathcal{M}_f(t)$ can be partitioned into maximal antichains of AC-terms, i.e. multisets of AC-terms which are pairwise uncomparable. This data structure improves the performance of the substitution algorithm compared to canonical forms, but does not behave significantly worse with respect to the algorithm comparison of AC-terms. It can be shown, in an algebraic complexity modell and by implementing the different data structures, that the implementation based on the AC preference ordering behaves favourably compared to the implementations by terms resp. by canonical forms.

Literature: Basic algebraic properties of equational theories are discussed in [1],[2], multiset orderings in [4]. In [3] an algebraic modell for implementations is presented which also serves as a complexity modell. Evaluation of the different data structures for AC-terms can be found in [2].

[1] Ehrig, H., Mahr, B.: *Fundamentals of algebraic specification*, 1, Springer, 1985

[2] Fortenbacher, A.: *Effizientes Rechnen in AC-Gleichungstheorien*, Dissertation, Universität Karlsruhe, 1989

[3] Goguen, J.A., Thatcher, J.W. und Wagner, E.G.: *An initial algebra approach to the specification, correctness and implementation of abstract data types*, in: *Current Trends in Programming Methodology*, 4, Prentice Hall, 1978

[4] Jouannaud, J.-P., Lescanne, P.: *On multiset orderings*, in: *Information Processing Letters*, Vol. 15, No. 2, 1982

An approach to goal driven strategies for the Knuth-Bendix completion procedure

Christoph Brzoska

SFB 314, "Künstliche Intelligenz - Wissensbasierte Systeme"
Institut für Logik, Komplexität und Deduktionssysteme,
Universität Karlsruhe, P.Box 6980,
D-7500 Karlsruhe1, F.R.Germany
e-mail: brzoska@ira.uka.de

We consider strategies for the Knuth-Bendix completion procedure ([KB70]) that constructs canonical term rewriting systems (TRS) for equational theories. In contrast to the critical pairs criteria ([BW83], [Küchlin85], [KMS88]) and to completion strategies, that minimize the size of axioms and rules with respect to the number of function symbols ([Küchlin82], [Wagner86]), we investigate heuristic completion strategies, that estimate during the completion process, which of the generated axioms and rules will be contained in R^{∞}, the TRS generated for the given equational theory. Our approach is justified by the fact, that the result of the KB procedure is uniquely determinated by the input (up to variable renaming).

Theorem1 [Metivier83] If R and R´ are two reduced canonical TRS for an equational theory E, and if R, R´ are both contained in some reduction ordering >, then R and R´ are equal (up to variable renaming). A TRS R is called *reduced*, if for any rule $l \rightarrow r \in R$ the right-hand side r is not reducible in R and the left-hand side l is not reducible in $R \setminus \{l \rightarrow r\}$.

Note: Any canonical TRS R can be reduced to a reduced canonical TRS R´ with $\xleftrightarrow{*}_R = \xleftrightarrow{*}_{R'}$ and $\rightarrow_{R'} \subseteq \xrightarrow{+}_R$. Most of the implemented completion procedures generate reduced TRS.

As a consequence, the result of the KB procedure can be characterized as follows:

Proposition 1 If > is a reduction ordering, E an equation theory and R a reduced canonical TRS for E, then:

a) for any rule $l \rightarrow r \in R$ the following is true:
   i) r is a least element of its congruence class, i.e. $t \in [r]_E$ implies $t > r$ or $t = r$
   ii) all proper subterms of l are least elements of their congruence classes.

b) for any term t with $|[t]_E| > 1$ there is a rule $l \rightarrow r \in R$, such that not $r > t$.

We use Proposition 1 to define a heuristic merit ordering on the rules generated during the completion process estimating, which of the rules have the best chance to lie in R^{∞}.

Definition If > is a reduction ordering then the merit ordering $>_{h(>)} \subseteq T_{\Sigma}(V)^2 \times T_{\Sigma}(V)^2$ is defined by $f(t_1,\dots,t_n),r_1) >_{h(>)} (g(s_1,\dots,s_m),r_2)$ iff $\{t_1,\dots,t_n\} >> \{s_1,\dots,s_m\}$ or $\{t_1,\dots,t_n\} =_{set} \{s_1,\dots,s_m\}$ and $r_1 > r_2$, where >> denote the extension of > to a multiset ordering and $=_{set}$ equality on sets.

Heuristic Huet's completion procedure

```
KB(E: set of equation, >: reduction ordering): TRS;
begin
$E_0 := E$; $R_0 := \emptyset$; $i := 0$; $p := 0$;
loop
    while $E_i \neq \emptyset$ do
        choose an equation $t_1 = t_2$ in $E_i$; remove $t_1 = t_2$ from $E_i$; $(t_1', t_2') := (t_1{\downarrow}_{R_i}, t_2{\downarrow}_{R_i})$;
        if $t_1' \neq t_2'$ then
            case $(t_1', t_2')$ of
            $t_1' > t_2'$ : $(l,r) := (t_1', t_2')$;
            $t_1' < t_2'$ : $(l,r) := (t_2', t_1')$;
            else        : exit with failure endcase;
            $E_{i+1} := E_i \cup \{l'' = r' \mid l' \to r' \in R_i, l'' = l'{\downarrow}_{R_i \setminus \{l' \to r'\} \cup \{l \to r\}}\}$; $p := p + 1$;
            $R_{i+1} := \{k{:}l' \to r'' \mid k{:}l' \to r' \in R_i, l'$ irreducible in $R_i \setminus \{l' \to r'\} \cup \{l \to r\}, r'' = r'{\downarrow}_{R_i \cup \{l \to r\}}\}$
                  $\cup \{p{:}l \to r\}$; $i := i + 1$;
        endif;
    endwhile;
    if all rules in $R_i$ are marked then exit with $R_i$ as canonical TRS for E
    else choose an unmarked rule $k{:}l \to r$ in $R_i$, which is minimal with respect to $>_{h(>)}$
    $E_{i+1} := \{p = q \mid p = q$ critical pair of rule $k{:}l \to r$ with rule $k'{:}l' \to r'$ in $R_i$ and $k' \leq k\}$;
    mark rule $k{:}l \to r$ in $R_i$; $i := i + 1$; endif
endloop
end KB ;
```

As pointed out in [Huet81] we have to guarantee the fairness of the heuristic completion procedure, i.e. any rule in $R^{\infty} := \cup_i \cap_{j \geq i} R_j$ has to be chosen for the creation of critical pairs in some iteration of the outer loop.

Proposition 2 If $\geq_{h(>)}$ is a globally finite well quasi ordering on $T_{\Sigma}(V)^2$ then the heuristic completion procedure is fair. Here $\geq_{h(>)}$ denotes the reflexive closure of $>_{h(>)}$ and a reflexive, transitive relation $\geq$ (quasi ordering) on a set M is called

- *globally finite* iff for any x in M the set $x_{\geq} := \{y \in M \mid x \geq y\}$ is finite.
- a *well quasi ordering (wqo)* iff $\geq$ is well founded (i.e. there exist no infinite descending sequences $x_1 \geq x_2 \geq \ldots$ such that $x_i \not\leq x_{i+1}$ for each $i \geq 1$) and each set of pairwise incomparable elements in M is finite.

Note: Huet's completion procedure generates variant free TRS's R_i. Hence for the fairness of KB procedure it is sufficient, that $\geq_{h(>)}$ is a globally finite wqo on each variant free subset R of $T_{\Sigma}(V)^2$.

Lemma 1 If $\sigma: V \mapsto T_{\Sigma}(V)$ is a substitution such that for all $x \in V$ $\sigma(x) = y$ for some $y \in V$, $\geq$ a globally finite wqo on $T_{\Sigma}(\{y\})^2$, then

a) $\geq_{[>]}$ defined by $(t_1,t_2) \geq_{[>]} (t_1',t_2')$ iff
   i) $(t_1,t_2)\sigma > (t_1',t_2')\sigma$ or
   ii) $(t_1,t_2)\sigma \approx (t_1',t_2')\sigma$ and $(t_1,t_2)\sigma >_{h(>)} (t_1',t_2')\sigma$
   is a globally finite wqo on each variant free subset R of $T_{\Sigma}(V)^2$, where $\approx := \geq \cap \leq$.

b) If $\geq_{h(>)}$ is wqo on $T_{\Sigma}(\{y\})^2$ then $\geq_{[>]}$ defined by
   $(t_1,t_2) \geq_{[>]} (t_1',t_2')$ iff $(t_1,t_2)\sigma \geq (t_1',t_2')\sigma$ and $(t_1,t_2)\sigma \geq_{h(>)} (t_1',t_2')\sigma$
   is a globally finite wqo on each variant free subset R of $T_{\Sigma}(V)^2$.

Empirical Results

Based on the system documented in [Dietrich85] an experimental implementation of the heuristic completion procedure was developed. Since the KB Algorithm implemented there is a variant of the [HO80] KB procedure, we use $>_{h(>)}$ to choose an axiom, which then is turned into a rule. Following Lemma 1a we extend $>_{h(>)}$ to a globally finite wqo on variant free subsets of $T_\Sigma(V)^2$. As globally finite wqo on $T_\Sigma(\{y\})^2$ we take the termsize ordering $\geq_{ts(n)}$ divided by n. More formally: $(l,r) \geq_{ts(n)} (l',r')$ iff $(|Occ(l)| + |Occ(r)|) \div n \geq (|Occ(l')| + |Occ(r')|) \div n$, where $\div$ denotes the integer division. Some examples from [KB70] were run so far.

Orientation of pairs through KBO ordering; heuristic completion strategy (smallest component strategy)					
Example	completion steps	matches performed	rewrites performed	unifications performed	pairs generated
l-group	15 (15)	14 382 (17 711)	195 (233)	449 (473)	117 (134)
r-group	17 (19)	17 275 (28 843)	235 (348)	512 (673)	135 (182)
lr-group	15 (15)	21 461 (28 595)	308 (361)	549 (570)	149 (163)
rl-group	22 (22)	69 231 (81 946)	776 (843)	1102 (1176)	317 (343)

References

[Brzoska88] Brzoska, C. *Untersuchung von Strategien für den Knuth-Bendix Vervollständigungsalgorithmus*, Tech. Report 88-12, Technical University Berlin, FB 20, 1988.

[Dietrich85] Dietrich, R. *Eine Programmierumgebung für Termersetzungssysteme*, Arbeitspapiere der GMD, Nr. 130, 1985.

[Huet81] Huet, G. *A Complete Proof of Correctness of the Knuth-Bendix Completion Algorithm*, J. Comp. Syst. Sci. 23 (1981), 11-21

[HO80] Huet, G. and Oppen, D. *Equations and Rewrite Rules: A Survey*. In *Formal Languages: Perspectives and Open Problems* (R. Book, Ed.), Academic Press 1980.

[KMN88] Kapur, D. and Musser D. R. and Narendran, P. *Only Prime Superpositions Need be Considered in the Knuth-Bendix Completion Procedure*, J. Symb. Comp. (1988) 6 ,19-36.

[KB70] Knuth, D. E. and Bendix, P. B. *Simple Word Problems in Universal Algebra*, In *Computational Problems in Abstract Algebra* (J. Leech, Ed.), Pergamon 1970, 263-297.

[Küchlin82] Küchlin, W. *A Theorem Proving Approach to the Knuth-Bendix Completion Procedure*, Proc. of EUROCAM 1982, Springer Verlag 1984, LNCS 144.

[Küchlin85] Küchlin, W. *A Confluence Criterion Based on the Generalized Newman Lemma*, Proc. of EUROCAL 1985, Springer Verlag 1985, LNCS 204.

[Wagner86] Wagner, E. *Strategien für den Knuth-Bendix Algorithmus*, Diplomarbeit, University Kaiserslautern, FB Informatik, 1986.

[Metivier83] Metivier, Y. *About the Term Rewriting Systems Produced by the Knuth-Bendix Completion Procedure*, Information Processing Letters 16 (1983), 31-34.

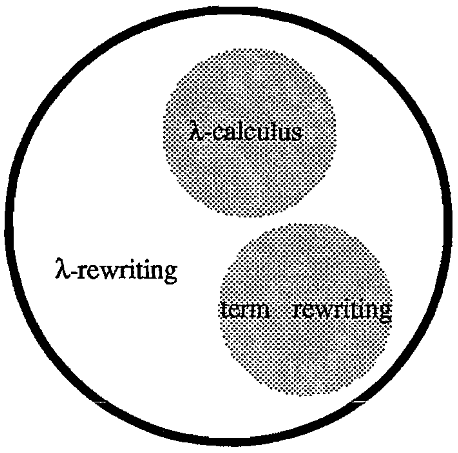
[WB83] Winkler, F. and Buchberger, B. *A Criterion for Eliminating Unnecessary Reductions in the Knuth-Bendix Algorithm*, Colloquim on Algebra, Combinatorics and Logic in Computer Science (Gyor, Hungary, Sept. 12-16, 1983).

λ-rewriting

(extended abstract)

Stefan Kahrs
Universität Bremen
Fachbereich Mathematik/Informatik
Postfach 33 04 40
D-2800 Bremen

February 1989



λ-rewriting combines λ-calculus with (one-sorted) term rewriting, but in a less general way than Klop's combinatory reduction systems, see [Klo 80]. Three types of such combinations are studied - each type being a special case of its successor.

Type 1, add term rewriting to λ-calculus (or vice versa)

In some sense, type 1 is the least calculus comprising λ-calculus and term rewriting. From the programmer's point of view, it is similar to functional programming languages with pattern matching (e.g. ML [HMM 86]). From a theoretical point of view it is a special case of the λδ-calculus [Bar 84] and Hindley's λ(a)-calculus [Hin 78] is very similar.

Terms of type 1 are extended by abstractions (w.r.t. TRS) or by function symbols (w.r.t. λ-calculus). Left-hand sides of type 1 rules are restricted to (the curryfied form of) terms allowed to be on the left-hand side of an ordinary TRS-rule. Right-hand sides are not restricted. Substitution has to be defined slightly more general than for TRS or λ-calculus, but more similar to substitution in the λ-calculus, because it has to take care of name conflicts.

Evaluation (called βδ-reduction) is a mixture of β-reduction (λ-calculus) and the relation defined by the rules, the δ-reduction. For any rule `left → right`, any context `C` and any substitution `sub` the term `C[sub(left)]` δ-reduces to `C[sub(right)]`. So the definition is the same as for ordinary TRS, if one ignores the more general definitions of terms and substitutions.

The particularity of type 1 systems can be shown by an example:

```
add 0 → λ x . x
add (succ x) → λ y . succ (add x y)
mul 0 → λ x . 0
mul (succ x) → λ y . add (mul x y) y
```

The λ-rewriting system above is one of several possibilities to define addition and multiplication on natural numbers where a natural number n is represented as $\texttt{succ}^n(\texttt{zero})$. Different from TRS and from functional programming languages with super-combinator

implementations [Tur 79, Joh 85] is the evaluation of functions. For example the system above evaluates (i.e. βδ-reduces) the term `mul(succ(succ zero))` to the normal form `λx.add x x`:

To the right there is a possible βδ-reduction sequence (here innermost-first evaluation strategy) of the first term in 7 steps. All the 3 β-reductions in the example have reduced non-closed terms inside of abstractions. In functional programming languages this is unusual.

```
                      mul(succ(succ 0))
              λy.add(mul(succ 0) y) y
   λy.add((λy.add(mul 0 y) y) y) y
λy.add((λy.add((λx.0) y) y) y) y
          λy.add((λy.add 0 y) y) y
        λy.add((λy.(λx.x) y) y) y
                  λy.add((λy.y) y) y
                          λy.add y y
```

Some results and observations about type 1 λ-rewriting (LRS) are the following:

- For confluent TRS mapped through currying to LRS (where each function symbol has a unique arity):
  - δ-reduction remains to be confluent, but it is not isomorphic to TRS-reduction - because of currying, some more terms can be δ-reduced.
  - βδ-reduction is confluent, if the TRS is additionally left-linear.
- ζ-transformation: the rule `a x → b`, such that `a` and `b` are terms and `x` is a variable not occurring free in `a`, can be ζ-transformed into the rule `a → λ x. b`. A LRS is ζ-normal, if no rule can be ζ-transformed. For example, the addmul-LRS is ζ-normal.
  - if X is a LRS and X' has been yielded by some ζ-transformations from X, then βδ(X) is a subrelation of βδ(X').
  - in general, confluence of βδ-reduction is lost by ζ-transformation.
- βδη-reduction on ζ-normal LRS:
  - satisfies extensionality.
  - unlike λ-calculus, the normal form property of βδη- and βδ-reduction differs, because η-reduction can introduce δ-redexes.
  - is confluent if βδ-reduction is.

Type 2, rewriting λ-calculus terms

LRS of type 2 deal with the same terms as type 1, but the rules are more general. Here the rules are not restricted on their left-hand sides, and altogether only by the usual restriction for term rewriting rules, that is FV(`left`)⊇FV(`right`) for rules `left → right`, where FV denotes the set of free variables in a term. Whereas a confluent LRS of type 1 only equalises the non-λ-calculus part of the terms and hence performs a consistent theory [Bar 84], type 2 is more powerful, but it is harder to prove several properties for LRS of type 2.

Type 2 rules can express λ-calculus' η-reduction:

```
λ x . y x → y
```

To do this in the λ-calculus, it is necessary to require x∉FV(y). For λ-rewriting of type 2 this condition holds implicitly by the definition of δ-reduction. In particular, the rule above

does not fit to the term `λx.x x`, because there is no substitution to make it equal to the left-hand side - as in λ-calculus, substitutions have to take care of name conflicts.

Type 3, extensions

There are several possibilities to make further extensions. Type 3 works with an extended set of terms. The idea is: rules are partial functions on terms - functions are abstractions - abstractions are terms - consequently rules are terms. Basically a term `(a→b) c` (where `a`, `b` and `c` are arbitrary terms) can be reduced to *sub*`(b)` if *sub* is a substitution, such that *sub*`(a)=c`. In fact this idea has been the starting point of the work on λ-rewriting presented here. The most general understanding of this idea leads to a non-confluent reduction, i.e. even for empty databases evaluation would not be determined. For example, evaluating the term `(x y→x) ((x→x) (a b))` can produce `a` as well as `x→x`. Therefore, the abstractions of type 3 are restricted on their left-hand sides to curryfied forms of linear TRS-terms.

Another possible extension of type 2 LRS would be to choose a different notion of reduction for type 2 rules. Consider the following rule of type 2:

```
λ x . f a → S (λ x . f) (λ x . a)
```

It is one of the 3 rules to map λ-calculus into combinatory logic, but for type 2 δ-reduction it only works incompletely, because it δ-reduces only those abstractions where the abstracted variables do not occur freely in the body. This behaviour is more restricted than necessary, because the variables `f` and `a` occur on the right-hand side in the same context of bound variables as on the left-hand side. A more liberal notion of reduction can be treated formally by introducing another kind of substitution. This new substitution does not care about name conflicts for a certain set of variables, i.e. these variables will not be renamed in a conflict situation.

References

[Cur 58] H. Curry, R. Feys: *Combinatory Logic, Vol. I;* North-Holland 1958

[Bar 84] H. P. Barendregt: *The Lambda-calculus* Volume 103, Elsevier Science Publishing Company, Amsterdam 1984

[HMM 86] R. Harper, D. MacQueen, R. Milner: *Standard ML;* Laboratory for Found. of Computer Science - University of Edinburgh, ECS-LFCS-86-2, March 1986

[Hin 78] R. Hindley: *Reductions of Residuals are Finite*; Transactions of the American Mathematical Society Vol. 240, June 1978, 345-361

[Joh 85] T. Johnsson: *Lambda-lifting - Transforming Programs to Recursive Equations*; 190-203 in LNCS 201

[Klo 80] J.W. Klop: *Combinatory Reduction Systems*; Mathematical Center Tracts 129, Amsterdam

[Klo 87] J. W. Klop: *Term Rewriting Systems: a Tutorial*; Bulletin of EATCS No. 32, June 1987

[Tur 79] D. A. Turner: *A new implementation technique for applicative languages;* Software Practice and Experience, 9:31-49, 1979

Jungle Evaluation for Efficient Term Rewriting [1]
(Extended Abstract) [2]

Berthold Hoffmann and Detlef Plump, Universität Bremen [3]

In a straightforward implementation of term rewriting, terms are represented by *trees*, and rewriting is realized by *subtree replacement.* Unfortunately, this may be very expensive, both in time and space: the application of a rule may require large subterms to be copied, and each copy of a term must be evaluated anew.

Jungle evaluation ([HKP 88], [HP 88]) provides an improved model for implementing term rewriting by graph rewriting where these sources of inefficiency can be avoided:

- Jungles are acyclic hypergraphs which allow terms to be represented such that multiple occurrences of a subterm can be shared. Acyclicity ensures that each jungle node represents a unique term and that structural induction on jungle nodes is available.
- Rewriting is performed by (hyper-)graph replacement, specified by *evaluation rules* according to the algebraic theory of graph grammars (see, e.g., [Ehr 79]). By applying these evaluation rules, new references to existing subterms are introduced instead of copying subterms.
- Additional hypergraph rules for *folding* multiple occurrences of terms allow each term in a jungle to be represented only once, so multiple evaluation can be avoided.

Example: Computation of Fibonacci Numbers

Consider the term rewrite rules

$$\begin{aligned}
\mathtt{fib(0)} &\rightarrow \mathtt{0}\\
\mathtt{fib(succ(0))} &\rightarrow \mathtt{succ(0)}\\
\mathtt{fib(succ(succ(x)))} &\rightarrow \mathtt{fib(succ(x))} + \mathtt{fib(x)}
\end{aligned}$$

specifying a function `fib` that computes Fibonacci Numbers, based on natural numbers with the constant `0`, successor function `succ`, and addition `+`.

[1]This work is partly supported by the Commission of the European Communities under Contract 390 (PROSPECTRA Project) in the ESPRIT Programme.

[2]For a full paper see [HP 88].

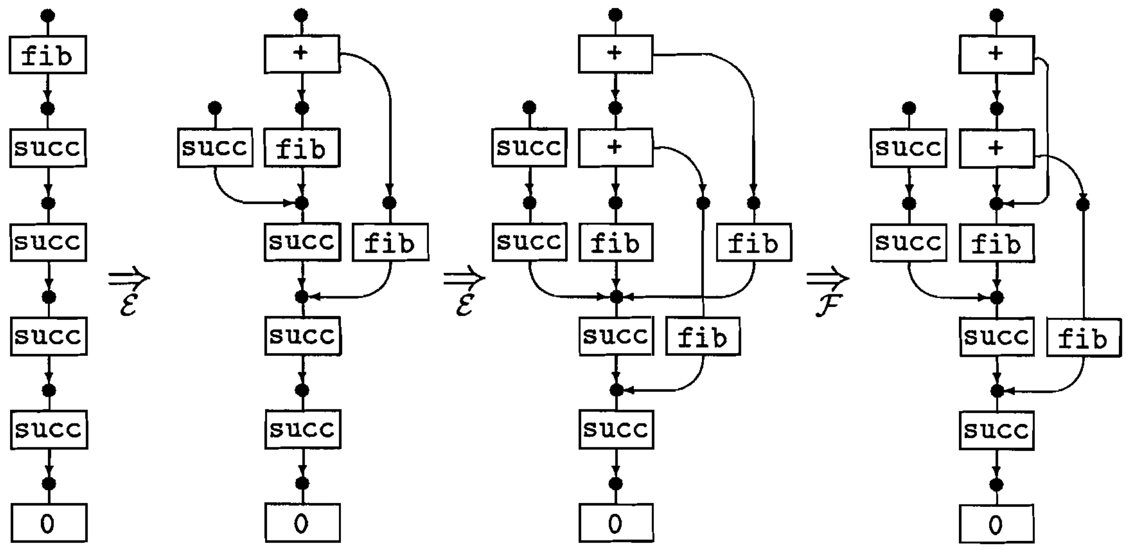
[3]Address: Fachbereich Mathematik und Informatik, Universität Bremen, Postfach 330 440, D-2800 Bremen 33. Usenet: `{hof,det}%Informatik.Uni-Bremen.de`

The first two steps for computing the Fibonacci Number of 4 by term rewriting are:

$$\begin{aligned} & \mathtt{fib}(\mathtt{succ}^4(\mathtt{0})) \\ \rightarrow\ & \mathtt{fib}(\mathtt{succ}^3(\mathtt{0})) + \mathtt{fib}(\mathtt{succ}^2(\mathtt{0})) \\ \rightarrow\ & \mathtt{fib}(\mathtt{succ}^2(\mathtt{0})) + \mathtt{fib}(\mathtt{succ}(\mathtt{0})) + \mathtt{fib}(\mathtt{succ}^2(\mathtt{0})) \end{aligned}$$

In both steps, subterms of the arguments of `fib` are copied. Furthermore, the resulting term contains two copies of $\mathtt{fib}(\mathtt{succ}^2(\mathtt{0}))$; each of them must be rewritten anew. As a consequence, rewriting a term $\mathtt{fib}(\mathtt{succ}^n(\mathtt{0}))$ to normal form requires space and a number of steps exponential in n.

Below we show corresponding jungle evaluation steps and a subsequent folding step. ($\underset{\mathcal{E}}{\Rightarrow}$ and $\underset{\mathcal{F}}{\Rightarrow}$ denote the application of an evaluation and a folding rule, respectively.)



The evaluation steps do not copy the arguments of `fib`, but merely introduce new references to them. Moreover, after the folding step the subterm $\mathtt{fib}(\mathtt{succ}^2(\mathtt{0}))$ is represented just once and thus has to be evaluated only once.

When performing evaluation and folding steps in this order, the evaluation of a term $\mathtt{fib}(\mathtt{succ}^n(\mathtt{0}))$ requires only a number of steps and space linear in n.

Results

Fully Collapsed Jungles: For each finite set of terms there is a (up to isomorphism) unique minimal jungle representing these terms most efficiently. Given an arbitrary jungle, the equivalent fully collapsed jungle can be generated by application of folding rules which eliminate multiple occurrences of terms.

Correctness: The translation of term rewrite rules into evaluation rules is correct in the sense that each application of an evaluation rule to a jungle rewrites the represented terms according to the underlying term rewrite rule. In general, a single evaluation step performs sequences of term rewrite steps in parallel.

Normal Forms: Exhaustive application of the evaluation rules to some jungle J yields a jungle $\overline{J}$ which represents normal forms of the terms represented by J, provided that the given rewriting system is left-linear. Moreover, the restriction of left-linearity can be dropped by allowing folding steps to be performed between evaluation steps.

Termination: Termination of term rewriting implies termination of jungle evaluation without restriction, even if folding is allowed. The proof of this result is nontrivial since for jungle evaluation, in contrast to term rewriting, "garbage" has to be considered (as the nodes representing $\mathtt{succ}^4(\mathtt{0})$ and $\mathtt{succ}^3(\mathtt{0})$ in the rightmost jungle of the above example) which may lead to additional evaluation steps.

Confluence: Unlike termination, confluence of term rewriting does not carry over to jungle evaluation. However, if the given rewriting system is terminating and confluent, then jungle evaluation is terminating and confluent too, provided the garbage produced by evaluation steps is ignored. Jungle evaluation without folding turns out to be strongly confluent for non-overlapping rewrite systems, where termination or left-linearity needs not to be required.

References

[Ehr 79] H. Ehrig: *Introduction to the Algebraic Theory of Graph Grammars.* Proc. 1st Graph Grammar Workshop, Lecture Notes in Comp. Sci. 73, 1-69 (1979)

[HKP 88] A. Habel, H.-J. Kreowski, D. Plump: *Jungle Evaluation.* Proc. Fifth Workshop on Specification of Abstract Data Types, Lecture Notes in Comp. Sci. 332, 92-112 (1988)

[HP 88] B. Hoffmann, D. Plump: *Jungle Evaluation for Efficient Term Rewriting.* Proc. Algebraic and Logic Programming, Akademie-Verlag, Berlin (GDR), 191-203 (1988). Long version containing all proofs published as Technical Report 4/88, Universität Bremen (1988)

Specification and Correctness of Code Generators - an Experiment with the CEC-System

(Extended Abstract)

Robert Giegerich

FB Informatik - Universität Dortmund
Postfach 500500
D-4600 Dortmund 50

1. Motivation

Using recent advances in executable specification languages and proof techniques, we approach an unsolved problem in the area of compiler construction: to provide high-level tools for the production of reliable code generators. Machine code generation appears to be a well-suited field for the application of algebraic specification techniques for two reasons:

- Target machine instruction sets, expecially with CISC architectures, are rather sizable data types. An (order-sorted) signature for the MC68000, for example, uses 42 sorts and 181 operators. Clearly, this calls for mechanical support, for checking the specification's consistency and completeness over several development steps.
- Most of a target machine description is plain syntax, and basic code selction can be done by pattern matching techniques. However, target machine programs must also satisfy certain context-sensitive constraints (regarding addressability of operands in the context of certain instructions, the number of registers used, etc.). These constraints seem quite independent, but may interact in a relatively complex fashion. This interaction, known as the danger of "semantic blocking" (in [GrGl77] and subsequent approaches), has so far not been captured in a satisfactory way.

Our approach is to describe the context sensitive properties of machine programs and code selection indpendently. Then we describe transformations that turn an (illegal) target program t into another target program t', such that

- t and t' are both correct target programs for the same intermediate program,
- t' satisfies a particular constraint,
- constraints already satisfied by t still hold for t'.

These are the properties of *correctness*, *efficacy* and *invariance* of the code generator specification, which are to be proved with equational proof methods. Together, they guarantee the absence of semantic blocking, i.e. the completeness of the code generator specification.

The study reported here was performed for a small, but nontrivial target language, consisting of 5 instructions and 4 address modes, combined in a totally non-orthogonal way.

2. An Algebraic Model of Code Generation

In our model, a code generator specification consists of five order-sorted specifications:

BS defines certain basic data types by (conditional) equational axioms, including Booleans, integers, but also machine specific ones like register numbers, register class identifiers, word length indicators, etc.

IL defines the intermediate language, extending BS by cell constructors, used by the compiler front-end to replace the source program variables by address subtrees according to the compiler's virtual machine.

TL defines the syntax of target programs, by extending BS. TL does not include any new axioms. Its sorts represent the address modes and operand classes of the target machine, its constructors model instructions and address calculations.

TM completes the target language description by enriching TL with predicates, (separately) describing operand binding restrictions, word length compatability laws, temporary register limits, and other peculiarities of the target machine. (See Example 1.)

D is the code selection specification. It enriches the union of IL and TL by one polymorphic operation *dd*, translating target programs into intermediate programs. The axioms defining *dd* must be in the form of (order-sorted) derivor equations.

For simplicity, we assume in the sequel that there are only two wellformedness-predicates specified by TM, *wb* and *wt* (for *well-bound* and *well-typed*). Now, the task of code generation can be specified formally as:

For a ground IL-term q (the given intermediate program) and a variable z of a suitable sort from TL, solve the equation system

$$dd(z) = q,$$
$$wt(z) = true,$$
$$wb(z) = true.$$

In principle, this could be solved by narrowing, but without some strategy tailored to the structure of the specification, this would be prohibitively expensive.

3. *Making the Specification (More) Operational*

The first equation, $dd(z) = q$, can be solved efficiently by using pattern matching techniques for derivor inversion, as explained in [GiSc88]. Applying the generated pattern matcher to q will yield a (generally infinite) stream of target programs $t_1, t_2, \ldots$ from $T_{TL}(V)$, ideally in the order of increasing cost (but we do not discuss this aspect here). They contain variables from V, for word lengths associated with instructions, or registers numbers yet to be assigned. In many situations, there exists a substitution σ such that $wb(t_i\sigma) = true$ and $wt(t_i\sigma) = true$ for some small value of i.

However, where wellformedness requires extra coercion instructions or loading of registers, such a substitution does not exist for many a t_i. (See Example 2.) One could consider disregarding t_i and continuing with t_{i+1}, etc. This, however, would give us no clue as to whether a solution exists at all. So instead, we further enrich the specification by "transformation" operators *mk_wb*, *mk_wt*, with the intent that if $wb(t) = true$ has no solution, $wb(mk\_wb(t)) = true$ will have one. (See Example 3.)

4. *Verification*

Up to this point, developing the specification using a tool like the CEC system [BGS88] has been a mere convenience. Now, to verify our last development step, the following equational

theorems have to be proved:

Efficacy:	$wt(mk\_wt(z)) = true$
	$wb(mk\_wb(z)) = true$
Correctness:	$dd(mk\_wt(z)) = dd(z)$
	$dd(mk\_wb(z)) = dd(z)$
Mutual Invariance:	$wb(z) = wb(mk\_wt(z))$
	$wt(z) = wt(mk\_wb(z))$

Efficacy means that it is always possible to satisfy each constraint by an equivalent target program, where target programs are equivalent when mapped to the same IL-program by *dd*. This we call correctness of the transformations. Invariance says that *mk_wt* does not destroy the achievements of *mk_wb*, and vice versa. (There are weaker formulations of invarance, which make transformations easier to write, but are harder to prove mechanically.)

Inductionless induction [HuHu80], as available by the completion procedure of the CEC-system, is sufficient to prove these theorems, basically since the transformations have a rather local effect, and so is their effect on the constraints, although it extends a little further. Without mechanical aid, it would be rather error-prone to carry out the neceessary proofs, even for a small target language as used in this study.

5. Conclusions

1. The application of our approach to code generator descriptions of realistic size may currently be out of reach for current proof systems (because of efficiency problems resulting from sheer specification size), but not without the reach of current proof techniques.
2. Striving for good code, when writing the transformations one intuitively makes use of properties of the intermediate language (such as commutativity of operations or the possibility to interchange two statements without harm). Not all of them may be implied by the axioms specified with IL. This deficiency is demonstrated by non-termination of the correctness proof, but sometimes in an obscure way.
3. This study was concerned with developing a form of code generator specification whose completeness could be verified mechanically. For a practical code generator generation tool, code selection, constraint checking and transformation should proceed in an interleaved fashion. To achieve this from a given specification, without sacrificing its provable completeness, is a challenging problem by itself.

References

[BGS88] *H.Bertling, H. Ganzinger, R. Schäfers:* CEC: A system for conditional equational completion. In Kaplan, Jouannaud (Eds.): Conditional term rewriting systems. Springer LNCS 308, 1987.

[GiSc88] *R. Giegerich, Karl Schmal:* Code selection techniques: tree parsing, pattern matching and inversion of derivors. Proceedings ESOP '88, Springer LNCS 300, 1988.

[GrGl77] *S. L. Graham, R. S. Glanville:* A new method for compiler code generation. Proceedings 5th Symposium on Principles of Programming Languages, 1977.

[HuHu80] *G. Huet, J.-M. Hullot:* Proofs by induction in equational theories with constructors. Proceedings 21st SFCS, Lake Placid, 1980.

Appendix: Examples

Example 1

For a 2-address add-to-register-instruction, our specification contains, among others, the following equations: (The small letters are variables, the rest is a mixfix notation for TL terms, and == is equality of register numbers or word length indicators like B(yte) or W(ord).)

$$wb(ADDR.l_1 \;\; Ri.l_0 \;\; Rj.l_1 \;\; opd) = (i == j)$$

$$wt(\, ADDR.l \;\; Ri.l_0 \;\; Rj.l_1 \;\; opd) = ((l == l_0) \text{ and } (l == l_1) \text{ and } (l == oplength(opd)))$$

Example 2

For $dd(z) = R1.L := R2.B + 1$, code selection would construct, among others, the solutions

z <-- $ADDR.l \;\; R1.L \;\; R2.B \;\; \#1,$ and

z <-- $MOV.l \;\; Ri.l_0 \;\; R2.B; ADDR.l_3 \;\; R1.L \;\; Ri.l_0 \;\; \#1,$

where for any instantiation of register number and word length variables, the second target program will not be well-typed, and the first will neither be well-typed nor well-bound.

Example 3

The transformation that yields a well-typable solution from the second one in Example 2 would be specified by an equation

$$l_1 < l_0 \Rightarrow mk\_wt(MOVE.l \;\; Ri.l_0 \;\; Rj.l_1) = SEXT.l_0.l_1 \;\; Ri.l_0 \;\; Rj.l_1$$

Note that if *wb* specifies $SEXT.l_0.l_1$ (sign extension from the shorter l_1 to the longer l_0) to be a 1-address instruction(while MOVE, of course, is 2-address), it will not be possible to prove *mk_wt* invariant with respect to *wb*.

Transfer in MT by Term-Rewriting

Wilhelm Weisweber, TU Berlin
Institute for Software and Theoretical Computer Science
Projectgroup KIT, Sekr. FR 5-12
Franklinstr. 28/29, D-1000 Berlin 10
E-mail: weisweb@db0tui11.bitnet

1) Introduction

The termination of the transfer process is an important question in Machine Translation (MT) and this paper makes a proposal of how to formulate transfer rules, which are in fact term-rewrite rules, in a way that the transfer process will terminate. The level of transfer in the project KIT-FAST [1] at the Technical University of Berlin is a semantic representation for sentences which expresses the logical relations within a sentence. Such logical relations are among others functor-argument relations, e.g. between the verb and its associated noun phrases. Thus the level of transfer is called Functor-Argument Structure (FAS). A FAS expression for a sentence is a derivation tree generated by a context-free grammar with complex symbols as non-terminals called FAS categories. A FAS category consists of a main category and a set of pairs of features and values. A(f) is used to denote the value of the feature f of the FAS category A.

Since the transfer process is to be formulated as a term-rewrite system, the FAS expressions have to be mapped into FAS terms. Therefore a signature of a term algebra for the FAS expressions of the source and target language is defined. Now the transfer rules can be formulated as term-rewrite rules and a term-rewrite system can be used to transfer the source FAS terms into target FAS terms, which again can be mapped into target FAS expressions with the help of the given signature.

2) An Ordering on FAS terms

In order to guarantee that the term-rewrite system terminates, a well-founded (partial) ordering $>_{FAS}$ on the set of source and target FAS terms has to be defined which does not allow infinite descending sequences of those FAS terms. In order to define a well-founded ordering $>_{FAS}$, the FAS categories of the source and target FAS grammar, henceforth source and target FAS categories respectively, are distinguished such that every source FAS category A and every target FAS category B share a common feature 'lang(uage)' where A(lang) $\neq$ B(lang). [2]

[1] This work has been developed in the project KIT-FAST (KIT = Künstliche Intelligenz und Textverstehen (Artificial Intelligence and Text Understanding); FAST = Functor-Argument Structure for Translation), which constitutes the Berlin component of the complementary research project of EUROTRA-D. It receives grants by the Federal Minister for Research and Technology under contract 1013211.

[2] In the following for all source FAS categories A(lang) = s and for all target FAS categories B(lang) = t is defined.

With this distinction the derivation sequence $t_S \rightarrow t_1 \rightarrow ... \rightarrow t_n \rightarrow t_T$ contains hybrid FAS terms t_i where $1 \leq i \leq n$ and t_S and t_T are the source and target FAS terms respectively. The FAS terms t_i are hybrid because source FAS categories as well as target FAS categories occur in them. Before the definition of the well-founded ordering can be given, the set S(t) of all source FAS categories and the set T(t) of all target FAS categories have to be defined. C(t) is used to denote the set of all FAS categories occuring in the term t.

***Definition 1*:** $S(t) = \{A \in C(t) \mid A(lang) = s\}$ and $T(t) = \{B \in C(t) \mid B(lang) = t\}$ where $S(t) \cap T(t) = \emptyset$ and $C(t) = S(t) \cup T(t)$.

***Definition 2a*:** Let t,u be FAS terms without variable occurrences then $t >_{FAS} u$ if and only if $(S(u) \subset S(t)) \vee (S(u) = S(t) \wedge T(u) \subset T(t))$.

The transfer rules may contain variables and in order to find out whether a given set of transfer rules is terminating or not, the above ordering $>_{FAS}$ on FAS terms has to be "lifted" to an ordering on FAS terms with variables. V(t) is used to denote the set of all variables occuring in the term t.

***Definition 2b*:** Let t,u be FAS terms with variable occurrences, then $t >_{FAS} u$ if and only if definition 2a holds and additionally $V(u) \subseteq V(t)$.

***Theorem 1*:** A transfer system R over a set of FAS terms is terminating if and only if $\lambda >_{FAS} \rho$ for each transfer rule $\lambda \rightarrow \rho$ in R.

Thus the transfer rules have to be defined according to the well-founded ordering $>_{FAS}$, i.e. the right-hand sides (rhs) of all transfer rules have to contain less source FAS categories than the corresponding left-hand sides (lhs) or, if the set of source FAS categories is equal on both sides, at least one of the target FAS categories has to be deleted, and every variable occuring on the rhs occurs too on the lhs. These conditions can be checked in preprocessing.

3) The Transfer System

A desirable feature of the transfer system would be that it works message-driven, i.e. it runs through the input structure and applies every possible transfer rule. Unfortunately, the lhs of the transfer rules are not necessarily local structures and therefore the transfer system has to work rule-driven, i.e. after each reduction it has to check every transfer rule for application. Additionally, the set of transfer rules need not be confluent, i.e. for a given source FAS term there may be one or more target FAS terms. For these reasons, the intrinsic application order of the transfer rules is made explicit in a preprocessing step, i.e. if the rhs of a transfer rule r and the lhs of a transfer rule r' share some non-variable common subterm, then

rule r has to be applied before rule r' ($r >_{app} r'$). If the lhs of a transfer rule r and the lhs of a transfer rule r' share some non-variable common subterm, then rule r and rule r' can be alternatively applicable ($r \vee_{alt} r'$). Before a formal definition can be given, some notations have to be introduced:

- $\sqcup$: unification of two terms. The result is the minimal unifier.
- M/u: non-variable subterm of the term M.

Now the formal definitions of the two relations $>_{app}$ and $\vee_{alt}$ can be given. Let R be the set of all transfer rules. $\forall$ r,r' $\in$ R where $r = (\lambda \rightarrow \rho)$ and $r' = (\lambda' \rightarrow \rho')$:

- superposition of the lhs of r' with the rhs of r: $(\lambda'/u \sqcup \rho \neq \emptyset) \vee (\rho/u \sqcup \lambda' \neq \emptyset) \Rightarrow r >_{app} r'$.
- superposition of the lhs of r' with the lhs of r: $(\lambda'/u \sqcup \lambda \neq \emptyset) \vee (\lambda/u \sqcup \lambda' \neq \emptyset) \Rightarrow r \vee_{alt} r'$.
- else: r and r' are independent of each other and may be applied in any order.

With the help of the relation $>_{app}$ the application sequence of the transfer rules can be computed taking into consideration the fact that $>_{app}$ also may contain cycles. The relation $\vee_{alt}$ is used to admit an alternative application of transfer rules only where necessary. In that way an efficient transfer algorithm can be defined which has been implemented at the Technical University of Berlin.

4) Literature

[Bläsius/Bürckert 85]: K.H. Bläsius, H.-J. Bürckert (eds.): "Deduktionssysteme, Automatisierung des logischen Denkens", Oldenbourg 1987, pp. 115 - 133

[Dershowitz 82]: N. Dershowitz: "Orderings for Term-Rewriting Systems", Theoretical Computer Science 17 (1982), North-Holland, pp. 279 - 301

[Dershowitz 85]: N. Dershowitz: "Termination", in: G.Goos, J. Hartmanis (eds.): "Rewriting Techniques and Applications", LNCS 202, Dijon, France, May 1985, pp. 180 - 224

[Ehrig/Mahr 85]: H. Ehrig, B. Mahr: "Fundamentals of Algebraic Specification 1", Springer, Berlin 1985

[Hauenschild et al. 78]: Ch. Hauenschild, E. Huckert, R. Maier: "SALAT: Entwurf eines automatischen Übersetzungssystems", in: Sprache und Datenverarbeitung 1978 (2), pp. 126 - 152

[Hauenschild et al. 79]: Ch. Hauenschild, E. Huckert, R. Maier: "SALAT: Machine Translation Via Semantic Representation", in: R. Bäuerle, U. Egli, A. von Stechow: "Semantics from Different Points of View", Springer 1979, pp. 324 - 352

[Hauenschild 86]: Ch. Hauenschild: "KIT/NASEV oder die Problematik des Transfers bei der Maschinellen Übersetzung", in: I. Bátori, H.J. Weber (eds.): "Neue Ansätze in Maschineller Sprachübersetzung: Wissensrepräsentation und Textbezug", Sprache und Information, Niemeyer 1986, pp. 167 - 196

[Hauenschild/Umbach 88]: Ch. Hauenschild, C. Umbach: "Funktor-Argument-Struktur, Die satzsemantische Repräsentations- und Transferebene im Projekt KIT-FAST", in: J. Schütz (ed.): "Workshop Semantik und Transfer", EUROTRA-D Working Papers No.6, Saarbrücken, Juni 1988, pp. 16 - 35

[Huet/Oppen 87]: G. Huet, D.C. Oppen: "Equations and Rewrite Rules", in: R.V. Book (ed.): "Formal Language Theory, Perspectives and Open Problems", Academic Press 1980, pp. 349 - 405

Term Transformations in Program Verification

- Extended Abstract -

Bettina Buth, Karl-Heinz Buth
Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40 - 60
D - 2300 Kiel 1

One of the main application areas for term rewriting techniques is automated theorem proving. Theorem proving, again, is especially suitable for the proofs of assertions arising in program verification, since these proofs are seldom of great mathematical depth and do not require much ingenuity.

So, it is not astonishing that there are already some applications of rewriting techniques in program verification, e.g. in the Boyer-Moore prover (cf. [Boyer/Moore 79]) or in SPADE (cf. [O'Neill et al. 88]). These systems have in common that they are used in a highly interactive way; the proofs must be directed by the user (e.g. by adding new lemmas in the Boyer-Moore system or by choosing a rule that shall be applied in SPADE).

We want to present an approach to program verification that also makes use of term transformations but works mostly non-interactive. We think that this is a sensible way of running a program verifier because it delivers the user from waiting in front of the terminal the whole proof long (which can be a very long time indeed).

Our method is situated in the scope of VDM (the "Vienna Development Method", cf. [VDM 87], [VDM 88]). Actually, the "programs" we want to verify are specifications written in an essentially functional subset of META IV, the VDM specification language of VDM. This subset is, nevertheless, so general that most imperative or functional languages contain a subset that is equivalent.

A program to be verified must be given as a system of recursive function definitions. The language elements that may be used inside a function are conditional expressions, sequences ("exp_1 ; exp_2"), constant declarations ("*let* *Id* = exp_1 *in* exp_2") and function calls. There are no loop constructs; loops can, however, be expressed via recursion.

Each of the functions defined must be equipped with a pair of pre- and postconditions. These form the invariants that are needed for the inevitably inductive proof of the "recursive program". Of course, they must be formulated strong enough, since otherwise, the proof will fail. The aim is to prove each function partially correct w.r.t. its pre- and postcondition. Unlike in the Boyer-Moore system, we do not have to set up a new induction scheme for each proof. One scheme, the correctness of which is proved once and for all beforehand, can be used for all proofs.

The assertions arising during the proofs are rewritten using term transformation rules that must be given as an input to the proof system. The proof has succeeded if the constant term *true* can be reached. The rules are directed, conditional and ordered rules; in contrast to usual rewrite rules, they may contain also higher order terms (a source for this complication is the denotational definition of programming language semantics).

We require the usual properties of correctness, confluence and termination. Without the first one, the proofs generated are incorrect as well, and without the others, there is a chance that correct assertions cannot be proved. The rules arise from generally valid laws (from arithmetic and logic), from the specification of primitive functions and from assumptions valid at the moment, i.e. from the induction hypotheses and from the conditions leading to the branch of a function under consideration at the moment. This last point makes it difficult to apply the usual algorithms for completion and termination checking, since they had to be applied over and over again at every change of the rules. So, for the time being, we do not check confluence and termination but assume that these properties are fulfilled.

We have implemented our method in a system called PAMELA (= "Proof Assistant for META IV-like Languages") which is a generalized version of the PACS system ("Proof Assistant for Code Generator Specifications", cf. [Buth/ Buth 88]). As an example, we have chosen code generator specifications used in the CAT compiler generating system (cf. [Schmidt/Völler 87]). These specifications are the basis for parts of compilers that are industrially used by Norsk Data.

Up to now, we have made the experience that we have not yet encountered an error in the specifications that could not be found. It is, however, extremely important to put up sufficiently strong invariants for the proof. But the definition of pre- and postconditions seems to be a quite natural way to

provide the invariants; therefore, it is not so very difficult to make them strong enough which means nothing else than to completely describe the behaviour of a function.

References:

[Boyer/Moore 79], Boyer, R.S., Moore, J.S., *A Computational Logic*, Academic Press, 1979

[Buth/Buth 88]: Buth, B, Buth, K.-H., "Correctness Proofs for META IV Written Code Generator Specifications Using Term Rewriting", in [VDM 88], pp. 406-433

[O'Neill 88]: O'Neill, I.M. et al., "The Formal Verification of Safety Critical Assembly Code", in: Ehrenberger, W.D. (ed.), *Proc. of the SAFECOMP '88*, IFAC Proceedings Series, Nr. 16, pp. 115-120

[Schmidt/Völler 87]: Schmidt, U., Völler, R., "Experience With VDM in Norsk Data", in [VDM 87], pp. 49-62

[VDM 87]: Bjørner, D. et al. (eds.), *Proc. of the VDM-Europe Symposium 1987*, Springer, LNCS 252, 1987

[VDM 88]: Bloomfield, R. et al. (eds.), *Proc. of the VDM-Europe Symposium 1988*, Springer, LNCS 328, 1988

EQTHEOPOGLES

A Theorem Prover for First Order Predicate Logic with Equality based on Rewrite -Techniques

J. Denzinger

Department of Computer Science
University of Kaiserslautern
6750 Kaiserslautern (FRG)

Introduction

EQTHEOPOGLES (german acronym for Theorem Prover for First Order Polynomial Equations with EQuality) is an automated theorem prover for first order predicate logic with equality. It's main feature is the use of rewriting techniques, e.g. reduction and critical pair completion, as inference rules on a two level Knuth-Bendix Completion Procedure. Using EQTHEOPOGLES without equality (the THEOPOGLES system [Mü 88], [De 87], [DM 87]) means working with a variation of the methods of Hsiang [Hs 85], Kapur and Narendran [KN 85] and Bachmair, Dershowitz [BD 87]. That is, a first order formula F to be proved valid is transformed into a system E of polynomial equations, s.th. F is valid iff E is unsolvable. The ideas to handle equality with EQTHEOPOGLES stems from approaches of Hsiang [Hs 87] and Rusinowitch [Ru 87]. But as in the definition for pure first order calculus EQTHEOPOGLES avoids the use of unnecessairy inference steps. In general E is divided into a system ($\mathcal{E}$,R) of polynomial equations $\mathcal{E}$ and term rewrite rules R. Some equations of $\mathcal{E}$ are also used as rewrite rules for polynomials. Completion is performed separately on $\mathcal{E}$ and R, and special inference rules are applied on elements of $\mathcal{E}$ and R to get the connection of the whole system.
In the following we will give a sketch of the theoretical technicalities, e.g. the inference rules, of EQTHEOPOGLES.
Details can be found in [Mü 87], [Mü 88], [De 88] for the theoretical aspects especially the completeness proofs, and in [De 87], [DM 87] for technical descriptions of the system.
EQTHEOPOGLES is implemented on an APOLLO-Workstation in Common-Lisp.

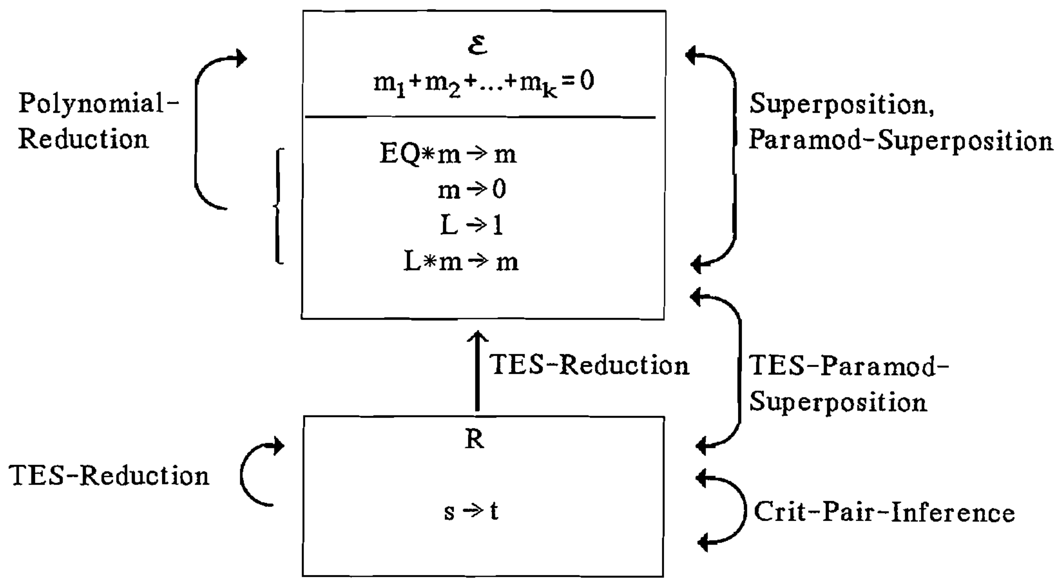
Theoretics

Polynomial equations are of the form $m_1+m_2+...+m_k=0$, where + is the operation XOR and the monomials m_i are conjunctions $L_1*L_2*...*L_n$ of atoms L_j. Given a formula F, it can be transformed into a set $E=\{p_i=0\}_{i=1..n}$ of polynomial equations, s.th. F is valid iff E has no solution, i.e. there is no interpretation I, s.th. I(p)=false for all $p=0 \in E$.
EQTHEOPOGLES divides the equational system E into two sets $\mathcal{E}$ and R. $\mathcal{E}$ represent the polynomial system and R the term rewrite system. An equation $p=0 \in E$ is transformed in a rule $s \rightarrow t$ in R, if $p \equiv EQ(s,t)+1$ and $s > t$ for a simplification ordering > on terms. All other equations in E belong to the polynomial system $\mathcal{E}$. In $\mathcal{E}$ every equation of the form m=0, EQ*m+m=0, L+1=0 or L*m+m=0 (where m a monomial, EQ an equality atom and L a non-equality atom) is used as rewriting rule on polynomial level : $m \rightarrow 0$, $EQ*m \rightarrow m$, $L \rightarrow 1$ or $L*m \rightarrow m$.

As usual these rewrite rules are used for mutual normalization and to simplify the polynomials in $\mathcal{E}$. The term rewrite rules in R reduce the terms of the atom arguments in $\mathcal{E}$ and they are also used for the interreduction of R itself (TES-Reduction). This constitutes the first class of inference rules. Note that the rewrite rules are very simple and can be efficiently implemented. On the other hand they are strong enough to reduce the search space for the (more expensive) critical pair generation drastically.
The second class of inference rules for $(\mathcal{E},R)$ is the critical pair generation. Let p_{EQ} denote polynomials consisting of EQ-Atoms and/or the atom 1. Then we have

- Superposition of one atom in m from an equation of the form $m*p_{EQ}=0$ with an atom of another equation in $\mathcal{E}$.
- Paramod-Superposition with EQ from an equation of the form $EQ*p_1+p_2=0$ with a term in an atom of another equation in $\mathcal{E}$.
- Factorization of equations of the form $m*p_{EQ}=0$ in $\mathcal{E}$.
- Standard superposition of rules in R (Crit-Pair-Inference).
- Superposition of a rule in R and a term of an atom of an equation in $\mathcal{E}$ (TES-Paramod-Superposition).

These superpositions for critical pair generation together with the simplification with polynomial rules and with term rewriting rules present the complete theorem prover EQTHEOPOGLES. The domains of every inference rule of EQTHEOPOGLES is shown below.



Now we want to compare EQTHEOPOGLES with other methods for the first order predicate logic with equality.
- The application of the critical pair generation is very restricted compared with other methods. For example, the first superposition rule might save an exponential factor relative to Hsiangs approach.
- We use two kinds of reduction relations and minimize both sets, $\mathcal{E}$ and R with these relations. So we always get a minimal representation of the system. Nevertheless we proved the soundness and completeness of EQTHEOPOGLES.
- Many authors (for example [Ru 87], [WRCS 67]) showed the advantages of an inference rule for the equality that is similar to the critical pair generation in the

Knuth-Bendix-Procedure, if the two involved clauses or polynomials are unit EQ-facts. As R is a term rewriting system and is completed by a Knuth-Bendix-Procedure (Crit-Pair-Inference), EQTHEOPOGLES includes these advantages.

- Instead of trying to find one general inference rule for the equality like paramodulation ([RW 69]) or Para-Superposition ([Hs 87]) we install three disjunct inference rules (Paramod-Superposition, Crit-Pair-Inference, TES-Paramod-Superposition). Therefore we get the possibility to prefer one rule over another. For example, we prefer the Crit-Pair-Inference over the Paramod-Superposition, because a uniquely terminated term rewriting system R (which is possibly generated) is very useful. Other preferences are also possible.
- The whole theory of term rewriting systems can be used to avoid unnessary Crit-Pair-Inferences or guide the choosing of "good" critical pairs. For example, the criteria of Kapur ([KMN 88]) or Winkler ([Wi 84]) can be integrated in EQTHEOPOGLES to reduce the number of inferences.

References

[BD 87]	Bachmair,L.; Dershowitz,N. : Inference Rules for Rewrite-Based First-Order Theorem Proving, 2nd LICS, 1987.
[De 87]	Denzinger,J. : Implementation of a Theorem Prover Based on Rewriting-Techniques (in German), project-report, University of Kaiserslautern.
[De 88]	Denzinger,J. : EQTHEOPOGLES (in German), M.S. thesis, University of Kaiserslautern.
[DM 87]	Denzinger,J.; Müller,J. : THEOPOGLES user manual, University of Kaiserslautern.
[Hs 85]	Hsiang,J. : Refutational Theorem Proving using Term Rewriting systems, AI 25, 1985, pp. 255-300.
[Hs 87]	Hsiang,J. : Rewrite Method for Theorem Proving in First Order Theory with Equality, J. of Symb. Comp., 1987.
[KMN 88]	Kapur,D.; Musser,D.R.; Narendran,P. : Only Prime Superpositions Need be Considered in the Knuth-Bendix Completion Procedure, J. of Symb. Comp. 6, 1988, pp. 19-36.
[KN 85]	Kapur,D.; Narendran,P. : An Equational Approach to Theorem Proving in First-Order Predicat Calculus, 84CRD322, GEC Research and Dev. Report, Schenactady, N.Y., 1985.
[Mü 87]	Müller,J. : THEOPOGLES - A Theorem Prover Based on First-Order Polynomials and a Special Knuth-Bendix Procedure, Proc. 11th GWAI, 1987, Spinger IFB 152.
[Mü 88]	Müller,J. : Theorem Proving with Rewrite-Techniques, - Methods, Strategies and Comparisons - (in German), Ph.D. Thesis, University of Kaiserslautern, 1988.
[Ru 87]	Rusinowitch,M. : Demonstration automatique par des techniques de reecriture, These de Doctorat d'Etat en Mathematique, Nancy, 1987.
[RW 69]	Robinson,G.; Wos,L. : First-order Theorem Proving with Equality, Mach. Intelligence, vol. 4, Edinburgh, 1969, pp. 135-150.
[Wi 84]	Winkler,F.: The Church-Rosser Property in Computer Algebra and Special Theorem Proving : an Investigation of Critical Pair Completion Algorithms, Ph.D. Thesis, J.-Kepler University Linz, 1984.
[WRCS 67]	Wos,L.; Robinson,G.; Carson,D.; Shalla,L. : The Concept of Demodulation in Theorem Proving, J. of ACM 14, 1967, pp. 698-709.