UNIF '89

Extended Abstracts of the 3rd Int. Workshop on Unification

H.-J. Bürckert & W. Nutt (Eds.)

SEKI Report SR-89-~~16~~ 17

UNIF'89

Third International Workshop on Unification 1989

PFALZAKADEMIE
Lambrecht, FR Germany

Monday, June 26th – Wednesday, 28th 1989

organized by H.-J. Bürckert and W. Nutt

sponsored by
Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI)
Universität Kaiserslautern, FB Informatik
Volkswagen-Stiftung

Preface

This is a collection of extended abstracts of the talks given at the Third International Workshop on Unification (UNIF'89) hold during June 26th - 28th, 1989 in the PFALZAKADEMIE at Lambrecht, Germany. The workshop is a forum for the researchers that are interested in Unification Theory and its applications, and we met since 1986 every year to exchange and discuss results, ideas and new trends in this area.

Topics of the unification workshops are

- Narrowing
- General E-Unification & Calculi
- Implementations
- Special Unification Algorithms
- Disunification
- Typed Unification
- Foundations
- Applications
- Combination Problems
- Constraint Solving

This 3rd Workshop on Unification had 49 participants from Belgium, France, Germany, Great Britain, and the USA. We had 10 short and 18 long talks, several system demonstrations, and an excursion to the Weingut Schönhof with a wine tasting presented by Philip Scammel and a banquet speech about living with limited vocabulary by Pierre Lescanne.

The volume is organized as follows. There are seven sections, each with three to five abstracts, essentially in the order of presentation at Lambrecht.

The first section is about ***General E-Unification & Combination*** and contains papers on *Higher Order E-Unification* (J. Gallier, W. Snyder, V. Tannen) and its connection to first-order rewriting and on the translation of proofs from one proof system into another one (*Proof Transformations For Simple Equational Theories*, T. Nipkow). These are followed by presentations about combination methods for regular and collapse-free theories applied to AC-unification (*A New Combination Technique for AC-Unification*, A. Boudet) and combination methods for general theories (*Optimizations of Schmidt-Schauß' General Unification Procedure*, M. Tepp).

The second section is on ***Applications*** of unification in the field of ***Completion*** of term rewriting systems. It begins with a paper by P. Watson and A.J.J. Dick on *Least Sorts in Order-Sorted Rewriting*. They attack the problem of non-uniquely determined sorts of terms during rewriting by introducing dynamic sorts. The second paper is by U. Martin and T. Nipkow on *Order-sorted Rewriting and Confluence*. Some ideas in constructing completion procedures are given by P. Lescanne in his paper *Completion Procedures as Transition Rules + Control* at the end of this section.

Third section is on ***Foundations*** starting with W. Nutt's *Unification in Modular Categories*. This is followed by *Unification in Commutative Theories, Hilbert's Basis Theorem, and Gröbner Bases* (F. Baader). The next paper is by F. Klay and

C. Kirchner, *A Note on Syntactic Theories*. The section closes with two papers on decidability: *On the Decidability of the Unification Problem* by A. Bockmayr and W. Nutt with *The Unification Hierarchy is Undecidable*.

The next section on ***Typed Unification*** begins with a paper by G. Smolka: *Polymorphically Order Sorted Unification*. L. Duponcheel presents a paper on *Typed Algebra*, followed by U. Waldmann with *Unitary Unification in Order-sorted Signatures*. Finally E. Domenjoud shows a way to do *AC-Unification Through Order Sorted AC1-Unification*.

The two sections on ***Special Algorithms*** start with a the contribution of H. Abdulrab and J.-P. Pecuchet on *Associative Unification* followed by two presentations on solving systems of diophantine equations. The first of them contains a generalization of Fortenbacher's algorithm to solve linear diophantine equations to a procedure that solves systems of linear diophantine equations in an efficient way (E. Contejean, H. Devie: *Solving Systems of Linear Diophantine Equations*). The second one gives some boundaries on the size of *Solutions of a Linear Diophantine System* (J.-F. Romeuf). Finally H.J. Ohlbach presents an advantageous representation of terms, *Abstraction Tree Indexing for Terms*.

The second section on ***Special Algorithms*** is started with S. Hölldobler's paper *Unification over Rational Trees*, that is unification of terms that may be cyclic. He is followed by L. Pottier and his presentation on *Term Generalizations in the AC Case*. H. Leiß comes up with a procedure for a process that might be seen as a common generalization of unification and matching: *A Semi-Unification Algorithm*. Finally B. Gramlich presents some ideas on the *Unification of Term Schemes*.

The last section, ***Equational Problems & Constraint Solving***, begins with C. and H. Kirchner's *Constrained Equational Reasoning*. The topic of Order Sorted Algebras is again taken up by H. Comon in his paper on *Equational Problems in Order Sorted Algebras*. H. Aït-Kaci presents his quite technical work on structured types *(Disjunctive ψ-Term Unification)*. The last paper of the workshop is by L. Puel and A. Suarez, *Unification of Restricted Terms*.

The volume is closed with an appendix containing the list of wines we tasted at Weingut Schönhof, an abstract of the banquet speech by P. Lescanne on the *Life with a Limited Vocabulary*, and of course the list of participants.

We would like to thank the authors, without whom the workshop would never have taken place. We also thank the staff of the PFALZAKADEMIE who provided an inspiring atmosphere and supported the organization of the workshop. Thanks also to Dorothea Kilgore and Patricia Sarach who had the not always easy job of assisting with organization and registration. Gebhardt Pzyrembel prepared the system demonstrations and SUN Germany provided machines for these demos. Finally we thank Volkswagen-Stiftung for their financial support of the workshop.

We hope to meet most people again at the next Workshop on Unification that will take place in Leeds, Great Britain, in July 1990 and will be organized by John K. Truss.

August, 1989 Hans-Jürgen Bürckert and Werner Nutt

Contents

Preface 3

Contents 5

Section 1 General E-Unification & Combination 7

Higher Order E-Unification *J. Gallier , W. Snyder, V. Tannen*

Proof Transformations for Some Simple Equational Theories *T. Nipkow*

A New Combination Technique for AC-Unification *A. Boudet*

Optimizations of Schmidt-Schauß' General Combination Procedure *M. Tepp*

Section 2 Applications – Completion 33

Completion-Time Optimization of Rewrite-Time Goal Solving *H. Bertling, H. Ganzinger*

Order-sorted Rewriting and Confluence *U. Martin, T. Nipkow*

Completion Procedures as Transition Rules + Control *P. Lescanne*

Section 3 Foundations 63

Unification in Modular Categories *W. Nutt*

Unification in Commutative Theories, Hilbert's Basis Theorem, and Gröbner Bases *F. Baader*

Solving Cyclic Equations in Equational Theories *F. Klay*

On the Decidability of the Unification Problem *A. Bockmayr*

The Unification Hierarchy is Undecidable *W. Nutt*

Section 4 Typed Unification 87

Polymorphically Order Sorted Unification *G. Smolka*

Order Sorted Algebra Revisited *L. Duponcheel*

Least Sorts in Order-sorted Rewriting *P. Watson, A.J.J. Dick*

AC-Unification Through Order Sorted AC1-Unification *E. Domenjoud*

Section 5 Special Algorithms I **115**

Associative Unification *H. Abdulrab, J.-P. Pecuchet*

Solving Systems of Linear Diophantine Equations *E. Contejean, H. Devie*

Solutions of Linear Diophantine Systems *J.-F. Romeuf*

Abstraction Tree Indexing for Terms *H.J. Ohlbach*

Section 6 Special Algorithms II **137**

Unification over Rational Trees *S. Hölldobler*

Pseudo-Unification for Type Inference *J.-P. Jouannaud*

A Semi-Unification Algorithm *H. Leiß*

Unification of Term Schemes *B. Gramlich*

Section 7 Equational Problems & Constraint Solving **159**

Constrained Equational Reasoning *C. Kirchner, H. Kirchner*

Equational Problems in Order Sorted Algebras *H. Comon*

Disjunctive ψ-Term Unification *H. Aït-Kaci*

Unification of Restricted Terms *L. Puel, A. Suarez*

Appendix **187**

Wine tasting *P. Scammel*

Life with a Limited Vocabulary (banquet speech) *P. Lescanne*

List of Participants

Section 1:
General E-Unification & Combination

J. Gallier , W. Snyder, V. Tannen:: Higher Order E-Unification

T. Nipkow: Proof Transformations for Equational Theories

A. Boudet: A New Combination Technique for AC-Unification

M. Tepp: Optimizations of Schmidt-Schauß' General Combination Procedure

Higher-Order E-Unification

Jean Gallier, Wayne Snyder, and Val Tannen

In this paper we investigate the problem of *Higher-Order E-Unification*, which seems to be the most general [illegible] of interest in automated deduction. A higher-order substitution θ is a *higher-order E-unifier* of two typed lambda terms e_1 and e_2 iff

$$\theta(e_1) \longleftrightarrow_{\beta\eta E} \theta(e_2),$$

where $\longleftrightarrow_{\beta\eta E}$ is the least congruence on lambda terms containing β- and η-reduction and every substitution instance of equations from E. The point here is that the first-order equations can only rewrite first-order portions of the lambda terms. The combination of higher-order types and first-order rewriting has been studied by Val Breazu-Tannen and Jean Gallier [1, 2], and has some interesting applications in reasoning simultaneously about functional programs and their data structures. Applications of this new form of (very) general unification remain to be investigated, but it appears that the results of [2] show that the combination of our sets of transformations $\mathcal{T}$ and $\mathcal{HT}$ will result in a complete set of transformations for this new problem, and it also appears that completeness can be proved along the lines of the proofs in [4] and [5]. Although we are preparing a preliminary report [hoeunif!num] of these (still tentative) results, much remains to be done in generalizing the notion of complete sets of solutions, finding restrictions on the set of transformations which improve efficiency while preserving completeness, and determining if the problem is practically interesting and our method is computationally feasible.

1 References

[1] Breazu-Tannen, V., "Combining Algebra and Higher-Order Types," LICS 1988, Edinburgh, Scotland.

[2] Breazu-Tannen, V., and Gallier, J., "Polymorphic Rewriting Conserves Algebraic Strong Normalization and Confluence," submitted to ICALP 1989. LICS 1988, Edinburgh, Scotland.

[3] Gallier, J.H., and Snyder, W., "A General Complete E-Unification Procedure," Conference on Rewriting Techniques and Applications, Bordeaux, France (1987).

[4] Gallier, J.H., and Snyder, W., "Complete Sets of Transformations for General E-Unification," to be published in a special issue of Theoretical Computer Science (1989).

[5] Gallier, J.H., and Snyder, W., "Higher Order Unification Revisited: Complete Sets of Transformations," to be published in a special issue of JSC (1989).

Proof Transformations For Equational Theories*
(*Extended Abstract*)

Tobias Nipkow
University of Cambridge
Computer Laboratory
Pembroke Street
Cambridge CB2 3QG
England

1 Introduction

We contrast two kinds of proof systems for a number of equational systems E. On the one hand there is the standard one obtained by combining E with the laws of equational logic. We know that the resulting system defines what we want it to define. However, it gives no indication how the word problem, matching, or unification can be solved in that theory. On the other hand we present proof systems that are not obviously complete for E but which immediately give rise to matching or even unification procedures.

Although new matching algorithms for associativity (A), associativity + commutativity (AC), and associativity + commutativity + identity ($AC1$) are presented, the emphasis is not so much on individual theories but on the general method of proof transformations as a tool for showing the equivalence of different proof systems. In particular a mechanizable equivalence test and its implementation are discussed. Although the test is not complete, it is powerful enough to deal with many equational systems.

The paper comes in two parts. Section 3 compares standard proof systems for the empty theory, commutativity, left/right-commutativity, A, and AC, with alternative ones. Equivalence is shown by presenting a terminating set of rewrite rules which translate proofs from one system into the other. However, these rewriting systems become more and more complex to construct and to prove terminating. Therefore Section 4 uses the notion of "resolvance" to present a uniform treatment of the theories in Section 3. It is shown that resolvant systems of equations directly yield alternative inference systems which, in certain cases, are terminating matching algorithms. Two powerful criteria for resolvance are developed and their implementation is discussed. This implementation is used to check a number of further equational systems for resolvance, which yields new matching algorithms for some of them. It is also shown that resolvance is a modular property, i.e. putting resolvant sets of equations together which do not have function symbols in common does again yield resolvant systems. Finally we discuss the relationship of our work with some recent results by Claude Kirchner.

The reader should be familiar with the basic notions of equational logic, as defined for example in [6].

2 Equational Theories, Unification, Matching, and Equality

Equational theories are defined by inference rules of the form

$$\frac{s_1 = t_1 \quad \ldots \quad s_n = t_n}{s = t}. \tag{1}$$

*This work was supported by the Alvey Diamond project, SERC grants GR/E/02369 and GR/F/10811. At MIT the author was supported in part by NYNEX, NSF grant CCR-8706652, and by the Advanced Research Projects Agency of the DoD, monitored by the ONR under contract N00014-83-K-0125.

A system of such rules inductively defines a predicate $=$. Furthermore, since all rules are Horn clauses, they constitute a logic program which can be used to solve unification, matching or word problems w.r.t. $=$. The only problem is termination. Since a set of Horn clauses merely yields a semidecision procedure, it depends on the particular set of rules whether they actually constitute a decision procedure for equality, matching or unification. Thus we can separate the question of correctness (does the given set of rules axiomatize the desired theory?) from that of termination.

Given a set of equations E, the equational theory induced by E is defined by E plus reflexivity, symmetry, transitivity and congruence. This system is denoted by E^+ and the predicate it defines by $=_E$. Although E^+ is by definition correct, i.e. defines the equational theory generated by E, it has a major termination problem: due to transitivity, any query (in the logic programming sense) will run forever even after all answer substitutions have been found.

The rest of the paper discusses alternative axiomatizations of various simple equational theories which yield decision procedures for matching and, in some trivial cases, even unification. Initially attention is focussed on correctness. In Section 4 the termination question is addressed and a fairly general answer is given.

Before we become technical, we have to fix some terminology. We call an equational theory *permutative* if all its equivalence classes are finite. A set of equations is permutative if its equational theory is. An equation $s = t$ is called *regular* if $\mathcal{V}(s) = \mathcal{V}(t)$, where $\mathcal{V}$ returns the set of variables in a term. An equation $s = t$ is called *collapse-free* if both s and t are proper terms.

3 Proof Transformation by Rewriting

In the following we examine different formulations of some well known equational theories. Each alternative axiomatization D_{rk} of E^+ consists of a collection of rules of the form (1), say D for "decomposition", together with congruence rules and reflexivity. D_{rk} and E^+ are shown to be equivalent by translating proofs from one into the other. The translation is expressed by rewrite rules on proof trees.

D and E are always chosen such that each element of D is a derived rule of E^+ and each equation in E is a derived rule of D_{rk}. Thus the transformation of proofs in D_{rk} into E^+ is trivial and is briefly described in Section 4. The translation in the opposite direction needs to get rid only of symmetry and transitivity, the two rules that cause termination problems and do not occur in D_{rk}.

3.1 The Empty Theory

We start with the basic laws of equational logic, reflexivity (r), symmetry (s), transitivity (t) and congruence (k) for a single binary function symbol ($\cdot$). The choice of function symbols is immaterial.

$$r \equiv \frac{}{x = x}$$

$$s \equiv \frac{x = y}{y = x}$$

$$t \equiv \frac{x = y \quad y = z}{x = z}$$

$$k \equiv \frac{x = u \quad y = v}{x \cdot y = u \cdot v}$$

In the sequel this system is called B. B axiomatizes equality in the empty theory. Of course r alone suffices for that. A constructive proof of this fact can be given by a set of transformation rules which eliminate all other rules from a proof in B.

$$\dfrac{s\dfrac{r\dfrac{}{x = x}}{x = x}}{x = x} \quad \longrightarrow \quad \frac{r\dfrac{}{x = x}}{x = x}$$

$$t\dfrac{r\dfrac{}{x=x}\quad r\dfrac{}{x=x}}{x=x} \longrightarrow r\dfrac{}{x=x}$$

$$k\dfrac{r\dfrac{}{x=x}\quad r\dfrac{}{y=y}}{x\cdot y=x\cdot y} \longrightarrow r\dfrac{}{x\cdot y=x\cdot y}$$

These rules are obviously terminating and cover all cases, i.e. reduce any proof in B to reflexivity.

In the sequel E will be some set of equations and D a set of inference rules. We want to show that for given D and E, any proof in $E^+ = E \cup B$ can be rewritten to one in $D_{rk} = D \cup \{r, k\}$. This transformation can be done on a rule by rule basis.

Axioms from E can be eliminated in one step because E and D will always be chosen such that all axioms in E are equivalent to some combination of rules in D_{rk}.

To simplify elimination of s and t, we show that B has some further properties:

$$t\dfrac{s\dfrac{x=y\quad y=z}{x=z}}{z=x} \longrightarrow t\dfrac{s\dfrac{y=z}{z=y}\quad s\dfrac{x=y}{y=x}}{z=x} \tag{2}$$

$$s\dfrac{k\dfrac{x=u\quad y=v}{x\cdot y=u\cdot v}}{u\cdot v=x\cdot y} \longrightarrow k\dfrac{s\dfrac{x=u}{u=x}\quad s\dfrac{y=v}{v=y}}{u\cdot v=x\cdot y} \tag{3}$$

$$t\dfrac{r\dfrac{}{x=x}\quad x=z}{x=z} \longrightarrow x=z \qquad t\dfrac{x=z\quad r\dfrac{}{z=z}}{x=z} \longrightarrow x=z \tag{4}$$

$$t\dfrac{k\dfrac{u=w\quad v=x}{u\cdot v=w\cdot x}\quad k\dfrac{w=y\quad x=z}{w\cdot x=y\cdot z}}{u\cdot v=y\cdot z} \longrightarrow k\dfrac{t\dfrac{u=w\quad w=y}{u=y}\quad t\dfrac{v=x\quad x=z}{v=z}}{u\cdot v=y\cdot z} \tag{5}$$

Rules (2) and (3) show that symmetry can be pushed through transitivity and congruence. Therefore symmetry can be pushed to the leaves of any proof in $B \cup E$, and hence can be eliminated altogether, provided that E is closed under s, i.e. $s{=}t \in E$ implies $t{=}s \in E$. In the sequel E will always have that property, and elimination of s is automatic. This is the formal justification of the fact that equational proofs don't need explicit symmetry if all axioms can be used in both directions.

The only remaining task is the elimination of t. The rules (4)-(5) show that it is sufficient to cover the cases $t(\rho, k)$, $t(k, \rho)$, and $t(\rho, \rho')$ for all rules ρ, ρ' in D.

Before we look at particular systems D and E, we simplify our notation. Although the above rewrite rules rules are fairly involved already, they are not quite precise in that they don't say how the subtrees are relocated. The proper formulation of, for example, rule (3) is

$$s\dfrac{k\dfrac{\dfrac{P}{x=u}\quad\dfrac{Q}{y=v}}{x\cdot y=u\cdot v}}{u\cdot v=x\cdot y} \longrightarrow k\dfrac{s\dfrac{\dfrac{P}{x=u}}{u=x}\quad s\dfrac{\dfrac{Q}{y=v}}{v=y}}{u\cdot v=x\cdot y}$$

where P and Q are the proof trees that prove $x = u$ and $y = v$. However, this rule is more complicated than it needs to be. All that is required is the pattern of proof rules as in

$$s(k(P, Q)) \longrightarrow k(s(P), s(Q)).$$

We now assume that r, s, t, and k are functions on proof trees or skeletons. The actual formulae being proved are determined by these proof skeletons. Under this new interpretation the above rewrite rules

translate to

$$\left.\begin{array}{rcl} s(r) & \longrightarrow & r \\ t(r,r) & \longrightarrow & r \\ k(r,r) & \longrightarrow & r \\ s(t(x,y)) & \longrightarrow & t(s(y),s(x)) \\ s(k(x,y)) & \longrightarrow & k(s(x),s(y)) \\ t(r,x) & \longrightarrow & x \\ t(x,r) & \longrightarrow & x \\ t(k(x,y),k(u,v)) & \longrightarrow & k(t(x,u),t(y,v)) \end{array}\right\} T$$

Notice that $x, y, z, \ldots$ stand for proof skeletons, not terms or formulae. The termination of T can now be shown by a mechanical system like LP [5].

3.2 Commutativity (C)

An alternative axiomatization of commutativity is obtained by adding

$$c \equiv \frac{x = v \quad y = u}{x \cdot y = u \cdot v}.$$

to r and k. This system is trivial and well known. For example Claude Kirchner [8] derives c automatically from commutativity. Commutativity is proved from c by composing both premises with reflexivity, i.e. the conclusion of $c(r,r)$ is $x \cdot y = y \cdot x$. The same device works for all subsequent equational theories.

The following further rewrite rules are needed to translate proofs:

$$\begin{array}{rcl} t(c(x,y),c(u,v)) & \longrightarrow & k(t(x,v),t(y,u)) \\ t(c(x,y),k(u,v)) & \longrightarrow & c(t(x,v),t(y,u)) \\ t(k(x,y),c(u,v)) & \longrightarrow & c(t(x,u),t(y,v)) \end{array}$$

Again, LP manages to show that the union of T with the above rules is a terminating system.

It is interesting to note that both for the empty theory and for commutativity, the system D_{rk} could be derived from $B \cup E$ automatically using the CEC system [2,4] which is a general purpose system for the completion of sets of conditional equations. The following equational theories are more complex and automatic tools failed to help.

3.3 Left/Right-Commutativity ($C_{l/r}$)

The first non-trivial example is left/right-commutativity. For simplicity we consider only $E = \{(x \cdot y) \cdot z = (x \cdot z) \cdot y\}$, right-commutativity. Left-commutativity is symmetric.

It turns out that if $D = \{c_r\}$, where

$$c_r \equiv \frac{x = w \cdot v \quad w \cdot y = u}{x \cdot y = u \cdot v},$$

then D_{rk} axiomatizes C_r. The elimination of t is achieved by the following rules:

$$\begin{array}{rcll} t(k(x,y),c_r(u,v)) & \longrightarrow & c_r(t(x,u),t(k(r,y),v)) & (6) \\ t(c_r(x,y),k(u,v)) & \longrightarrow & c_r(t(x,k(r,v)),t(y,u)) & (7) \\ t(c_r(x,k(y_1,y_2)),c_r(u,v)) & \longrightarrow & t(c_r(x,r),c_r(t(k(y_1,y_2),u),v)) & (8) \\ t(c_r(x,c_r(y_1,y_2)),c_r(u,v)) & \longrightarrow & t(c_r(x,r),c_r(t(c_r(y_1,y_2),u),v)) & (9) \\ t(c_r(x,r),c_r(r,v)) & \longrightarrow & k(t(x,v),r) & (10) \\ t(c_r(x,r),c_r(k(u_1,u_2),v)) & \longrightarrow & k(t(x,t(k(u_1,r),v)),u_2) & (11) \\ t(c_r(x,r),c_r(c_r(u_1,u_2),v)) & \longrightarrow & c_r(t(x,c_r(u_1,r)),t(c_r(r,u_2),v)) & (12) \end{array}$$

The first two rules cover the cases that either of t's subtree is labelled with k. The next two rules translate from $t(c_r(.,.),c_r(.,.))$ to $t(c_r(.,r),c_r(x,.)$, and the last three rules translate the latter pattern, depending on the form of x.

The above set of rules is not easily proved to terminate and systems like LP fail to do so. The point is that the termination argument is hidden in the equations that are being proved, which are not part of the proof skeletons. To exploit this information, we view t as a function defined on terms over $\{r, k, c_r\}$. Looking at the form of the transformation rules we can see that their termination, or equivalently totality of t, can be shown by proving that some measure function on the arguments of t decreases when going from left to right. This measure is the size of the equation (or either side of it) that forms the invisible conclusion of the proof tree rooted in t. One way to see that the measure decreases is to decorate the rewrite rules with the equations being proved. For example rule (10) becomes

$$\dfrac{\dfrac{x = u\cdot v \quad \overset{r}{\overline{u\cdot y = u\cdot y}}}{x\cdot y = (u\cdot y)\cdot v}\,c_r \quad \dfrac{\overset{r}{\overline{u\cdot y = u\cdot y}} \quad u\cdot v = z}{(u\cdot y)\cdot v = z\cdot y}\,c_r}{x\cdot y = z\cdot y}\,t \longrightarrow \dfrac{x = u\cdot v \quad u\cdot v = z}{x = z}\,t \quad \overset{r}{\overline{y = y}}\;\Big/\; x\cdot y = z\cdot y\;k$$

On the rhs t proves $x = z$, which is strictly smaller than $x\cdot y = z\cdot y$, the conclusion of the tree labelled by t on the lhs.

Alternatively, we notice that the size of the conclusion of k and c_r is strictly greater then the size of their hypotheses. Hence in any rule with lhs $t(\ldots k(x, y)\ldots)$ or $t(\ldots c_r(x, y)\ldots)$, the measure of x and y is smaller than the measure of the full lhs. Hence x and y are ok as arguments to t on the rhs.

If the termination proof is done in complete detail, one notices that in rules (8) and (9) the outer occurrence of t on the rhs proves the same formula as the t on the lhs, i.e. the measure is not decreased. However, none of the two rules can be applied twice in a row. Therefore the complexity measure becomes a pair. Its first component is what we had before, which all other rules decrease. The second component indicates the applicability of rule (8) or (9). It is decreased by those two rules, which do not change the first component.

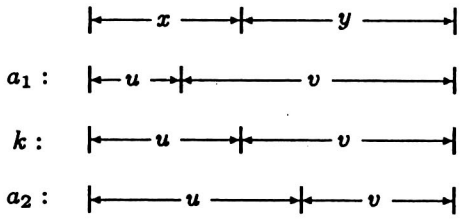
As we have seen, both the transformation rules and their termination proofs become quite complex even for very simple equational theories. For that reason, both are omitted in the following examples. The author has carried them out by hand and found them very similar to the what we saw in this section, only more tedious. This prompted him to look for automatic methods which are presented in Section 4.

3.4 Associativity (A)

Associativity can be axiomatized with k, r,

$$a_1 \equiv \frac{x = u\cdot w \quad w\cdot y = v}{x\cdot y = u\cdot v} \text{ and } a_2 \equiv \frac{x\cdot w = u \quad y = w\cdot v}{x\cdot y = u\cdot v}.$$

A geometric interpretation of this fact can be given where terms are interpreted as strings or lines and $\cdot$ is concatenation. If the two lines $x\cdot y$ and $u\cdot v$ are equal, there are three cases:



It can be shown that Plotkin's associative unification algorithm [13] is an optimization of the algorithm embodied in the inference rules k, r, a_1 and a_2. Optimization means that a particular search strategy has been imposed.

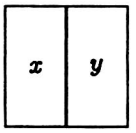
3.5 Associativity + Commutativity (AC)

For AC let $D = \{c, a_1, a_2, ac_1, ac_2, ac\}$, where

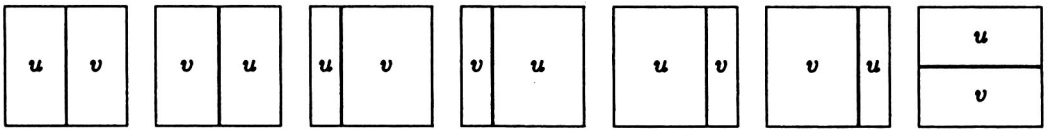
$$ac_1 \equiv \frac{x = v\cdot w \quad w\cdot y = u}{x\cdot y = u\cdot v} \qquad ac_2 \equiv \frac{x\cdot w = v \quad y = w\cdot u}{x\cdot y = u\cdot v}$$

$$ac \equiv \frac{x = x_1 \cdot x_2 \quad y = y_1 \cdot y_2 \quad x_1 \cdot y_1 = u \quad x_2 \cdot y_2 = v}{x \cdot y = u \cdot v}.$$

Again there is a geometric interpretation of these rules, this time in terms of areas or multisets. If $x \cdot y$ is of the form



and $u \cdot v$ is the same area, there are 7 ways in which they can cover each other:



k c a_1 ac_1 a_2 ac_2 ac

4 Resolvant Theories

The collection of equational theories presented above and the complexity of some of the completeness proofs raises the question whether there is some hidden principle behind all of them which might even be automated. For some of the simpler examples (C, $C_{l/r}$) this question was first answered by Claude Kirchner in [7,8] using the notion of *resolvant* theories.

A set of equational axioms E is *resolvant* if $s =_E t$ implies that there is an equational derivation of this fact which uses at most one application of an equation at the root of a term. Notice that by a "derivation" $s =_E t$ we refer to a list of terms $s = s_0, \ldots, s_n = t$ such that s_{i+1} can be obtained from s_i by replacing a subterm which is an instance of one side of an equation in E by the corresponding instance of the other side. In particular we call the step $s_k =_E s_{k+1}$ a *peak* if s_k is an instance of one side of an equation in E and s_{k+1} the corresponding instance of the other side. With this terminology we can say that E is resolvant if $s =_E t$ implies that there is an equational derivation of this fact with at most one peak. For a formal definition of resolvance we need the following simple predicates:

$$\begin{aligned}
s \mathrel{\dot{=}}_E t &\Leftrightarrow \exists f, s_i, t_i.\ s = f(s_1, \ldots, s_n) \wedge t = f(t_1, \ldots, t_n) \wedge s_1 =_E t_1 \wedge \ldots \wedge s_n = t_n \\
s \doteq_E t &\Leftrightarrow \exists p{=}q \in E, \sigma.\ s = \sigma p \wedge t = \sigma q \\
s \mathrel{\dot{\doteq}}_E t &\Leftrightarrow \exists s', t'.\ s \mathrel{\dot{=}}_E s' \doteq_E t' \mathrel{\dot{=}}_E t \\
s \mathrel{\dot{\doteq}}_E t &\Leftrightarrow s \mathrel{\dot{=}}_E t \vee s \mathrel{\dot{\doteq}}_E t
\end{aligned}$$

Notice that in general only $\doteq_E$ is decidable, provided E is finite. If E is also permutative, all four predicates are in principle decidable, although in practice a complexity theoretic barrier may quickly be reached.

Definition 1 *E is called* resolvant *iff $s =_E t$ implies $s \mathrel{\dot{\doteq}}_E t$.*

An equational theory is called *syntactic* in [8] if it is generated by a finite set of resolvant axioms. In the sequel we assume tacitly that all equational theories are collapse-free. The theory can be made to work without that restriction but that requires some further case distinctions.

The point is that each equational theory presented above is syntactic but only C and $C_{l/r}$ are resolvant. Although [8] gives some sufficient conditions for syntacticness, they are not met by $C_{l/r}$, A or AC. The aim of this section is to present a more powerful criterion for resolvance, its implementation, and its application to both the examples above and some further theories.

For resolvant theories there is a simple translation from equational axioms to inference rules: an equation $f(s_1, \ldots, s_m) = g(t_1, \ldots, t_n)$ yields

$$\frac{x_1 = s_1 \ \ldots \ x_m = s_m \quad t_1 = y_1 \ \ldots \ t_n = y_n}{f(x_1, \ldots, x_m) = g(y_1, \ldots, y_n)} \tag{13}$$

where the x_i and y_j are new variables. In [8] this translation is called *Gen*. All the inference rules presented above can be generated this way. However, in many cases this leads to rules with trivial equations of the form $x = y$ among the hypotheses. These can be deleted w.l.o.g. if x is replaced by y everywhere else.

Starting with C, $C_{l/r}$, or A we obtain the rules shown in Sections 3.2-3.4. For AC we needed three more rules. Those are generated by the equations

$$P = \{(x{\cdot}y){\cdot}z = (x{\cdot}z){\cdot}y,\ x{\cdot}(y{\cdot}z) = y{\cdot}(x{\cdot}z),\ (x{\cdot}y){\cdot}(u{\cdot}v) = (x{\cdot}u){\cdot}(y{\cdot}v)\}$$

The reason is that AC by itself is not resolvant but $AC^+ = AC \cup P$ is. Note that all elements of P are equational consequences of AC.

In the sequel let D be the set of inference rules obtained from E as above, let K be the set of all congruence rules, and let $D_{rk} = D \cup \{r\} \cup K$. In particular let k_f be the congruence rule for function symbol f.

We can now give a general scheme for translating proofs in D_{rk} to those in E^+, the direction that had not been tackled in Section 3. For every rule $\rho \in D$, $\rho(r, \ldots, r)$ is the proof of some equation $e \in E$. If ρ is rule (13), we have the following translation from D_{rk} to E^+:

$$\rho(p_1, \ldots, p_m, q_1, \ldots, q_n) \longrightarrow t(k_f(p_1, \ldots, p_m), t(e, k_g(q_1, \ldots, q_n))) \tag{14}$$

The importance of resolvant theories stems from the following theorem.

Theorem 1 *If E is resolvant, D_{rk} is a sound and complete inference system for $=_E$.*

Proof Soundness of D_{rk} follows from the fact that each rule in D is a derived rule in E^+, as witnessed by (14). The completeness proceeds by induction on the length of equational proofs. Let $s =_E t$ and distinguish 3 cases.

If $s = t$, this has an immediate proof by reflexivity.

If $s \mp_E t$, it follows that $s = f(s_1, \ldots, s_n)$, $t = f(t_1, \ldots, t_n)$, and $s_i =_E t_i$ for all i. Since the derivation of the latter equations are shorter than the derivation of $s =_E t$, there is a proof skeleton p_i in D_{rk} for each of them. Hence there is a proof of $s =_E t$ with the skeleton $k_f(p_1, \ldots, p_n)$.

If $s \doteqdot_E t$, it follows that there is an equation $f(s_1, \ldots, s_m) = g(t_1, \ldots, t_n)$ in E and

$$s = f(r_1, \ldots, r_m) \mp_E f(s'_1, \ldots, s'_m) \doteq_E g(t'_1, \ldots, t'_n) \mp_E g(u_1, \ldots, u_n) = t$$

such that s'_i and t'_j are instances of s_i and t_j respectively and that $s_i =_E s'_i$ and $t'_j =_E t_j$ for all i and j. Since the derivation of the latter equations are shorter than the derivation of $s =_E t$, there are proof skeletons p_i and q_j for each of them. Hence the skeleton $\rho(p_1, \ldots, p_m, q_1, \ldots, q_n)$, where ρ is the rule (13) derived from $f(s_1, \ldots, s_m) = g(t_1, \ldots, t_n)$, proves $s =_E t$. □

More than that:

Theorem 2 *If E is permutative, the interpretation of D_{rk} as a Prolog program yields a terminating matching algorithm.*

Proof An equation $l = r$ is called a *matching problem* if $\mathcal{V}(l) = \{\}$. Because E is permutative, it must be regular, and hence any solution σ to a matching problem $l = r$ must be such that $\mathcal{V}(\sigma r) = \{\}$.

Interpreting D_{rk} as a Prolog program means that the goals are kept as a list of equations. We use the ML notation for lists, i.e. $[e_1, \ldots, e_n]$ is a list of length n, and @ denotes list concatenation.

First we establish that the current goal is always a matching problem if it was one initially. Resolution of a rule in D_{rk} with a matching problem $l = r$ will always lead to a list of subgoals of the form

$$H = H_l @ H_r = [l_1 = s_1, \ldots, l_m = s_m, t_1 = r_1, \ldots, t_n = r_n] \tag{15}$$

such that each $l_i = s_i$ is a matching problem, and $\bigcup_{j=1}^{n} \mathcal{V}(t_j) \subseteq \bigcup_{i=1}^{m} \mathcal{V}(s_i)$. The first property follows from the form of the rules, the second one is a consequence of the fact that E is regular. Both properties together imply that during execution the current goal will always be a matching problem: either it was one to start with ($l_i = s_i$), or a solution of the goals to the left of it have instantiated all variables on its lhs by ground terms ($t_j = r_j$).

To prove termination, we need some more definitions. If $S \subseteq \mathbf{N}$, $max(S) = m$ if $m \in S$ and all elements in S are less or equal m, and $max(S) = 0$ if there is no such m. The function *depth* computes the depth of a term, where the depth of variables and constants is 1. The function

$$maxd_E(s) = max\{depth(t) \mid s =_E t\}$$

computes the maximal depth of any E-equivalent term. Since E is permutative, $maxd_E(s)$ is never 0. The ordering $<$ on $\mathbf{N}$ is extended to multisets over $\mathbf{N}$ in the canonical way [3]. Multiset union is written as $\sqcup$. The function C takes two lists of equations and produces a multiset of natural numbers:

$$\begin{aligned} C([],L) &= \{\} \\ C([l=r]@R,L) &= \{max\{maxd_E(\sigma l) \mid \sigma \vdash_E L\}\} \sqcup C(R, L@[l=r])\} \end{aligned}$$

where $\sigma \vdash_E L$ means that σ solves all equations $s = t$ in L, i.e. $\sigma s =_E \sigma t$.

Finally we define the *complexity* of a list of equational goals G by $C(G,[])$. In words: each goal $l = r$ in G is assigned the maximal depth of l under instantiations resulting from a solution of all goals to the left; the overall complexity is the multiset of all these integers.

Some important facts about C are

1. $C(\sigma R, \sigma L) \leq C(R,L)$
2. If $\sigma \vdash_E H$ implies $\sigma \vdash_E H'$ for all σ, then $C(R,H) \leq C(R,H')$.

Each resolution step transforms the list of goals from $[l = r]@G$ to $H@G'$, where $G' = \theta G$, θ is the unifying substitution, and H is defined in (15) above. Since $C(H@G',[]) = C(H_l,[]) \sqcup C(H_r,H_l) \sqcup C(G',H)$ and $C([l = r]@G,[]) = C([l = r],[]) \sqcup C(G,[l = r])$, it suffices to show $C(G',H) \leq C(G,[l = r])$, $C(H_l,[]) < C([l = r],[])$, and $C(H_r,H_l) < C([l = r],[])$ in order to prove $C(H@G',[]) < C([l = r]@G,[])$.

$C(G',H) \leq C(G,[l = r])$ follows from the two facts about C above because the rules in D_{rk} guarantee that $\sigma \vdash_E H$ implies $\sigma l =_E \sigma r$.

To show $C(H_l,[]) < C([l = r],[])$ and $C(H_r,H_l) < C([l = r],[])$ we notice that $C([l = r],[]) = maxd_E(l) \neq 0$. Looking at the equations in H_l, we find that all l_i are proper subterms of l and hence that $maxd_E(\sigma l_i) = maxd_E(l_i) < maxd_E(l)$, which establishes $C(H_l,[]) < C([l = r],[])$. If H_r is nonempty, resolution must have taken place with a rule in D derived from an equation $s = t$ in E. Thus $\sigma \vdash_E H_l$ implies $l =_E \sigma s =_E \sigma t$. Hence σt_j is ground and a proper subterm of σt, and therefore $maxd_E(\sigma t_j) < maxd_E(\sigma t) = maxd_E(l)$. This proves $C(H_r,H_l) < C([l = r],[])$ and concludes the termination proof. □

Theorem 2 is important because it is the first time that a subclass of resolvant theories has been identified which yield *terminating* matching algorithms. On the other hand, there is a trivial terminating matching algorithm for permutative theories: in trying to match the pattern r to the variable free term s, enumerate the finite set of t's with $s =_E t$ and try to match r and t in the empty theory. It is beyond the scope of this paper to compare the time and space complexity of both algorithms.

We will now stop to think in terms of inference rules and confine our attention to equations.

4.1 A Generalized Criterion

Given a set of equations E we want to test whether E is resolvant. Our criterion actually shows how to go from an arbitrary derivation $s =_E t$ to one with at most one peak. The transformation is an inductive process which combines adjacent peaks.

If we write $e, e' \in E$ in the sequel, we really mean that there are two equations e and e_1 in E, such that $e' = \sigma e_1$, where σ is a renaming of the variables in e_1 away from those in e. We also assume that E is closed under symmetry, i.e. $s{=}t \in E$, implies $t{=}s \in E$.

Defining

$$p{=}q \Downarrow_E u{=}v \;=\; \forall\sigma.\; \sigma q \mathrel{\dot{=}}_E \sigma u \;\Rightarrow\; \sigma p \mathrel{\dot{=}}_E \sigma v$$

we get

Lemma 1 *E is resolvant iff $p{=}q \Downarrow_E u{=}v$ holds for all equations $p{=}q, u{=}v \in E$ where $q = f(\ldots)$ and $u = f(\ldots)$.*

Proof We concentrate on the "if"-part as the "only if"-part is trivial. For E to be resolvant, $s =_E t$ must imply $s \mathrel{\dot{=}}_E t$. We show that under the given assumptions any derivation $s =_E t$ can be reduced to $s \mathrel{\dot{=}}_E t$.

The reduction merges adjacent peaks. This means we translate a derivation $s_0 \doteq_E s_1 \dot{=}_E s_2 \doteq_E s_3$ into one of the form $s_0 \dot{=}_E s_3$. By induction the number of peaks in any derivation can always be reduced to 0 or 1.

A derivation $s_0 \doteq_E s_1 \dot{=}_E s_2 \doteq_E s_3$ must be of the form $\sigma p \doteq_E \sigma q \dot{=}_E \sigma u \doteq_E \sigma v$ for some $p{=}q, u{=}v \in E$ with $q = f(\ldots)$ and $u = f(\ldots)$. By assumption this implies $\sigma p \dot{=}_E \sigma v$, which is the required reduction. □

We will now concentrate on ways of turning the resolvance criterion embodied in this lemma into a finite test.

In the sequel let $\bar{s}$ denote the instantiation of every variable in s by a unique free constants. Thus $\bar{s}$ is equivalent to τs, where τ is a fixed injective substitution from variables to free constants. Therefore any first-order formula P holds iff σP holds for all σ.

Lemma 2 *Let $p{=}q, u{=}v \in E$ such that $q = f(q_1, \ldots, q_n)$, and $u = f(u_1, \ldots, u_n)$, and let $E' = E \cup \{\overline{q_1} = \overline{u_1}, \ldots, \overline{q_n} = \overline{u_n}\}$. Then $\bar{p} \dot{=}_{E'} \bar{v}$ implies $p{=}q \Downarrow_E u{=}v$.*

Proof In the following we make use of the fact that if $A \vdash P \Rightarrow Q$ then $A \vdash P$ implies $A \vdash Q$.

$$
\begin{array}{ll}
 & \bar{p} \dot{=}_{E'} \bar{v} \\
\Leftrightarrow & E \cup \{\overline{q_1} = \overline{u_1}, \ldots, \overline{q_n} = \overline{u_n}\} \vdash \bar{p} \doteq \bar{v} \\
\Leftrightarrow & E \vdash \overline{q_1} = \overline{u_1} \wedge \ldots \wedge \overline{q_n} = \overline{u_n} \Rightarrow \bar{p} \doteq \bar{v} \\
\Leftrightarrow & E \vdash \bar{q} = \bar{u} \Rightarrow \bar{p} \doteq \bar{v} \\
\Rightarrow & \bar{q} =_E \bar{u} \Rightarrow \bar{p} \dot{=}_E \bar{v} \\
\Leftrightarrow & \forall \sigma.\ \sigma q =_E \sigma u \Rightarrow \sigma p \dot{=}_E \sigma v \\
\Leftrightarrow & p{=}q \Downarrow_E u{=}v
\end{array}
$$

□

Example 1 In the presence of commutativity in E, we need to test $x{\cdot}y{=}y{\cdot}x \Downarrow_E u{\cdot}v{=}v{\cdot}u$. By Lemma 2 it suffices to show $\bar{x}{\cdot}\bar{y} \dot{=}_{E'} \bar{v}{\cdot}\bar{u}$ where $E' = E \cup \{\bar{y} = \bar{u}, \bar{x} = \bar{v}\}$: $\bar{x}{\cdot}\bar{y} \dot{=}_{E'} \bar{v}{\cdot}\bar{u}$.

If E contains $x{\cdot}y = y{\cdot}x$, $(x{\cdot}y){\cdot}z = x{\cdot}(y{\cdot}z)$, and $(x{\cdot}y){\cdot}z = (x{\cdot}z){\cdot}y$ we need to show $(x{\cdot}y){\cdot}z{=}x{\cdot}(y{\cdot}z) \Downarrow_E u{\cdot}v{=}v{\cdot}u$ and hence $(\bar{x}{\cdot}\bar{y}){\cdot}\bar{z} \dot{=}_{E'} \bar{v}{\cdot}\bar{u}$ where $E' = E \cup \{\bar{x} = \bar{u}, \bar{y}{\cdot}\bar{z} = \bar{v}\}$: $(\bar{x}{\cdot}\bar{y}){\cdot}\bar{z} \dot{=}_{E'} (\bar{y}{\cdot}\bar{x}){\cdot}\bar{z} \doteq_{E'} (\bar{y}{\cdot}\bar{z}){\cdot}\bar{x} \dot{=}_{E'} \bar{v}{\cdot}\bar{u}$

However, Lemma 2 is not necessary for resolvance:

Example 2 Right-commutativity is resolvant and $(x{\cdot}y){\cdot}z{=}(x{\cdot}z){\cdot}y \Downarrow_{C_r} (u{\cdot}v){\cdot}w{=}(u{\cdot}w){\cdot}v$ holds although Lemma 2 is not applicable: $(\bar{x}{\cdot}\bar{y}){\cdot}\bar{z} \dot{=}_{E'} (\bar{u}{\cdot}\bar{w}){\cdot}\bar{v}$, where $E' = C_r \cup \{\bar{x}{\cdot}\bar{z} = \bar{u}{\cdot}\bar{v}, \bar{y} = \bar{w}\}$, does not hold.

The problem stems from the fact that in E' we only assume that $\overline{q_i} = \overline{r_i}$, without any further distinctions. We have to take into account what the derivation of $q_i =_E r_i$ looks like. For right-commutativity it is sufficient to take a closer look at $x{\cdot}z =_E u{\cdot}v$. We can assume that this derivation has at most one peak.

If $x{\cdot}z \dot{=}_E u{\cdot}v$, we have $E'_1 = E' \cup \{\bar{x} = \bar{u}, \bar{z} = \bar{v}\}$ and therefore $(\bar{x}{\cdot}\bar{y}){\cdot}\bar{z} \dot{=}_{E'_1} (\bar{u}{\cdot}\bar{w}){\cdot}\bar{v}$.

Otherwise there are terms r,s,t such that $x{\cdot}z \dot{=}_E (r{\cdot}s){\cdot}t \doteq_E (r{\cdot}t){\cdot}s \dot{=}_E u{\cdot}v$, which gives rise to $E'_2 = E' \cup \{\bar{x} = \bar{r}{\cdot}\bar{s}, \bar{z} = \bar{t}, \bar{r}{\cdot}\bar{t} = \bar{u}, \bar{s} = \bar{v}\}$. Therefore $(\bar{x}{\cdot}\bar{y}){\cdot}\bar{z} \dot{=}_{E'_2} ((\bar{r}{\cdot}\bar{w}){\cdot}\bar{s}){\cdot}\bar{t} \doteq_{E'_2} ((\bar{r}{\cdot}\bar{w}){\cdot}\bar{t}){\cdot}\bar{s} \dot{=}_{E'_2} (\bar{u}{\cdot}\bar{w}){\cdot}\bar{v}$.

Thus we have shown that $(\bar{x}{\cdot}\bar{y}){\cdot}\bar{z} \dot{=}_{E'_i} (\bar{u}{\cdot}\bar{w}){\cdot}\bar{v}$ holds for $i = 1, 2$.

The case distinction of the previous example can be formalized as follows:

$$
\begin{array}{rcl}
C_E(f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)) & = & \{\{s_1 = t_1, \ldots, s_n = t_n\}\} \cup \\
 & & \{\{s_1 = u_1, \ldots, v_1 = t_1, \ldots\} \mid f(u_1, \ldots, u_n){=}f(v_1, \ldots, v_n) \in E\} \\
C_E(f(s_1, \ldots, s_m) = g(t_1, \ldots, t_n)) & = & \{\{s_1 = u_1, \ldots, v_1 = t_1, \ldots\} \mid f(u_1, \ldots, u_m){=}g(v_1, \ldots, v_n) \in E\} \\
C_E(x = y) & = & \{\{x = y\}\}
\end{array}
$$

The three clauses that define C_E are ordered in the sense of ML or Prolog. For example the last one, which is the default case, applies only if the first two do not. The next two facts show that the definition

of $C_E(s = t)$ returns exactly those equalities between subterms that can arise if $s' \doteqdot_E t'$ for some instances s' and t' of s and t respectively.

$$C \in C_E(s = t) \Rightarrow \bar{s} \doteqdot_{E \cup \bar{C}} \bar{t} \quad (16)$$

$$\sigma s \doteqdot_E \sigma t \Rightarrow \exists C \in C_E(s = t). \forall p{=}q \in C.\ \sigma p =_E \sigma q \quad (17)$$

With these facts we can prove the following lemma which formalizes the procedure outlined in Example 2.

Lemma 3 *If $\bar{p} \doteqdot_{E \cup \bar{C}} \bar{v}$ holds for all $p{=}q, u{=}v \in E$ such that $q = f(q_1, \ldots, q_n)$ and $u = f(u_1, \ldots, u_n)$, and all $C \in M = \{E_1 \cup \ldots \cup E_n \mid E_i \in C_E(q_i = u_i)\}$, then $p{=}q \Downarrow_E u{=}v$ holds for all $p{=}q, u{=}v \in E$, i.e. E is resolvant.*

Proof The proof is similar to the one of Lemma 1 in that it proceeds by reducing peaks. However, we are more specific about the order in which adjacent peaks are removed from a derivation and its subderivations: we reduced a derivation $\sigma p \doteq_E \sigma q \mathrel{\dot{=}}_E \sigma u \doteq_E \sigma v$, where $p{=}q, u{=}v \in E$, $q = f(q_1, \ldots, q_n)$ and $u = f(u_1, \ldots, u_n)$, only if all subderivations $\sigma q_i =_E \sigma u_i$ are normalized, i.e. of the form $\sigma q_i \doteqdot_E \sigma u_i$. In that case we know from (17) that there is a $C_i \in C_E(q_i = u_i)$ such that $\sigma s =_E \sigma t$ holds for all $s{=}t \in C_i$. Letting $C = C_1 \cup \ldots \cup C_n \in M$ we obtain

$$\begin{aligned} \bar{p} \doteqdot_{E \cup \bar{C}} \bar{v} &\Leftrightarrow E \vdash (\forall i, s{=}t \in C_i.\ \bar{s} = \bar{t}) \Rightarrow \bar{p} \doteq \bar{v} \\ &\Rightarrow (\forall i, s{=}t \in C_i.\ \bar{s} =_E \bar{t}) \Rightarrow \bar{p} \doteqdot_E \bar{v} \\ &\Rightarrow (\forall i, s{=}t \in C_i.\ \sigma s =_E \sigma t) \Rightarrow \sigma p \doteqdot_E \sigma v \\ &\Leftrightarrow \sigma p \doteqdot_E \sigma v \end{aligned}$$

the required reduction. □

This concludes the exposition of the theoretical basis for our resolvance checker.

4.2 Automating It

The procedures suggested by Lemmas 2 and 3 lead to a fairly large number of cases that have to be examined. The principal problem with automating this procedure is the undecidability of the concepts involved, i.e. the predicates $\mathrel{\dot{=}}_E$ etc. As remarked above, these predicates, and hence Lemmas 2 and 3, become decidable for permutative E. However, in practice it turns out that the resulting search space is far too large for a naive enumeration procedure.

A prototype system for automating these tests has been implemented in Prolog. To overcome the problems just mentioned, a number of heuristics for testing $\doteqdot_E$ were implemented. If these turn out to be insufficient (as they have for a number of equational theories) the predicate $\doteqdot_E$ can be implemented separately for each equational theory.

4.3 More Theories

Using the implementation described in the previous section, the following list of theories was shown to be resolvant:

$$\begin{aligned} AC1^+ &= \{(x{\cdot}y){\cdot}(u{\cdot}v) = (x{\cdot}u){\cdot}(y{\cdot}v), 1{\cdot}x = x, x{\cdot}1 = x\} \\ D &= \{x{\cdot}(y+z) = x{\cdot}y + x{\cdot}z\} \\ I &= \{x{\cdot}x = x\} \\ CI &= C \cup I \\ DC &= D \cup \{x + y = y + x\} \\ DC_r &= D \cup \{(x+y)+z = (x+z)+y\} \\ DA &= D \cup \{(x+y)+z = x+(y+z)\} \\ C_S &= \{(x{\cdot}x){\cdot}y = y{\cdot}(x{\cdot}x)\} \\ C_P &= \{(x{\cdot}y){\cdot}z = z{\cdot}(x{\cdot}y)\} \end{aligned}$$

I and CI yield terminating unification algorithms, the rest only terminating matching algorithms. For D this has already been exploited in [11].

We also conjecture very strongly that $ACI^+ = AC^+ \cup I$ and $ACI1^+ = AC1^+ \cup I$ are resolvant theories. However, the methods of this paper seem inadequate to prove it.

4.4 Combining Resolvant Theories

In this section we investigate the combination of resolvant theories. We are given a collection of equational presentations E_i, $i \in I$, over pairwise disjoint signatures Σ_i. Let $E = \bigcup_{i \in I} E_i$ and $\Sigma = \bigcup_{i \in I} \Sigma_i$. Adapting the terminology of [10] we call a property P of equational presentations *modular* if E has property P iff all E_i have property P.

As an immediate consequence of Lemma 1 we obtain that

Lemma 4 *Resolvance is modular.*

Proof Assume that all E_i are resolvant. Given two equations $p{=}q, u{=}v \in E$ such that $q = f(\ldots)$ and $u = f(\ldots)$, disjointness of the Σ_i implies that both equations must come from the same E_k. Hence $p{=}q \Downarrow_{E_k} u{=}v$ and thus $p{=}q \Downarrow_E u{=}v$ must hold, which implies resolvance of E.

Now assume that E is resolvant and let s, t be two terms over Σ_k with $s =_{E_k} t$ and thus in particular $s =_E t$. Resolvance of E implies $s \doteq_E t$. Using disjointness of the signatures and Fact 1 in [12] $s \doteq_{E_k} t$ follows. Thus E_k is resolvant. □

This lemma does for the combination of resolvant theories what [15,14,12] do for the combination of arbitrary unification or matching algorithms. The simplicity of its proof is partly due to the fact that it says nothing about termination: even if each E_i yields a terminating unification or matching algorithm, E might not. For permutative theories however we are able to show that termination carries over. The reason is that

Lemma 5 *Permutativity is modular.*

Proof This proof relies heavily on notions defined in [12] which appear in quotation marks.

Assume that all E_i are permutative. The proof is by induction on the "theory height" of terms over Σ, i.e. the maximum number of alternations of signature along some path in a mixed term considered as a DAG. Any term s over Σ can be written as $C[s_1, \ldots, s_m]$, where C is a proper "context" over some Σ_k and the s_i are the "immediate alien subterms". By induction hypothesis each s_i has a finite E-equivalence class. Since E is both regular and collapse-free, Fact 2 in [12] shows that any term t with $s =_E t$ is of the form $D[t_1, \ldots, t_n]$ such that each t_j is E-equivalent to some s_i (and vice versa), and $C[[s_1]_{=_E}, \ldots] =_{E_k} D[[t_1]_{=_E}, \ldots]$. Thus there are only finitely many different t_j and D, and therefore only a finite number of t with $s =_E t$, i.e. E is also permutative.

The reverse implication is trivial. □

Thus we can safely combine permutative resolvant theories to obtain a matching algorithm for the joint theory.

5 Related Work

As was mentioned earlier on, this paper is strongly related to the work of Claude Kirchner, who coined the term "resolvant" [8]. He also suggested some criteria for checking resolvance and a completion procedure which turns a non-resolvant theory into a resolvant one. However, for most equational theories discussed above his criteria are too weak to determine that they are resolvant. This prompted the development of the improved tests in this paper. Recently, Claude Kirchner and Francis Klay have given a nice and simple characterization of resolvant theories:

Theorem 3 ([9]) *If E is a collapse-free set of equations over the signature Σ, the set*

$$\{\sigma f(x_1, \ldots) = \sigma g(y_1, \ldots) \mid f, g \in \Sigma, \sigma \in cU_E(f(x_1, \ldots) = g(y_1, \ldots))\}$$

is a resolvant set of axioms which generates the same equational theory as E. (The x_i and y_j are all distinct and cU_E denotes a complete set of E-unifiers).

Thus any equational theory with a finitary unification problem has a finite resolvant presentation. This presentation can be computed by means of a unification algorithm for the theory.

Although this theorem yields very short proofs for the resolvance of the theories in Sections 3.1 to 3.5, it is not easy to apply to some of the theories in Section 4.3. For example D has a fairly involved unification algorithm (see [1]), and the author is not aware of a published unification algorithm for DC

or DA. The problem is that a notion that was conceived to explain and automate the generation of unification algorithms, namely resolvance, is itself reduced to unification.

If a complete unification algorithm for some theory E is not known, it may still be possible to compute a resolvant presentation by combining Theorem 3 with the results of this paper: instead of computing a complete set of unifiers, enumerate all unifiers one by one and keep on testing whether the resulting presentation is yet resolvant. Of course this procedure may still not succeed because we can reach a resolvant presentation without realizing it, due to the incompleteness of the test we described.

Acknowledgements

Part of this research was carried out at the Laboratory for Computer Science at MIT. I would like to express my gratitude to both John Guttag and Larry Paulson for giving me the freedom to pursue this line of research. Larry Paulson also helped to improve the presentation.

References

[1] S. Arnborg, E. Tidén: *Unification Problems with One-Sided Distributivity*, Proc. 1st Conf. Rewriting Techniques and Applications, LNCS 202 (1985), 398-406.

[2] H. Bertling, H. Ganzinger, R. Schäfer: *CEC: A System for Conditional Equational Completion*, PROSPECTRA-Report M.1.3-R-7.0, Universität Dortmund (1988).

[3] N. Dershowitz, Z. Manna: *Proving Termination with Multiset Orderings*, CACM 22 (1979), 465-476.

[4] H. Ganzinger, Private system demonstration, June 1989.

[5] S.J. Garland, J.V. Guttag: *An Overview of LP, The Larch Prover*, in: Proc. 3rd Intl. Conf. Rewriting Techniques and Applications, LNCS 355 (1989), 137-151.

[6] G. Huet, D.C. Oppen: *Equations and Rewrite Rules - A Survey*, in: Formal Languages: Perspectives and Open Problems, R. Book (ed.), Academic Press (1982).

[7] C. Kirchner: *Méthodes et outils de conception systématique d'algorithmes d'unification dans les théories équationnelles*, Thèse d'état de l'Université de Nancy I (1985).

[8] C. Kirchner: *Computing Unification Algorithms*, Proc. Symp. on Logic in Computer Science (1986), Cambridge, MA, 206-216.

[9] C. Kirchner, F. Klay: *A Note on Syntacticness*, This Volume.

[10] A. Middeldorp: *Modular Aspect of Properties of Term Rewriting Systems Related to Normal Forms*, Proc. 3rd Intl. Conf. Rewriting Techniques and Applications, LNCS 355 (1989), 263-277.

[11] J. Mzali: *Matching with Distributivity*, Proc. 8th Intl. Conf. on Automated Deduction, LNCS 230 (1986), 496-505.

[12] T. Nipkow: *Combining Matching Algorithms: The Regular Case*, Proc. 3rd Intl. Conf. Rewriting Techniques and Applications, LNCS 355 (1989), 343-358.

[13] G.D. Plotkin: *Building-in Equational Theories*, in: Machine Intelligence, Vol. 7, Halsted Press (1972), 73-90.

[14] E. Tidén: *Unification in Combinations of Collapse-Free Theories with Disjoint Sets of Function Symbols*, in: Proc. CADE-8, LNCS 230 (1986), 431-449.

[15] K. Yelick: *Unification in Combinations of Collapse-Free Regular Theories*, JSC 3 (1987), 153-181.

A new Combination Technique for AC Unification

Alexandre Boudet
LRI, Université Paris-Sud, Bât 490 91405 ORSAY Cedex, France

1 The Rules

The following set of transformations allow to solve equations in a combination of regular collapse-free theories (like AC theories) provided a complete unification algorithm is known for each theory.

Definition 1 *A unification problem P is of the form:*

$$P = P_V \cup P_H \cup P_0 \cup P_1 \cup P_2 \cup \cdots \cup P_n$$

where
P_V is the subproblem of non proper equations $x \stackrel{?}{=} y$ in P.
P_H is the subproblem of heterogeneous equations $s \stackrel{?}{=} t$ in P.
P_i is the subproblem of proper equations $s \stackrel{?}{=} t$ in P that are pure in E_i.
$V(P)$ denotes the set of the variables occurring in P.

Definition 2
A unification problem P is in solved form *if it is of the form: $\{x_1 \stackrel{?}{=} t_1, \ldots, x_n \stackrel{?}{=} t_n\}$ where $x_i \in X$ and x_i occurs nowhere else in P for $i \in [1..n]$.*

VA
$s \stackrel{?}{=} t \quad \vdash \quad C[x_1, \ldots, x_n] \stackrel{?}{=} t, x_1 \stackrel{?}{=} s_1, \ldots, x_n \stackrel{?}{=} s_n$
if $C[x_1, \ldots, x_n]$ is a maximal pure term such that $s = C[x_1, \ldots, x_n]\{x_1 \mapsto s_1, \ldots, x_n \mapsto s_n\}$.

E-Res
$P_i \quad \vdash \quad \{x_1 \stackrel{?}{=} t_1, \ldots, x_n \stackrel{?}{=} t_n\}$
if P_i is a pure subsystem in a theory E_i and P_i is not in a solved form,
and $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ is a most general E_i-unifier of P_i away from $V(P)$.

Clash
$s \stackrel{?}{=} t \quad \vdash$ fail
if the top function symbols of s and t are constrained by two different theories.

Merge
$x \stackrel{?}{=} s, x \stackrel{?}{=} t \quad \vdash$ fail
if the top function symbols of s and t are constrained by two different theories.

Var-Rep
$\{x \stackrel{?}{=} y\} \cup P \quad \vdash \quad \{x \stackrel{?}{=} y\} \cup P\{x \mapsto y\}$

if both x and y occur in P

Definition 3
A compound cycle *is a cycle* $x_1 \stackrel{?}{=} s_1, x_1 \stackrel{?}{=} s_2, \ldots, x_n \stackrel{?}{=} s_n$ *where* $x_i \in V(s_{i-1})$ *for* $i \in [2..n]$ *and* $x_1 \in V(s_{2n})$ *and two of* $s_1, \ldots, s_n$ *are non-variable terms pure in different theories.*

Combined Occur-Check
$P \vdash$ fail
if P contains a compound cycle.

Rep
$\{x \stackrel{?}{=} s\} \cup P \vdash \{x \stackrel{?}{=} s\} \cup P\{x \mapsto s\}$
if x occurs in P.

Checking that that x does not occur in P is not necessary here, because of the particular control described below.

2 Termination

Proposition 1 *If the set*
$S =$ {**VA**, *E*-**Res**, **Var-Rep**, **Clash**, **Merge**, **Combined Occur-check**, **Rep**} *terminates for some input* P, *then the normal form of* P *is either "fail", or an equivalent problem in solved form.*

We restrict our attention to the following class of controls:

1. **Apply as long as possible the rules in** $S' = S \setminus \{\textbf{Rep}\}$.
2. **Apply Rep as long as possible.**

Definition 4 *A unification problem* P *is* separated *if it is of the form*

$$P = P_V \cup P_0 \cup P_1 \cup P_2 \cup \cdots \cup P_n$$

where

- P_i *is solved for* $i \in [0..n]$ *and*
- $\nexists x \stackrel{?}{=} s, x \stackrel{?}{=} t$ *in two different subproblems with* $s \notin X$ *and* $t \notin X$
- *If* $x \stackrel{?}{=} y \in P_V$, *then one of* x *and* y *occurs nowhere else in* P.

A separated problem is either "fail", or a solved form, or a problem to which only Term Replacement may apply. If Term Replacement is removed from S, then normal forms are separated problems.

To prove the termination of S', we use the following measures:

Definition 5 $TH_{mul}(P)$ *denotes the multiset of* theory heights *of the heterogeneous terms in* P, *where the theory height of a term* t *is the maximal number of times the theory constraining the function symbols changes in a path of* t.

Lemma 1 **VA** *decreases strictly* $TH_{mul}(P)$. *No rule of* S' *increases* $TH_{mul}(P)$.

Shared variables are crucial in our termination proof. Intuitively, x and y are shared variables whenever **Var-Rep**, applied to the equation $x \stackrel{?}{=} y$ yields a subproblem P_i unsolved:

Definition 6

- $x \sim_{P_i} y$ *if both* x *and* y *occur in* P_i.
- *Let* $P = P_V \cup P_H \cup P_0 \cup P_1 \cup \cdots \cup P_n$. *The variables* x *and* y *are* identifiable outside P_i, *denoted* $x \approx_i^P y$ *if* (x, y) *is in the transitive closure of the relation*

$$=_V \cup \sim_{P_0} \cup \sim_{P_1} \cup \cdots \cup \sim_{P_{i-1}} \cup \sim_{P_{i+1}} \cup \cdots \cup \sim_{P_n}$$

where $=_V$ *is the equivalence on variables generated by the equations in* P_V.

When x and y are identifiable outside P_i, it is possible that some applications of E-**Res** to problems other than P_i and some applications of **Var-Rep** yield the equation $x \stackrel{?}{=} y$. For example, E_j-**Res** may yield the equations $x \stackrel{?}{=} x'$ and $y \stackrel{?}{=} x'$, and **Var-Rep** generate the equation $x \stackrel{?}{=} y$. Hence, the following definition:

Definition 7 *A variable* x *is* shared *in* P_i *if* $x \in V(P_i)$ *and* $x \approx_i^P y$ *for some variable* $y \in V(P_i)$, $y \neq x$. $SV(P)$ *is the multiset* $\{m_0, m_1, \ldots, m_n\}$ *where* m_i *is the number of variables shared in* P_i.

Here is an example to illustrate the definition. Assume the original unification problem is:

P_V	P_0	P_1
$x \stackrel{?}{=} y$	$x \stackrel{?}{=} f(u)$	$u \stackrel{?}{=} u_1 + u_2$
	$y \stackrel{?}{=} f(v)$	$v \stackrel{?}{=} v_1 + v_2$
	$z \stackrel{?}{=} a$	

In this problem x and y are shared in P_0, u and v are shared in P_0 and P_1, hence $SV(P) = \{4, 2\}$. **Var-Rep** applies to $x \stackrel{?}{=} y$ and yields:

P_V	P_0	P_1
$x \stackrel{?}{=} y$	$x \stackrel{?}{=} f(u)$	$u \stackrel{?}{=} u_1 + u_2$
	$x \stackrel{?}{=} f(v)$	$v \stackrel{?}{=} v_1 + v_2$
	$z \stackrel{?}{=} a$	

Note that P_0 is not solved anymore and that x and y are not shared anymore in P_0. We now have $SV(P) = \{2, 2\}$, the application of **Var-Rep** has decreased $SV(P)$. E-**Res** applies to P_0 and replaces its two first equations by the equations $u \stackrel{?}{=} u'$, $v \stackrel{?}{=} u'$. The obtained problem is:

P_V	P_0	P_1
$x \stackrel{?}{=} y$	$z \stackrel{?}{=} a$	$u \stackrel{?}{=} u_1 + u_2$
$u \stackrel{?}{=} u'$		$v \stackrel{?}{=} v_1 + v_2$
$v \stackrel{?}{=} u'$		

The variables u and v are still shared in P_1, but no longer in P_0. Now $SV(P) = \{0, 2\}$. **Var-Rep** applied to $v \stackrel{?}{=} u'$ replaces u' by v and yields:

P_V	P_0	P_1
$x \stackrel{?}{=} y$	$z \stackrel{?}{=} a$	$u \stackrel{?}{=} u_1 +_1 u_2$
$u \stackrel{?}{=} u'$		$v \stackrel{?}{=} v_1 +_1 v_2$
$v \stackrel{?}{=} u$		

This does not change the shared variables. **Var-Rep** applied to $v \stackrel{?}{=} u$ yields:

P_V	P_0	P_1
$x \stackrel{?}{=} y$	$z \stackrel{?}{=} a$	$u \stackrel{?}{=} u_1 + u_2$
$u \stackrel{?}{=} v$		$u \stackrel{?}{=} v_1 + v_2$

Again, an application of **Var-Rep** has made a subproblem unsolved, and as before, it has removed two shared variables. There are no shared variables anymore in P_0 or P_1. This is a general property as shown below.

Lemma 2
E-**Res** *does not increase* $SV(P)$.

proof :
Assume E_i-**Res** is applied to

$$P = P_V \cup P_H \cup P_0 \cup P_1 \cup \cdots \cup P_i \cup \cdots \cup P_n$$

yielding

$$P' = P'_V \cup P_H \cup P_0 \cup P_1 \cup \cdots \cup P'_i \cup \cdots \cup P_n$$

Note first that E_i res does not change $\sim_{P_j}$ for $j \neq i$. We can assume that E_i-**Res** first replaces P_i by a solved form, then moves the non proper equations from P'_i to P_i. The proof proceeds by induction on the number of fresh variables occurring in P'_V. The new equations occurring in P'_V must be of the form $x \stackrel{?}{=} x'$ where x is a fresh variable, by definition of *E*-**Res**.
Basic case: $P'_V = P_V$. The variables shared in P'_i were already shared in P_i (fresh variables added to P'_i cannot be shared here since they only occur in P'_i).
Inductive case: Assume some equations $x_1 \stackrel{?}{=} x', \ldots, x_p \stackrel{?}{=} x'$ are added to P_V while $x_1, \ldots, x_p$ are removed from P_i. We proceed by introspecting all the variables shared in P'.

1. x' is shared in P'_i. By definition, there exists a variable $z \in P'_i$ such that $x' \approx_i^{P'} z$. Let $x' =_V x \approx_i^{P'} z$ be a shortest proof, forbidding x' to occur in $x \approx_i^{P'} z$. Hence $x \approx_i^{P} z$. By assumption $x \in V(P_i)$ as well as z. Hence x was shared in P_i and does not occur anymore in P'_i, hence is no more shared in P_i. Note that x is associated to x'.

2. Let y be a variable of P_i shared in P'_i. We show that y is already shared in P_i. Assume $y \approx_i^{P'} z$ where $z \in V(P'_i)$. By singling out the first application of a new equation $x \stackrel{?}{=} x' \in P'_V$ (if none $y \approx_i^{P} z$ and y is shared in P_i), the proof must be of the form:

$$y \approx_i^{P} x =_V x' \approx_i^{P'} z$$

   Note that $y \neq x$ since x does not occur anymore in P'_i, hence y is shared in P_i.

3. Let y be a variable shared in P'_j for $j \neq i$. Then $y \approx_j^{P'} z$ for some $z \in V(P_j)$. Again, by singling out every use of step in $\approx_j^{P'} \setminus \approx_j^{P}$ in a proof we get:

$$y \approx_j^{P} x_1 \, (=_{V'} \cup \sim_{P'_i}) \, x' \, (=_{V'} \cup \sim_{P'_i}) \, x_2 \approx_j^{P} \cdots \approx_j^{P} z$$

   because x' is a fresh variable occurring only in P'_V and P'_i. x_1 and x_2 were both in P_i, hence $x_1 \approx_j^{P} x_2$ and y was already shared in P_i.

□

Lemma 3
Var-Rep *does not increase* $SV(P)$.

proof : Assume **Var-Rep** applies to the equation $x \stackrel{?}{=} y$ in P_V and replaces x by y. Note first that x cannot be shared anymmore because it occurs only in $P_{V'}$. For the other variables, it is enough to note that any proof $z \approx_i^{P'} v$ can be mapped back to a proof in P by replacing appropriate occurrences of y by occurrences of x. □

Definition 8 $USP(P)$ *is the number of unsolved subproblems in* P.

Lemma 4 *If* **Var-Rep** *increases* $USP(P)$, *then it decreases strictly* $SV(P)$.

proof : For **Var-Rep** to make a subproblem P_i unsolved, it must have replaced a variable x by a variable y, both occurring in P_i. Hence x and y were shared in P_i because of the equation $x \stackrel{?}{=} y$ in P_V, and x has disappeared from P_i, hence is no more shared. □

Definition 9 $PVR(P)$ *denotes the number of potential applications of Variable Replacement. A potential application of Variable Replacement is an equation* $x \stackrel{?}{=} y \in P$ *such that both* x *and* y *occur elsewhere in* P.

The *weight* of a unification problem is $W(P) =< TH_{mul}(P), SV(P), USP(P), PVR(P) >$. Let $<_W$ be the ordering on weihgts obtained by comparing lexicographically its components from left to right, using the multiset extension of $<$ for the two first, and $<$ for the two others, where $<$ denotes the ordinary ordering on natural numbers. The ordering $<_W$ is well founded, as a lexicographic extension of well founded orderings.

Theorem 1 S' *terminates and yields either a separated problem with no cycle in the graph of the occur-check relation, or "fail".*

proof : First note that if a problem is not separated, then some rule of S' applies necessarily. If there is a cycle in the occur-check graph, then E-**Res** or **Combined Occur-Check** apply. If one of the subproblems is not solved, then E-**Res** applies. If a variable x appears in two equations $x \stackrel{?}{=} s$ and $x \stackrel{?}{=} t$, then one of **Var-Rep**, E-**Res**, **Merge** or **Clash** applies.
The termination proof is summarized in the following table:

	$TH_{mul}(P)$	$SV(P)$	$USP(P)$	$PVR(P)$
VA	$\downarrow$			
E-**Res**	$=$	$=\downarrow$	$\downarrow$	
Var-Rep (1)	$=$	$=\downarrow$	$=$	$\downarrow$
Var-Rep (2)	$=$	$\downarrow$	$\uparrow$	$\downarrow$

There is no problem with the rules that return "fail".
Lemma 1 shows that **VA** decreases $W(P)$ and that no rule increases $TH_{mul}(P)$. This proves the first column of the table.
Lemmas 2 and 3 show that no rule except **VA** increase $SV(P)$.
E-**Res** decreases $USP(P)$.
Var-Rep decreases $PVR(P)$. If an application of **Var-Rep** does not increase $USP(P)$, then it decreases $W(P)$ (case 1). Otherwise, lemma 4 shows that every time **Var-Rep** increases $USP(P)$, it decreases strictly $SV(P)$ (case 2). □

3 A Bound on the Number of E-Resolutions

Lemma 4 shows that a shared variable must disappear every time a subproblem is made unsolved. In the worst case, E-**Res** is applied to all subproblems and then some applications of **Var-Rep** make them all unsolved. in the worst case, this can happen as many times as the number of shared variables minus one. Hence the number of calls to restricted AC unification (unification for one AC symbol) is bounded by the number of different AC symbols times the number of shared variables minus one. This is so because only Variable Replacement may make a solved subproblem unsolved by identifying two shared variables.
One may think this is nonsense since the definition of shared variables depends on the state of the system. Remember though that only Variable Abstraction can increase $SV(P)$, hence it is enough to apply **VA** as long as possible (which is done in linear time) for knowing the total number of shared variables.

Theorem 2 *The procedure:*

1. *Apply as long as possible the rules in S'.*
2. *Apply Term Replacement as long as possible.*

terminates for any input P and computes an equivalent problem in solved form.

This applies to AC unification, provided a complete unification algorithm is known for solving equations in $T(\{+\}, X)$, modulo the associativity and commutativity of $+$.

References

[Bou Jou Sch 88] Boudet, A., Jouannaud, J.-P., and Schmidt-Schauß, M. "Unification in Free Extensions of Boolean Rings and Abelian Groups " *Proc. 3rd IEEE Symp. on Logic in Computer Science (Edimburgh 1988). (Also to appear in JSC, Special issue on Unification).*

[Fages 84] Fages, F. "Associative Commutative Unification " *Proc. 7th Int. Conf. on Automated Deduction. Springer-Verlag (1984).*

[Kirchner 85] Kirchner, C. "Méthodes et outils de conception systématique d'algorithmes d'unification dans les théories équationnelles " *Thèse de Doctorat d'Etat en Informatique. Université de Nancy I (1985).*

[Kirchner 87] Kirchner, C. "From Unification in a Combination of Equational Theories to a new AC-unification algorithm " *Proc. CREAS, Austin, to appear.*

[Lin Chris 88] Lincoln, P. and Christian ,J. "Adventures in Associative-Commutative Unification (A Summary) " *In Proc. 9th Int. Conf. on Automated Deduction, pp.358-367, Springer-Verlag (1988).*

[Mar Mon 82] Martelli, A. and Montanari, U. "An Efficient Unification Algorithm " *ACM Transactions On Programming Languages And Systems, vol.4, n 2*

[Sch 88] Schmidt-Schauß, M. "Unification in a Combination of Arbitrary Disjoint Equational Theories " *In Proc. 9th Int. Conf. on Automated Deduction, pp.378-396, Springer-Verlag (1988).*

[Siekmann 78] Siekmann G. "Unification and Matching Problems " *Memo CSM-4-78. University of Essex (1978).*

[Stickel 81] Stickel, M.E. "A complete unification Algorithm for Associative-Commutative functions " *Journal of the ACM, vol.28, pp423-434*

[Tidén 86] Tidén, E. "Unification in combinations of collapse-free theories with disjoint sets of function symbols " *Proc. 8th Int. Conf. on Automated deduction. Springer-Verlag (1986).*

[Yelick 85] Yelick, K. "Combining unification algorithms for confined equational theories " *Proc. First Int. Conf. on Rewriting Techniques and Applications. Springer-Verlag (1985).*

Optimizations of Schmidt-Schauß' General Combination Procedure

M. Tepp, Univ. Kaiserslautern, 6750 Kaiserslautern, FR Germany

In this abstract we want to present a modification of the algorithm of [Schmidt-Schauß] for unification in the combination of arbitrary disjoint equational theories. This algorithm takes as input a system of multiequations, an equational theory E and a partition of this theory into disjoint subtheories for which already special unification procedures exist. These special procedures are used to compute the output of the algorithm which consists of a complete set of solutions for the system of equations modulo E. In the following we refer to the special subtheories of the input simply as 'theories'. Computation proceeds in six non deterministic steps, where each of them is performed exactly once. The steps are:

1. UNFOLDING: Replace subterms of terms in the multiequations by new variables and introduce a new equation for each of the replaced terms containing the term and the replacing variable such that all terms in the resulting system of multiequations belong to one of the special theories and no two terms that are contained in different theories have some variable in common.
2. UNIFICATION AND IDENTIFICATION: Use the special procedures to unify all terms belonging to the same theory in each multiequation of the system. Variables are viewed to belong to the theory of those terms they occur in (this is unique). Choose one partition of the set of multiequations and identify all equations that are in the same set of the partition. Then again unify all terms of same theory in the newly created equations. That identification is done in order to be able to assume in further processing that existing multiequations are inequal to each other.
3. THEORY LABELING: For each multiequation choose one theory as a label.
4. COLLAPSING: For each multiequation choose one new variable x. This new variable is regarded as a constant in all terms that do not belong to the theory E_i the equation is labelled with. After that each term of the multiequation that is not contained in E_i is matched agaist the variable x (this can be done using the special unification procedures). So x is a variable that, in contrast to others, can occur in terms of different theory. But only in terms of theory E_i it is indeed treated as a variable but everywhere else as a constant (note: this can only be done because we may regard different multiequations to be inequal). In this way we still preserve some kind of variable disjointness for different theories. After this step each equation contains exactly one variable and one term.
5. CHOOSE CONSTANT ELIMINATION PROBLEM: Choose one set of pairs (t,x) where t is a term in a multiequation and x is a variable that is regarded as a constant in terms belomging to the same theory as t.
6. SOLVE CONSTANT ELIMINATION PROBLEM: Let $\{(t_1,x_1),\ldots,(t_n,x_n)\}$ be the set chosen in the preceding step. Use the special procedures to compute a substitution σ such that x_i is not contained in $\sigma(t_i)$ for all i and apply it to the system.

This is the original algorithm. Now our intention in creating an optimized version was based on three things:

1. We wanted to loosen the constraint of the original procedure to work only with disjoint equational theories. If we already know algorithms for combinations of some of the disjoint theories we do not always want to unfold terms that are contained in one of those combinations. Assume for example we had a procedure for the combination of free constants with semigroups and abelian semigroups. Further on let there exist a program for the combination of free constants, semigroups and boolean rings. Now consider the following example where *a* is a free constant, *g* belongs to an abelian semigroup, *h* is the function symbol of a semigroup and * is the 'and' of a boolean ring (*v,w,x,y,z* are variables):

$$g(x,h(z,v),a) = g(y,y,y,y)$$
$$u*h(v,z) = a*w$$

In this example it is not necessary at all to unfold the terms into disjoint theories, since we can solve it only by unification with our special programs. On the contrary, if we would use the original procedure we had to transform the above system of equations by unfolding and get:

$$g(x,v_1,v_2) = g(y,y,y,y)$$
$$u*v_3 = v_4*w$$
$$v_1 = h(z,v)$$
$$v_2 = v_4 = a$$
$$v_3 = h(v,z)$$

Now the problem $g(x,v_1,v_2) = g(y,y,y,y)$ gives us 32677 (see [Bürckert et al.]) minimal unifiers that cause as much new systems of equations still to solve.

So our first intention was to reduce unfolding.

2. Our second reason was to avoid unnecessary identification of terms. Let us consider the following example:

$$f_1(v_{11},v_{12}) = f_2(v_{21},v_{22})$$
$$f_3(v_{31},v_{33}) = f_4(v_{41},v_{42})$$
$$f_5(v_{51},v_{52}) = f_6(v_{61},v_{62})$$

In step 2 of the algorithm we get five alternatives for this system. Besides the above one itself there are these four:

$$f_1(v_{11},v_{12}) = f_2(v_{21},v_{22}) = f_3(v_{31},v_{33}) = f_4(v_{41},v_{42})$$
$$f_5(v_{51},v_{52}) = f_6(v_{61},v_{62})$$

$$f_1(v_{11},v_{12}) = f_2(v_{21},v_{22})$$
$$f_3(v_{31},v_{33}) = f_4(v_{41},v_{42}) = f_5(v_{51},v_{52}) = f_6(v_{61},v_{62})$$

$$f_1(v_{11},v_{12}) = f_2(v_{21},v_{22}) = f_5(v_{51},v_{52}) = f_6(v_{61},v_{62})$$
$$f_3(v_{31},v_{33}) = f_4(v_{41},v_{42})$$

$$f_1(v_{11},v_{12}) = f_2(v_{21},v_{22}) = f_3(v_{31},v_{33}) = f_4(v_{41},v_{42}) = f_5(v_{51},v_{52}) = f_6(v_{61},v_{62})$$

It is obvious that the additional four systems of equations are superfluous.

We remember that identification in step 2 is necessary in order to be able to regard variables as constants in step 4. In our new version we preferred a lazy way of identification. If two such 'variable-constants' never occur in the same subproblem (such as collapsing one term or solving a constant elimination problem) then what on earth should induce you to identify them. Therefore we will do identification only when several such variables occur in one subproblem we want to solve. We give two examples, one for the case that we do identification and one where we do not need it. In both examples we use function symbols f,g and h and a symbol i for which $i(x,x,y) = y$ holds. The terms a and b denote variables that are regarded as constants in the theory of i.

In our first example the subproblem is to match $i(a,v_1,v_2)$ with the variable x in the system:

$$i(a,v_1,v_2) = f(v_3,v_4)$$
$$a = g(v_5,v_6)$$
$$b = h(v_7,v_8)$$

The solution is obviously $\{v_1 \leftarrow a,\ v_2 \leftarrow x\}$ and no identification is needed at all.

In the second example the subproblem consists of matching $i(a,b,v_2)$ with x in:

$$i(a,b,v_2) = f(v_3,v_4)$$
$$a = g(v_5,v_6)$$
$$b = h(v_7,v_8)$$

Here the solution can only be obtained by identifying a and b in $i(a,b,v_2)$. This leads to the term $i(a,a,v_2)$ and the solution of the whole process is $\{b \leftarrow a,\ v_2 \leftarrow x\}$.

> **Please note:** From the point of view of the equational system a and b are **variables**. From the point of view of the special algorithm for the theory of i those two symbols are **constants**. So the process of identification may be seen as some kind of interface between equational system and special algorithm.

3. The third intention was to give up the fixed order of rules to be applied. In the algorithm of [Schmidt-Schauß] each rule was used exactly once and then never more. That is why each transformation had to anticipate all cases that could appear in one of the following steps. So, for example there had to be done all possible identifications of equations in one step in order to provide that no identification would be needed in further processing or all possible theory labellings of equations had to be calculated at one point in order to assume fixed theories for all equations for the rest of computation. Assume for example we had the ten theories $E_1,\ldots,E_{10}$ and there are ten variables $x_1,\ldots,x_{10}$ where x_i occurs in terms of theory E_i. Now suppose there is a multiequation in our system to solve that looks like:

$$x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = x_7 = x_8 = x_9 = x_{10}$$

We do not know what theory this equation will obtain in a solved system. So we have to take into account all ten alternatives, that is the multiequations $x = x_i$ for $i \in \{1,\ldots,10\}$ where x is a new variable that is viewed as a constant in the theories $\{E_1,\ldots,E_{10}\} \setminus E_i$.

Now we allow to apply all of our rules in arbitrary order what leads to interaction between different steps instead of anticipation of all following computation. So in the above example we can wait until some of the variables are replaced by non variable terms.

After having given these motivating notes we go into the actual description of the new version (for details see [Tepp]).

Our version of the algorithm takes as input the system of multiequations to solve, an equational theory E, a partition of this theory for which special unification procedures exist (this partition needs not to be disjoint but must have a disjoint continuation) and a disjoint continuation of the partition. The output again consists of a complete set of solutions modulo E for the system of equations. The procedure consists of seven rules that are ordered by a priority table. For a given system that rule with the highest priority is tried to apply. If it is not applicable the next one is used. This way we get several descendents for one equational system. The non determinism in this version is given by the different choices which one to take next. If no rule at all is applicable then the system is in solved form and can directly be transformed into a substitution. In the following 'theory' denotes an element of the non disjoint partition whereas one of the disjoint continuation is called 'elementary theory'. The seven rules are:

1. MERGING: If there is a multiequation that has some term in common with other equations then all these are identified into a new one.
2. UNITARY UNIFICATION: If there is a multiequation containing several variables or several free terms then in every equation of the system all free terms are unified and all free variables are unified with one another.
3. UNIFICATION: In every multiequation unify all terms whose topsymbol belongs to the same elementary theory and that are contained in some common theory.
4. COMMONIZATION: If there is a multiequation containing two terms with topsymbols belonging to the same elementary theory but there is no common theory for these terms then do unfolding (see original algorithm) for one term until there is a common theory.
5. COLLAPSING: If there are two terms in a multiequation whose topsymbols belong to different elementary theories then choose one of them. Let the elementary theory of its topsymbol be E_i. Now we have two alternatives: label the multiequation to belong to theory E_i or to belong not to theory E_i. The first possibility is processed by the next rule. For the second one take the chosen term and replace all occurences of one of its subterms whose topsymbols are not contained in the signature of E_i by a new variable x. Then match the term with the variable and add a new equation with x and the subterm. This is done for each of those subterms and for the term itself without replacing. The variable x is then regarded as a constant for the theory E_i.

   The second alternative is also taken if an equation is already labelled not to belong to an elementary theory E_i but contains a term with topsymbol in E_i.
6. UNIQUE COLLAPSING: If there is a multiequation labelled to belong to a theory E_i and it contains a term whose topsymbol does not belong to that elementary theory then replace all occurences of one subterm whose topsymbol lies in E_i by a variable x. Then proceed exactly as in the rule

COLLAPSING. We get one new system for each subterm with topsymbol in E_i and one for matching the non replaced term itself with x. The new variable is regarded as a constant for the elementary theory of the topsymbol of the matched term.

7. CYCLE ELIMINATION: This rule is applied when the current system of equations is cyclic. There are two kinds of cycles. One kind consists of a variable and a term containing this variable and both of them are contained in the same multiequation. This sort of cycle is simply solved by unification using the appropriate special procedure. The other kind is formed by terms $t_1,...,t_m\ (= t_1)$ where each term t_i contains a variable x_i that occurs in the same equation as t_{i+1}. In this case we first completely unfold the whole system into disjoint theories just like in the original algorithm. Moreover we transform it such that two neighboring terms in the cycle are in different elementary theories. This way we get a modified cycle $t_1',...,t_n'\ (= t_1')$ with variables $x_1',...,x_n'$. Now there are two alternatives to proceed. In the first case choose one t_i' and label the corresponding multiequation not to belong to the elementary theory of t_i'. Here again we have $n-1$ possibilities each leading to a new system. In the second case we label for each t_i' the involved multiequation containing this term to belong to the elementary theory of t_i'. After that we choose one t_i' and solve the constant elimination problem (t_i',x_i'), where x_i' is regarded as a constant. Once more we have $n-1$ possibilities. Elimination problems once solved are recorded and always composed with new ones and solved all together to prevent that old cycles are reintroduced by solving a current one.

This was a short description of our version of a combination procedure for unification algorithms. With our modification of the original algorithm we were able to achieve more efficiency for theory unification in many cases and additionally we attained a more natural behaviour by not introducing avoidable complexity for easily solvable problems as we hope to have shown in our examples.

Literature:

[Bürckert et al.] H.-J. Bürckert, A. Herold, D. Kapur, J. H. Siekmann, M. E. Stickel, M. Tepp, H. Zang; Opening the AC-Unification Race, JAR 1988

[Schmidt-Schauß] M. Schmidt-Schauß; Unification in a Combination of Arbitrary Disjoint Equational Theories, JSC 1989

[Tepp] M. Tepp; Kombinationsverfahren für Unifikationsalgorithmen, Diploma Thesis 1989

Sextion 2:
Applications – Completion

P. Watson, A.J.J. Dick: Least Sorts in Order-Sorted Term Rewriting

U. Martin, T. Nipkow: Ordered Rewriting and Confluence

P. Lescanne: Completion Procedures as Transition Rules + Control

Least Sorts in Order-Sorted Term Rewriting Extended Abstract

Phil Watson*and Jeremy Dick†

July 28, 1989

Abstract

We consider a problem common to all order-sorted term rewriting systems, namely the requirement for completeness and soundness that the system must be compatible and sort-decreasing (also called fair).

These difficulties arise because the semantic and syntactic notions of what it means for a term to belong to a sort are different. Although in general it is only semi-decidable whether a term is semantically meaningful, we are able to remove the requirements of compatibility and sort-decreasingness with a new definition of the least sort of a term, which bridges the gap between semantics and syntax. This far we follow [WD89].

We then consider the consequences for unification of using this new definition.

Finally we consider some of the other approaches to solving this problem.

1 Introduction - Order-Sorted Term Rewriting

We follow [SNGM87] in our definition of an order-sorted signature, initial algebra, term rewriting system and so on, and the reader is referred to this paper for all the definitions used here. However, two points are of particular importance to us.

We shall assume that substition of 'equals for equals' in the term algebra is always sound and always leads to terms which are semantically meaningful (even if sometimes syntactically ill-formed). This is the key assumption of the 'Smolka Semantics' which we use. We might add that without this assumption, i.e. in any semantics where substitution of equals for equals is not always allowed, this work and the other approaches detailed later are all unsound.

The standard definition of what it means for a term to belong to a sort is as follows. For any term t the least sort of t, LS(t) is defined to be the least sort to which we can mechanically deduce that t belongs, using just the signature and knowledge of the least sort of every proper subterm of t. LS(t) is a purely syntactic measure of the sort(s) to which t belongs. We assume that the signature is regular, namely that LS(t) is unique for every t.

*Supported by the Science and Engineering Council under grant GR/E83634, at Department of Computer Science, Royal Holloway and Bedford New College, University of London

†Informatics Division, Rutherford-Appleton Labs. (now at RACAL Research Ltd.)

We visualise Knuth-Bendix completion operating on the given signature and an initial set of equations (or axioms) in a series of stages. At each stage the existing axioms are oriented into rules by a given term ordering, and these rules are used to generate more axioms in the form of critical pairs.

2 Problems of Order-Sorted Rewriting

In [SNGM87] three problems which are specific to order-sorted rewriting are identified, but all three cases reduce to the same basic problem: terms which may be meaningful semantically (typically by substitution of equals for equals) may be ill-sorted syntactically.

2.1 Example

$$f : S \rightarrow S'$$
$$a : \phi \rightarrow S$$
$$g : S' \rightarrow S'$$

Suppose $< f(a), a >$ is a critical pair. Then any well-founded term ordering will order $f(a) > a$, by the subterm property.

However if we make $f(a) \rightarrow a$ a rule then at a later stage we may have to rewrite $g(f(a))$ to $g(a)$. $g(f(a))$ is well-sorted but $g(a)$ is not well-sorted, although it is semantically meaningful since it is equal to $g(f(a))$. Thus we find ourselves rewriting from a syntactically well-formed term to an ill-sorted one.

The problem here is that our rewrite rule $l \rightarrow r$ has LS(l) <LS(r) and LS(r) is not in the domain of g.

2.2 Example(G.Smolka)

$$a : \phi \rightarrow A$$
$$b : \phi \rightarrow B$$

x is a variable of sort A
$f : B \to B$

Let

$$a \to b$$

and

$$f(x) \to x$$

be rules.
Then our rewrite system gives no critical pairs, but is not locally confluent, e.g.

$$b \leftarrow a \leftarrow f(a) \to f(b)$$

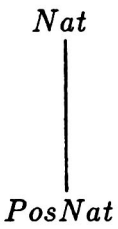
$f(b)$ is in normal form because the rule $f(x) \to x$ cannot be used to reduce $f(b)$.

Note 2.1 *We note in passing that a modified version of order-sorted unification solves the above problem: if we match the left-hand sides of rules of the form $l \to r$ where $LS(l) \not\geq LS(r)$ with variables in the other rule then in the example above $< b, f(b) >$ is a critical pair and confluence is achieved. Note also that the example above is completely general, because we have not assumed any other properties of a, b or f, and the right hand side of the second rule and the fact that $B > A$ are both irrelevant. However this does not solve the problem of rewriting to syntactically ill-formed terms.*

2.3 Sort-Decreasing Systems

A restriction is generally put on order-sorted term rewriting systems to avoid the problems above, namely that the rules of the system must be sort-decreasing, i.e. if $l \to r$ is a rule then LS(l) $\geq$LS(r). However this is an unwanted restriction on the class of problems we may solve with term rewriting, and thus on its usefulness.

Example 2.1 *Worse still, even without rewriting, semantically meaningful terms can be syntactically ill-sorted, e.g.*



$$/ : NatPosNat \to Nat$$
$$- : NatNat \to Nat$$
$$6 : 0 \to PosNat$$
$$3 : 0 \to PosNat$$

Let $6 - 3 = 3$ be an axiom. Then the term $6/(6 - 3)$ is ill-sorted, even though semantically meaningful.

3 Syntactic and Semantic Sorts

The only notion we have regarding the sort(s) of a term t is the syntactic measure LS(t), defined above. In general this is not the same as the semantic sort defined as follows.

Definition 3.1 *Define the semantic sort of a term to be* $SS(t) = \cap_{t==u} LS(u)$ *where* $==$ *means equality (congruence) in the term algebra. We see without difficulty that* $LS(t) \geq SS(t)$ *in all cases.*

Problem 3.1 *In general it is not possible to find the exact position of SS(t) in the sort lattice generated by the sorts in the signature and closed under intersection. Since the sorts are just domains of (typically partial recursive) functions, finding SS(t) is exactly as hard as the Halting Problem.*

Proposition 3.1 *If two terms are proved equal as an axiom of our term rewriting system they have the same semantic sort.*

Proof 1 *Let* $s = t$ *be an axiom and let* s *have semantic sort* A. *Then for any function* f *whose domain includes* A, $f(s)$ *is well-defined in the term algebra. Now we can define* $f(t) = f(s)$ *and* $f(t)$ *is semantically meaningful because we remain in a conservative extension of the term algebra through substitution of equals for equals. The other properties of equality (reflexivity, symmetry and transitivity) are trivially proved to apply also to semantic sort.*

3.1 Example

$$a : \phi \rightarrow A$$
$$b : \phi \rightarrow B$$

Then $a = b$ leads us to deduce $\mathrm{SS}(a) = \mathrm{SS}(b) = B$.

3.2 Approximation

We can define an effective approximation to the semantic sort as follows.

Definition 3.2 *For any term* t, *define the approximate least sort of* t *at stage* $n + 1$ *of the Knuth-Bendix procedure,* $ALS(t, n + 1)$ *to be the greatest lower bound in the sort lattice of:*

(i) $ALS(t, n)$

(ii) $ALS(u, n)$: $u = t$ *has been derived by stage* $n + 1$

Then obviously $LS(t) \geq ALS(t, n) \geq SS(t)$ *for all* t, n.

Furthermore provided Knuth-Bendix does not fail and is fair,

$$Lim_{n \to \inf} \text{ALS}(t, n) = \text{SS}(t)$$

for all t. So for every t there exists an n such that $\text{ALS}(t, n) = \text{SS}(t)$, and if the Knuth-Bendix procedure halts at stage n with a canonical rewrite system then obviously $\text{ALS}(t, n) = \text{SS}(t)$.

3.3 Sort-Decreasingness

The main use of our method is to painlessly remove the requirement of sort-decreasingness in order-sorted term rewriting. There is no longer any danger of rewriting a term to a term of higher or incomparable sort, because as soon as two terms are proved equal (before they need to be oriented into a rule) they take the same ALS, and after this they always have the same ALS and therefore the same SS. Thus in orienting equations into rules we can follow the term ordering in every case. We have effectively reduced the rewriting to single-sorted rewriting, while retaining the expressiveness and flexibility of an order-sorted signature.

Theorem 3.1 *Rewriting using ALS is complete and sound under the 'Smolka semantics'.*

Proof 2 *This is effectively what is proved by Gallier and Isakowitz in [GI89] and [IG88].*

3.4 Formalisation

A formalisation into a set of induction rules for calculating the sorts of terms and a modified Knuth-Bendix procedure with ALS are given in [WD89].

3.5 Problems

There are inevitably some problems associated with this new method:

- Time has to be spent calculating the ALS of terms at each stage; [WD89] shows how this can be minimalised (indeed made effective) but there is still some loss of efficiency.
- If we prove two terms of incomparable ALS to be equal then we must place them in the intersection sort, which may not exist in the original signature. In the worst case we may begin with n sorts and finish with 2^n sorts.
- Unification is affected by this method of defining sorts.

4 Consequences for Unification

4.1 Example

Let

$$f(...,t,...) \rightarrow r1$$

and

$$f(...,x,...) \rightarrow r2$$

be rules, where x is a variable of sort B and t is a term of sort A. We can never unify these two rules using the accepted methods of unification with the standard definition of least sort, LS. However if we prove $\mathrm{SS}(t) \leq B$, we want to unify or we lose completeness.

Example 4.1 *Let A, B, C, V, W be incomparable sorts.*

$$f : V \rightarrow A$$
$$f : W \rightarrow B$$
$$g : B \rightarrow C$$
$$h : V \rightarrow W$$

Let t be a term in V, x be a variable in B. Let the rules be

$$f(t) \rightarrow r1$$
$$g(x) \rightarrow r2$$
$$h(t) \rightarrow t$$

Then there are no critical pairs using unification by LS, but

$$g(r1) = r2$$

in the algebra, but we cannot show that

$$g(r1) \leftrightarrow r2.$$

It is clear from these two examples that we need unification with ALS.

Problem 4.1 *In the first example above, if we attempt to unify $f(...,t,...) \rightarrow r1$ and $f(...,x,...) \rightarrow r2$ early in the completion procedure, we cannot unify. However, later we may prove that $SS(t) \leq B$, so then we can unify. This threatens to make the completion procedure very messy.*

4.2 Obvious Method

The solution to this given in [WD89] is as follows. Add to the completion procedure an instruction to the effect that at the end of each stage, any rule $l \rightarrow r$ for which l has a subterm t of which some instance $\sigma(t)$ has changed its sort during that stage must be returned to the equation set, and hence be unified with all other rules once again.

This is not satisfactory as it is horribly expensive.

4.3 Better Method

A more careful examination of exactly which unifications must be attempted a second time leads to the following, much more efficient, method.

Let $l \to r$ be the rule in question and let t_0 be the smallest subterm of a given instance $\sigma(l)$ of l whose sort has changed during the previous stage. Let t_{j+1} be the largest subterm of $\sigma(l)$ whose sort has changed, i.e.

$t_{i+1} = f_i(..., t_i, ...)$

for some f_i, every $i = 0, 1, ..., j$.

Then we only have to attempt to match l 'at the second attempt' with terms with a variable (x) of sort

$$\geq f_i(..., ALS(t_i, n+1), ...)$$

but

$$\not\geq f_i(..., ALS(t_i, n), ...)$$

for every i, or of sort

$$\geq ALS(t_0, n+1)$$

and

$$\not\geq ALS(t_0, n)$$

and in each case t_{i+1} or t_0 (respectively) must be substituted for x.

4.4 (In)Efficiency

- We have already calculated the ALS of every subterm of the left hand side of every rule in order to find the ALS of the whole l.h.s. during the last stage anyway. Very little extra work is done in re-using this information.

- It is easy to keep a record for every rule of the sorts of the variables occurring in the left hand side, and their positions. N.B. Variables are the only terms which we can be certain will never change their sorts!

- Knowing even one substitution $(x \to t_i)$ cuts down the number of possible unifiers considerably.

- j may be small, even 0, and we need not try to unify subterms whose ALS has not changed.

With these advantages we believe this method will be practical.

4.5 Other approaches to removing sort-decreasingness

The problem of sort-decreasingness has been widely recognised, and there have been a number of different attempts to remove it, within the 'Smolka semantics' where substitution of equals for equals preserves semantic meaning. We restrict our interest to the context of term rewriting; there have been many attempts to reason with sort information outside this field.

Probably the earliest version is that of Gallier and Isakowitz [GI89] and [IG88] who showed that rewriting through syntactically ill-formed, but semantically meaningful, terms preserves soundness and completeness. They did not pursue this idea as far as giving a completion procedure, or consider the consequences for unification. The trick used above to solve Smolka's example is adequate to make this method practical. However we do not feel it is aesthetically pleasing to allow rewriting through ill-sorted terms without explicitly calculating the sort of the resulting term.

An approach similar to this paper is taken by [Du89], except that he makes an explicit declaration of each piece of sort information and uses this to build a unification algorithm.

Explicit declarations are also used by [Ga88], in which sort information is used as the conditions in conditional rewriting in order to reduce order-sorted rewriting to many-sorted (effectively single-sorted) rewriting. The drawback to this is that use of conditional rewriting means completeness can only be proved under certain (undecidable) conditions. However, Ganzinger's is the only approach which has yet been implemented, in the system CEC.

4.6 Acknowledgements

Thanks to Brian Matthews (R.A.L.) and Mike Lai (R.H.B.N.C.) for useful discussions, especially on the trick used to solve Smolka's example.

4.7 References

[Ba87] L. Bachmair - "Proof Methods for Equational Theories", Ph.D. Thesis, Univ. of Illinois at Urbana-Champaign, 1987

[Du89] L. Duponcheel - "Typed Algebra (Back to the future)", these proceedings, 1989

[GI89] J.H. Gallier and T. Isakowitz - "Rewriting in Order-Sorted Equational Logic, preprint

[Ga88] H. Ganzinger - "Order-Sorted Completion: The Many-Sorted Way", Technical Report no. 274, Forschungsberichte des Fachbereichs Informatik, Univ. Dortmund, 1988

[GJM85] J.A. Goguen, J.-P. Jouannaud and J. Meseguer - "Operational Semantics of Order-Sorted Algebras", Proc. ICALP 1985, Springer LNCS 194, 221-231

[IG88] T. Isakowitz and J.H. Gallier, "Congruence Closure in Order-Sorted Algebra", preprint

[SNGM87] G. Smolka, W. Nutt, J.A. Goguen and J. Meseguer - "Order-Sorted Equational Completion", SEKI Report SR-87-14, Univ. Kaiserslautern, 1987

[WD89] P. Watson and A.J.J. Dick, Univ. of London, Royal Holloway and Bedford New College, Tech. Report TR-CSD-606, 1989

Ordered Rewriting and Confluence

Ursula Martin* and Tobias Nipkow†

Department of Computer Science, RHBNC, University of London
Egham, Surrey TW20 0EX, *UK*
and
University of Cambridge, Computer Laboratory,
Pembroke Street, Cambridge CB2 3QG, *UK*

1 Introduction

One of the major problems in term rewriting theory is what to do with an equation which cannot be ordered into a rule. Many solutions have been proposed, including the use of special unification algorithms [7] or of unfailing completion procedures [1,6].

If an equation cannot be ordered we can still use any instances of it which can be ordered for rewriting. Thus for example

$$x * y = y * x$$

cannot be ordered, but if a, b are constants with $b * a > a * b$ we may rewrite

$$b * a \longrightarrow a * b.$$

This idea is used in unfailing completion, and also appears in Boyer-Moore [2]. In this paper we define and investigate completeness with respect to this notion of rewriting and show that many familiar systems are complete rewriting systems in this sense. This allows us to decide equality without the use of special unification algorithms. We prove completeness by proving termination and local confluence. We describe a confluence test based on recursive properties of the ordering.

1.1 Summary

In this section we summarize our results. Precise definitions are given below. An *ordered rewriting system* consists of a set of equations E and a monotonic ordering on terms $>$ which is total on ground terms. We say a term s rewrites to a term t, denoted by $s \longrightarrow t$, if there is an equation $r = l$ or $l = r$ in E, a substitution σ and a subterm σl of s such that $\sigma l > \sigma r$ and t is s with that subterm replaced by σr. Thus for example if $x * y = y * x$ is in E and $a * b > b * a$ then $a * b \longrightarrow b * a$. We observe that the usual notion of a rewriting system can be regarded as a special case of our concepts in the case when the ordering allows all the equations to be ordered into rules.

A *ground complete ordered rewriting system* is one which is terminating and confluent on ground terms. This means that any ground term can be rewritten to a unique canonical form, and we can decide equality between ground terms, and hence between variable terms by regarding the variables as generalised constants. This process uses only unification in the empty theory. In section 4 we give examples of ground complete ordered rewriting systems including AC, ACI, Boolean rings, Distributivity and Abelian Groups.

Example 1 As an example let E be

$$(x * y) * z \quad = \quad x * (y * z) \tag{1}$$

*The author acknowledges support of the UK SERC under grant GR/E 83634.

†This research was supported in part by NYNEX, NSF grant CCR-8706652, and by the Advanced Research Projects Agency of the DoD, monitored by the ONR under contract N00014-83-K-0125.

$$x * y = y * x \quad (2)$$
$$x * (y * z) = y * (x * z) \quad (3)$$

and let $>$ be any monotonic ordering on terms which is total on ground terms and satisfies for all ground terms x, y, z

$$(x * y) * z > x * (y * z) \quad (4)$$
$$x * y > y * x \quad \text{if } x > y \quad (5)$$
$$x * (y * z) > y * (x * z) \quad \text{if } x > y. \quad (6)$$

Then $(E, >)$ is an ground complete ordered rewriting system. For example suppose that $>$ is the lexicographic path ordering (see section 3) and a, b, c are constants with $c > b > a$. Then

$$b * (c * (b * a)) \longrightarrow b * (c * (a * b)) \longrightarrow b * (a * (c * b)) \longrightarrow a * (b * (c * b)) \longrightarrow a * (b * (b * c)).$$

To prove completeness we need as usual to prove termination and confluence. Termination is generally proved by showing that the ordering is well-founded[1], and the orderings we need to do this for our examples are discussed in section 3. To prove confluence we need to prove local confluence, and in section 2.1 we prove the necessary version of the critical pairs lemma

The usual notion of rewriting and confluence allows confluence to be checked automatically by computation of normal forms. At first sight this is not so for ordered rewriting, which appears to require infinitely many calculations. However we shall explain in section 2.2 how the confluence test may indeed be automated in many cases by axiomatising the properties of the orderings and rewritings that we need. This automation works for most of the examples of section 4.

Consider the example above. Computing critical pairs between

$$(x * y) * z \longrightarrow x * (y * z)$$
$$x * y = y * x$$

we obtain

$$z * (x * y) \longleftarrow (x * y) * z \longrightarrow x * (y * z),$$

so we have to prove that $z * (x * y)$ and $x * (y * z)$ are joinable for all ground terms x, y, z. Now since x, y, z are ground terms we may consider the possible relationships between them under $>$. For example if $x > z > y$ then

$$x * (y * z) \longrightarrow y * (x * z) \longrightarrow y * (z * x) \longleftarrow z * (y * x) \longleftarrow z * (x * y)$$

While our technique allows the computation of canonical forms without a special matching algorithm we note that it is not always as powerful as rewriting with an equational matching algorithm.

Example 2 Consider the example above together with the equation $f(x * x) = 1$. Rewriting with an AC matching algorithm shows that $f(a * (a * (b * b))) = 1$. However, there is no equivalent ground complete ordered rewriting system as any such system would have to contain infinitely many equations to deal with

$$f(a * a), \quad f(a * (a * (b * b))), \quad f(a * (a * (a * (a * (b * b)))))$$

and so on.

On the other hand the advantages of our method are that

- Ground rewriting is possible without E-matching algorithms.
- Ground rewriting is sufficient for theorem proving.
- Completion is possible without E-completion.

It shares these features with unfailing completion as described for example in [6]. Indeed, it is very similar to unfailing completion in two technical aspects: both sides on an equation may give rise to critical pairs, and complete systems need only be confluent for ground terms. In contrast to [6], our method is specially designed to test for confluence of ground terms. Thus we obtain many complete systems which their approach fails to recognize as complete.

[1] In fact, it is sufficient to show well-foundedness within equivalence classes. If all equivalence classes are finite, this follows because any ordering an a finite set is well-founded.

2 Critical Pairs, Confluence, and Completion

We assume that all concepts and definitions are as in [5] or [3]. Let Σ be a set of function symbols and V a set of variables. The set of all terms in $\Sigma \cup V$ is denoted by $\mathcal{T} = \mathcal{T}(\Sigma \cup V)$, and the set of all ground terms is the subalgebra $\mathcal{T}_G = \mathcal{T}(\Sigma)$. The function $\mathcal{V}$ returns the set of variables in a term. A term can be represented as subterms in a *context* by writing $C[s_1, \ldots, s_n]$. The context C is a λ-term $\lambda x_1, \ldots, x_n.t$, and $C[s_1, \ldots, s_n]$ denotes application, i.e. the simultaneous the replacement of all x_i by s_i. In particular we assume that every x_i occurs exactly once in t.

An ordering $>$ on a set S is a relation which is irreflexive and transitive, so that it is false that $x > x$, and if $x > y$ and $y > z$ then $x > z$. An ordering on $\mathcal{T}$ is monotonic if for all function symbols f and terms $s_1, \ldots, s_n, s, t$ we have $f(s_1, \ldots, s, \ldots s_n) > f(s_1, \ldots, t, \ldots s_n)$ if $s > t$.

An *ordered rewriting system* is a pair $(E, >)$ where E is a set of equations in $\mathcal{T}$ and $>$ is a monotonic ordering on $\mathcal{T}$ which is total on ground terms. The notation $s \doteq t \in E$ is short for $s = t \in E \vee t = s \in E$. If for some $l \doteq r \in E$ we have $\sigma l > \sigma r$ for all substitutions σ we write $l \longrightarrow r$ and call it a rule.

The ordered rewriting system $(E, >)$ induces a relation $\longrightarrow$ defined as

$$C[\sigma s] \longrightarrow C[\sigma t] \quad \text{if } s \doteq t \in E \ \wedge \ \sigma s > \sigma t$$

Since $>$ is monotonic, $p \longrightarrow q$ implies $p > q$. The restriction of $\longrightarrow$ to ground terms is denoted by $\Longrightarrow$. We use $\longrightarrow^*$ to denote the reflexive transitive closure of $\longrightarrow$. Two terms s and t are called *joinable*, written $s \downarrow t$, iff there is a term u such that $s \longrightarrow^* u$ and $t \longrightarrow^* u$. They are called *ground joinable*, written $s \Downarrow t$, iff any two ground instances σs and σt are joinable. An ordered rewriting system is called *ground terminating* if there is no sequence of ground terms $\{a_i | i \in \mathbf{N}\}$ such that $a_i \Longrightarrow a_{i+1}$ for all i. An ordered rewriting system is called *ground confluent* if whenever r, s, t are ground terms with $r \Longrightarrow^* s$ and $r \Longrightarrow^* t$ then $s \downarrow t$. If the terminating or confluence conditions hold for all terms rather than just ground terms we call $(E, >)$ terminating or confluent respectively. An ordered rewriting system which is terminating and confluent is called *complete*; one which is ground terminating and ground confluent is called *ground complete*. It follows from Newman's lemma that if $(E, >)$ is complete then each term s has a unique normal form, and if $(E, >)$ is ground complete this is true for ground terms. Thus if s, t are terms (ground terms) and $(E, >)$ is complete (ground complete) then $s =_E t$ if and only if their normal forms are identical. If s and t are arbitrary terms we may still use a ground complete system to decide equality if we regard the variables occurring in s and t as new constants.

In the sequel let $(E, >)$ denote an ordered rewriting system and let $\longrightarrow$, $\Longrightarrow$ etc. be the rewrite relations it generates. If there is a second ordering, say $\succ$, we write $\longrightarrow_\succ$, $\Downarrow_\succ$ etc. to denote the relations induced by $(E, \succ)$.

2.1 The Critical Pair Lemma

This section deals with the extension of the critical pair lemma to ordered rewriting.

Definition 1 Given two equations $C[u] = t$ and $v = w$, where $u \notin V$, and a most general unifier σ of u and v such that $\sigma(C[u]) \not\leq \sigma t$ and $\sigma v \not\leq \sigma w$, then $(\sigma t, \sigma(C[w]))$ is a *critical pair*.

The set of all critical pairs of E is the set of all critical pairs between any two equations $p \doteq q, s \doteq t \in E$.

Note that because of the symmetry of $\doteq$ both sides of an equation can give rise to critical pairs. If $>$ orders every equation in E into a rule our definition of critical pairs reduces to the usual one.

Lemma 1 *If $>$ is total within equivalence classes of ground terms, both sides of an equation $s \doteq t \in E$ are ground joinable.*

Proof For any ground substitution σ we have $\sigma s > \sigma t$, $\sigma s = \sigma t$ or $\sigma s < \sigma t$, which implies $\sigma s \longrightarrow \sigma t$, $\sigma s = \sigma t$ or $\sigma s \longleftarrow \sigma t$ and hence $\sigma s \downarrow \sigma t$. □

We now come to the proof of the critical pair lemma. We cannot directly appeal to the theorems in [4] because we deal with ordered rewriting. In particular this means that we may have $\sigma l \longrightarrow \sigma r$ but not $\tau l \longrightarrow \tau r$ for two substitutions σ and τ. Fortunately, $\longrightarrow$ is still *compatible*: if $s \longrightarrow t$ then $C[s] \longrightarrow C[t]$ for any context C.

Lemma 2 *An ordered rewriting system $(E, >)$ is locally ground confluent iff all critical pairs are ground joinable.*

Proof The proof is very similar to the one in [4], except that we need to take $>$ into account when rewriting. The $\Rightarrow$-direction of the proposition is trivial. For the other direction let all critical pairs be ground joinable, and let $r \Longrightarrow s$ and $r \Longrightarrow t$. Thus there are equations $p \doteq q, u \doteq v \in E$, matching substitutions σ and τ, and contexts M and N such that $\sigma p \Longrightarrow \sigma q$, $\tau u \Longrightarrow \tau v$, and $r = M[\sigma p] = N[\tau u]$, $s = M[\sigma q]$, and $t = N[\tau v]$. We distinguish 3 cases.

Case 1: the two rewrites are at disjoint occurrences. Then $r = C[\sigma p, \tau u]$, $s = C[\sigma q, \tau u]$, $t = C[\sigma p, \tau v]$, and therefore $s \Longrightarrow C[\sigma q, \tau v] \Longleftarrow t$.

Case 2: the two rewrites overlap each other. W.l.o.g. let $r = C[\sigma p]$ where τu is a subterm of σp. Then $s = C[\sigma q]$.

Case 2a: there is a variable x in p such that $\sigma x = D[\tau u]$. Let $p = A[x^m]$ and $q = B[x^n]$, i.e. p and q contain m and n distinct occurrences of x respectively. Then $\sigma p = A'[\sigma(x)^m] = A'[D[\tau u]^m]$ and $\sigma q = B'[\sigma(x)^n] = B'[D[\tau u]^n]$. By compatibility $\sigma x \Longrightarrow D[\tau v]$ and thus $s \Longrightarrow^* C[B'[D[\tau v]^n]] =: s'$ and $t = C[A'[\sigma x, \ldots, \sigma x, D[\tau v], \sigma x, \ldots, \sigma x]] \Longrightarrow^* C[A'[D[\tau v]^m]] =: t'$. Thus $s' = C[\sigma' q]$ and $t' = C[\sigma' p]$ for $\sigma' = \sigma + [x \rightarrow D[\tau v]]$. Because $>$ is total on equivalence classes of ground terms, lemma 1 implies that $\sigma' p \downarrow \sigma' q$. By compatibility $s \downarrow t$ holds as well.

Case 2b: otherwise σp must be the instance of a proper overlap of p and u. Therefore $(\sigma p, t_1)$, where $t = C[t_1]$, is a ground instance of a critical pair between $p = q$ and $u = v$. Thus $\sigma p \downarrow t_1$ which implies $s \downarrow t$ by compatibility. □

The proof of this lemma relies on the totality of $>$ within equivalence classes of ground terms. The following example shows that this requirement cannot be dropped:

Example 3 Let $\Sigma = \{*\} \cup C$, where C is a set of constants, $E = \{x * y = y * x\}$, and $s > t$ iff the leftmost constant in s is $>$ the leftmost constant in t. Clearly $>$ is not total because a and $a * b$ are incomparable. Assume that $a < b$. Then the term $r = (b * a) * a$ can be rewritten to $s = a * (b * a)$ and $t = (a * b) * a$. However, s and t are not joinable because s rewrites only to $a * (a * b)$, which is in normal form, and t is in normal form already.

On the other hand there is only a single critical pair $(y * x, y * x)$ in E which is trivially ground joinable. This shows that for non-total $>$ the consideration of critical pairs does not suffice to determine local ground confluence.

Corollary 1 *A terminating ordered rewriting system is ground confluent iff all critical pairs are ground joinable.*

Proof If all critical pairs are ground joinable we know by lemma 2 that $\Longrightarrow$ is locally ground confluent. Termination implies confluence of $\Longrightarrow$. The other direction is trivial. □

2.2 Automating It

In contrast to ordinary rewriting systems, where critical pairs are required to be joinable, we need the weaker criterion of ground joinability. It is not at all clear how a test of the latter property can be automated since it talks about an infinite set of ground instances. In fact we believe that ground joinability is in general undecidable. The purpose of this section is to give some sufficient criteria which are easily implementable and powerful enough to solve some non-obvious examples. On the other hand they are far from complete. Section 4.9 contains an example which is easily proved to be ground joinable but which is not covered by our method.

The principle idea underlying the automation has already been sketched in the introduction: given two terms s and t, we consider all possible relationships between the variables in s and t under $>$ and $=$ and try to join s and t for each of them. Since there are only finitely many relationships, namely all linear orderings, we only have to consider a finite, albeit possibly very large, number of cases. It remains to be explained how rewriting of terms with variables is to proceed if we do not know what the variables stand for, only how they are related to each other with respect to $>$. As an example take the term $y * x$ with the constraint $x < y$. It requires some intimate knowledge of $>$ to determine whether this implies that $y * x > x * y$, i.e. whether commutativity is applicable.

Instead of working with a particular ordering and inferring some of its properties, we assume a small set of properties of the ordering which allow us to order enough terms for proving ground confluence. For the AC case we have seen in the introduction that the implications (4)-(6) are sufficient for joining one of the critical pairs under a particular set of constraints. In section 4.2 we show that the equations (1)-(3) together with any ordering satisfying (4)-(6) are ground confluent.

The advantage of this approach is its generality: ground confluence is proved for any ordering satisfying the properties we have assumed. However, it means that one has to be careful in the choice of properties. For example they must not violate well-foundedness.

We will now describe a test for ground joinability based on the above ideas. Formally, the "properties" of the ordering are given as a closure operator C on $\mathcal{T} \times \mathcal{T}$ subject to the restriction

$$(s,t) \in C(>) \quad \Rightarrow \quad (\sigma s, \sigma t) \in C(\sigma(>)) \tag{7}$$

where $\sigma(>) = \{(\sigma u, \sigma v) \mid u > v\}$. The intuition is that C takes a relation on terms and returns the set of consequences implied by the properties we assumed of the ordering. The above restriction ensures that C is well behaved with respect to substitutions. This enforces for example that if $x * y > y * x$ follows from $x > y$, then $x' > y'$ must imply $x' * y' > y' * x'$. We say that an ordering $>$ is *compatible* with C if $C(>) = >$. As a consequence of restriction (7) we obtain:

Lemma 3 *Let E be a set of equations, let $>$ and $\succ$ be two relations on $\mathcal{T}$, and let σ be a substitution such that $\sigma(\succ) \subseteq >$. Then $u \longrightarrow_{C(\succ)} v$ implies $\sigma u \longrightarrow_{C(>)} \sigma v$ for all terms u, v.*

Proof ¿From $u \longrightarrow_{C(\succ)} v$ it follows that $u = C[\tau l]$, $v = C[\tau r]$ and $(\tau l, \tau r) \in C(\succ)$ for some $l \doteq r \in E$. From $(\tau l, \tau r) \in C(\succ)$ it follows by (7) that $(\sigma\tau l, \sigma\tau r) \in C(\sigma(\succ))$. Since C is a closure operator and $\sigma(\succ) \subseteq >$ we also have $(\sigma\tau l, \sigma\tau r) \in C(>)$. Thus $\sigma u = \sigma(C)[\sigma\tau l] \longrightarrow_{C(>)} \sigma(C)[\sigma\tau r] = \sigma v$. □

Ordering the variables in a term with respect to $=$ and $>$ is equivalent to providing a total order on equivalence classes of variables. If ρ is an equivalence on a set of variables, $\hat{\rho}$ denotes a substitution which maps each variable to some fixed representative of its equivalence class. Testing for ground joinability of two terms s and t by considering all total orders on equivalence classes of variables in s and t leads to the following definition. If $\hat{\rho}s \downarrow_{C(\succ)} \hat{\rho}t$ holds for all equivalences ρ on the variables in s and t and all total orders $\succ$ on the range of $\hat{\rho}$, then we write

$$s \Downarrow_C t.$$

Restriction (7) ensures that this definition is independent of the particular choice of representatives of ρ-equivalence classes.

The next lemma shows that $s \Downarrow_C t$ does imply ground joinability:

Lemma 4 *If $s \Downarrow_C t$ then $s \Downarrow_> t$ holds for all orderings $>$ compatible with C.*

Proof Let $>$ be compatible with C and let σ be some ground substitution with $dom(\sigma) = \mathcal{V}(s) \cup \mathcal{V}(t)$. We have to show that $\sigma s \downarrow_> \sigma t$.

Let $\rho = ker(\sigma)$ and define $x \succ y$ iff $\sigma x > \sigma y$ for x, y in the range of $\hat{\rho}$. Then $\succ$ is a total order and $s \Downarrow_C t$ implies $\hat{\rho}s \downarrow_{C(\succ)} \hat{\rho}t$. Since $>$, $\succ$, and σ satisfy the assumptions of lemma 3 it follows that $\sigma s = \sigma\hat{\rho}s \downarrow_{C(>)} \sigma\hat{\rho}t = \sigma t$. Since $>$ is compatible with C we have $\sigma s \downarrow_> \sigma t$. □

¿From this lemma and the definition of $\Downarrow_C$ it follows directly that

Corollary 2 *If $C(\succ)$ is recursive and well-founded for all recursive and well-founded $\succ$, then $\Downarrow_C$ is a sufficient recursive criterion for ground joinability with respect to all orderings compatible with C.*

This is the first step towards automating the test for ground joinability. The second ingredient is lemma 1. Combining all these criteria we obtain the following set of rules:

$$\begin{aligned} s \Downarrow t &\Leftarrow s = t \\ s \Downarrow t &\Leftarrow s \Downarrow_C t \\ s \Downarrow t &\Leftarrow \exists l \doteq r \in E, \sigma.\ \sigma l = s \ \wedge\ \sigma r = t \\ f(s_1, \ldots, s_n) \Downarrow f(t_1, \ldots, t_n) &\Leftarrow \forall i.\ s_i \Downarrow t_i \end{aligned}$$

The first clause is obvious, the second and third ones are consequences of lemmas 4 and 1 respectively, and the last one follows from compatibility of rewriting.

The prototype implementation of this test is written in Prolog and follows exactly the above four Horn clauses. C is just another predicate. In all our examples C consists of the implications (4)-(6) and further clauses specific to the example.

2.3 Completion

The critical pair lemma in the preceding section leads to a completion algorithm in the usual way: critical pairs which are not ground joinable are added as new equations. Formally this can be expressed as an inference rule between sets of equations:

$$\frac{E}{E \cup \{s = t\}} \quad \text{if } (s,t) \text{ is a critical pair of } E \text{ and not } s \Downarrow t.$$

If this process terminates because all critical pairs are ground joinable, we have obtained a ground complete ordered rewriting system. In addition one may want to obtain a reduced rewriting system by simplifying the right or left hand sides of equations by other equations. This can be achieved by the following rule:

$$\frac{E \cup \{s \doteq t\}}{E \cup \{s = u\}} \quad \text{if } t \longrightarrow_E u \ \wedge \ s \doteq t \notin E$$

Since we are only interested in ground confluence, ground joinable equations can be removed:

$$\frac{E \cup \{s \doteq t\}}{E} \quad \text{if } s \Downarrow t$$

The applications of these three rules may be interleaved arbitrarily.

A prototype implementation of this completion procedure has been written in Prolog and was used for all the examples in section 4.

3 Orderings

Our notations and concepts are taken from Dershowitz [3].

An ordering is called well-founded if there is no set $\{a_i | i \in \mathbf{N}\}$ with $a_i > a_{i+1}$ for each i. We have

Lemma 5 *Let $(E, >)$ be an ordered rewriting system. If $>$ is well-founded then $(E, >)$ is terminating.*

Proof The monotonicity condition ensures that if $s \longrightarrow t$ then $s > t$, so if $>$ is terminating there can be no infinite chain of rewrites. □

The following orderings will be used in the sequel.

Lexicographic Path Ordering

Let $s = f(s_1, \ldots, s_m), t = g(t_1, \ldots, t_n)$. Let $>$ be an ordering on function symbols. Then

$s > t$ if and only if

- $s_i \geq t$ for some $i = 1, \ldots, m$, or
- $f > g$ and $s > t_j$ for all $j = 1, \ldots, n$, or
- $f = g$ (so $n = m$) and $(s_1, \ldots, s_n)$ is greater than $(t_1, \ldots, t_n)$ in the lexicographic ordering from the left on sequences induced by $>$, and $s > t_i$ for $i = 2, \ldots, n$.

Then we have

Lemma 6

1. *The lexicographic path ordering is well-founded, and is total on ground terms if the operator precedence is total.*
2. *If f, g are binary function symbols with $f > g$ and x, y, u, v are any terms and $f(x,y) > u, f(x,y) > v$ then $f(x,y) > g(u,v)$.*

Proof

1. This is just Theorem 22 of [3]
2. Follows from the definitions.

□

Knuth-Bendix Orderings

The essence of the Knuth-Bendix orderings is to compare terms first by weight and then lexicographically by an operator precedence. For details see [8] or [10], where proofs will be found of

Lemma 7 *The Knuth-Bendix ordering is monotonic and well-founded, and is total on ground terms if the operator precedence is total.*

Lexicographic Orderings

Let $\Sigma = \{f, a_1, \ldots, a_k\}$ where f is binary and $a_1, \ldots a_n$ are constants. Assume $a_1 < a_2 \ldots < a_k$. Define $>_t$ for $t = 1, 2$ by

$$\begin{array}{rcll} a_i & >_t & a_j & \text{if and only if} \quad i \geq j \\ f(x,y) & >_t & f(z,t) & \text{if and only if} \quad x > z \text{ or } x = z \text{ and } y > t \\ f(a_i,x) & >_1 & a_j & \text{if and only if} \quad i \geq j \\ a_j & >_1 & f(a_i,x) & \text{if and only if} \quad j > i \\ f(a_i,x) & >_2 & a_j & \text{for all } i,j = 1,\ldots,k \end{array}$$

where x, y, z, t are arbitrary ground terms. Then

Lemma 8 *For each of the orderings $>_1, >_2$*

1. *$>_t$ is a monotonic ordering and total on ground terms*
2. *$f(f(x,y),z) >_t f(x,f(y,z))$ and if $x >_t y$ then $f(x,y) > f(y,x)$ for all ground terms x, y, z.*
3. *$>_t$ is not well-founded.*

Proof The proof is straightforward. For (3) notice that we have

$$f(a_2,a_1) >_t f(a_1,f(a_2,a_1)) >_t f(a_1,f(a_1,f(a_2,a_1))) >_t \ldots.$$

□

Notice that $>_1$ is described by Boyer and Moore [2], where it is expressed in terms of projecting onto strings by

$$s(a_i) = a_i, s(f(x,y)) = fs(x)s(y)$$

where f denotes function application, and ordering the strings lexicographically.

To enable us to use the automatic confluence test described in section 2.2 we need to identify orderings with certain properties.

Definition 2 An ordering is called *AC compatible* for the binary operator f if it is monotonic, well-founded and total on ground terms, and satisfies for all ground terms x, y, z

$$\begin{array}{rcll} f(f(x,y),z) & > & f(x,f(y,z)) & \\ f(x,y) & > & f(y,x) & \text{if } x > y \\ f(x,f(y,z)) & > & f(y,f(x,z)) & \text{if } x > y \end{array}$$

Lemma 9 *Let $>$ be the Knuth-Bendix ordering or the lexicographic path ordering and f any binary function symbol. Then $>$ is AC-compatible for f.*

Proof Follows from the definitions. □

4 Examples

We present here examples of ground complete ordered rewriting systems. 4.1-4.9 are standard algebraic systems. In 4.10 we investigate combination of rewriting systems. In 4.11 we investigate an alternative ordering for which AC has a ground complete system containing two rules only, and in we give a ground complete ordered rewriting system for abelian groups.

Examples 4.2-4.9 are all ground complete for any ordering $>$ which

1. is AC-compatible for all AC operators in the system, and

2. satisfies $s > t$ for all rules $s \longrightarrow t$.

For examples 4.1-4.6 the Knuth-Bendix orderings and the lexicographic path orderings have the required properties. These examples were all proved ground complete using the method of section 2.2. The closure operator C was induced by 1 and 2 above.

Intuitively one reason why all the examples involving AC work is that we we are doing is using a sorting algorithm. Any ground term is equal to a product of irreducibles and the AC rules (1)-(3) are sorting these irreducibles into increasing order using bubble sort. The two rule version is merely using a different sorting algorithm.

4.1 Commutativity

Let E be $\{x * y = y * x\}$ and $>$ any monotonic ordering total within equivalence classes of ground terms. Then $(E, >)$ is a ground complete ordered rewriting system. It is confluent because there are no (non-trivial) critical pairs. It is terminating since each equivalence class is finite, and so any infinite chain of rewrites would contain a loop, which would imply that $>$ was not irreflexive.

4.2 Associativity and Commutativity

This example has been discussed in the introduction. Let E be

$$\begin{aligned} (x * y) * z &\longrightarrow x * (y * z) \\ x * y &= y * x \\ x * (y * z) &= y * (x * z) \end{aligned}$$

$(E, >)$ is also ground complete if $>$ is either of the lexicographic orderings $>_i$.

4.3 Associativity and Commutativity — Another Version

In the introduction we observed that one of the critical pairs generated by 4.2 was $(z * (x * y), x * (y * z)$. We may use this to obtain another three rule ground complete ordered rewriting system for AC. Let E be

$$\begin{aligned} (x * y) * z &\longrightarrow x * (y * z) \\ x * y &= y * x \\ z * (x * y) &= x * (y * z) \end{aligned}$$

and $>$ any ordering which satisfies (5) and $z * (x * y) > x * (y * z)$ if $z > x$. The lexicographic path ordering and the Knuth-Bendix ordering have this property.

4.4 Associativity, Commutativity, and Idempotence

Let E be

$$\begin{aligned} (x * y) * z &\longrightarrow x * (y * z) \\ x * y &= y * x \\ x * (y * z) &= y * (x * z) \\ x * x &\longrightarrow x \\ x * (x * y) &\longrightarrow x * y. \end{aligned}$$

4.5 Groups of Exponent Two

Let E be

$$\begin{aligned} (x * y) * z &\longrightarrow x * (y * z) \\ x * y &= y * x \\ x * (y * z) &= y * (x * z) \end{aligned}$$

$$
\begin{aligned}
x * x &\longrightarrow 1 \\
x * (x * y) &\longrightarrow y \\
x * 1 &\longrightarrow x \\
1 * x &\longrightarrow x
\end{aligned}
$$

Then $(E, >)$ is a ground complete ordered rewriting system for groups of exponent two.

4.6 Groups of Exponent Two in Disguise

We want to prove that the two laws

$$
\begin{aligned}
(x * x) * y &= y \\
(x * y) * z &= (y * z) * x
\end{aligned} \tag{8}
$$

axiomatize groups of exponent two. Starting from this system, the completion procedure generated the following list of critical pairs, ordering some of them into rules:

$$
\begin{aligned}
(x * y) * x &\longrightarrow y \\
(x * y) * y &\longrightarrow x \\
x * x &= y * y && (9) \\
x * x &\longrightarrow 1 && (10) \\
1 * x &\longrightarrow x \\
x * 1 &\longrightarrow x \\
x * (x * y) &\longrightarrow y \\
x * y &= y * x \\
(x * y) * z &\longrightarrow x * (y * z) \\
x * (y * z) &= y * (x * z)
\end{aligned}
$$

Notice that (10) is the result of "dividing" (9), i.e. introducing the new constant 1. The final set of equations (all the ones below and including (10)) is the same as in section 4.5. All the other equations are now joinable.

In [9] the same problem is attacked with the help of the term rewriting system Reve. Because (8) cannot be oriented into a rule, Reve cannot deal with it directly. Martin obtained the result by working with consequences of (8) that can be ordered.

4.7 Distributivity

Let E be

$$
\begin{aligned}
(x * y) * z &\longrightarrow x * (y * z) \\
x * y &= y * x \\
x * (y * z) &= y * (x * z) \\
x * (y + z) &\longrightarrow x * y + x * z \\
(x + y) * z &\longrightarrow x * z + y * z
\end{aligned}
$$

and let $>$ be any ordering which is AC-compatible for both $+$ and $*$. For example the lexicographic path ordering fits the bill. Then $(E, >)$ is a ground complete ordered rewriting system.

4.8 Boolean Rings

The following is a ground complete set of ordered rewrite rules for Boolean rings.

$$\begin{array}{rcl@{\qquad}rcl}
x+y & = & y+x & x*y & = & y*x \\
x+(y+z) & = & y+(x+z) & x*(y*z) & = & y*(x*z) \\
x+x & \longrightarrow & 0 & x+0 & \longrightarrow & x \\
0+x & \longrightarrow & x & x*x & \longrightarrow & x \\
1*x & \longrightarrow & x & x*1 & \longrightarrow & x \\
x*0 & \longrightarrow & 0 & 0*x & \longrightarrow & 0 \\
(x*y)*z & \longrightarrow & x*(y*z) & (x+y)+z & \longrightarrow & x+(y+z) \\
x*(y+z) & \longrightarrow & x*y+x*z & (x+y)*z & \longrightarrow & x*z+y*z \\
x*(x*y) & \longrightarrow & x*y & x+(x+y) & \longrightarrow & y
\end{array}$$

The ordering must be AC-compatible for both $+$ and $*$. The lexicographic path ordering has these properties. Ground confluence can be checked by the technique of of section 2.2.

4.9 Another System

The equation

$$(x*x)*y = y*(x*x) \tag{11}$$

is an example of a system that is ground confluent for any ordering total and well-founded on ground terms. The reason is that the only nontrivial critical pair

$$y*((x*x)*(x*x)) = ((x*x)*(x*x))*y$$

is an instance of (11). By lemma 1 this implies ground joinability.

If (11) is generalized slightly to

$$(x*y)*z = z*(x*y) \tag{12}$$

and we assume that $x*y > z$ implies $(x*y)*z > z*(x*y)$, the criteria of section 2.2 fail to prove ground confluence, although there is a very simple proof. The two critical pairs are

$$\begin{array}{rcl}
(z*(x*y))*u & = & u*((x*y)*z) \\
((x*y)*z)*u & = & u*(z*(x*y)).
\end{array} \tag{13}$$

Let us just consider the first one. If $(x*y) = z$, (13) is an instance of (12). If $(x*y) > z$ or $(x*y) < z$, (13) can be rewritten to $(z*(x*y))*u = u*(z*(x*y))$ or $((x*y)*z)*u = u*((x*y)*z)$, both of which are instances of (12). Again lemma 1 implies ground joinability. The proof for the second critical pair is practically identical.

The tests in section 2.2 cannot cope with these critical pairs because the proof of ground joinability is based on a case distinction which compares not just variables but whole subterms, namely $x*y$ and z.

4.10 Combination of Systems

In this section we discuss how a ground complete ordered rewriting system may be combined with a ground complete rewriting system in the usual sense.

Lemma 10 *Suppose that*

1. *R is a ground complete rewriting system in the usual sense over a set of function symbols Σ and $R' = \{l = r \mid l \longrightarrow r \in R\}$.*
2. *$(E, >)$ is a ground complete ordered rewriting system over a set of function symbols Γ*
3. *There are no critical pairs between E and R'.*
4. *There is a well-founded monotonic total ordering $\succ$ on $T(\Sigma \cup \Gamma)$ such that $\succ \supseteq >$ and $\sigma l \succ \sigma r$ for each rule $l \longrightarrow r \in R$ and ground substitution σ.*

Then $(E \cup R', \succ)$ is a ground complete ordered rewriting system.

Proof The conditions ensure that $(E \cup R', \succ)$ is terminating. Since E and R have no non-constant function symbols in common the only critical pairs are those of E or of R and hence are ground joinable. Thus $(E \cup R', \succ)$ is ground complete. □

As a corollary we see immediately that the combination of any of the theories we have considered above with new free function symbols (so R is empty) gives a ground complete ordered rewriting system.

If R and E are both proved terminating using the lexicographic path ordering or Knuth-Bendix ordering, and assuming that the operator precedences are consistent on $\Sigma \cap \Gamma$, we may obtain a total ordering $\succ$ by constructing a total operator precedence on $\Sigma \cup \Gamma$ which subsumes the two partial precedences. Thus we may combine any of the examples above with any such R.

4.11 AC with Two Rules

In this section we show how with a suitable choice of ordering two rules suffice for a ground complete AC rewriting system. Let E be

$$\begin{aligned} (x * y) * z &\longrightarrow x * (y * z) \\ x * y &= y * x \end{aligned}$$

and $>$ any monotonic ordering which is total on ground terms and satisfies for all ground terms x, y, z and all constants a, b

$$\begin{aligned} (x * y) * z &> x * (y * z) \\ x * y &> y * x \qquad \text{if } x > y \\ a &> b * x \qquad \text{if } a > b. \end{aligned}$$

We show that $(E, >)$ is ground complete. Notice that the ordering $>_2$ of the previous section has the required properties. (In fact it is not hard to see that any ordering with these properties is not well-founded).

We must first prove termination. Suppose that $s_1 \Longrightarrow s_2 \Longrightarrow \cdots$ is an infinite chain of rewrites. Since each equivalence class is finite it must contain a loop $s_i \Longrightarrow s_{i+1} \Longrightarrow \cdots \Longrightarrow s_{i+k} \Longrightarrow s_i$. But since $s \Longrightarrow t$ implies $s > t$ we have $s_i > s_{i+1} > \cdots > s_i$, which contradicts the definition of $>$. Thus $(E, >)$ is terminating.

To prove ground confluence we first observe

Lemma 11 *Let x and y be ground terms and let S_x be the multiset of all constants occurring in X. If $x =_E y$ then $S_x = S_y$.*

Proof S_x is invariant when applying the equations of E. □

Then we can prove

Theorem 1 *Let $(E, >)$ be as above. Then*

1. *If $w \in T_G$ then*
$$w \Longrightarrow^* a_1 * (a_2 * \cdots * (a_{n-1} * a_n) \ldots)$$
where $S_w = \{a_1, \ldots, a_n\}$ and $a_1 \leq a_2 \leq \cdots \leq a_n$, and this expression is irreducible.
2. *$(E, >)$ is a ground complete ordered rewriting system.*

Proof

1. The proof is by induction on $n = |S_w|$. If $n \leq 2$ the result is clear. Now by applying associativity we may assume $w \Longrightarrow^* a * v$ where $S_v = S_w - \{a\}$. By induction we may assume that v has the required form, so that in particular $v = b * u$, where $b \leq c$ for each $c \in S_u = S_v - \{b\}$. Now if $a \leq b$ we are done, so assume that $a > b$. Then
$$w \Longrightarrow^* a * (b * u) \Longrightarrow^* (b * u) * a \Longrightarrow^* b * (u * a).$$
Now by induction $u * a$ rewrites to the required form, $a_2 * (a_3 \cdots * a_n)$, and since $b < a$ and $b \leq c$ for each $c \in S_u$ we have $b \leq a_i$ for each $i = 2, \ldots, n$. So
$$w \Longrightarrow^* a_1 * (a_2 * (\cdots * a_n)),$$
where $a_1 = b$. It is clear that this expression is irreducible.

2. Suppose that $v =_E w$. Then $S_v = S_w$, so by part (i)

$$v \Longrightarrow^* a_1 * (a_2 * (\cdots * a_n))$$

and

$$w \Longrightarrow^* a_1 * (a_2 * (\cdots * a_n))$$

where $S_v = \{a_1, \ldots, a_n\}$ and $a_1 \leq \cdots a_n$. Thus v and w are joinable. Hence $(E, >)$ is locally ground confluent, and as it is terminating it is ground complete.

□

4.12 Abelian Groups

Let E be

$$\begin{aligned}
x * y &= y * x \\
x * (y * z) &= y * (x * z) \\
(x * y) * z &= x * (y * z) \\
x * i(x) &= 1 \\
1 * x &= x \\
x * 1 &= x \\
x * (i(x) * y) &= y \\
i(x * y) &= i(x) * i(y) \\
i(i(x)) &= x \\
i(1) &= 1
\end{aligned}$$

and $>$ any AC compatible ordering which orders the last seven equations from left to right for all ground terms x, y, and has

$$a_1 < i(a_1) < a_2 < i(a_2) < \cdots < a_n < i(a_n)$$

where the constants are $a_1, \ldots, a_n$. The lexicographic path ordering with precedence $* < i < a_1 < \ldots < a_n$ will do this. We shall show that $(E, >)$ is ground complete.

Unfortunately our automated ground confluence checking procedure fails in this case as it has to reduce arbitrary terms of the form $x * (y_1 * (y_2 * (\cdots (y_n * (i(x) * z)\ldots)$. But we may prove ground completeness using the technique of the previous example.

If a is a constant and t is a term we define the polarity of a in t as $p(a, \_) : \mathcal{T}_G \longrightarrow \mathbf{Z}$ inductively as

$$\begin{aligned}
p(a, a) &= 1 \\
p(a, b) &= 0 \\
p(a, i(x)) &= -p(a, x)) \\
p(a, s * t) &= p(a, s) + p(a, t)
\end{aligned}$$

where s and t are terms and b is any constant distinct from a. Thus for example $p(a, i(i(a) * (t * a))) = -p(t)$.

Lemma 12 *If x and y are ground terms with $x =_E y$ then for each i $p(a_i, x) = p(a_i, y)$.*

Proof It is easy to check that $p(a_i, x)$ is invariant under application of any of the equations. □

Now we have

Theorem 2 *Let $(E, >)$ be as above. Then*

1. *If $w \in \mathcal{T}_G$ and $w \neq_E 1$ then*

$$w \Longrightarrow^* e_1^{p_1} * e_2^{p_2} \cdots * e_n^{p_n},$$

(assumed associated to the right) where for each i we have $p_i = |p(a_i, w)|$, and $e_i = a_i$ if $p(a_i, w) \geq 0, e_i = i(a_i)$ if $p(a_i, w) < 0$. Furthermore each such expression is irreducible.

2. *$(E, >)$ is a ground complete ordered rewriting system.*

Proof

1. If $w \in \mathcal{T}_G$ then by applications of the last seven equations it is easy to see that

$$w \Longrightarrow^* e_1 * (e_2 * \cdots * e_n) \ldots)$$

where each $e_i \in \{a_j, i(a_j) \mid j = 1, \ldots, n\}$. Now as $a_1 < i(a_1) < a_2 < i(a_2) \cdots < a_n < i(a_n)$ an argument similar to the previous theorem shows that we may assume $e_1 \leq e_2 < \cdots < e_n$. Now applying (7) we see that each expression of the form $u * (a_i * (i(a_i) * z))$ reduces to $u * v$, and thus

$$w \Longrightarrow^* e_1^{p_1} * e_2 p_2 * \cdots * e_n^{p_n},$$

where each a_j is a_j or $i(a_j)$. Now by the lemma each $p_i = |p(a_i, w)|$. It is clear that this expression is irreducible.

2. Since the ordering is well-founded, $(E, >)$ is terminating.
To prove confluence suppose that $u =_E v$. We have by the lemma that $p(a_j, u) = p(a_j, v)$ for each j. But then by the first part u and v are joinable. Thus $(E, >)$ is locally ground confluent and hence ground confluent. □

Acknowledgements

This paper was written while the second author was at the Laboratory for Computer Science at MIT and the first author was visiting there. We would both like to acknowledge the generous hospitality of John Guttag.

References

[1] L. Bachmair, N. Dershowitz, D. Plaisted: *Completion Without Failure*, Proc. Coll. on Resolution of Equations in Algebraic Structures (1987).

[2] R.S. Boyer, J.S. Moore: *A Computational Logic Handbook*, Academic Press (1988).

[3] N. Dershowitz: *Termination of Rewriting*, Journal of Symbolic Computation (1987) 3, 69-116.

[4] G. Huet: *Confluent Reductions: Abstract properties and Applications to Term Rewriting Systems*, Journal ACM 27, 4 (1980), 797-821.

[5] G. Huet, D.C. Oppen: *Equations and Rewrite Rules - A Survey*, in: Formal Languages: Perspectives and Open Problems, R. Book (ed.), Academic Press (1982).

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

[7] J.-P. Jouannaud, H. Kirchner: *Completion of a Set of Rules Modulo a Set of Equations*, SIAM Journal of Computing 15 (1986), 1155-1194.

[8] D.E. Knuth, P.B. Bendix: *Simple Word Problems in Universal Algebras*, in: Computational Problems in Abstract Algebra, ed J. Leech, Pergamon 1970, 263-297

[9] U. Martin: *Doing Algebra with Reve*, Report UMCS-86-10-4, Dept. of Comp. Sci., University of Manchester (1986).

[10] U. Martin: *How to Choose the Weights in the Knuth Bendix Ordering*, in: Rewriting Techniques and Applications, LNCS 256, 1987, 42-53.

Completion Procedures as Transition Rules + Control

Extended Abstract

Pierre LESCANNE*
Centre de Recherche en Informatique de Nancy
LORIA
Campus Scientifique, BP 239,
54506 Vandœuvre-lès-Nancy, France
email: lescanne@crin.crin.fr

1 Completion procedures as sets of transitions rules

The interest of rewriting techniques in programming, algebraic and computer algebra specifications is well-known as is its ability to provide proof environments essentially based on completion procedures [FG84,KS83,Fag84,GG89,Les83]. In this introduction, I suppose the reader is familiar with this concept. Indeed my goal is not to present it, but to study how methods developed essentially with a theoretical purpose, namely proving completeness can be used to present simple short and understandable programs. This paper can also be seen as a set of exercises on the use of a functional language to program high level procedures and as a bridge between theory and practice. Readers who want to get more introductory informations are invited to look at Dershowitz survey *Completion and its Applications* [Der87]. The completion procedure is a method used in equational logic to built from a set of identities an equivalent canonical set of rewrite rules i.e., a confluent, noetherian and interreduced set of rules used to compute normal forms. If one tracks the history of the presentation of this procedure, one can notice different methods of description. In their seminal paper [KB70] Knuth and Bendix describe essentially the procedure in natural language, in [Hue80] Huet uses a style similar to Knuth's book, *The Art of Computer Programming*, in [Hue81] he uses a program structured by while loops, in [Kir84] H. Kirchner uses a recursive procedure and in [For84] Forgaard proposes an organization of the procedure around tasks to be performed. In the following, a completion will be seen as a set of *inference rules* or more precisely a set of *transition rules* acting on a data structure. The idea of using *inference rules* when dealing with completion is not new and leads to the beautiful proofs of completeness proposed by Bachmair and Dershowitz [Bac87,BDH86,BD87] and their followers [GKK88,Gan87]. The *completeness* is the ability of the procedure to eventually generate a proof by normalization or a rewrite proof for every equational theorem. In this paper, I want to show how this description leads to actual, nice and elegant programs when used as a programming method and I illustrate that by an actual *CAML* implementation [FOR87a]. Actually the inference rules one considers in completion are specific in the sense that they transform a t-uple of

*The research was sponsored by PRC "programmation avancée et outils de l'intelligence artificielle", CNRS and INRIA

objects into a t-uple of objects with the same structure. This is why I refer to them as *transition rules.* Thus the basic components of such a procedure are four,

- **a data structure** on which the transition rules operate, sometimes called the universe,
- **a set of transition rules,** that are the basic operations on the data structure,
- **a control,** that is a description of the way the transition rules are invoked,
- **a toolkit** that is shared by several completion procedures.

When one wants to describe a specific completion procedure, usually one uses the following method. First one chooses the data structure, then one chooses transition rules and often at the same time the control. The toolkit is something that remains from one procedure to the other in many cases, it was partly borrowed from the "*CAML* Anthology" [FOR87b] as a natural attempt to reuse pieces of codes already debugged and tested. As we will see the control is typically data driven and can be easily expressed by rewrite rules. In the following, the influence of these choices on the efficiency of the procedure will be illustrated through three refinements of the Knuth-Bendix completion procedures and a two unfailing completions. Indeed, we will see how, starting from a naive implementation of the completion, improvements can be obtained by changing the data structure and consequently the transition rules and the control. These ideas are implemented in my program *ORME.* A long version of this abstract can be found in [Les89].

In this extended abstract I am going to give only two controls. One which is relatively simple is called KB-completion since its reflects the completion presented in the original paper due to Knuth and Bendix. Another called the ANS-completion performs optimizations namely with respect to simplifications and computation of critical pairs. In the full paper, I present other completions that are intermediate between the KB-completion and the ANS-completion. I also present unfailing completion.

2 The KB-completion

This completion tries to stick to the control given by Knuth in his paper. The name of the inference rules are taken from Dershowitz [Der87] except that some rules are naturally extended. *Normalize* computes the normal form of all the identities in E, *Delete_all* removes form E all the trivial identities, *Orient_all* transforms all the identities that can be oriented into rules, *Deduction* computes all the critical pairs of R and put them into E.

3 The ANS-completion

The *KB-completion* can be improved into respects. It should privilege simplifications. This is the role of the set S where the rules enter as soon as they are oriented. There first task is to simplify all the other rules. Since the *KB-completion* computes at each loop, all the critical pairs between all the rules in R. It can be improved by computing the critical pairs between only two rules in R. Thus R is split into four parts. T is a waiting room where rules that have already been used for simplification wait before being used for computation of critical pairs. The computation of critical pairs is done by three sets C, N and A. C contains zero or one rule; if it contains no rule this means one is not processing critical pair

```
let rec KB_Completion ordering = COMP
where COMP (R, E) =

    match E with

        [] -> (R, []) (* success *)

    | (_::_) ->  COMP (Delete_all
                       (Normalize
                       (Delete_all
                       (Deduction
                       ((repeat Collapse)
                       ((repeat Compose)
                       ((Orient_all ordering) (R, E))))))))
```

Figure 1: *The KB-completion*

computation, if C contains one rule, this rules will be confronted with all the rules in N for computation of critical pairs. After this confrontation, the rules go from N into A. When N is empty, the computation of critical pairs ends with the computation of the critical pairs resulting form the superposition of the rule in C on itself.

- **E** the set of identities,
- **S** the set of simplifiers, where identities enter after being oriented into rules,
- **T** is a set of rules coming from S and waiting to enter C,
- **C** is a set that contains one or zero rule and whose critical pairs have to be computed with one in N,
- **N** is the part of R whose critical pairs have not been computed with C, but whose critical pairs with $A \cup N$ have been computed,
- **A** is a set whose critical pairs with $A \cup N \cup C$ have been computed.

The transition rules are adapted to work with this new data structure and three new rules are introduced. *Deduction* computes the critical pairs between the smallest rule in N and the rule in C. *Internal_Deduction* computes the critical pairs obtained by superposing the rule(s) in C on itself (themselves). *A_C2N* moves the rules in A and C into N to start a new "loop" of computation of critical pairs, according to the emptyness of the components of the data structure. The procedure has now clearly six parts, namely *success, simplification, orientation, deduction, internal deduction* and *beginning of a new loop* of computation of critical pairs. Typically this cannot be easily structured by a while loop because at each time the iteration on the computations of the critical pairs can be interrupted by a simplification. A data driven control is then much better (Figure 2).

```
let rec ANS_Completion crit ordering  =  COMP
where rec COMP (A,N,C,T,S,E) =
      let ORIENT = Orientation crit ordering in
match (N,C,T,S,E) with
  (_,[],[],[],[]) -> (A,N,C,T,S,E)

 | (_,_,_,_::_,_) -> (COMP (A',N',C',T'@S',[],E')
             where A',N',C',T',S',E' =
                  repeat_list [Simpl_left_A_by_S;Simpl_right_A_by_S;
                               Simpl_left_N_by_S;Simpl_right_N_by_S;
                               Simpl_left_C_by_S;Simpl_right_C_by_S;
                               Simpl_left_T_by_S;Simpl_right_T_by_S] (A,N,C,T,S,E))

 | (_,_,_,[],_::_) -> let  A',N',C',T',S',E' = clean_E(A,N,C,T,S,E) in
            (match E' with
                [] ->  COMP(A',N',C',T',S',E')
              | _ ->  (try(COMP(ORIENT (A',N',C',T',S',E')))
                       with _ -> try Deduction(A',N',C',T',S',E')
                                 with _ -> try Internal_Deduction(A',N',C',T',S',E')
                                           with _ -> failwith "non orientable equation"))

| (_::_,_::_,_,[],[]) -> COMP (Deduction(A,N,C,T,S,E))

 | ([],_::_,_,[],[]) -> COMP (A_C2N crit (Internal_Deduction (A,N,C,T,S,E)))

 | (_,[],_::_,[],[]) -> (COMP([],A@N,[r],T',[],[])
          where r,T' = least crit T)
```

Figure 2: *The ANS-completion*

4 Conclusion

The main idea of the approach presented here is to decompose the algorithm into basic actions and to describe some kind of abstract machine where these actions as the instructions. This may remind either Forgaard's description of *REVE* based on tasks [For84], or *ERIL* [Dic85] where users have access to the basic operations or Huet's first description [Hue80]. The rigorous and formal approach of this paper gives precision and concision and leads to a better understanding of the program and therefore to a better confidence. Since one is closer to the proof of completeness there are more chance that the implementation is both correct and complete. Another important aspect of this approach is that modifications and improvements are easily done. Basically this level of programming allows to study very high level optimizations [Ben82] and when an efficient procedure is discovered, a low level implementation can be foreseen. Here I made many implementation choices that still can be discussed, but since they are rather explicit this discussion is easy and changes can be quickly made. However, as well illustrated by the *ER_completion* compared with the *unfailing completion*, it should also be noticed that the complexity of the completion

procedures described by transition rules increases exponentially with the size of the number of components of the data structure, which implies that some kind of modularity has to be found.

Another interesting aspect of the programming by transition rules is that simple snapshots exist, therefore the process can easily be stopped after each rule and restarted on this state. Thus backtracking on the choice of the orderings as implemented in *REVE* or any kind of backtracking to insure fairness [DMT88], backups, breakpoints or integration of an already completed rewrite system in another equational theory can be easily handled.

But this approach does not address low level controls, for instance refinements that computes one critical pair at a time. This indeed requires a level of granularity in the actions that cannot be handled by the current form of the data structure. Attempts to fully formalize all the tasks, including substitutions and unifications should answer this question [GS88,HJ88].

All the procedures presented in this paper are a part of *ORME*, a set of *CAML* procedures that were run for completing a set of examples. Both the programs and the examples can be obtained from the author upon request.

I would like to remember Alain Laville from whom I received wise advices on how to use *CAML*. I thank Leo Bachmair, Françoise Bellegarde, Jieh Hsiang, Jean-Pierre Jouannaud, Jean-Luc Remy, Michael Rusinowitch and the EURECA group at CRIN who provided me with stimulating discussions and Gérad Huet who gave me access to the *CAML Anthology*.

References

[Bac87] L. Bachmair. *Proof methods for equational theories.* PhD thesis, University of Illinois, Urbana-Champaign, 1987. Revised version, August 1988.

[BD87] L. Bachmair and N. Dershowitz. Completion for rewriting modulo a congruence. In *Proceedings Second Conference on Rewriting Techniques and Applications*, Springer Verlag, Bordeaux (France), May 1987.

[BDH86] L. Bachmair, N. Dershowitz, and J. Hsiang. Orderings for equational proofs. In *Proceedings Symp. Logic in Computer Science*, pages 346–357, Boston (Massachusetts USA), 1986.

[Ben82] J. L. Bentley. *Writing Efficient Programs.* Prentice Hall, 1982.

[Der87] N. Dershowitz. Completion and its applications. In *Proceedings Colloquium on Resolution of Equations in Algebraic Structures*, MCC, 3500 West Balconies Center Drive, Austin, Texas 78759-6509, May 4-6 1987.

[Dic85] A.J.J. Dick. ERIL equational reasoning: an interactive laboratory. In B. Buchberger, editor, *Proceedings of the EUROCAL Conference*, Springer-Verlag, Linz (Austria), 1985.

[DMT88] N. Dershowitz, L. Marcus, and A. Tarlecki. Existence, uniqueness and construction of rewrite systems. *SIAM J. Comput.*, 17(4):629–639, August 1988.

[Fag84] F. Fages. *Le système KB. Manuel de référence, présentation et bibliographie, mise en œuvre.* Technical Report, Greco de Programmation, Bordeaux, 1984.

[FG84] R. Forgaard and J. Guttag. *REVE: A term rewriting system generator with failure-resistant Knuth-Bendix.* Technical Report, MIT-LCS, 1984.

[For84] R. Forgaard. *A program for generating and analyzing term rewriting systems.* Technical Report 343, Laboratory for Computer Science, Massachusetts Institute of Technology, 1984. Master's Thesis.

[FOR87a] Projet FORMEL. *CAML: the reference Manuel.* Technical Report, INRIA-ENS, March 1987.

[FOR87b] Projet FORMEL. *The CAML Anthology.* July 1987. Internal Document.

[Gan87] H. Ganzinger. A completion procedure for conditional equations. In *Proceedings 1st International Workshop on Conditional Term Rewriting Systems*, pages 62–83, Springer-Verlag, 1987.

[GG89] S.J. Garland and J.V. Guttag. An overview of LP, the Larch Prover. In N. Dershowitz, editor, *Proceedings of the 3rd Conference on Rewriting Techniques and Applications, Chapel Hill, North Carolina, USA*, pages 137–151, Springer-Verlag, April 1989. Lecture Notes in Computer Science, volume 355.

[GKK88] I. Gnaedig, C. Kirchner, and H. Kirchner. Equational completion in order-sorted algebras. In M. Dauchet and M. Nivat, editors, *Proceedings of the 13th Colloquium on Trees in Algebra and Programming*, pages 165–184, Springer-Verlag, Nancy (France), 1988.

[GS88] J. Gallier and W. Snyder. Complete sets of transformations for general E-unification. *Theoretical Computer Science*, 1988.

[HJ88] J. Hsiang and J.-P. Jouannaud. General E-unification revisited. In C. Kirchner and G. Smolka, editors, *Proceeding of UNIF'88*, page 51, CRIN Report 89R38, 1988.

[Hue80] G. Huet. *A complete proof of correctness of the Knuth-Bendix completion algorithm.* Technical Report 25, INRIA, August 1980.

[Hue81] G. Huet. A complete proof of correctness of the Knuth and Bendix completion algorithm. *Journal of Computer Systems and Sciences*, 23:11–21, 1981.

[KB70] D. Knuth and P. Bendix. *Simple Word Problems in Universal Algebra*, pages 263–297. Pergamon Press, 1970.

[Kir84] H. Kirchner. A general inductive completion algorithm and application to abstract data types. In R. Shostak, editor, *Proceedings 7th international Conference on Automated Deduction*, pages 282–302, Springer-Verlag, Lecture Notes in Computer Science, 1984.

[KS83] D. Kapur and G. Sivakumar. Experiments with an architecture of RRL, a rewrite rule laboratory. In *Proceedings of an NSF Workshop on the Rewrite Rule Laboratory*, pages 33–56, 1983.

[Les83] P. Lescanne. Computer experiments with the REVE term rewriting systems generator. In *Proceedings, 10th ACM Symposium on Principles of Programming Languages*, pages 99–108, ACM, 1983.

[Les89] P. Lescanne. Completion procedures as transition rules + control. In M. Diaz and F. Orejas, editors, *TAPSOFT'89*, page xx, Springer-Verlag, Lecture Notes in Computer Science, 1989.

Section 3:
Foundations

W. Nutt: Unification in Modular Categories

F. Baader: Unification in Commutative Theories, Hilbert's Basis Theorem, and Gröbner Bases

C Kirchner, F. Klay: A Note on Syntacticness

A. Bockmayr: On the Decidability of the Unification Problem

W. Nutt: The Unification Hierarchy is Undecidable

Unification in Modular Categories

Werner Nutt

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

Abstract

Recently, two classes of equational theories that are intimately related have been investigated for their unification properties. Monoidal theories as introduced in [Nutt 1988] are defined in terms of signatures and identities. To every monoidal theory corresponds a characteristic algebraic structure, a semiring, such that solving unification problems in the theory is equivalent to solving linear equation systems in the characteristic semiring.

Baader [1988] called a theory commutative if the category of free algebras and homomorphisms is semi-additive. He proved unification properties of such theories using mainly categorical arguments.

It has been proved [Nutt 1988] that every monoidal theory is commutative and that every commutative theory can be turned into a monoidal theory by a signature transformation.

Rydeheard and Burstall [1985] reformulated the basics of unification theory for categories. In this paper we show that the basic results on monoidal and commutative theories can already be obtained in a purely categorical framework. The appropriate generalization of these theories are modular categories which are defined as having biproducts and being generated by a single object.

As for monoidal theories, it turns out that for every modular category there exists a semiring determining the structure of unification problems. More precisely, the category of matrices over this semiring is equivalent to the given category. As a consequence, unification problems can be treated using methods from linear algebra as they were already applied to monoidal theories.

On the other hand, we can use this fact to make sure that during our excursion to category theory we didn't generalize too far. Since every semiring is the characteristic semiring of some monoidal theory, we know that every unification problem occurring in a modular category can as well occur as an equational unification problem.

References

Baader, F. (1988). "Unification in Commutative Theories". Presented at the 2nd Workshop on Unification, Val d'Ajol, France. Also in Kirchner, C. (ed.), *Special Issue on Unification, Journal of Symbolic Computation.*

Nutt, W. (1988). "Unification in Monoidal Theories". Presented at the 2nd Workshop on Unification, Val d'Ajol, France. Also *SEKI Report*, Universität Kaiserslautern, forthcoming.

Rydeheard, D. E., Burstall, R. M. (1985). A Categorical Unification Algorithm. *Proceedings of the Workshop on Category Theory and Computer Programming, LNCS* **240**.

Unification in Commutative Theories, Hilbert's Basis Theorem and Gröbner Bases

Franz Baader

Institut für Mathematische Maschinen und Datenverarbeitung (Informatik) 1, Universität Erlangen-Nürnberg, Martensstraße 3, 8520 Erlangen, F.R.G.

Extended Abstract

1. Unification in Commutative Theories

Unification in the empty theory (which is unitary) plays an important rôle in automated theorem proving, term rewriting and logic programming. Generalizations to E-unification usually require that E is finitary. A finitary theory most used in this context is the theory of abelian monoids, i.e. the theory of an associative, commutative binary operation with a neutral element. Unification algorithms for this theory and unification algorithms for the theory of abelian groups and for the theory of idempotent abelian monoids have similar structures. They depend on the following properties which the three theories have in common:

(1.1) There is a binary associative, commutative operation.

(1.2) The finitely generated free objects are direct products of the free objects in one generator.

(1.3) Unifiers correspond to solutions of systems of linear equations in a certain semiring. For the theory of abelian monoids we have semiring $\mathbb{N}$, for the theory of abelian groups $\mathbb{Z}$ and for the theory of idempotent abelian monoids the 2-element boolean semiring $\mathcal{B}$.

Category theory can be used as an appropriate level of abstraction to express the common structures. Let E be an equational theory and let C(E) be the category which has the finitely generated E-free algebras $F_E(X)$ as objects and the homomorphisms between these objects as morphisms. An E-unification problem can be written as a pair $< \sigma = \tau >_E$ of morphisms $\sigma, \tau: F_E(I) \rightarrow F_E(X)$ in the category C(E) and an E-unifiers of the unification problem $< \sigma = \tau >_E$ is a morphism δ such that $\sigma\delta = \tau\delta$ (see Baader (1988)).

We can now characterize a class of equational theories E by properties of the corresponding category C(E): A theory E is called commutative iff C(E) is a semiadditive category. Semiadditive categories are categories which have a zero object, all binary coproducts and allow an associative, commutative binary operation "+" on morphisms distributing with the composition of morphisms. It can be shown that the binary coproducts in semadditive categories are also products (this corresponds to 1.2 above) and that the operation on morphisms induces an associative, commutative binary implicit operation "$*$" in the class of all finitely generated E-free algebras (vid. 1.1) (see Baader (1988) or Herrlich-Strecker (1973) for the exact definition and properties of semiadditive categories).

Werner Nutt (Nutt (1988)) observed that commutative theories are (modulo a translation of the signature) what he calls monoidal theories and that unification in a monoidal theory E may be reduced to solving linear equations in a certain semiring S(E) (vid. 1.3). In the categorical framework this semiring can be obtained as follows:

Let **1** be an arbitrary set of cardinality 1. The definition of semiadditive categories yields

that $\hom(F_E(\mathbf{1}),F_E(\mathbf{1}))$ with addition "+" and composition as multiplication is a semiring, which shall be denoted by S(E). Any $F_E(x)$ is isomorphic to $F_E(\mathbf{1})$ and for $|X| = n$, $F_E(X)$ is n-th power and copower of $F_E(\mathbf{1})$ with projections p_x and injections u_x. A morphism σ: $F_E(X) \to F_E(Y)$ is uniquely determined by the $|X|\times|Y|$-matrix $M_\sigma = (u_x \sigma p_y)_{x\in X, y\in Y}$ and the entries of M_σ may all be considered as elements of S(E). Hence all morphisms of C(E) can be written as matrices over the semiring S(E). Addition and composition of morphisms correspond to addition and multiplication of matrices over S(E), i.e. $M_{\sigma+\tau} = M_\sigma + M_\tau$ and $M_{\sigma\delta} = M_\sigma \cdot M_\delta$.

The characterizations of unification type unitary for unification without constants (finitary for unification with constants) for commutative theories, which were given in Baader (1989), can now be translated into algebraic conditions on S(E):

(2.1) A commutative theory E is unitary w.r.t unification without constants iff the corresponding semiring S(E) satisfies the following condition: For any n, $m \geq 1$ and any pair M_σ, M_τ of m×n-matrices over S(E) the set

$$U(M_\sigma,M_\tau) := \{ \underline{x} \in S(E)^n; M_\sigma \underline{x} = M_\tau \underline{x} \}$$

is a finitely generated right S(E)-semimodule, i.e. there are finitely many $\underline{x}_1, ..., \underline{x}_r \in S(E)^n$ such that $U(M_\sigma,M_\tau) = \{ \underline{x}_1 s_1 + ... + \underline{x}_r s_r; s_1, ..., s_r \in S(E) \}$.

(2.2) Let E be a commutative theory which is unitary w.r.t unification without constants. Then E is finitary w.r.t. unification with constants, if the following condition holds in S(E):

Let A be any m×n-matrices over S(E) and let $\underline{b}$ be any element of $S(E)^m$. Then the set $M := \{ \underline{x} \in S(E)^n; A\underline{x} = \underline{b} \}$ is a finite union of cosets of the (finitely generated) right S(E)-semimodule $N := \{ \underline{x} \in S(E)^n; A\underline{x} = 0 \}$, i.e. there exist finitely many $\underline{m}_1, ..., \underline{m}_k \in S(E)^n$ such that $M = \{ \underline{m}_i + \underline{n}; \underline{n} \in N \text{ and } 1 \leq i \leq k \}$.

(2.3) Let E be a unitary commutative theory such that S(E) is a ring. Then E is unitary w.r.t. unification with constants

2. The Theory AGnHC of Abelian Groups with n Commuting Homomorphisms.

It is easy to see that S(AGnHC) is isomorphic to the ring $\mathbb{Z}[X_1,...,X_n]$, i.e. the polynomial ring over $\mathbb{Z}$ in the (commuting) indeterminates $X_1, ..., X_n$. To establish Condition 2.1, we have to consider systems of homogeneous linear equations in $\mathbb{Z}[X_1,...,X_n]$, i.e. systems $f_{1i}x_1 + ... + f_{ki}x_k = 0$ ($i = 1, ..., s$), where the coefficients f_{ij} and the desired solutions are elements of $\mathbb{Z}[X_1,...,X_n]$. The set of solutions $\underline{I} \subseteq (\mathbb{Z}[X_1,...,X_n])^k$ is a $\mathbb{Z}[X_1,...,X_n]$-module, which is finitely generated by Hilbert's Basis Theorem and the fact that $\mathbb{Z}$ is a noetherian ring. Thus AGnHC is unitary w.r.t. unification without constants. Since $\mathbb{Z}[X_1,...,X_n]$ is a ring, AGnHC is also unitary w.r.t. unification with constants (see 2.3). This argument does not yield an AGnHC-unification algorithm, because we still do not know how to solve linear equations in $\mathbb{Z}[X_1,...,X_n]$ effectively. Buchberger (1985) describes an effective method, which constructs finitely many generators of the solutions of a single equation $f_1x_1 + ... + f_kx_k = 0$, where the f_i and the desired solutions are elements

of $K[X_1,...,X_n]$ for a field K. This method may also be used for $\mathbb{Z}[X_1,...,X_n]$, but the proof of its correctness becomes more involved (see Baader (1989a)).

3. The Theory AGnH of Abelian Groups with n Non-commuting Homomorphisms.

It is easy to see that S(AGnH) is isomorphic to the ring $\mathbb{Z}\langle X_1,...,X_n\rangle$, i.e. the polynomial ring over $\mathbb{Z}$ in the non-commuting indeterminates X_1, ..., X_n. Unfortunately, for $n \geq 2$ this ring is not noetherian and the membership problem for finitely generated two-sided ideals is undecidable. Fortunately, we are not interested in two-sided ideals, but only in right ideals. The solutions of a homogeneous equation $f_1x_1 + ... + f_rx_r = 0$ are only closed under right multiplication and the inhomogeneous equation $f_1x_1 + ... + f_rx_r = f_0$ has a solution iff f_0 is a member of the right ideal generated by f_1, ..., f_r. Though, for $n \geq 2$, $\mathbb{Z}\langle X_1,...,X_n\rangle$ is not even right noetherian (i.e. there are right ideals in $\mathbb{Z}\langle X_1,...,X_n\rangle$, which are not finitely generated), the set of solutions of a homogeneous equation $f_1x_1 + ... + f_rx_r = 0$ is a finitely generated right $\mathbb{Z}\langle X_1,...,X_n\rangle$-semimodule and the membership problem for finitely generated right ideals is decidable in $\mathbb{Z}\langle X_1,...,X_n\rangle$. This is proved in Baader (1989a) by construction of a Gröbner base algorithm for finitely generated right ideals in $\mathbb{Z}\langle X_1,...,X_n\rangle$. The computation of Gröbner bases for finitely generated right ideals in $K\langle X_1,...,X_n\rangle$, where K is a field, is very easy (Mora (1986)). For $\mathbb{Z}\langle X_1,...,X_n\rangle$ one has to be much more careful to obtain a terminating algorithm.

Conclusion

The categorical reformulation of E-unification allows to characterize the class of commutative theories by properties of the category C(E) of finitely generated E-free objects. The definition of semiadditive categories provides an algebraic structure on the morphism sets, which can be used to obtain algebraic characterizations of the unification types. This shows the connection between unification in commutative theories and equation solving in linear algebra. The very common syntactic approach to equational unification, which only uses the defining axioms, is thus replaced by a more semantic approach, which works with algebraic properties of the defined algebras. Hence unification algorithms for commutative theories can be derived with the help of well-known algebraic methods.

References

Baader, F. (1988). Unification in Commutative Theories. To appear in *J. Symbolic Computation.*

Baader, F. (1989). Unification Properties of Commutative Theories: A Categorical Treatment. Proceedings of the Summer Conference on Category Theory and Computer Science, Manchester (England).

Baader, F. (1989a). Unification in Commutative Theories, Hilbert's Basis Theorem and Gröbner Bases. Submitted to J. ACM.

Buchberger, B. (1985). Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory. In Bose, N. K. (Ed.). *Recent Trends in Multidimensional System Theory.*

Mora, F. (1986). Gröbner Bases for Non-Commutative Polynomial Rings. Proceedings of the AAECC3. *Springer Lec. Notes Comp. Sci.* **228.**

Nutt, W. (1988). Talk at the Second Workshop on Unification, Val d' Ajol (France).

A Note on Syntacticness

Claude Kirchner
LORIA & INRIA

Francis Klay
LORIA & CRIN

BP 239
54506 Vandœuvre-Les-Nancy Cedex
France
E-mail: {ckirchner,klay}@loria.crin.fr

July 30, 1989

Abstract

We present in this note a necessary and sufficient condition for a collapse free theory to be syntactic.

1 Preliminaries

We are using the standard notations in equational logic [4] and unification [6,8]. $T(F, X)$ denotes the free F-algebra on X, where F is any finite set of operator symbols. If σ is a substitution then $I(\sigma)$ is the set of variables of the codomain of σ. Given a subset V of X, we define $\sigma \leq_A \sigma'$ $[V]$ iff $\exists \sigma''$, $\forall x \in V, \sigma'(x) =_A \sigma''.\sigma(x)$. The qualification $[V]$ is omitted when $V = X$. If A is a set of identities and U is a unificand then **SU(A,U)** is the set of all A-unifiers of U, **CSU(A,U)** is a complete set of A-unifiers of U and **MCSU(A,U)** a minimal complete set of A-unifiers of U. We say that the unificand U_2 is **A-dependent** of U_1, or equivalently that U_1 **A-extends** U_2, iff for any complete set of A-solutions Σ of U_1, $\Sigma_{|Var(U_2)}$ is a complete set of A-unifiers of U_2 and $U_1 \Rightarrow_A U_2$ means that U_1 A-extends U_2. $|B|$ denotes as usually the cardinal of a set B.

Definition 1 *Let W be a set of "protected" variables, f, g in F and $V = \{v_1, \ldots, v_{n+m}\}$ where n and m are the arities of f and g, and v_i's are distinct variables. Then $U^W(f, g, V)$ (resp. $\mu U^W(f, g, V)$) denotes a complete (resp. minimal complete) set of idempotent unifiers, away from W, of the equation:*

$$f(v_1, \ldots, v_n) \doteq g(v_{n+1}, \ldots, v_{n+m})$$

We call these equations **general** and note them $Ge(f, g, V)$.

Example 1 *If we consider the set identities A reduced to the commutativity of the symbol $+$ then $\mu U^W(+, +, V)$ is the minimal complete set of unifiers of the equation $v_1 + v_2 \doteq v_3 + v_4$ and*

it is equal to:

$$\mu U^W(+,+,V) = \left\{ \begin{array}{l} \left\{ \begin{array}{l} v_1 \mapsto x_1 \\ v_2 \mapsto x_2 \\ v_3 \mapsto x_1 \\ v_4 \mapsto x_2 \end{array} \right. \\ \\ \left\{ \begin{array}{l} v_1 \mapsto x_1 \\ v_2 \mapsto x_2 \\ v_3 \mapsto x_2 \\ v_4 \mapsto x_1 \end{array} \right. \end{array} \right.$$

Definition 2 *Let A be a set of identities, $V = \{v_1, v_2, \ldots\}$ and $W = \{w_1, w_2, \ldots\}$ two countable sets of distinct new variables. For all f, g in F we define the sets of substitutions $\Sigma(A, V, W)$ and $\Sigma(f, g, A, V, W)$ as follows:*

- *If $f \neq g$ then:*

$$\Sigma(f, g, A, V, W) = \{(v_i \mapsto t_i)_{i=1..m} \mid (f(t_1, \ldots, t_n) = g(t_{n+1}, \ldots, t_m)) \in A\}$$

- *If $f = g$ then:*

$$\begin{array}{rl} \Sigma(f, f, A, V, W) = & \{(v_i \mapsto t_i)_{i=1..2n} \mid (f(t_1, \ldots, t_n) = f(t_{n+1}, \ldots, t_{2n})) \in A\} \bigcup \\ & \{(v_i \mapsto w_i)_{i=1..n} \cup (v_{n+i} \mapsto w_i)_{i=1..n}\} \end{array}$$

$$\Sigma(A, V, W) = \bigcup_{(f,g) \in F \times F} \Sigma(f, g.A, V, W)$$

Note that by definition the $\Sigma(f, g, A, V, W)$ are subsets of A-solutions of the general equation $Ge(f, g, V)$.

1.1 Definitions of syntactic theories

Syntactic theories are very interesting because we may deduce automatically a matching algorithm [7] or, under some conditions, a unification procedure for a theory from its identities [5]. We suppose that the theories considered are collapse free, which means that any A-equal terms are both non variable. A collapse identity is an equation of the form $x = t$ where x is a variable and t a non variable term. Every presentation of a collapse free theory does not contain collapse identities [2]. This excludes identities like idempotency ($x + x = x$) or involution ($-(-(x)) = x$).

In order to define syntactic theories, we first need to introduce some terminology. Let

$$A(f, f') = \{\{l = r\} \in A \mid l(\varepsilon) = f \text{ and } r(\varepsilon) = f'\}$$

be the subset of A whose top symbols are f and f'.

A set A of identities is **resolvent** iff for all terms $t = f(t_1, \ldots, t_n)$ and $t' = f'(t'_1, \ldots, t'_p)$ we have:

$$t =_A t' \Leftrightarrow \left\{ \begin{array}{l} \bullet \left\{ \begin{array}{l} f = f' \text{ and} \\ \forall j \in [1..n], t_j =_A t'_j \end{array} \right. \\ \textit{or} \\ \bullet \left\{ \begin{array}{l} \exists l = r \in A(f, f'), \exists \sigma \text{ such that} \\ \quad -\forall j \in [1..n], t_j =_A \sigma(l/j) \\ \quad \text{and} \\ \quad -\forall k \in [1..p], t'_k =_A \sigma(r/k) \end{array} \right. \end{array} \right.$$

An equational theory is said **syntactic**[1] if it is generated by a finite and resolvent set of identities. It can be characterized precisely by the form of the A-equality proof of two terms:

Proposition 1 *[5] Let A be a set of collapse free identities. A is resolvent iff for any terms $t = f(t_1, \ldots, t_n)$ and $t' = f(t'_1, \ldots, t'_n)$ such that $t =_A t'$ there is a proof*

$$t = s_0 \vdash_{m_1} s_1 \vdash \ldots \vdash_{m_k} s_k = t'$$

with at the most one of the m_j equal to ε for $j \in [1..k]$.

Example 2 *If we consider the set of identities A reduced to the identity of commutativity of the symbol $+$ then A is resolvent since:*

$$t_1 + t_2 =_A t_3 + t_4 \Leftrightarrow \begin{cases} \bullet\ t_1 =_A t_3 \text{ and } t_2 =_A t_4 \\ \text{or} \\ \bullet\ t_1 =_A t_4 \text{ and } t_2 =_A t_3 \end{cases}$$

2 Condition for syntacticness

Lemma 1 *Let $e = (f(t_1, \ldots, t_n) \doteq g(t_{n+1}, \ldots, t_m))$ an equation and W be a set of variables such that $Var(e) \subseteq W$. The following transformation rule is sound and complete in the theory A:*

$$\frac{f(t_1, \ldots, t_n) \doteq g(t_{n+1}, \ldots, t_m)}{\bigvee_{\sigma \in \mu U^W(f,g,V)} \bigwedge_{i=1..m} \sigma v_i \doteq t_i}$$

Proof: Using variable abstraction which is known to be a sound and complete transformation, we get

$$f(t_1, \ldots, t_n) \doteq g(t_{n+1}, \ldots, t_m) \Leftarrow_A \begin{cases} v_1 & \doteq & t_1 \\ & \vdots & \\ v_m & \doteq & t_m \\ f(v_1, \ldots, v_n) & \doteq & g(v_{n+1}, \ldots, v_m) \end{cases}$$

Since replacing a part of a system by an A-dependent unificand preserves A-dependance [6], we get

$$f(t_1, \ldots, t_n) \doteq g(t_{n+1}, \ldots, t_m) \Leftarrow_A \bigvee_{\sigma \in \mu U^W(f,g,V)} \begin{cases} v_1 & \doteq & t_1 \\ & \vdots & \\ v_m & \doteq & t_m \\ v_1 & \doteq & \sigma(v_1) \\ & \vdots & \\ v_m & \doteq & \sigma(v_m) \end{cases}$$

By definition of W, $(V \cup I(\sigma)) \cap \mathrm{Var}(e) = \emptyset$, thus we can replace v_i's by their values and then suppress the equations $v_i \doteq \sigma(v_i)$ and we get

$$f(t_1, \ldots, t_n) \doteq g(t_{n+1}, \ldots, t_m) \Leftarrow_A \bigvee_{\sigma \in \mu U^W(f,g,V)} \begin{cases} \sigma(v_1) & \doteq & t_1 \\ & \vdots & \\ \sigma(v_m) & \doteq & t_m \end{cases}$$

□

[1] The definition of syntactic theory has been extended to almost syntactic in [3]

Definition 3 *If E is a collapse free theory generated by A then $\mathcal{U}$ and $\mathcal{A}$ are the following set of identities:*

$$\begin{aligned}\mathcal{U} &= \bigcup_{(f,g)\in F\times F} \bigcup_{\sigma\in\mu U(f,g,V)} \sigma(f(v_1,\ldots,v_n)) = \sigma(g(v_{n+1},\ldots,v_m)) \\ \mathcal{A} &= \mathcal{U}\cup A\end{aligned}$$

Lemma 2 *The set of identities $\mathcal{A}$ is such that $(=_A) = (=_{\mathcal{A}})$.*

Proof: $=_{\mathcal{A}}\supseteq=_A$ is obvious by definition of $\mathcal{A}$ and $=_{\mathcal{A}}\subseteq=_A$ stands because for all $l = r \in \mathcal{A}$, $l =_A r$ by definition of $\mathcal{A}$. □

Lemma 3 *$\mathcal{A}$ is a resolvent set of identities.*

Proof: The transformation rule presented in lemma 1 can be rewritten as

$$\frac{f(t_1,\ldots,t_n) \doteq g(t_{n+1},\ldots,t_m)}{\bigvee_{f(s_1,\ldots,s_n)=g(s_{n+1},\ldots,s_m)\in\mathcal{U}} \bigwedge_{i\in[1..m]} s_i \doteq t_i}$$

Since $\mathcal{U}\subseteq\mathcal{A}$ and since this transformation is sound and complete, we get

$$\begin{array}{l}\sigma(f(t_1,\ldots,t_n)) =_{\mathcal{A}} \sigma(g(t_{n+1},\ldots,t_m)) \\ \Leftrightarrow \\ \left\{\begin{array}{l} f = g \text{ and } \forall i\in[1..n],\ \sigma(t_i) =_{\mathcal{A}} \sigma(t_{n+i}) \\ \text{or} \\ \exists f(s_1,\ldots,s_n) = g(s_{n+1},\ldots,s_m)\in\mathcal{A},\ \exists\rho,\ \forall i\in[1..m],\ \rho(s_i) =_{\mathcal{A}} \sigma(t_i)\end{array}\right.\end{array}$$

and this is exactly the definition of a resolvent set of identities. □

Lemma 4 *Let E be a collapse free theory. A is a resolvent presentation of E iff for all f, g in F and for all σ in $U(f,g,V)$ there is a proof:*

$$\sigma f(v_1,\ldots,v_n) \vdash * \dashv \sigma g(v_{n+1},\ldots,v_m)$$

with at most one step in ε.

Proof:

$\Rightarrow$

Since A is a resolvent set of identities, by the proposition 1, there is a proof

$$\sigma f(v_1,\ldots,v_n) \vdash * \dashv \sigma g(v_{n+1},\ldots,v_m)$$

with at most one step in ε.

$\Leftarrow$

The notation are those of Figure 1. Since we can define the A-unifier $\rho = (v_i \mapsto u_i)_{i=1..m}$ of $Ge(f,g,V)$ from any E-equality (2), there is a proof of (2) with at most one step in ε because:

- Since $U(f,g,V)$ is a complete set of A-solutions of $Ge(f,g,V)$, there is a substitution σ in $U(f,g,V)$ such that $\rho =_E \varphi\sigma$.

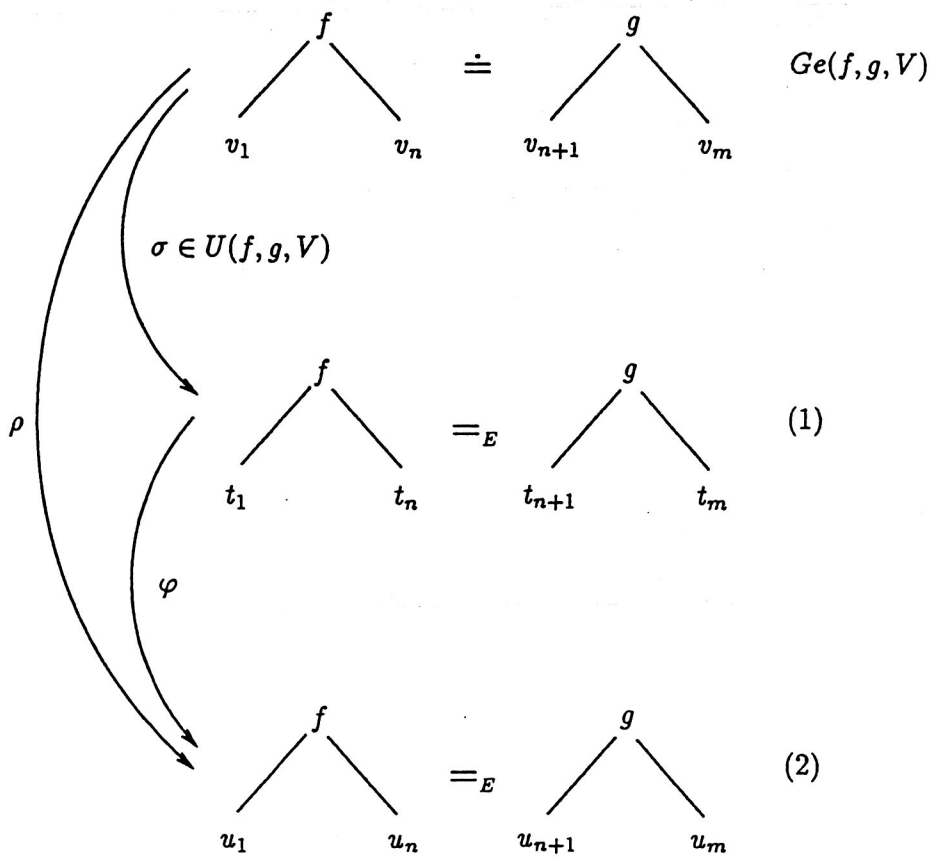


Figure 1: Composition diagram

- As there is a proof

$$\sigma f(v_1, \ldots, v_n) \vdash * \dashv \sigma g(v_{n+1}, \ldots, v_m)$$

with at most one step in ε, there is a proof

$$\varphi\sigma f(v_1, \ldots, v_n) \vdash * \dashv \varphi\sigma g(v_{n+1}, \ldots, v_m)$$

with at most one step in ε. Since $\rho =_E \varphi\sigma$, there is a proof

$$\rho f(v_1, \ldots, v_n) \vdash * \dashv \rho g(v_{n+1}, \ldots, v_m)$$

with at most one step in ε and by the proposition 1 we deduce that A is resolvent.

□

Lemma 5 *Let E be a collapse free theory. A is a resolvent presentation of E iff:*

$$\forall f, g \in F \times F,\ \forall \sigma \in U(f, g, V),\ \exists \rho \in \Sigma(f, g, A, V, W),\ \sigma \geq_E \rho\ [V]$$

Proof: We have to prove (by the proposition 1) that:

for some terms $t = f(t_1, \ldots, t_n)$ and $t' = g(t_{n+1}, \ldots, t_m)$ such that $t =_E t'$ there is a proof:

$$t = s_0 \vdash_{m_1} s_1 \vdash \ldots \vdash_{m_k} s_k = t'$$

with at most one of the m_j equal to ε for $j \in [1..k]$ iff

$$\forall \sigma \in U(f,g,V),\ \exists \rho \in \Sigma(f,g,A,V,W),\ \sigma \geq_E \rho\ [V]$$

$\Rightarrow$
Let $\sigma \in U(f,g,V)$ with $\sigma = (v_i \mapsto t_i)_{i=1..m}$. By hypothesis the proof of $\sigma f(v_1, \ldots, v_n) =_E g(v_{n+1}, \ldots, v_m)$ has at most one step in ε.

- If no step occurs in ε then $f = g$ and $\forall i \in [1..n]$, $t_i =_E t_{n+i}$. In this case there is $\rho \in \Sigma(f,f,A,V,W)$ such that $\sigma \geq_E \rho\ [V]$ since:

$$\rho = (v_i \mapsto w_i)_{i=1..n} \cup (v_{n+i} \mapsto w_i)_{i=1..n} \quad and \quad \sigma = (w_i \mapsto t_i)_{1=1..n} \circ \rho$$

- If one step occurs in ε then there is $f(u_1, \ldots, u_n) = g(u_{n+1}, \ldots, u_m)$ in A and φ such that $\forall i \in [1..m]$, $\varphi u_i = t_i$. Since $\rho = (v_i \mapsto u_i)_{i=1..m}$ is in $\Sigma(f,g,A,V,W)$ we have $\sigma =_E \varphi\rho\ [V]$.

$\Leftarrow$
Suppose that for all $\sigma \in U(f,g,V)$ there is $\rho \in \Sigma(f,g,A,V,W)$ such that $\sigma \geq_E \rho\ [V]$. By definition of $\Sigma(f,g,A,V,W)$ we have:

$$\rho f(v_1, \ldots, v_n) \vdash_\varepsilon \rho g(v_{n+1}, \ldots, v_m)$$

or

$$\rho f(v_1, \ldots, v_n) = \rho f(v_{n+1}, \ldots, v_{2n})$$

as $\sigma \geq_E \rho\ [V]$, we deduce that for all $\sigma \in U(f,g,V)$ there is a proof

$$\sigma f(v_1, \ldots, v_n) \vdash * \dashv \sigma g(v_{n+1}, \ldots, v_m)$$

with at most one step in ε. This means by the lemma 4 that A is resolvent. □

Lemma 6 *Let E be a collapse free theory. If there are $f, g \in F$ such that $|\mu U(f,g,V)| = \infty$ then E is not syntactic.*

Proof: Let $f, g \in F$ be symbols of function such that $|\mu U(f,g,V)| = \infty$. If we suppose that there is a finite and resolvent presentation A then since $\Sigma(f,g,A,V,W)$ is finite we can deduce by lemma 5 and the pigeon hole principle that:

$$\exists \sigma_1, \sigma_2 \in \mu U(f,g,V),\ \exists \rho \in \Sigma(f,g,A,V,W),\ \sigma_1 \neq \sigma_2\ and\ \sigma_1 \geq_E \rho\ and\ \sigma_2 \geq_E \rho$$

As ρ is an A-unifier of $Ge(f,g,V)$ there is $\sigma_3 \in \mu U(f,g,V)$ such that $\rho \geq_E \sigma_3$. By transitivity we have $\sigma_1 \geq_E \sigma_3$ and $\sigma_2 \geq_E \sigma_3$. Since $\mu U(f,g,V)$ is a minimal complete set of A unifiers we deduce $\sigma_1 = \sigma_2 = \sigma_3$. This contradicts the fact that $\sigma_1 \neq \sigma_2$. □

Definition 4 *Let E be a theory and two substitions σ and ρ.*

- $\sigma \equiv_E \rho\ [V]$ *iff* $\sigma \geq_E \rho\ [V]$ *and* $\rho \geq_E \sigma\ [V]$.
- $\sigma >_E \rho [V]$ *iff* $\sigma \geq_E \rho\ [V]$ *and* $\sigma \not\equiv_E \rho\ [V]$.

Lemma 7 *Let t and t' be two terms with their variables in V. If the minimal complete set of A-solutions $MCSU(A, t \doteq t')$ does not exist then there is in all $CSU(A, t \doteq t')$ a decreasing chain:*

$$\sigma_1 \geq_E \sigma_2 \geq_E \ldots \geq_E \sigma_n \geq_E \ldots$$

in $CSU(A, t \doteq t')$ without lower bound in $CSU(A, t \doteq t')$.

Proof: (this proof is inspired by [1]) We proceed by contradiction. Suppose that all decreasing chains in $CSU(A, t \doteq t')$ have a lower bound in $CSU(A, t \doteq t')$ (i.e. for all decreasing chains a $\geq_E$-minimal element exists). Let M' be the set of all $\geq_E$-minimal elements of $CSU(A, t \doteq t')$. If we define the set M by $M = M'/_{\equiv_E}$ then:

- $M \subseteq CSU(A, t \doteq t')$ by definition of M.
- Since all decreasing chains in $CSU(A, t \doteq t')$ have a lower bound in $CSU(A, t \doteq t')$ we have:

  $$\forall \rho \in CSU(A, t \doteq t'),\ \exists \sigma \in M,\ \rho \geq_E \sigma\ [V]$$

  and since $CSU(A, t \doteq t')$ is a complete set of A-unifiers of $t \doteq t'$ we deduce:

  $$\forall \rho \in SU(A, t \doteq t'),\ \exists \sigma \in M,\ \rho \geq_E \sigma\ [V]$$

- Since M' is a set of $\geq_E$-minimal elements we have:

  $$\forall \sigma_1, \sigma_2 \in M,\ \sigma_1 \not\geq_E \sigma_2\ [V]$$

  and since M is equal to $M'/_{\equiv_E}$ we have:

  $$\forall \sigma_1, \sigma_2 \in M,\ \sigma_1 \geq_E \sigma_2 \ \Rightarrow\ \sigma_1 = \sigma_2\ [V]$$

Since the three properties above are the definition of a minimal complete set of A-solutions this contradicts the hypothesis of this lemma. □

Lemma 8 *If the minimal complete set of A-solutions $MCSU(A, t \doteq t')$ does not exist then there is in all $CSU(A, t \doteq t')$ a properly decreasing chain:*

$$\sigma_1 >_E \sigma_2 >_E \ldots >_E \sigma_n >_E \ldots$$

in $CSU(A, t \doteq t')$ without lower bound in $CSU(A, t \doteq t')$.

Proof: By the lemma 7 we know there is a decreasing chain without lower bound in $CSU(A, t \doteq t')$. Since the lower bound does not exist in $CSU(A, t \doteq t')$ we can always build a properly decreasing subchain in $CSU(A, t \doteq t')$ without lower bound in $CSU(A, t \doteq t')$. □

Lemma 9 *Let C be a properly decreasing chain of substitution without lower bound and ρ any substitution.*

$$C = \sigma_1 >_E \sigma_2 >_E \ldots >_E \sigma_n >_E \ldots$$

1. *If $\exists \sigma_i \in C,\ \sigma_i \equiv_E \rho$ then there is a subchain C' of C without lower bound such that $\forall \sigma'_j \in C',\ \sigma'_j \not\geq_E \rho$.*
2. *If $\exists \sigma_i \in C,\ \rho >_E \sigma_i$ then there is a subchain C' of C without lower bound such that $\forall \sigma'_j \in C',\ \sigma'_j \not\geq_E \rho$.*

Proof: In both case consider the subchain:

$$C' = \sigma_{i+1} >_E \sigma_{i+2} >_E \ldots >_E \sigma_{i+n} >_E \ldots$$

In case 1 it is clear that C' satisfies the required condition.
In case 2, suppose that $\exists j > i, \sigma_j \geq_E \rho$. Then, $\sigma_j \geq_E \rho >_E \sigma_i$ by hypothesis, which contradicts the fact that C is strictly decreasing.

□

Lemma 10 *Let E be a collapse free theory. If there are $f, g \in F$ such that $\mu U(f, g, V)$ does not exist then E is not syntactic.*

Proof: Since $\mu U(f, g, V)$ does not exist, lemma 8 allows us to say that in all $U(f, g, V)$ there is a properly decreasing chain without lower bound in $U(f, g, V)$. We define this chain as:

$$C = \sigma_1 >_E \sigma_2 >_E \ldots >_E \sigma_n >_E \ldots$$

We now suppose that there is a finite and resolvent presentation A of E. Since $\Sigma(f, g, A, V, W)$ is finite we deduce a subchain C' of C as follows:

1. $\Sigma_0 = \Sigma(f, g, A, V, W)$, $C_0 = C$, i=0
2. **if** $\Sigma_i = \emptyset$ **then** $C' = C_i$ **end**
3. $\Sigma_{i+1} = \Sigma_i - \{\rho_i\}$
4. **if** $\exists \sigma \in C_i,\ \sigma \geq_E \rho_i$
   then deduce C_{i+1} from C_i and ρ_i by using the lemma 9
   else $C_{i+1} = C_i$
5. $i = i + 1$ **goto** 2.

Since C_{i+n} is a subchain of C_i we have:

$$\forall \sigma'_j \in C_i,\ \sigma'_j \not\geq_E \rho_i \implies \forall \sigma'_j \in C_{i+n},\ \sigma'_j \not\geq_E \rho_i$$

and we deduce:

$$\forall \sigma_j \in C',\ \forall \rho \in \Sigma(f, g, A, V, W),\ \sigma_j \not\geq_E \rho$$

therefore $\exists \sigma \in U(f, g, V),\ \forall \rho \in \Sigma(f, g, A, V, W),\ \sigma \not\geq_E \rho$. This leads to a contradiction by the lemma 5. □

As a consequence of the previous lemmas, we get the following result:

Theorem 1 *Let E be a collapse free theory, $\forall (f, g) \in F \times F,\ \mu U(f, g, V)$ exists and is finite iff E is syntactic.*

As an immediate consequence, we get

Corollary 1 *Every collapse free and finitary unifying theory is syntactic.*

3 Applications

3.1 AC-theories are syntactic

If $+$ is an associative and commutative operator then we obtain the following minimal complete set of A-solutions of the equation $v_1 + v_2 \doteq v_3 + v_4$ [9]:

$$\mu U(+,+,V) = \left\{ \begin{array}{llll} \left\{ \begin{array}{l} v_1 \mapsto u_1 \\ v_2 \mapsto u_2 \\ v_3 \mapsto u_1 \\ v_4 \mapsto u_2 \end{array} \right. & \left\{ \begin{array}{l} v_1 \mapsto u_1 \\ v_2 \mapsto u_2 \\ v_3 \mapsto u_1 \\ v_4 \mapsto u_2 \end{array} \right. & \left\{ \begin{array}{l} v_1 \mapsto u_1 + u_2 \\ v_2 \mapsto u_3 \\ v_3 \mapsto u_1 \\ v_4 \mapsto u_2 + u_3 \end{array} \right. & \left\{ \begin{array}{l} v_1 \mapsto u_1 \\ v_2 \mapsto u_2 + u_3 \\ v_3 \mapsto u_1 + u_3 \\ v_4 \mapsto u_2 \end{array} \right. \\ \left\{ \begin{array}{l} v_1 \mapsto u_1 + u_2 \\ v_2 \mapsto u_3 \\ v_3 \mapsto u_1 + u_3 \\ v_4 \mapsto u_2 \end{array} \right. & \left\{ \begin{array}{l} v_1 \mapsto u_1 \\ v_2 \mapsto u_2 + u_3 \\ v_3 \mapsto u_3 \\ v_4 \mapsto u_1 + u_2 \end{array} \right. & \left\{ \begin{array}{l} v_1 \mapsto u_1 + u_3 \\ v_2 \mapsto u_2 + u_4 \\ v_3 \mapsto u_1 + u_4 \\ v_4 \mapsto u_2 + u_3 \end{array} \right. & \end{array} \right.$$

Since we can compute the resolvent set of identities from $\mu U(f,f,V)$ we obtain the following finite and resolvent presentation:

$$\mathcal{A} = \left\{ \begin{array}{ll} u_1 + u_2 & = u_1 + u_2 \\ u_1 + u_2 & = u_2 + u_1 \\ (u_1 + u_2) + u_3 & = u_1 + (u_2 + u_3) \\ u_1 + (u_2 + u_3) & = (u_1 + u_3) + u_2 \\ (u_1 + u_2) + u_3 & = (u_1 + u_3) + u_2 \\ u_1 + (u_2 + u_3) & = u_3 + (u_1 + u_2) \\ (u_1 + u_3) + (u_2 + u_4) & = (u_1 + u_4) + (u_2 + u_3) \end{array} \right.$$

3.2 A non-syntactic theory

Finally let us present a simple [2] (also called strict [6]) theory which is not syntactic:

$$A = \{fg(x) = f(x)\}$$

Such a theory is not syntactic since $\mu U(f,f,V)$ (the minimal complete set of A-solutions of the equation $f(v_1) \doteq f(v_2)$) is the infinite set:

$$\sigma_0 = \left\{ \begin{array}{lll} v_1 & \mapsto & x \\ v_2 & \mapsto & x \end{array} \right. , \sigma_1 = \left\{ \begin{array}{l} v_1 \mapsto x \\ v_2 \mapsto g(x) \end{array} \right. , \sigma_2 = \left\{ \begin{array}{l} v_1 \mapsto x \\ v_2 \mapsto g(g(x)) \end{array} \right. , \cdots \sigma_n = \left\{ \begin{array}{l} v_1 \mapsto x \\ v_2 \mapsto g^n(x) \end{array} \right. , \cdots$$

Of course by theorem 1 one gets many other examples of collapse free and non-syntactic theories like nullary theories [1].

4 Conclusion

Syntactic theories have been introduced in unification theory in order to automaticaly build unification algorithms [5]. We have studied in this paper some of the properties of syntactic theories and in particular we have shown that a resolvant presentation of a syntactic theory can be obtained as unifier instances of special kinds on equations called general. This gives a better understanding of what the syntactic theories are and in particular it allows to characterize them accordingly to the unification type of the general equations. This opens among other the

following questions on which we are now working: how to compute minimal complete set of solutions of general equations, how to prove the termination of the unification algorithm which is obtained from the resolvant set of identities, is it possible to extend our results to collapse theories?

Acknowledgements: This work has been partly supported by the GRECO de programmation du CNRS. We would like to thank Hélène Kirchner, E. Domenjoud, P. Marchand for fruitful discussions concerning this work.

References

[1] F. Baader. Characterizations of unification type zero. In N. Dershowitz, editor, *Proceedings of the 3rd Conference on Rewriting Techniques and Applications, Chapel Hill, North Carolina, USA*, pages 2–14, Springer-Verlag, April 1989. Lecture Notes in Computer Science, volume 355.

[2] H-J. Bürckert, A. Herold, and M. Schmidt-Schauß. On equational theories, unification and decidability. In P. Lescanne, editor, *Proceedings of the Second Conference on Rewriting Techniques and Applications*, Springer-Verlag, Bordeaux (France), May 1987.

[3] H. Comon. Unification et disunification. Théories et applications. Thèse de l'Institut Polytechnique de Grenoble, 1988.

[4] G. Huet and D. Oppen. Equations and rewrite rules: a survey. In R. Book, editor, *Formal Language Theory: Perspectives and Open Problems*, pages 349–405, Academic Press, New-York, 1980.

[5] C. Kirchner. Computing unification algorithms. In *Proceeding of the first symposium on Logic In Computer Science, Boston (USA)*, pages 206–216, 1986.

[6] C. Kirchner. Méthodes et outils de conception systématique d'algorithmes d'unification dans les théories équationnelles. Thèse d'état de l'Université de Nancy I, 1985.

[7] T. Nipkow. Proof transformation for simple equational theories. In H-J. Bürckert and W. Nutt, editors, *Proceedings of UNIF'89, third international workshop on unification*, Lambrecht (FR Germany), June 1989.

[8] J. Siekmann. Unification theory. *Journal of Symbolic Computation*, 3-4(7):207–274, 1989. Special issue on unification. Part one.

[9] M.E. Stickel. A unification algorithm for associative-commutative functions. *Journal of the Association for Computing Machinery*, 28:423–434, 1981.

On the Decidability of the Unification Problem

Alexander Bockmayr (e-mail: bockmayr@ira.uka.de)
SFB 314, Universität Karlsruhe, P.O.Box 6980, D-7500 Karlsruhe 1, F.R.Germany

Unification is a fundamental operation in a number of fields of computer science including theorem proving, logic programming, and natural language processing. One of the major theoretical questions is the decidability of the unification problem.

Let Σ be a signature and E a set of Σ-equations (see e.g. [Huet/Oppen 80]).
A system of Σ-equations

$$<t_1 = u_1, \dots, t_n = u_n>,\ n \geq 1, \text{ with terms } s_i, t_i \in T(\Sigma,X)$$

is called *E-unifiable,* iff there is a substitution $\sigma: X \rightarrow T(\Sigma,X)$, such that

$$\sigma(t_i) \equiv_E \sigma(u_i) \text{ for } i = 1,\dots,n.$$

Here $\equiv_E$ denotes the congruence on $T(\Sigma,X)$ generated by E.
We say that the *unification problem is decidable for E*, iff we can decide for any system of Σ-equations $<t_1 = u_1, \dots, t_n = u_n>$ whether it is E-unifiable.

In this note we investigate the decidability of the unification problem from the viewpoint of mathematical logic. The key to our approach is the following proposition:

PROPOSITION. Two terms $t, u \in T(\Sigma,X)$ are E-unifiable iff the existential closure $\exists(t = u)$ of the first-order formula $t = u$ is valid in the first-order theory E, i.e. $E \models \exists(t = u)$.

This proposition shows that unification problems correspond to certain first-order formulas.
In mathematical logic the following classification of formulas is common:

DEFINITION. Let $\mathcal{K}$ be a class of Σ-algebras.
The *elementary theory* of $\mathcal{K}$ is the set of all closed formulas in first-order predicate logic with equality that are valid in $\mathcal{K}$. Any such formula can be represented in prenex disjunctive normal form

$$Q_1x_1 \dots Q_mx_m ((t_{11} = \neq u_{11} \wedge \dots \wedge t_{1j_1} = \neq u_{1j_1}) \vee \dots\dots \vee (t_{q1} = \neq u_{q1} \wedge \dots \wedge t_{qj_q} = \neq u_{qj_q})),$$

where each Q_i is a universal or existential quantifier, $x_1, \dots, x_m$ are variables and each atomic formula has the form $t_{ij} = u_{ij}$ or $t_{ij} \neq u_{ij}$ with terms $t_{ij}, u_{ij} \in T(\Sigma,X)$.
The *universal theory* of $\mathcal{K}$ is the collection of those closed formulas valid in $\mathcal{K}$ that are of the form

$$\forall x_1 \dots \forall x_m ((t_{11} = \neq u_{11} \wedge \dots \wedge t_{1j_1} = \neq u_{1j_1}) \vee \dots\dots \vee (t_{q1} = \neq u_{q1} \wedge \dots \wedge t_{qj_q} = \neq u_{qj_q})).$$

The *existential theory* of $\mathcal{K}$ is the collection of those closed formulas valid in $\mathcal{K}$ that are of the form

$$\exists x_1 \dots \exists x_m ((t_{11} = \neq u_{11} \wedge \dots \wedge t_{1j_1} = \neq u_{1j_1}) \vee \dots\dots \vee (t_{q1} = \neq u_{q1} \wedge \dots \wedge t_{qj_q} = \neq u_{qj_q})).$$

The *positive theory* of $\mathcal{K}$ is the collection of those closed formulas valid in $\mathcal{K}$ that are of the form

$$Q_1x_1 \ldots Q_mx_m ((t_{11} = u_{11} \wedge \ldots \wedge t_{1j_1} = u_{1j_1}) \vee \ldots\ldots \vee (t_{q1} = u_{q1} \wedge \ldots \wedge t_{qj_q} = u_{qj_q})),$$

The *diophantine theory* of $\mathcal{K}$ is the collection of those closed formulas valid in $\mathcal{K}$ that are of the form

$$\exists x_1 \ldots \exists x_m (t_1 = u_1 \wedge \ldots \wedge t_n = u_n)$$

The *equational theory* of $\mathcal{K}$ is the collection of those closed formulas valid in $\mathcal{K}$ that are of the form

$$\forall x_1 \ldots \forall x_m (t = u)$$

There are numerous results in the literature on the decidability and undecidability of these fragments, which seem to be widely unknown in unification theory. We may profit from these results using the following immediate consequence of the above proposition.

COROLLARY. Let E be a set of Σ-equations. The unification problem for E is decidable if and only if the diophantine theory of the class of models of E is decidable.

Now we give some applications. The first theorem, based on [Vazhenin 74] and [Rozenblat 85], shows how adding a constant symbol to the signature may affect the decidability of the unification problem.

THEOREM. The unification problem for the theory

$$\{(x*y)*z = x*(y*z),\ (x^{-1})^{-1} = x,\ x = x*x^{-1}*x,\ x^{-1}*x*y^{-1}*y = y^{-1}*y*x^{-1}*x\ \}$$

of *inverse semigroups* is decidable in the signature $\Sigma = < *, {}^{-1}, a >$ and undecidable in the signature $\Sigma' = < *, {}^{-1}, a, b >$.

The next result establishes a connection between the decidability of the word problem and the decidability of the unification problem. In general, the decidability of the equational theory does not imply the decidability of the diophantine theory, even in theories defined by a canonical term rewriting system [Bockmayr 87, Heilbrunner/Hölldobler 87]. In *locally finite* varieties however, i.e. in equational theories for which every finitely generated model is finite, we can deduce from [Vazhenin/Rozenblat 83] the following theorem.

THEOREM. In a locally finite variety E over a finite signature Σ the unification problem is decidable if the word problem is decidable. If moreover E is finitely axiomatizable, the word problem is decidable.

EXAMPLE. The varieties

$$AI = \{ (x*y)*z = x*(y*z),\ x*x = x \}$$

of *idempotent semigroups* and

$$ACI = \{ (x*y)*z = x*(y*z),\ x*y = y*x,\ x*x = x \}$$

of *semilattices* are locally finite.

Therefore, these varieties have a decidable word and unification problem. In fact, they even have a decidable positive theory, whereas the elementary theory is undecidable [Vazhenin/Rozenblat 83].

Now we present some results from group theory.
Consider the signature

$$\Sigma = < * , ^{-1}, e , a_1,..., a_n >, n \geq 1.$$

[Makanin 83] showed that unification is decidable in the theory

$$\{ (x*y)*z = x*(y*z), e*x = x, x^{-1}*x = e \}$$

of groups on Σ.
Later he proved the decidability of the positive theory and the universal theory of a free group [Makanin 85]. The decidability of the elementary theory of a free group seems still to be open.
For the decidability of the elementary theory of a variety of groups commutativity plays an important role. It is well-known that the variety of abelian groups has a decidable elementary theory [Szmielew 55]. [Zamyatin 78] showed that every variety of groups that contains at least one non-abelian group has an undecidable elementary theory.
What about the unification problem in non-abelian varieties of groups ?
The axiom of commutativity can be weakened in several ways.
As an abbreviation we use the commutator

$$[x,y] =_{def} x^{-1}y^{-1}x\ y.$$

A group G is abelian iff

$$[x,y] = e, \text{ for all } x, y \in G.$$

A group G is *nilpotent of class 2* , iff

$$[[x,y], z] = e, \text{ for all } x, y, z \in G,$$

and *metabelian* iff

$$[[x,y], [u,v]] = e \text{ for all } x, y, u, v \in G.$$

Obviously, an abelian group is nilpotent of class 2, and a nilpotent group of class 2 is metabelian.
The next results follow from [Roman'kov 79] and [Repin 85].

THEOREM.

1. In the theory of abelian groups of signature Σ, the unification problem is decidable.
2. In the theory of nilpotent groups of class 2 of signature Σ, the unification problem for equations in one unknown is decidable.
3. In the theory of metabelian groups of signature Σ, the unification problem is undecidable.

REFERENCES.

[Bockmayr 87] A. Bockmayr: A note on a canonical theory with undecidable unification and matching problem. J. Autom. Reas. 3 (1987), 379 - 381

[Heilbrunner/Hölldobler 87] S. Heilbrunner, S. Hölldobler: The undecidability of the unification and matching problem for canonical theories. Acta Inform. 24 (1987), 157-171

[Huet/Oppen 80] G. Huet, D. C. Oppen: Equations and Rewrite Rules, A Survey.
in: Formal Language Theory (Ed. R. V. Book), Academic Press 1980

[Makanin 83] G. S. Makanin: Equations in a free group. Math.USSR Izv. 21 (1983), 483 - 546

[Makanin 85] G. S. Makanin: Decidability of the universal and positive theories of a free group. Math.USSR Izv. 25 (1985), 75 - 88

[Repin 85] N. N. Repin: The solvability problem for equations in one unknown in nilpotent groups. Math.USSR Izv. 25 (1985), 601 - 618

[Roman'kov 79] V. A. Roman'kov: Equations in free metabelian groups. Sib. J. Math. 20,1 (1979), 469 - 471

[Rozenblat 85] B. V. Rozenblat: Diophantine theories of free inverse semigroups. Sib. J. Math. 26,6 (1985), 860 - 864

[Szmielew 55] Elementary properties of abelian groups. Fund. Math. 41 (1955), 203 - 271

[Vazhenin 74] Y. M. Vazhenin: On the elementary theory of free inverse semigroups. Semigroup Forum 9 (1974), 189 - 195

[Vazhenin/Rozenblat 83] Y. M. Vazhenin, B. V. Rozenblat: On positive theories of free algebraic systems. Sov. Math. (Izv. VUZ) 27,3 (1983), 88 - 91

[Zamyatin 78] A. P. Zamyatin: A non-abelian variety of groups has an undecidable elementary theory. Algebra and Logic 17 (1978), 13 - 17

The Unification Hierarchy is Undecidable

WERNER NUTT

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

1 Introduction

In unification theory, equational theories are classified according to the existence and cardinality of minimal complete solution sets for equation systems. For unitary, finitary, and infinitary theories minimal complete solution sets always exist and are singletons, finite, or possibly infinite sets, respectively. In nullary theories, minimal complete sets do not exist for some equation systems. These classes form the unification hierarchy.

Although it is widely believed that it is not possible to decide from a finite presentation where a given equational theory resides in the unification hierarchy no proof of this fact exists in the literature. We give such a proof using undecidability results from group theory. Moreover, we show that the classes of nullary and infinitary theories are not even semi-decidable.

2 Basic Definitions and Notation

If nothing else is said, we assume standard definitions and notation of unification theory and group theory (cf. [Bürckert et al. 1989, Rotman 1973]).

A **finite presentation** of an equational theory is a pair $\mathcal{E} = (\Sigma, E)$ consisting of a signature Σ and a finite set E of identities between Σ-terms. By abuse of notation we will identify a presentation with its corresponding theory.

Siekmann [1978] classified equational theories according to the existence and cardinality of minimal sets of unifiers. Let $\mathcal{E}$ be a theory, then:

$\mathcal{E} \in \mathcal{U}_1$	iff	minimal sets of unifiers exist and have at most one element ($\mathcal{E}$ is **unitary**)
$\mathcal{E} \in \mathcal{U}_\omega$	iff	finite minimal sets of unifiers exist and $\mathcal{E} \notin \mathcal{U}_1$ ($\mathcal{E}$ is **finitary**)
$\mathcal{E} \in \mathcal{U}_\infty$	iff	minimal sets of unifiers exist and $\mathcal{E} \notin \mathcal{U}_1 \cup \mathcal{U}_\omega$ ($\mathcal{E}$ is **infinitary**)
$\mathcal{E} \in \mathcal{U}_0$	iff	there is some equation system for which no minimal set of unifiers exists ($\mathcal{E}$ is **nullary**).

Analogously, one defines a matching hierarchy with classes $\mathcal{M}_1$, $\mathcal{M}_\omega$, $\mathcal{M}_\infty$, and $\mathcal{M}_0$.

An **alphabet** is a finite set of symbols. For an alphabet Σ, we denote the set of words over Σ by Σ^* and the empty word by e. A **monoid presentation** (or Thue system) is a pair $\mathcal{M} = (\Sigma, \Delta)$ consisting

of an alphabet Σ and a finite set Δ of word identities over Σ. Group presentations are defined as special monoid presentations. By abuse of notation we will identify a presentation $\mathcal{M}$ with the monoid or group generated by it, speaking of the **finitely presented monoid** or **group** $\mathcal{M}$.

A monoid presentation $\mathcal{M}$ can be converted into the presentation $\mathcal{E}$ of an equational theory by considering every symbol of the alphabet as a unary function symbol and turning every word identity into a corresponding term identity. This transformation leaves identities invariant, that is if u, v are words that are converted into terms ux, vx, then $u =_{\mathcal{M}} v$ iff $ux =_{\mathcal{E}} vx$.

3 The Order Problem

Our proof that the unification types unitary, finitary, and infinitary are undecidable will be based on an undecidability result from group theory (cf. [Rotman 1973, Lyndon/Schupp 1977]).

If $\mathcal{G} = (\Sigma, \Delta)$ is a finitely presented group and $w \in \Sigma^*$, then $ord(w)$, the **order** of w, is defined as the least nonnegative integer n such that $w^n =_{\mathcal{G}} e$, if such an integer exists, and as $ord(w) = \infty$ otherwise.

Theorem 3.1. *There exists a finitely presented group $\mathcal{G} = (\Sigma, \Delta)$ such that the following problems are undecidable:*

INSTANCE: a word $w \in \Sigma^*$
QUESTION 1: is $ord(w) = 1$?
QUESTION 2: is $1 < ord(w) < \infty$?
QUESTION 3: is $ord(w) = \infty$?
Furthermore, the problem whether $ord(w) = \infty$ is not even semi-decidable.

4 Undecidability of Types Unitary, Finitary, and Infinitary

We reduce the problem of deciding the order of a group element to the problem of deciding the unification type of an equational theory.

Construction 4.1. Let $\mathcal{G} = (\Sigma, \Delta)$ be a finitely presented group. For every $w \in \Sigma^*$ we define a finitely presented monoid

$$\mathcal{G}_w := (\Sigma \cup \{f\}, \Delta \cup \{fw \doteq f\})$$

where f is a symbol not occurring in Σ. We denote the finitely presented theory corresponding to $\mathcal{G}_w$ as $\mathcal{E}_w$. □

The idea in defining $\mathcal{E}_w$ is best illustrated by the following example.

Example 4.2. Consider the $\mathcal{E}_w$-unification problem

$$\langle fx \doteq fy \rangle.$$

Then for every $k \in \mathbf{Z}$ the substitution $\sigma_k = [x/w^k y, y/y]$ is a unifier. If $ord(w) = \infty$, then all σ_k are

$\mathcal{E}_w$-different on $\{x, y\}$. If $ord(w) = n < \infty$, then $\sigma_0, \ldots, \sigma_{n-1}$ are pairwise $\mathcal{E}_w$-different on $\{x, y\}$, and $\sigma_k =_{\mathcal{E}_w, \{x,y\}} \sigma_{k+n}$ for $k \in \mathbf{Z}$.

Moreover, every $\mathcal{E}_w$-unifier of our problem is an $\mathcal{E}_w$-instance of some σ_k. No σ_l is an $\mathcal{E}_w$-instance of σ_k if $ord(w)$ is infinite and $k \neq l$, or if $ord(w)$ is finite and $0 \leq k < l < ord(w)$. Hence, the set $\{\sigma_k \mid k \in \mathbf{Z}\}$ is a minimal complete set of unifiers if $ord(w) = \infty$, and the set $\{\sigma_k \mid 0 \leq k < ord(w)\}$ is minimal and complete if $ord(w) < \infty$.

In summary, the $\mathcal{E}_w$-unification problem $\langle fx \doteq fy \rangle$ has a minimal complete set of unifiers of cardinality $ord(w)$. □

The same kind of argument that we used in the discussion of the above example applies to arbitrary unification and matching problems. To be more precise, one can show that for a single equation in $\mathcal{E}_w$ minimal complete sets of unifiers and matchers, respectively, have at most cardinality $ord(w)$ and that there exist such sets having exactly cardinality $ord(w)$. Since no theory $\mathcal{E}_w$ is nullary, the position of $\mathcal{E}_w$ in the hierarchy depends on the order of w.

Proposition 4.3.

- $\mathcal{E}_w \in \mathcal{M}_1 \iff \mathcal{E}_w \in \mathcal{U}_1 \iff ord(w) = 1$
- $\mathcal{E}_w \in \mathcal{M}_\omega \iff \mathcal{E}_w \in \mathcal{U}_\omega \iff 1 < ord(w) < \infty$
- $\mathcal{E}_w \in \mathcal{M}_\infty \iff \mathcal{E}_w \in \mathcal{U}_\infty \iff ord(w) = \infty$

Having established the correspondence between the order of group elements and the type of an equational theory, using Theorem 3.1 we can reduce the order problem for groups to the problem of locating a theory in the hierarchy.

Theorem 4.4.

- *$\mathcal{U}_\infty$ and $\mathcal{M}_\infty$ are not semi-decidable*
- *$\mathcal{U}_1$, $\mathcal{U}_\omega$, $\mathcal{U}_\infty$, $\mathcal{M}_1$, $\mathcal{M}_\omega$, and $\mathcal{M}_\infty$ are undecidable*

5 Undecidability of Type Nullary

We reduce the consistency problem for equational theories to the decision problem whether a theory is nullary. Since the disjoint combination of a consistent theory and a nullary theory is again nullary, by combination of an arbitrary theory $\mathcal{E}$ with a nullary theory we can construct a theory that is nullary if and only if $\mathcal{E}$ is consistent.

We start with a well-known result from equational logic, proved by Perkins [1967]. Recall that a theory $\mathcal{E}$ is inconsistent iff $x =_{\mathcal{E}} y$ for two distinct variables x and y, and consistent otherwise.

Lemma 5.1. *The following problem is not semi-decidable:*
INSTANCE: *a finitely presented equational theory $\mathcal{E}$*
QUESTION: *is $\mathcal{E}$ consistent ?*

Theorem 5.2. *$\mathcal{U}_0$ and $\mathcal{M}_0$ are not semi-decidable.*

Proof. Let $\mathcal{E}_0$ be a nullary theory. Then there exist an $\mathcal{E}_0$-unification problem Γ and a complete set of unifiers U_0 of Γ such that U_0 has no minimal complete subset.

Suppose $\mathcal{E}$ is a consistent theory. By a lemma due to Tidén [1986] and Schmidt-Schauß [1989], U_0 is also a complete set of unifiers of Γ for the disjoint combination of theories $\mathcal{E}^+ := \mathcal{E}_0 \uplus \mathcal{E}$, and U_0 has no minimal complete subset if $\mathcal{E}$ is consistent. Hence, $\mathcal{E}^+$ is nullary if $\mathcal{E}$ is consistent. Conversely, the combined theory $\mathcal{E}^+$ is inconsistent if $\mathcal{E}$ is inconsistent. Thus $\mathcal{E}^+$ is nullary iff $\mathcal{E}$ is consistent.

Were $\mathcal{U}_0$ semi-decidable, then in particular the problem whether for a given finitely presented theory $\mathcal{E}$ the combination $\mathcal{E}_0 \uplus \mathcal{E}$ is nullary would be semi-decidable. Hence, consistency of $\mathcal{E}$ would be semi-decidable, contradicting Lemma 5.1.

The proof that $\mathcal{M}_0$ is not semi-decidable is similar. □

Corollary 5.3. *$\mathcal{U}_0$ and $\mathcal{M}_0$ are not decidable.*

References

H.-J. Bürckert, A. Herold, and M. Schmidt-Schauß, "On Equational Theories, Unification, and Decidability," *Proc. 2nd Conference on Rewriting Techniques and Applications, LNCS* **256**, Springer, 1987, 240–250. Also to appear in C. Kirchner (ed.), *J. Symbolic Computation, Special Issue on Unification*, 1989.

C. Kirchner (ed.), *J. Symbolic Computation, Special Issue on Unification*, 1989, to appear.

R. C. Lyndon and P. E. Schupp, *Combinatorial Group Theory*, Springer, 1977.

W. Nutt, "The Unification Hierarchy is Undecidable," *SEKI Report SR-89-06*, Universität Kaiserslautern, 1989.

P. Perkins, "Unsolvable Problems for Equational Theories," *Notre Dame J. Formal Logic*, **8**, 1967, 175–185.

J. J. Rotman, *The Theory of Groups*, Allyn and Bacon, 1973.

M. Schmidt-Schauß, "Unification in a Combination of Arbitrary Disjoint Equational Theories,", to appear in C. Kirchner (ed.), *J. Symbolic Computation, Special Issue on Unification*, 1989.

J. H. Siekmann, *Unification and Matching Problems*, PhD. Thesis, University of Essex, 1978.

E. Tidén, "Unification in Combinations of Collapse-Free Theories with Disjoint Sets of Function Symbols," *Proc. 8th Conference on Automated Deduction, LNCS* **230**, Springer, 1986, 431–450.

Section 4:
Typed Unification

G. Smolka: Polymorphically Order Sorted Unification

L. Duponcheel: Typed Algebra

U. Waldmann: Unitary Unification in Order-Sorted Signatures

E. Domenjoud: AC-Unification Through Order Sorted AC1-Unification

Polymorphically Order-Sorted Unification

Gert Smolka
IBM Deutschland, IWBS
Postfach 80 08 80
7000 Stuttgart 80
West Germany
smolka@ds0lilog.bitnet

The type discipline of the logic programming language TEL [1, 2] combines the notion of parametric polymorphism known from the functional programming language ML with the notion of subsorting known from the specification language OBJ2. In TEL one can define sort functions such as lists or pairs and also define sorts as the nondisjoint union of other sorts. The model-theoretic semantics of TEL interprets sorts as sets of ground terms and sort functions as functions that are monotonic with respect to set inclusion.

To execute TEL-programs, constraints consisting of equations and memberships need to be solved, where a membership has the form $x{:}\sigma$ and constrains the variable x to the elements of the set denoted by the sort term σ. While the equations of a constraint can be easily solved with the usual unification rules, solving memberships turned out to be a tough problem whose decidability is open in general. The difficulties are due to the presence of subsorts and sort variables in memberships.

Since TEL-programs are required to be well-typed (a property that is checked automatically before execution), the constraints that need to be solved during execution also enjoy a certain weak well-typedness property. Fortunately, for such constraints there exists an efficient solution method [2].

For weakly well-typed constraints most of the conditions imposed by the memberships are redundant. In fact, if a membership contains no subsort it can be completely ignored. To avoid as much redundant sort computation as possible, our method does not work with the actual sort terms but with approximations retaining only the nonredundant information.

[1] G. Smolka, TEL (Version 0.9), Report and User Manual. SEKI Report SR-87-11, FB Informatik, Universität Kaiserslautern, West Germany, February 1988.

[2] G. Smolka, Logic Programming over Polymorphically Order-Sorted Types. Dissertation, FB Informatik, Universität Kaiserslautern, West Germany, May 1989.

Typed Algebra

Luc Duponcheel
Alcatel Bell Research Centre
Francis Wellesplein 1
B-2018 Antwerp
Belgium

August 4, 1989

Abstract

In this paper we introduce a theory of specifications in which types and equality are treated in a similar way. We present sound and complete rules for inferring types and equality. We discuss regularity of signatures and we present a unification algorithm for regular signatures.

1 Introduction

Many Sorted Algebra has been introduced as an alternative to (One Sorted) Algebra for dealing with specifications. In the Many Sorted Algebra framework equality declarations are well sorted. There is no difference between syntactic and semantic sorts.

We introduce Typed Algebra as an alternative to Many Sorted Algebra for dealing with specifications in a more flexible way. In the Typed Algebra framework equality declarations need not be well typed. There is a difference between syntactic and semantic types.

Our framework is similar to the ones presented in [1] and [2]. Our denotational semantics is simpler since we deal with total functions.

We present sound and complete rules for inferring types and equality. We discuss regularity of signatures and we present a unification algorithm for regular signatures.

Our framework is very well suited to deal with the problem of least sorts in Order Sorted Algebra. In contrast with the approach of [4] completion procedures generate critical equality declarations (for dealing with peaks occurring from functional overlaps) *and* critical type declarations (for dealing with peaks occurring from variable overlaps). Completion is not treated in this paper. For more information on completion we refer to [3].

2 Typed Algebra

2.1 S Algebra

2.1.1 Symbols

A *symbol set* S consists of

- a finite *type symbol set* T,
- a finite graded *function symbol set* $F = \sqcup_{i \in N} F_i$,
- an infinite *variable symbol set* X.

Type symbols are denoted by $\tau, \ldots$, function symbols by $g, \ldots$, and variable symbols by $x, \ldots$.

A function symbol g is either a *constant* function symbol c in F_0 or a function symbol f in $F_+ = \sqcup_{i \geq 1} F_i$ (say f in F_n).

2.1.2 T Sets

A T *set* is a set A together with

- a function $\tau \mapsto A_\tau$ from T to the powerset of A.

We use the abbreviation

- '$a.\tau$' for '$a \in A_\tau$'.

Every quotient set $[A]$ of a T set A is a T set using $\tau \mapsto [A_\tau]$ and is called a T *quotient set* of A.

A T *function* from A to B is a function h from A to B for which

- $a.\tau$ implies $h(a).\tau$.

The set of T functions from A to B is denoted by $A \rightarrow_T B$.

2.1.3 F Algebras

An F *algebra* is a set A together with

- a graded function $g \mapsto g_A$ from F to the graded function space on A.

An equivalence $\cong$ on an F algebra A is a *congruence* if

- $a_1 \cong b_1, \ldots, a_n \cong b_n$ implies $f_A(a_1, \ldots, a_n) \cong f_A(b_1, \ldots, b_n)$.

A quotient $[A]$ for a congruence on an F algebra A is an F algebra using

- $c_{[A]}() = [c_A()]$,
- $f_{[A]}([a_1], \ldots, [a_n]) = [f_A(a_1, \ldots, a_n)]$,

and is called an F *quotient algebra* of A.

2.1.4 Free F Algebras

With the function and variable symbols introduced in 2.1.1 we construct *terms.*

- every x is a term, called a variable (term),
- every $c()$ is a term, called a constant (term),
- every $f(t_1, \ldots, t_n)$ is a term whenever $t_1, \ldots, t_n$ are terms.

$V(t)$ is the *set of variable symbols* of the term t. If $V(t) = \emptyset$, then t is called a *ground* term.

$T(V)$ is the set of terms with $V(t) \subset V$. This is an F algebra in the usual way, called the *free* F algebra over V.

The following rules are sound and complete for inferring *syntactic equality* on $T(V)$.

- $\frac{}{\vdash x{=}x}$
- $\frac{}{\vdash c(){=}c()}$
- $\frac{\vdash s_1{=}t_1 \wedge \ldots \wedge \vdash s_n{=}t_n}{\vdash f(s_1,\ldots,s_n){=}f(t_1,\ldots,t_n)}$

2.1.5 S Algebras

An S *algebra* is an F algebra and a T set.

Every F quotient algebra $[A]$ of an S algebra A is an S algebra as defined in 2.1.2 and is called an S *quotient algebra* of A.

2.2 S Algebra

2.2.1 Declarations

A *declaration set* D consists of

- a finite *variable type declaration set* D_T^X,
- a finite *term type declaration set* D_T,
- a finite *term equality declaration set* D_E.

A variable type declaration is of the form '$x : \tau$', a term type declaration of the form '$t : \tau$' and a term equality declaration of the form '$s \doteq t$'.

$V(d)$ is the set of variable symbols of the declaration d.

We assume that for every x in X there exists at most one $x : \tau$ in D_T^X.

Every subset V of the variable symbol set X is a T set as follows

- $x.\tau$ iff $x : \tau \in D_T^X$.

With every t in $T(V)$ and every S algebra A we associate a function t_A^V from $V \rightarrow_T A$ to A in the usual way. If A is an S algebra $T(W)$, then $\theta \in V \rightarrow_T T(W)$ is called a *typed substitution* and we write θt instead of $t_{T(W)}^V(\theta)$.

We use the abbreviations

- '$t_A^V.\tau$' for '$t_A^V(\theta).\tau$ for all θ in $V \rightarrow_T A$',
- '$s_A^V = t_A^V$' for '$s_A^V(\theta) = t_A^V(\theta)$ for all θ in $V \rightarrow_T A$'.

Let A be an S algebra.

- $d = t : \tau$ in D_T is *valid* in A if $t_A^{V(d)}.\tau$,
- $d = s \doteq t$ in D_E is *valid* in A if $s_A^{V(d)} = t_A^{V(d)}$.

2.2.2 Σ Algebras

A *signature* Σ consists of a

- a symbol set S,
- a declaration set D with $D_E = \emptyset$.

An S algebra A is a Σ algebra if all term declarations of Σ are valid in A.

An S quotient algebra $[A]$ of a Σ algebra A is a Σ algebra and is called a Σ *quotient algebra* of A.

2.2.3 Free Σ Algebras

Consider an S algebra $T(V)$. We use the abbreviation

- '$s \cdot \tau$' for 'there exists a $d = t : \tau$ in D_T and a θ in $V(d) \rightarrow_T T(V)$ such that $s = \theta t$'.

$T(V)$ is *type reasoning compatible* if

- $x.\tau$ (in V) implies $x.\tau$ (in $T(V)$),
- $t \cdot \tau$ implies $t.\tau$.

Fact 1 *A type reasoning compatible S algebra $T(V)$ is a Σ algebra.*

We use the abbreviation

- '$\Sigma \models t : \tau$' for '$t : \tau$ is valid all Σ algebras'.

Consider the following S algebra $T(V)$ (denoted by $T_\Sigma(V)$)

- $t.\tau$ iff $\Sigma \models t : \tau$.

Fact 2 *$T_\Sigma(V)$ is type reasoning compatible.*

Thus $T_\Sigma(V)$ is a Σ algebra, called the *free* Σ algebra over V.

Fact 3 *$\Sigma \models t : \tau$ iff $t : \tau$ is valid in $T_\Sigma(V)$.*

2.2.4 Inferring Syntactic Types

S algebras $T(V)$ are ordered in the usual way :

- $T(V)_1 \leq T(V)_2$ iff
  - $t._1\tau$ implies $t._2\tau$.

Fact 4 *$T_\Sigma(V)$ is the least type reasoning compatible S algebra $T(V)$.*

Syntactic types on $T(V)$ are defined as follows

- $t._\Sigma\tau$ iff $t.\tau$ in $T_\Sigma(X)$.

We use the abbreviation

- '$\Sigma \vdash t.\tau$' for 'from Σ one can infer that t has type τ'.

If θ is a substitution in $W \rightarrow T(V)$, then

- 'from Σ one can infer that θ is typed' means
  '$\Sigma \vdash \theta x.\tau$ for all x in W with $x.\tau$'.

We use the abbreviation

- '$\Sigma \vdash \theta.$' for 'from Σ one can infer that θ is typed'.

It follows from fact 4 that the following rules are sound and complete for inferring syntactic types on $T(V)$.

- $\frac{}{\Sigma \vdash x.\tau}$ if $x : \tau \in D_T^X$
- $\frac{\Sigma \vdash \theta.}{\Sigma \vdash \theta t.\tau}$ if $t : \tau \in D_T$

2.2.5 $\mathcal{S}$ Algebras

A *specification* $\mathcal{S}$ consists of a

- a symbol set S,
- a declaration set D.

An S algebra A is an $\mathcal{S}$ algebra if all term declarations of $\mathcal{S}$ are valid in A.

A quotient S algebra $[A]$ of an $\mathcal{S}$ algebra A is also an $\mathcal{S}$ algebra and is called an $\mathcal{S}$ *quotient algebra* of A.

2.2.6 Free $\mathcal{S}$ Algebras

Consider an S algebra $T(V)$. We use the abbreviation

- '$u \leftrightarrow v$' for 'there exists a $d = s \doteq t$ in D_E and a θ in $V(d) \rightarrow_T T(V)$ such that $u = \theta s$ and $v = \theta t$'.

An S quotient algebra $[T(V)]$ is *reasoning compatible* if

- $x.\tau$ (in V) implies $[x].\tau$ (in $[T(V)]$),

- $t \cdot \tau$ implies $[t].\tau$,
- $s \leftrightarrow t$ implies $[s] = [t]$.

Fact 5 *A reasoning compatible S quotient algebra $[T(V)]$ is an $\mathcal{S}$ algebra.*

We use the abbreviations

- '$\mathcal{S} \models t : \tau$' for '$t : \tau$ is valid all $\mathcal{S}$ algebras',
- '$\mathcal{S} \models s \doteq t$' for '$s \doteq t$ is valid all $\mathcal{S}$ algebras'.

Consider the following S quotient algebra $[T(V)]$ (denoted by $T_{\mathcal{S}}(V)$)

- $[t].\tau$ iff $t_A^V.\tau$ for all $\mathcal{S}$ algebras A,
- $[s] = [t]$ iff $s_A^V = t_A^V$ for all $\mathcal{S}$ algebras A.

Notice that (for well known reasons) we did *not* use the definition

- $[t].\tau$ iff $\mathcal{S} \models t : \tau$,
- $[s] = [t]$ iff $\mathcal{S} \models s \doteq t$.

Fact 6 *$T_{\mathcal{S}}(V)$ is reasoning compatible.*

Thus $T_{\mathcal{S}}(V)$ is an $\mathcal{S}$ algebra, called the *free* $\mathcal{S}$ algebra over V.

Fact 7 *$\mathcal{S} \models t : \tau$ iff $d = t : \tau$ is valid in $T_{\mathcal{S}}(V(d))$*
and $\mathcal{S} \models s \doteq t$ iff $d = s \doteq t$ is valid in $T_{\mathcal{S}}(V(d))$.

2.2.7 Inferring Types and Equality

S quotient algebras $[T(V)]$ are ordered in the usual way :

- $[T(V)]_1 \leq [T(V)]_2$ iff
  - $[t]_1.\tau$ implies $[t]_2.\tau$,
  - $[s]_1 = [t]_1$ implies $[s]_2 = [t]_2$.

Fact 8 *$T_{\mathcal{S}}(V)$ is the least reasoning compatible S quotient algebra $[T(V)]$.*

Types and *equality* on $T(V)$ are defined as follows

- $t._{\mathcal{S}}\tau$ iff $[t].\tau$ in $T_{\mathcal{S}}(V)$,
- $s =_{\mathcal{S}} t$ iff $[s] = [t]$ in $T_{\mathcal{S}}(V)$.

We use the abbreviations

- '$\mathcal{S} \vdash t.\tau$' for 'from $\mathcal{S}$ one can infer that t has type τ',
- '$\mathcal{S} \vdash s = t$' for 'from $\mathcal{S}$ one can infer that s and t are equal'.

If θ is a substitution in $W \to T(V)$, then

- 'from $\mathcal{S}$ one can infer that θ is typed' means
  '$\mathcal{S} \vdash \theta x.\tau$ for all x in W with $x.\tau$'.

We use the abbreviation

- '$\mathcal{S} \vdash \theta.$' for 'from $\mathcal{S}$ one can infer that θ is typed'

It follows from fact 8 that the following rules are sound and complete for inferring types and equality on $T(V)$.

- $\frac{}{\mathcal{S} \vdash x.\tau}$ if $x : \tau \in D_T^X$
- $\frac{\mathcal{S} \vdash \theta.}{\mathcal{S} \vdash \theta t.\tau}$ if $t : \tau \in D_T$
- $\frac{}{\mathcal{S} \vdash x=x}$
- $\frac{}{\mathcal{S} \vdash c()=c()}$
- $\frac{\mathcal{S} \vdash s_1=t_1 \wedge \ldots \wedge \mathcal{S} \vdash s_n=t_n}{\mathcal{S} \vdash f(s_1,\ldots,s_n)=f(t_1,\ldots,t_n)}$
- $\frac{\mathcal{S} \vdash s=t}{\mathcal{S} \vdash t=s}$
- $\frac{\mathcal{S} \vdash r=s \wedge \mathcal{S} \vdash s=t}{\mathcal{S} \vdash r=t}$
- $\frac{\mathcal{S} \vdash \theta.}{\mathcal{S} \vdash \theta s=\theta t}$ if $s \doteq t \in D_E$
- $\frac{\mathcal{S} \vdash s=t \wedge \mathcal{S} \vdash t.\tau}{\mathcal{S} \vdash s.\tau}$

If p is a *position* in the term t, then $t[p \leftarrow s]$ is the term obtained by *replacing* the subterm $t|_p$ of u at p by the term s.

In the foregoing inference rules we may replace rules number 5 and 8 by the rule

- $\frac{\mathcal{S} \vdash \theta.}{\mathcal{S} \vdash u[p \leftarrow \theta s]=u[p \leftarrow \theta t]}$ if $s \doteq t$ *in* D_E.

2.2.8 Alternative rules

If $\mathcal{S}$ is a specification, then we write Σ for the signature part of $\mathcal{S}$.
A Σ quotient algebra $[T_\Sigma(V)]$ is *equality reasoning compatible* if

- $u \leftrightarrow v$ implies $[u] = [v]$.

Fact 9 *An equality reasoning compatible Σ quotient algebra $[T_\Sigma(V)]$ is an $\mathcal{S}$ algebra.*

Consider the following Σ quotient algebra $[T_\Sigma(V)]$ (denoted by $\langle T_\Sigma(V)\rangle$)

- $\langle s\rangle = \langle t\rangle$ iff $s_A^V = t_A^V$ for all $\mathcal{S}$ algebras A.

Notice that (for well known reasons) we did *not* use the definition

- $\langle s\rangle = \langle t\rangle$ iff $\mathcal{S} \models s \doteq t$.

Fact 10 $\langle T_\Sigma(V)\rangle$ *is equality reasoning compatible.*

From fact 9 and fact 10 we may conclude that $\langle T_\Sigma(V)\rangle$ is an $\mathcal{S}$ algebra.

Fact 11 $\mathcal{S} \models t : \tau$ *iff* $d = t : \tau$ *is valid in* $\langle T_\Sigma(V(d))\rangle$
and $\mathcal{S} \models s \doteq t$ *iff* $d = s \doteq t$ *is valid in* $\langle T_\Sigma(V(d))\rangle$.

Fact 12 $T_\mathcal{S}(V)$ *and* $\langle T_\Sigma(V)\rangle$ *are isomorphic.*

Σ quotient algebras $[T_\Sigma(V)]$ are ordered in the usual way :

- $[T_\Sigma(V)]_1 \leq [T_\Sigma(V)]_2$ iff
  - $[s]_1 = [t]_1$ implies $[s]_2 = [t]_2$.

Fact 13 $\langle T_\Sigma(V)\rangle$ *is the least equality reasoning compatible Σ quotient algebra* $[T_\Sigma(V)]$.

It follows from fact 13 that the following alternative rules are sound and complete for inferring equality on $T(V)$.

- $\dfrac{}{\mathcal{S} \vdash x{=}x}$
- $\dfrac{}{\mathcal{S} \vdash c(){=}c()}$
- $\dfrac{\mathcal{S} \vdash s_1{=}t_1 \wedge \ldots \wedge \mathcal{S} \vdash s_n{=}t_n}{\mathcal{S} \vdash f(s_1,\ldots,s_n){=}f(t_1,\ldots,t_n)}$

- $\frac{\mathcal{S} \vdash s=t}{\mathcal{S} \vdash t=s}$
- $\frac{\mathcal{S} \vdash r=s \wedge \mathcal{S} \vdash s=t}{\mathcal{S} \vdash r=t}$
- $\frac{\Sigma \vdash \theta.}{\mathcal{S} \vdash \theta s=\theta t}$ if $s \doteq t \in D_T$

In the foregoing inference rules we may replace rules number 3 and 6 by the rule

- $\frac{\Sigma \vdash \theta.}{\mathcal{S} \vdash u[p \leftarrow \theta s]=u[p \leftarrow \theta t]}$ if $s \doteq t \in D_T$.

It is now also clear that the following rules are sound and complete for inferring types on $T(V)$.

- $\frac{\Sigma \vdash t.\tau}{\mathcal{S} \vdash t.\tau}$
- $\frac{\mathcal{S} \vdash s=t \wedge \mathcal{S} \vdash t.\tau}{\mathcal{S} \vdash s.\tau}$

3 Typed Unification

In this section we use *syntactic* types on terms and write $s.\tau$ instead of $s._\Sigma\tau$.

As a variable x may have different types. Whenever we write '$x.\tau$' this means 'as a variable symbol x has type τ'.

Types are ordered in the natural way.

Terms are ordered as follows

- $s \leq t$ if $s = \theta t$

3.1 Equations

We work with *equations* $s \doteq t$, where s and t are terms. If $s \notin X$ or $t \notin X$, then $s \doteq t$ and $t \doteq s$ are considered to be the same equation.

An *equation set* is a finite set $E = \{e_1, \ldots, e_n\}$ of equations.

An equation $s \doteq t$ is *trivial* if $s \in X$ and $s = t$.

An equation $s \doteq t$ is *solved* if $s \in X$ or $t \in X$ and $s \doteq t$ is nontrivial.

An unsolved equation $s \doteq t$ is *decomposable* if

- $s = c()$ and $t = c()$ or
- $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$.

The decomposition of e (denoted by $D(e)$) is defined as follows

- $\emptyset$ is the decomposition of $c() \doteq c()$,
- $\{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}$ is the decomposition of $f(s_1, \ldots, s_n) \doteq f(t_1, \ldots, t_n)$.

Let E be an equation set. For $x \in X$ and $y \in X$ we write

- $x \succ_E y$ if there exists a solved $x \doteq s$ in E with $s \notin X$ and $y \in V(s)$,
- $x \approx_E y$ if there exists a solved $x \doteq y$ or $y \doteq x$ in E.

Let $\geq_E$ be the transitive closure of $\succ_E$ and $\approx_E$ and let $=_E$ be the transitive closure of $\approx_E$. We write

- $x >_E y$ if $x \geq_E y$ and $x \neq_E y$.

E is *circular* if $>_E$ is not irreflexive.

3.2 Unification

A substitution θ is a *unifier* of E if $\theta s = \theta t$ for all equations $s \doteq t$ of E.

3.2.1 Solved Case

A solved $e = x \doteq s \in E$ is *isolated* if $x \notin V(s)$ and $x \notin V(e')$ for all $e' \neq e$ in E.

An equation set E is *solved* if all its equations are solved and isolated.

If $E = \{x_1 \doteq s_1, \ldots, x_n \doteq s_n\}$ is a solved equation set, then $\mu(E)$ is the substitution $[x_1/s_1, \ldots, x_n/s_n]$.

3.2.2 Transformation Rules

Equation sets are transformed using rules $E \longrightarrow E'$ as follows. (We use a special equation set ∇ which is not unifyable.)

For circular E

- $E \longrightarrow \nabla$

For noncircular E

- $E \cup e \longrightarrow E$ if e is trivial

- $E \cup e \longrightarrow \nabla$ if e is indecomposable
- $E \cup e \longrightarrow E \cup D(e)$ if e is decomposable
- $E \cup e \longrightarrow E[p \leftarrow s] \cup e$ if $e = x \doteq s$ is solved, and $E|_p = x$

These rules are terminating.[1]

3.2.3 General Case

Let E be an equation set, and let $(E_n)_{n \leq s}$ be a finite sequence of equation sets such that $E_0 = E$ and $E_k \longrightarrow E_{k+1}$ for all $k < s$ using the rules of 3.2.2 and such that no rule can be applied to E_s. If $E_s \neq \nabla$, then it is a solved equation set and $\mu(E) = \mu(E_s)$, is a *most general* unifier of E.

3.3 Strongly Typed Unification

We use the abbreviation

- '$t._l\tau$' for 't has a *least type* τ'.

θ is a *strongly typed substitution* if $\theta x._l\tau$ for all x in X with $x.\tau$. θ is a *strongly typed unifier* of E if it is a unifier of E and a strongly typed substitution.

We also work with sets $\mathbf{E} = \{E_1, \ldots, E_n\}$ of equation sets. θ is a strongly typed unifier of $\mathbf{E}$ if it is a strongly typed unifier of one of the equation sets of $\mathbf{E}$.

3.3.1 Strongly Typed Case

A solved equation $x \doteq s$ is *strongly typed* if $x.\tau$ implies $s._l\tau$. A solved equation set is *strongly typed* if all its equations are strongly typed.

If $E = \{x_1 \doteq s_1, \ldots, x_n \doteq s_n\}$ is a strongly typed solved equation set, then $\mu(E)$ is the strongly typed substitution $[x_1/s_1, \ldots, x_n/s_n]$.

[1] Let $c(E)$ be the multiset of all (function and variable) symbols in E. Let $(E_n)_{n \in N}$ be an infinite sequence with $E_k \longrightarrow E_{k+1}$ for all k. Let $>_k$ be the following ordering on (function and variable) symbols :
- $x >_k g$ if x is a variable symbol and g is a function symbol and
- $x >_k y$ if x and y are variable symbols with $x >_{E_k} y$.

There exists an ordering $>$ and an m in N such that $>_k = >$ for all $k \geq m$. It is clear that $c(E_k) >_M c(E_{k+1})$ for all $k \geq m$, where $>_M$ is the multiset extention of $>$.

3.3.2 Transformation rules

If E is an arbitrary equation set, then $Solve(E)$ is a solved equation set obtained from E using the rules of 3.2.2. *In this subsection we work with solved equation sets.* All solved equation sets E are a disjoint union of a 'to be worked of' equation set E_* and a 'worked of' equation set. All strongly typed equations are 'worked of'.

All rules $E \longrightarrow \mathbf{E}$, where E is an equation set and $\mathbf{E}$ is a set of equation sets must be interpreted as $E \cup \mathbf{E}' \longrightarrow \mathbf{E} \cup \mathbf{E}'$.

Let $e = x \doteq s$ be a 'to be worked off' equation with $x.\tau$. There are two cases :

- s is a variable or a ground term,
- s is not a variable and not a ground term.

In the first case we use the rule

- $E \cup e \longrightarrow \emptyset$.

Consider now the second case. If $s.\sigma$ for some $\sigma < \tau$, then we use the rule

- $E \cup e \longrightarrow \emptyset$.

Let $T^s_\tau = \max\{t \in T(X) \setminus X \mid t : \tau \in D_T\}$.[2] (We use new variable symbols and new variable type declarations for the terms in $T^s_\tau(X)$.)

We use the rule

- $E \cup e \longrightarrow \bigcup_{t \in T^s_\tau} Solve(E \cup e \cup s \doteq t)$

and e becomes 'worked of'.

3.3.3 Constraints

Σ is *linear* if all t with $t : \tau \in D_T$ are linear.

p is an *open function position* in a term t if the subterm $t|_p$ of t at p is not a variable and not a ground term. Let $P_o(t)$ be the set of all open function positions in t.

We write

[2] If there does not exist declarations $s : \tau$ and $t : \tau$ in D_T with $s \leq t$, then, obviously $T^s_\tau = \{t \in T(X) \setminus X \mid t : \tau \in D_T\}$.

- $f \succ g$ if there exists a $f(t_1, \ldots, t_n) : \tau$ in D_T and a $p \in P_o(f(t_1, \ldots, t_n))$ with $t|_p = g(s_1, \ldots, s_m)$.

Σ is circular if the transitive closure of $\succ$ on F is not irreflexive.

From now on we assume that

- Σ is noncircular,
- Σ is linear.

The rules of 3.3.2 are terminating.[3]

Conjecture 1 *The linearity constraint is not needed.*

3.3.4 General Case

Let E be an equation set with $V = V(E)$ and let $(\mathbf{E}_n)_{n \leq s}$ be a finite sequence of sets of solved equation sets such that $\mathbf{E}_0 = \{Solve(E)\}$ and $\mathbf{E}_k \longrightarrow \mathbf{E}_{k+1}$ for all $k < s$ using the rules of 3.3.2 and such that no rule can be applied to $\mathbf{E}_s$. $\mathbf{E}_s$ is a set of strongly typed solved equation sets and $\bigcup_{E_s \in \mathbf{E}_s} \mu(E_s)|_V$, is a most general set of strongly typed unifiers of E.

3.4 Regularity

A signature Σ is *regular* if every term has a least type.

A renaming of variables ρ is a *weakening* on V if for all x in V

- $x.\tau$ implies $\rho x.\sigma$ for some $\sigma \leq \tau$.

s is a *weakening* of t if there exists a weakening ρ on $V(t)$ with $s = \rho t$.

Fact 14 *A signature Σ is regular iff for all least strongly typed common instances s of weakenings s_1 and s_2 of terms t_1 and t_2 with $t_1 : \tau_1$ and $t_2 : \tau_2$ in D_T there exists a σ with $\sigma \leq \tau_1$ and $\sigma \leq \tau_2$ such that $s.\sigma$.*

3.5 Typed Unification

From now on we assume that Σ is regular.

θ is a *typed substitution* if $\theta x.\tau$ for all x in X with $x.\tau$. θ is a *typed unifier* of E if it is a unifier of E and a typed substitution.

θ is a typed unifier of $\mathbf{E}$ if it is a typed unifier of one of the equation sets of $\mathbf{E}$.

[3] Let $c(E)$ be the multiset of all open function symbols in E_*. $E \longrightarrow F$ implies $c(E) >_M c(F)$, where $\succ_M$ is the multiset extention of $\succ$.

3.5.1 Typed Case

A solved equation $x \doteq s$ is *typed* if $x.\tau$ implies $s.\tau$. A solved equation set is *typed* if all its equations are typed.

If $E = \{x_1 \doteq s_1, \ldots, x_n \doteq s_n\}$ is a typed solved equation set, then $\mu(E)$ is the typed substitution $[x_1/s_1, \ldots, x_n/s_n]$.

3.5.2 Transformation rules

The transformation rules are similar to the ones in 3.3.2.

There are three cases :

- s is a variable,
- s is a ground term,
- s is not a variable and not a ground term.

Consider now the first case (say s is a variable y with $y.\sigma$). Let $x \wedge y$ consist of new variable symbols z with new variable type declarations $z : \pi$ for all π in $\max(]-, \tau] \cap]-, \sigma])$.

We use the rule

- $E \cup e \longrightarrow \bigcup_{z \in x \wedge y} Solve(E \cup x \doteq z \cup y \doteq z)$

and both $x \doteq z$ and $y \doteq z$ become 'worked off'.

In the second case we use the rule

- $E \cup e \longrightarrow \emptyset$.

Consider now the third case.

Let $T_\tau = max\{t \in T(X) \setminus X \mid \exists \sigma \leq \tau \;\; t : \tau \in D_T\}$ (We use new variable symbols and new variable type declarations for the terms in $T_\tau(X)$.)

We use the rule

- $E \cup e \longrightarrow \bigcup_{t \in T_\tau} Solve(S \cup e \cup s \doteq t)$

and e becomes 'worked of'.

3.5.3 Termination

Under the assumptions of 3.3.3 the rules of 3.5.2 are terminating.

3.5.4 General Case

Let E be an equation set with $V = V(E)$ and let $(\mathbf{E}_n)_{n \leq s}$ be a finite sequence of sets of solved equation sets such that $\mathbf{E}_0 = \{Solve(E)\}$ and $\mathbf{E}_k \longrightarrow \mathbf{E}_{k+1}$ for all $k < s$ using the rules of 3.5.2 and such that no rule can be applied to $\mathbf{E}_s$. $\mathbf{E}_s$ is a set of typed solved equation sets and $\bigcup_{E_s \in \mathbf{E}_s} \mu(E_s)|_V$, is a most general set of typed unifiers of E.

References

[1] Smolka G., Nutt W., Goguen J.A., Meseguer J.
Order-Sorted Equational Computation
SEKI Report SR-87-14
Universität Kaiserslautern, 1987.

[2] Schmidt-Schauss M.
Computational Aspects of an Order-Sorted Logic with Term Declarations
SEKI Report SR-88-10
Universität Kaiserslautern, 1988.

[3] Duponcheel L.
Typed Completion
(to appear).

[4] Watson P., Dick J.
Least Sorts in Order-Sorted Term Rewriting
Technical Report CSD-TR-606
Royal Holloway And Bedford New College, 1989.

Unitary Unification in Order-Sorted Signatures
Extended Abstract

Uwe Waldmann
Lehrstuhl Informatik V, Universität Dortmund
Postfach 50 05 00, 4600 Dortmund 50, West Germany
E-Mail: uwe@unidoi5.ls5.informatik.uni-dortmund.de

June 1989

1 Foundations

In this paper we give a decidable necessary and sufficient criterion for unitary unification in order-sorted algebras. By lack of space we omit all the proofs; these can be found in [Wal89].

We shall first summarize some basic results about order-sorted signatures and algebras, following [SNGM87].

An order-sorted signature is a triple $(S, \leq, \Sigma)$, where S is a set of sorts, $\leq$ a partial ordering over S, and Σ a family $\{ \Sigma_{w,s} \mid w \in S^*, s \in S \}$ of (not necessarily disjoint) sets of operator symbols. The ordering $\leq$ is extended componentwise to strings $s_1 \dots s_n \in S^*$, so we have $s_1 \dots s_n \leq s'_1 \dots s'_n$ if and only if $s_i \leq s'_i$ for $1 \leq i \leq n$. In order to make the notation simpler and more intuitive we shall often write $f : w \to s$ instead of $f \in \Sigma_{w,s}$ and $f : \to s$ instead of $f \in \Sigma_{\epsilon,s}$. We shall also use Σ as an abbreviation for both $(S, \leq, \Sigma)$ and $\bigcup_{w,s} \Sigma_{w,s}$.

An S-sorted variable set is a family $V = \{ V_s \mid s \in S \}$ of disjoint sets. A variable $x \in V_s$ is written $x : s$. We shall use V as an abbreviation for $\bigcup_{s \in S} V_s$.

Let $(S, \leq, \Sigma)$ be an order-sorted signature and V be a variable set disjoint from Σ. The set $\mathrm{T}_\Sigma(V)_s$ of terms over Σ and V with sort s is the least set with the following properties:

(i) $x \in \mathrm{T}_\Sigma(V)_s$ if $x : s_0 \in V$ and $s_0 \leq s$.

(ii) $f \in \mathrm{T}_\Sigma(V)_s$ if $f : \to s_0$ and $s_0 \leq s$.

(iii) $f(t_1, \dots, t_n) \in \mathrm{T}_\Sigma(V)_s$ if $f : s_1 \dots s_n \to s_0$ such that $s_0 \leq s$ and $t_i \in \mathrm{T}_\Sigma(V)_{s_i}$ for every $i \in \{1, \dots, n\}$.

$\mathrm{T}_\Sigma(V) := \bigcup_{s \in S} \mathrm{T}_\Sigma(V)_s$ denotes the set of all terms over Σ and V. The set of all ground terms over Σ is $\mathrm{T}_\Sigma := \mathrm{T}_\Sigma(\emptyset)$. The set of all variables in a term $t \in \mathrm{T}_\Sigma(V)$ is abbreviated by $\mathrm{Var}(t)$.

Let $(S, \leq, \Sigma)$ be an order-sorted signature. A $(S, \leq, \Sigma)$-algebra A consists of a family $\{ A_s \mid s \in S \}$ of sets and a function $A_f : \mathrm{D}_f^A \to \mathrm{C}_A$ for every $f \in \Sigma$ such that the following conditions are fulfilled:

(i) $A_s \subseteq A_{s'}$, if $s \leq s'$.

(ii) D_f^A is a subset of $(\mathrm{C}_A)^*$ where $\mathrm{C}_A := \bigcup_{s \in S} A_s$.

(iii) If $f \in \Sigma_{w,s}$, then $A_w \subseteq \mathrm{D}_f^A$ and $A_f(A_w) \subseteq A_s$.

We use $A_{s_1 \dots s_n}$ as an abbreviation for $A_{s_1} \times \dots \times A_{s_n}$, A_ϵ is some one-point set. (A function with domain A_ϵ may be considered as a constant.)

Obviously we can make T_Σ (which we shall abbreviate by T) a Σ-algebra. We define $T_s := T_{\Sigma,s}$; for $f \in \Sigma$ the function T_f maps the tuple $(t_1, \ldots, t_n) \in D_f^T$ to $f(t_1, \ldots, t_n)$, where D_f^T is the union of all T_w such that $f : w \to s$.

Let A and B be two $(S, \leq, \Sigma)$-algebras. An $(S, \leq, \Sigma)$-homomorphism $h : A \to B$ is a function $h : C_A \to C_B$ so that $h(A_s) \subseteq B_s$ for each $s \in S$ and so that $h(D_f^A) \subseteq D_f^B$ and $h\big(A_f(a_1, \ldots, a_n)\big) = B_f\big(h(a_1), \ldots, h(a_n)\big)$ for all $f \in \Sigma$ and $(a_1, \ldots, a_n) \in D_f^A$. A homomorphism $h : A \to B$ is called an isomorphism, if and only if a homomorphism $h' : B \to A$ exists satisfying $h' \circ h = \mathrm{id}_A$ and $h \circ h' = \mathrm{id}_B$.

As in the unsorted case, the term algebra T_Σ is the initial Σ-algebra, it is determined uniquely (up to isomorphism).

An assignment ν from a variable set V into a Σ-algebra A is an S-sorted family of functions $\{\nu_s : V_s \to A_s \mid s \in S\}$. The algebra $T_\Sigma(V)$ is the free Σ-algebra generated by V, i.e., for every Σ-algebra A and every assignment ν from V to A there is exactly one homomorphism $\nu^* : T_\Sigma(V) \to A$ that extends ν.

A substitution σ is an assignment from a variable set Y into the term algebra $T_\Sigma(X)$. In general the uniquely determined extension $\sigma^* : T_\Sigma(Y) \to T_\Sigma(X)$ of σ will also be denoted by σ. If the substitution $\sigma : \{x_1, \ldots, x_n\} \to T_\Sigma(X)$ maps the variables $x_1, \ldots, x_n$ to the terms $t_1, \ldots, t_n$, respectively, we write $\sigma = (x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n)$.

A substitution $\sigma : X \to T_\Sigma(Y)$ is called a renaming, if it is injective and if it maps every variable $x : s$ from X to a variable of the same sort s. A substitution $\sigma : X \to T_\Sigma(Y)$ is called a specialization, if it is injective and if it maps every variable $x : s$ from X to a variable (of the same or of a smaller sort).

2 Unifiers and Weakenings

From now on we shall always assume that $(S, \leq, \Sigma)$ is a signature such that there is a ground term $t \in T_{\Sigma,s}$ for every sort $s \in S$.

Let $t \in T_\Sigma(Y)$, $t' \in T_\Sigma(Y')$. We say that t' is an instance of t (abbreviated by $t' \geq t$), if there is a substitution $\theta : Y \to T_\Sigma(Y')$ such that $t' = \theta(t)$. Let $\sigma : X \to T_\Sigma(Y)$ and $\sigma' : X \to T_\Sigma(Y')$ be two substitutions. We say that σ' is an instance of σ (abbreviated by $\sigma' \geq \sigma$), if there exists a substitution $\theta : Y \to T_\Sigma(Y')$ such that $\sigma' = \theta \circ \sigma$. The quasi-ordering $\geq$ is called subsumption ordering.

An equation system over $T_\Sigma(X)$ is a finite multiset of equations $t \doteq t'$, where $t, t' \in T_\Sigma(X)$. Let Γ be an equation system over $T_\Sigma(X)$. A substitution $\sigma : X \to T_\Sigma(Y)$ is said to be a unifier of Γ, if $\sigma(t) = \sigma(t')$ holds for all equations $t \doteq t' \in \Gamma$. The set of all unifiers of Γ is denoted by $\mathrm{SU}(\Gamma)$. A unifier σ of an equation system $\{t \doteq t'\}$ is also called a unifier of t and t'.

A subset $U \subseteq \mathrm{SU}(\Gamma)$ is called complete, if for every unifier $\sigma' : X \to T_\Sigma(Y')$ there exists a $\sigma : X \to T_\Sigma(Y)$ from U such that σ' is an instance of σ. We write CSU as an abbreviation for a complete set of unifiers.

A signature Σ is said to be unitary unifying, if for all terms $t, t' \in T_\Sigma(X)$ there is a CSU containing at most one element; it is called finitary unifying, if there exists a finite CSU for all terms $t, t' \in T_\Sigma(X)$.

Let $(S, \leq, \Sigma)$ be a signature, t a term in $T_\Sigma(X)$, and s a sort in S. A substitution $\sigma : X \to T_\Sigma(Y)$ is called a weakening from t to s, if $\sigma(t) \in T_\Sigma(Y)_s$. The set of all weakenings from t to s is denoted by $\mathrm{SW}(t, s)$. Complete sets of weakenings (CSWs) are defined analogously to CSUs.

It is well-known that order-sorted unification may be infinitary even if the signature is finite [SNGM87]. In order to avoid the problems arising from infinite minimal CSUs we have to restrict the class of signatures to be considered:

A signature Σ is called regular, if every term $t \in T_\Sigma(V)$ has a least sort, which is denoted by $\mathrm{LS}(t)$. (We assume that V contains at least one variable of every sort.) As proved by G. Smolka, the regularity of a finite signature is decidable [Smo86].

Regular signatures have the following important property: If $(S, \leq, \Sigma)$ is a finite and regular

signature, then for every term $t \in T_\Sigma(X)$ and every sort $s \in S$ there is a finite complete set of weakenings from t to s such that each of these weakenings is a specialization.

3 Computation of a CSU

The unification algorithm that will now be demonstrated is split into several passes. The first pass, which is described by the following inference system, makes no use of the sorts of the terms to be unified, thus it does not differ from an unsorted unification algorithm.

Inference System 3.1 Let Σ be a regular signature. The following rules transform an equation system over $T_\Sigma(X)$ into a new equation system.

(U1) Decomposition $\dfrac{\Gamma \cup \{f(t_1,\ldots,t_n) \doteq f(t'_1,\ldots,t'_n)\}}{\Gamma \cup \{t_1 \doteq t'_1,\ldots,t_n \doteq t'_n\}}$

(U2) Merging $\dfrac{\Gamma \cup \{x \doteq t,\ x \doteq t'\}}{\Gamma \cup \{x \doteq t,\ t \doteq t'\}}$ if $\mathrm{depth}(t) \leq \mathrm{depth}(t')$ and $v(t) < v(x)$.

(U3) Commutation $\dfrac{\Gamma \cup \{t \doteq x\}}{\Gamma \cup \{x \doteq t\}}$ if $v(t) < v(x)$.

(U4) Deletion $\dfrac{\Gamma \cup \{t \doteq t\}}{\Gamma}$

Without loss of generality let $X = \{x_1, x_2, \ldots\}$; the function v is defined by $v(x_i) := i$ and $v(t) := 0$ for $t \notin X$.

Lemma 3.2 If an equation system Γ_i is transformed into an equation system Γ_{i+1} using one of the inference rules (U1)–(U4), then Γ_i and Γ_{i+1} have the same set of unifiers. The inference system terminates, i.e., starting with an arbitrary equation system Γ_0 it yields after finitely many steps an equation system Γ_k, to which no rule can be applied. Provided that Γ_k does not contain an equation of the form $f(t_1,\ldots,t_n) \doteq g(t'_1,\ldots,t'_m)$ where $f \neq g$ or $n \neq m$, then all equations in Γ_k have the form $x \doteq t$ where $v(t) < v(x)$, and every variable occurs at most once on the left hand side of an equation.

In the unsorted case an equation $x \doteq t$ where $x \neq t$ is unifiable if and only if x does not occur in t, the most general unifier is $\sigma = (x \leftarrow t)$. In order-sorted signatures it may happen that $\mathrm{LS}(t) \not\leq \mathrm{LS}(x)$ holds (thus the substitution $(x \leftarrow t)$ would not be well-formed), on the other hand there may exist instances of t having a sort that is less or equal to $\mathrm{LS}(x)$. In such cases it is therefore necessary to compute a CSW from t to $\mathrm{LS}(x)$.

Lemma 3.3 Let Σ be a finite regular signature and let $\Gamma = \{y_1 \doteq t_1,\ldots,y_m \doteq t_m\}$ be an equation system over $T_\Sigma(X)$ such that y_j does not occur in t_i for $j \leq i$ and such that every variable occurs at most once on the left hand side of an equation. For $i \leq m$ we recursively define sets U_i of substitutions as follows:

$$U_0 := \{\mathrm{id}_X\}$$

$$U_i := \{(\sigma|_{Y\setminus\{y'_i\}} \cup (y'_i \leftarrow \sigma t'_i)) \circ \theta \mid \theta : X \to T_\Sigma(Y) \in U_{i-1},\ y'_i := \theta y_i,\ t'_i := \theta t_i,\ \sigma : Y \to Y' \in \mathrm{CSW}(t'_i, \mathrm{LS}(y'_i))\},$$

where $\mathrm{CSW}(t,s)$ denotes a fixed complete set of weakenings from t to s such that each of these weakenings is a specialization. Then U_m is a CSU of Γ.

The subsequent theorem originates from M. Schmidt-Schauß [SS85].

Theorem 3.4 Given a finite regular signature $(S, \leq, \Sigma)$ for every equation system Γ over $T_\Sigma(X)$ a finite CSU is effectively computable.

4 Downward Uniqueness

In finite regular signatures unification is finitary. But unfortunately in certain signatures the problem to determine whether two terms are unifiable in NP-complete, just as the problem to determine whether there is a weakening from a term to a sort. Even if there are only two sorts (then the weakening problem is solvable in linear time), the size of a minimal CSU or CSW can grow exponentially with the number of variables in the terms [SS88]. We shall now give a criterion, under which circumstances the unification in a signature is unitary.

Definition 4.1 Let S' be a subset of $(S, \leq)$. The set $\mathrm{lbd}(S')$ of the lower bounds of S' is defined by $\mathrm{lbd}(S') := \{ s \in S \mid s \leq s' \text{ for each } s' \in S' \}$.

Definition 4.2 A set of sorts $(S, \leq)$ is called downward complete, if for all $s_1, s_2 \in S$ the set $\mathrm{lbd}(\{s_1, s_2\})$ is either empty or contains a greatest element.

Definition 4.3 A regular signature $(S, \leq, \Sigma)$ is called downward unique, if $(S, \leq)$ is downward complete and if for all $s \in S$, $w \in S^*$, $f : w^\circ \to s^\circ$ where $s \leq s^\circ$ and $w \leq w^\circ$ the set

$$\{ w' \in S^* \mid \exists \overline{w} \in S^*, \overline{s} \in S \text{ so that } f : \overline{w} \to \overline{s}, \overline{s} \leq s, w' \leq w, w' \leq \overline{w} \}$$

is empty or contains a greatest element.

Lemma 4.4 Given a regular signature $(S, \leq, \Sigma)$ the following properties are equivalent:

(i) $(S, \leq, \Sigma)$ is downward unique.

(ii) For each term $t \in \mathrm{T}_\Sigma(X)$ and every sort $s \in S$ we have: Either there is no weakening from t to s or there exists a substitution σ_s^t such that $\mathrm{LS}(\sigma t) \leq s \iff \sigma \geq \sigma_s^t$ for all substitutions $\sigma : X \to \mathrm{T}_\Sigma(Y)$.

For a term t and a sort s the most general weakening σ_s^t can be computed in $O(n \log m)$ time where n is the size of t and m is the cardinality of $\mathrm{Var}(t)$.

Theorem 4.5 Let $(S, \leq, \Sigma)$ be a regular signature, then $(S, \leq, \Sigma)$ is downward unique if and only if for every equation system Γ over $\mathrm{T}_\Sigma(X)$ there is a complete set of unifiers having at most one member.

5 Remarks

Meseguer, Goguen, and Smolka have shown [MGS87], that unification in an order-sorted signature is unitary, provided that $(S, \leq)$ is downward complete and that for all $f \in \Sigma$, $s \in S \cup \{max\}$, and $n \in \mathbf{N}$ the set $\{ s_1 \ldots s_n \in S^* \mid f : s_1 \ldots s_n \to \tilde{s}, \tilde{s} \leq s \}$ is either empty or contains a greatest element, where $s \leq max$ holds for each $s \in S$. This criterion is sufficient, however, it is necessary only if unsorted variables are permitted in the terms that are to be unified, and at least one of the terms to be unified is not an unsorted variable. The following example demonstrates how unsorted variables may affect unitary unifiability.

Example 5.1 Let $(S, \leq, \Sigma)$ be defined by

$$(S, \leq) = \{ s_2 \leq s_1, s_2' \leq s_1' \}$$

$$\Sigma = \{ f : s_1 \to s_1, \; f : s_1' \to s_1, \; f : s_2 \to s_2, \; f : s_2' \to s_2 \}$$

The equation $f(x : \mathit{untyped}) \doteq \tilde{x} : s_2$ has the minimal complete set of unifiers $\{(x \leftarrow y : s_2, \tilde{x} \leftarrow f(y)), (x \leftarrow y' : s_2', \tilde{x} \leftarrow f(y'))\}$. We can never encounter such a situation, however, if the variable x has a sort: If x has the sort s_1 or s_2, it cannot be mapped to a variable with the sort s_2', hence only the first substitution remains, similarly the first substitution must be excluded, if x has the sort s_1' or s_2'.

References

[MGS87] J. Meseguer, J. A. Goguen, and G. Smolka. Order-sorted unification. Technical Report CSLI-87-86, Center for the Study of Language and Information, 1987.

[Smo86] G. Smolka. Order-sorted Horn logic: Semantics and deduction. Draft, Universität Kaiserslautern, Fachbereich Informatik, 1986.

[SNGM87] G. Smolka, W. Nutt, J. A. Goguen, and J. Meseguer. Order-sorted equational computation. SEKI-Report SR-87-14, Universität Kaiserslautern, 1987.

[SS85] M. Schmidt-Schauß. A many-sorted calculus with polymorphic functions based on resolution and paramodulation. In A. Joshi, editor, *Proceedings of the 9th International Joint Conference on Artificial Intelligence, Los Angeles*, pages 1162–1168, 1985.

[SS88] M. Schmidt-Schauß. *Computational Aspects of an Order-Sorted Logic with Term Declarations.* Dissertation, Universität Kaiserslautern, Fachbereich Informatik, 1988.

[Wal89] U. Waldmann. Unification in order-sorted signatures. Forschungsbericht 298, Universität Dortmund, Fachbereich Informatik, 1989.

AC Unification Through Order-Sorted AC1 Unification*

Extended abstract

Eric Domenjoud
LORIA & CRIN
BP 239 F-54506 Vandœuvre Cedex
FRANCE
e-mail domen@crin.crin.fr

May 25, 1989

Unification in the Associative Commutative theory is a major topic in automated deduction since many mathematical operators have these properties. AC-unification is a very complex problem which was shown NP-complete by Kapur and Narendran in 1986 [10], and M. Stickel [17] and M. Livesey & J. Siekmann [15] discovered concurrently in 1975 the first AC unification algorithm whose termination was proved by F. Fages [5] in 1984. After them, many authors studied intensively this problem, including A. Fortenbacher [3], A. Herold & J. Siekmann [6], C. Kirchner [11], J. Christian & P. Lincoln [2], D. Kapur, H. J. Bürckert et al. [1].

The AC unification algorithm involves usually the solving of a linear homogeneous diophantine equation over $\mathbb{N}^n$ followed by a combination step of its solutions. Each of these steps was studied independently but only the first one was significantly improved. In 1978, Huet [7] proposed an algorithm to solve a linear homogeneous diophantine equation over $\mathbb{N}^n$ by searching for the solutions in a bounded domain, and the bound was then improved by Lambert in 1987 [14]. At last, in 1987, Clausen & Fortenbacher [3] proposed a new very efficient algorithm. The only significant improvements of the combination step were proposed by J.M. Hullot in 1979 [8] and J. Christian & P. Lincoln in 1987 [2]. Hullot gave criteria to eliminate early some combinations which do not lead to a solution, while J. Christian & P. Lincoln proposed a particular very fast AC unification algorithm for the case of equations without repeated variables. This last solution may however not be used as a general AC unification algorithm. In the general case, even simple equations may be solved by no current algorithm because of the huge amount of their minimal AC-unifiers. For example, if + is associative and commutative, then

$$x + x + x + x \doteq y_1 + y_2 + y_3 + y_4 \qquad \text{has } 34\,359\,607\,481 \text{ minimal unifiers}$$

as shown in [4], and the computation of these minimal unifiers, required in an implementation of the equational Knuth Bendix's completion procedure [13,9] for example, is thus untractable. A solution to such problems is to introduce an identity for +, and to perform unification modulo Associativity, Commutativity and Identity, as proposed by Herold & Siekmann [6]. The theory defined by these axioms, called AC1, is then unitary unifying when only variables and one AC1-operator are involved in terms to unify. This works well as long as no other operators are involved and no other axioms are taken into account but in general, we do not get a conservative extension of the quotient algebra. For example, if we consider the theory defined by the following axioms

$$\begin{aligned} (x \cdot y) \cdot z &= x \cdot (y \cdot z) \\ x \cdot y &= y \cdot x \\ x \cdot x &= A \\ x \cdot A &= A \end{aligned}$$

*This work was partly supported by the GRECO de programmation (CNRS)

where A is a constant, and if we assume the existence of an identity for $\cdot$, that is to say, if we add the axiom $x \cdot 1 = x$, where 1 is a new constant, then we can prove $u = A$ for any term u as follows:

$$\begin{aligned} u &= u \cdot 1 \\ &= u \cdot (1 \cdot 1) \\ &= u \cdot A \\ &= A \end{aligned}$$

but this does not hold modulo the previous axioms because no axiom may be applied to u. The trouble is due to the fact that the new axiom $x \cdot 1 = x$ may be overlapped with the previous ones, by instanciating some variables with 1. Such instanciations should thus be forbidden. The first solution to this new problem is to compute the AC unifiers from the AC1 unifiers when they are needed, as made by Herold & Siekmann [6]. We loose then the benefit of the Identity, and complexity of unification does not decrease as we would expect. An other solution is to handle constraints expressing that some variables should not be instanciated with the identity. Although this solution works well, it has some drawbacks:

- We do not really perform unification in an algebra.
- The constraints are disequations which may, in general, never be removed because unlike for equations, instances of solutions of a disequation are not necessarily solutions.
- We must deal with constrained equational logic. See [12] for more details.

We found a third solution by studying the combination step of Stickel's algorithm. when solving $x + y \doteq z + t$ for example, we first solve the diophantine equation $X_1 + X_2 = X_3 + X_4$ over $\mathbb{N}^4$. The matrix of minimal solutions of this equation is

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

and we must then find all subset of the set of this minimal solutions which form a matrix without a zero column. In our case, there are 7 such subsets. Considering each column, we see that a matrix without a zero column is obtained if and only if we take the 1^{st} or the 2^{nd} row, the 3^{rd} or the 4^{th}, the 1^{st} or the 3^{rd}, and the 2^{nd} or the 4^{th}. That is to say, we have to take rows 1 and 4 or rows 2 and 3, other rows being *"free"* in each case. For each of these two possibilities, we associate with each constrained row, a *"normal"* new variable u_i, and with each *"free"* row, a *"special"* new variable u_i^+ which may be instanciated with the identity. We get then two solutions

$$\begin{cases} x = u_1 + u_2^+ \\ y = u_3^+ + u_4 \\ z = u_1 + u_3^+ \\ t = u_2^+ + u_4 \end{cases} \quad \text{and} \quad \begin{cases} x = u_1^+ + u_2 \\ y = u_3 + u_4^+ \\ z = u_1^+ + u_3 \\ t = u_2 + u_4^+ \end{cases}$$

instead of seven. We see on this example that we have variables of two sorts: *"special"* variables may be instanciated with the identity while *"normal"* ones may not. Since the second sort is embedded in the first one, we will work in an order-sorted framework with the semantic introduced by G.Smolka in [16]. Applying the same method,

$$x + x + x + x \doteq y_1 + y_2 + y_3 + y_4 \qquad \text{has 437 minimal unifiers}$$

and this may now be used for practical purpose. One may notice that this third point of view is very similar to the second one, but as we will see later, sorting terms avoids the drawbacks mentioned above. Indeed, we deal with constrained equational logic, but without disequations.

This idea is developed in the paper, and given an equational theory, we design an order-sorted extension of this theory in which every AC-operator has an identity. We prove then that this extension is conservative, and give a unification procedure for this theory. The soundness, completeness and termination of the procedure are then proved and some comparisons are given which show clearly that the complexity of the unification drastically decreases.

References

[1] H-J. Burckert, A. Herold, D. Kapur, J. Siekmann, M. Stickel, M. Tepp, and H. Zhang. Opening the ac-unification race. *Journal of Automated Reasoning*, 1988.

[2] J. Christian and P. Lincoln. *Adventures in Associative-Commutative Unification.* Technical Report ACA-ST-272-87, MCC, 1987.

[3] M. Clausen and A. Fortenbacher. *Efficient Solution of Linear Diophantine Equations.* Interner Bericht 32/87, Universität Karlsruhe, 1987.

[4] E. Domenjoud. *Number of Minimal Unifiers of the Equation* $\alpha x_1 + \cdots + \alpha x_p \doteq_{AC} \beta y_1 + \cdots + \beta y_q$. Research Report 89 R 2, CRIN, Nancy (France), 1989.

[5] F. Fages. Associative-commutative unification. In R. Shostak, editor, *Proceedings 7th international Conference on Automated Deduction*, Springer-Verlag, Napa Valley (California, USA), 1984.

[6] H. Herold and J. Siekmann. *Unification in Abelian Semigroups.* Memo SEKI-85-III-KL, universität Kaiserslautern, 1985.

[7] G. Huet. An algorithm to generate the basis of solutions to homogenous linear diophantine equations. *Information Processing Letters*, 7(3):144–147, 1978.

[8] J.M. Hullot. Associative-commutative pattern matching. In *Proceedings 9th International Joint Conference on Artificial Intelligence*, 1979.

[9] J.P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *Proceedings 11th ACM Symposium on Principles Of Programming Languages, Salt Lake City*, 1984.

[10] D. Kapur and P. Narendran. NP-completeness of the set unification and matching problems. In J. Siekmann, editor, *Proceedings 8th Conference on Automated Deduction*, Springer-Verlag, 1986.

[11] C. Kirchner. *From Unification in Combination of Equational to A New AC-Unification algorithm.* Technical Report 87-R-132, Centre de Recherche en Informatique de Nancy, 1987. To appear in the Proceedings of the CREAS (Acadenic Press 1989).

[12] C. Kirchner and H. Kirchner. Constrained equational reasoning. In *Proceeding of ISSAC'89*, page , July 1989.

[13] D. Knuth and P. Bendix. *Simple Word Problems in Universal Algebra*, pages 263–297. Pergamon Press, 1970.

[14] J-L. Lambert. *Une borne pour les generateurs des solutions entieres positives d'une equation diophantienne lineaire.* Technical Report 334, Université de Paris-Sud, Centre d'Orsay, février 1987.

[15] M. Livesey and J. Siekmann. *Termination and Decidability Results for Stringunification.* Technical Report Memo CSM-12, University of Essex, 1975.

[16] G. Smolka, W. Nutt, J.A. Goguen, and J. Meseguer. Order sorted equational computation. In *Proceedings of the Colloquium on Resolution of Equations in Algebraic Structures*, Austin (Texas), May 1987.

[17] M.E. Stickel. A complete unification algorithm for associative-commutative functions. *Proceedings 4th International Joint Conference on Artificial Intelligence, Tbilissi*, 1975.

Section 5:
Special Algorithms I

H. Abdulrab, J.-P. Pecuchet: Associative Unification

E. Contejean, H. Devie: Solving Systems of Linear Diophantine Equations

J.-F. Romeuf: Solutions of a Linear Diophantine Systems

H.J. Ohlbach: Abstraction Tree Indexing for Terms

ASSOCIATIVE UNIFICATION

Habib Abdulrab* Jean-Pierre Pécuchet**
Faculté des Sciences, B.P. 118, 76134 Mont-Saint-Aignan Cedex †

Notations

An algebra equipped with a single associative law is a **semigroup**. It is a **monoid** when it has a unit. The free monoid generated by the set A (also called **alphabet**) is denoted by A^*. Its elements are the **words** written on the alphabet A, the neutral element being the empty word denoted by 1. The operation is the concatenation denoted by juxtaposition of words. The **length** of a word w (the number of letters composing it) is denoted by $|w|$. For a word $w = w_1 \dots w_n$, with $|w| = n$, we denote by $w[i] = w_i$ the letter at the ith position.

In this terminology, the term algebra (in the sense of Fages & Huet (1986), Kirchner (1987)) built on a set of variables V, a set C of constants, and a set of operators constituted of an associative law, is nothing else than the free monoid $T = (V \bigcup C)^*$ over the alphabet of **letters** $L = V \bigcup C$.

A unifier of two terms $e_1, e_2 \in T$ is a monoid morphism $\alpha : T \longrightarrow T$ (i.e. a mapping satisfying $\alpha(mm') = \alpha(m)\alpha(m')$ and $\alpha(1) = 1$), leaving the constants invariant (i.e. satisfying $\alpha(c) = c$ for every $c \in C$) and satisfying the equality $\alpha(e_1) = \alpha(e_2)$.

The pair of words $e = (e_1, e_2)$ is called an **equation** and the unifier α is a **solution** of this equation.

A solution $\alpha : T \longrightarrow T'$ **divides** a solution $\beta : T \longrightarrow T''$ if there exists a **continuous** morphism $\theta : T' \longrightarrow T''$ (i.e. satisfying $\theta(x) \neq 1$ for every x) such as $\beta = \alpha\theta$. We also say that α is **more general** than β. A solution α is said to be **principal** (or **minimal**) when it is divided by no other but itself (or by an **equivalent** solution, i.e. of the form $\alpha' = \alpha\theta$ with θ, an isomorphism).

---

* Laboratoire d'Informatique de Rouen et LITP. E.m.: inria!geocub!abdulrab.

** Laboratoire d'Informatique de Rouen, LITP . E.m.: inria!geocub!pecuchet.

† This work was also supported by the Greco de Programmation du CNRS and the PRC Programmation Avancee et Outils pour l'Intelligence Artificielle.

Main problems

The three main problems concerning systems of equations are the existence of a solution, the computation of the set of minimal solutions (denoted by μCSU_A in Fages & Huet (1986)) and the computation of the rank. All these problems reduce to the case of a single equation, as by Albert & Lawrence (1985) every infinite system of equations is equivalent to one of its finite subsystems, and a finite system can be easily encoded in a single equation (cf. Hmelevskii (1971)).

The study of properties and structure of the set of solutions of a word equation was initiated by Lentin and Schützenberger (1967), Lentin (1972) in the case of constant-free equations ($C = \emptyset$).

In particular, Lentin shows that every solution is divided by a unique minimal one and gives a procedure (known as the **pig-pug**: paire initiale gauche, paire ultime gauche) allowing to enumerate the set of minimal solutions. This procedure extends without difficulty to the general case of an equation with constants (cf. Plotkin (1972), Pécuchet (1981)). The minimal solutions are obtained as labels of some paths of a graph. When this graph is finite, as in the case when no variable appears more than twice, we obtain a complete description of all solutions.

It can be shown that the rank of an equation is the maximum rank of its principal solutions, that is, the maximum number of variables V' appearing in its principal solutions. The computation of the rank of certain types of constant-free equations was solved by Lentin (1972). Then Makanin (1977) showed that one can compute the rank of an equation with constants in the case of four variables and in the general case (cf. Makanin (1978)). Pécuchet (1984) shows that one can compute the rank of any equation as soon as one knows how to decide the existence of a solution, showing thus that the two algorithms of Makanin (1977, 1978) given by Makanin are not independent.

The problem of the existence of a solution was first tackled by Hmelevskii (1971) who solved it in the case of three variables, then by Makanin (1977) who solved the general case. He gave an algorithm to decide whether a word equation with constants has a solution or not.

All the problems linked to word equations are very close to those linked to equations in the free group. Thus the computation of minimal solutions by the pig-pug method uses transformations close to those used by Nielsen (1918) in the study of automorphisms in free groups, then taken up again by Lyndon (1960). Finally, an adaptation of his first algorithm allowed Makanin (1983) to solve the problem of the existence of a solution to an equation in a free group. This paper also gives a bound on the length of a solution of minimal length in the case of free monoid, allowing to use the pig-pug method as an algorithm to solve the problem. However, no proof of this bound independent of Makanin's algorithm is known, and the value of this bound makes the pig-pug method almost inoperating in the general case. So the original algorithm of Makanin seems still to give the most reasonable implementation.

We present here the pig-pug method, we assume first without loss of generality, that the alphabets of variables $V = \{v_1 \dots v_n\}$ and of constants $C = \{c_1 \dots c_m\}$ are finite and disjoint. We make the convention to represent the variables by lower-case letters,

as $x, y, z \ldots$, and the constants by upper-case letters as $A, B, C \ldots$. We call **length** of an equation $e = (e_1, e_2)$ the integer $d = |e_1 e_2|$.

The **projection** of an equation e over a subset Q of V is the equation obtained by "erasing" all the occurrences of $V \setminus Q$. Consequently, an equation has 2^n projections $(\Pi_Q e_1, \Pi_Q e_2)$ where $\Pi_Q : (V \bigcup C)^* \longrightarrow (Q \bigcup C)^*$ is the projection morphism.

One easily proves the following proposition which reduces the research of a solution to that of a continuous one.

PROPOSITION 1.1 *An equation e has a solution iff one of its projections has a continuous solution.* ∎

The pig-pug method consists in searching for a continuous solution α in the following manner: it visits the lists $e_1[1], \ldots, e_1[|e_1|]$ and $e_2[1], \ldots, e_2[|e_2|]$ of symbols of e from left to right and at the same time, one tries to guess how their images can overlap. At each step, one makes a non deterministic choice for the relative lengths of the images of the first two symbols $e_1[1]$ et $e_2[1]$. According to the choice made :

$$|\alpha(e_1[1])| < |\alpha(e_2[1])|, \quad |\alpha(e_1[1])| = |\alpha(e_2[1])|, \quad |\alpha(e_1[1])| > |\alpha(e_2[1])|$$

one applies to the equation one of the three substitutions to variables :

$$e_2[1] \leftarrow e_1[1]e_2[1], \quad e_2[1] \leftarrow e_1[1], \quad e_1[1] \leftarrow e_2[1]e_1[1].$$

The process is repeated until the **trivial** equation (1,1) is obtained.

The following result, a proof of which can be found in Pécuchet (1981), shows that this graph enumerates all the minimal solutions.

THEOREM 1.2 (cf. Lentin (1972)) *The set of minimal solutions of a word equation is given by the labels of the paths linking the root to the trivial equation in the pig-pug graph.* ∎

When the graph associated with an equation is finite, the pig-pug method provides a particularly simple unification algorithm. That is the case when no variable appears more than twice. In fact, it is easy to prove the following proposition:

PROPOSITION 1.3 *When each variable appearing in the equation e has at most two occurrences, the length of all the equations derived by the pig-pug method is bounded by the length of e.*

In the general case, the pig-pug's graph will be infinite. However one can always decide the existence of a solution by:

THEOREM 1.4 (cf. Makanin (1983)) *One can construct a recursive function F such that, if an equation of length d has a solution, then there exists one in which the lengths of the components are bounded by $F(d)$.* ∎

Note that the only known function F is that derived from Makanin's algorithm (1977). Another reason for the study of this algorithm is that it leads to a better pruning of the graph, and is more efficient than the pig-pug method in some cases.

Note also that this bound concerns the solutions of minimal length, and allows to use the pig-pug as a decision procedure. When the set of minimal solutions is finite, the pig-pug's graph is not necessarily finite, and no bound concerning the length of minimal solutions is known. In this case the pig-pug cannot be used as a unification procedure, because it is not known how to stop the graph construction. The only known unification procedure is that of Jaffar (1989), based on Makanin's algorithm.

Application in a programming system

We describe here an application of the resolution of word equations in a programming system.

It is well-known that the algorithm of unification (cf. Robinson (1965)) is the heart of PROLOG language. PROLOG-3 propose a major modification to this algorithm. In this section the modification is discussed and the potential role of Makanin's algorithm in this area is demonstrated.

Here is an example, given in Colmeraur (1987), of a program written in PROLOG-3:

$$\{z : 10,\ < A, B, C > .z\ =\ z. < B, C, A >\}?$$

The result of running of this program is

$$\{z\ = < A, B, C, A, B, C, A, B, C, A >\}$$

More precisely, this program computes the list z, which produces the same list if it is appended at the right of the list $< A, B, C >$ and at the left of the list $< B, C, A >$, such that the length of z is equal to 10.

In other terms, this program computes a solution α to the equation $ABCz = zBCA$, such that $|\alpha(z)| = 10$. A solution satisfying this condition is $z = ABCABCABCA$.

It must be observed from this example that PROLOG-3 takes into account the associativity of the operation of concatenation (denoted by .). In equational terms, this implies that the value of a variable given by the solution α may be a sequence of terms, not simply one term as in the case of a classic PROLOG. This major difference provides a very powerful tool of formal computation.

However, there is an important restriction: it is necessary to specify the length l of each variable x, used in the operation of concatenation, by the condition $x : l$. In the program given above, $z : 10$ is an example of a constraint which must be given explicitly. Note that the classic unification algorithm is replaced by an algorithm solving a **system of constraints** which becomes the heart of PROLOG-3. Of course, this must be efficient, which is why Colmeraur (1987) justifies, the necessary condition on

the length of variables mentioned above. Note that this restriction avoids solving word equations which have no solution.

New results and implementaion

In our paper (1989), we gave a complete description of Makanin's algorithm which allows to decide whether a word equation has a solution or not.

New concepts, such as equations with schemes and position equations are introduced in our description. A new presentation of Makanin's algorithm permitting to all its steps to be illustrated graphically, is described in this paper. The fundamental point is that our version has been effectively implemented.

When a word equation has a solution, one can use the pig-pug method to compute the set of all the minimal solutions of e. This set is presented as a language accepted by a deterministic automaton (not necessarily finite). The families of languages accepted by such automata are described.

Some improvements concerning the reduction of the number of types of position equations, and the reduction of the complexity of the algorithm by one exponential, are given in (1989). Other technical results concerning our work in this area can be found in Abdulrab (1987a) Pécuchet (1981). We prove, for example, that one of the three conditions of the normalization is always satisfied. The size of the tree $\mathcal{A}$ is greatly reduced in our version. Our construction of $\mathcal{A}$ is based on the notion of a solution of an equation with scheme, which allows the elimination of some types of equations with schemes which have no solution.

The first implementation of the version of Makanin's algorithm is described in Abdulrab (1987a,b).

This implementation is an interactive system written in LISP and running on VAX780 under UNIX, and on LISP Machine. This system visualizes the position equations, computes effectively a solution of the initial equation whenever there exists one, and provides a tool permitting to understand, experiment and study introdthe algorithm. Our implementation has also been coded in CAML by Rouaix (1987).

We show in Abdulrab (1987b) that the present implementation is not efficient enough to be used as a unification module in a programming system. We also describe the characteristics of a new version which could realize this goal.

The purpose of the algorithm being to decide whether an equation admits a solution or not, we provide an algorithm (cf. Abdulrab (1989)) which, by taking advantage of the tree $\mathcal{A}$, effectively computes a solution to the initial equation. The idea is to compute a solution to the equation e, from the equation with scheme which generates the root of the subtree containing a simple position equation.

Here is in milliseconds some execution results on a LMI LISP Machine.

1) Equation $(zxzyCBzxzx, yAByzxB)$ has no solution: (433 ms).

2) Equation $(xAByCBzxtzux, yABytzuxB)$ has no solution: (757 ms).

3) Equation $(xxAyBy, CAyvABD)$ has solution $x = CAABD$,
$v = CAABDAABDB$, $y = ABD$: (19 ms).

4) Equation $(BlABlABlA, rouDouDou)$ has solution $l = ADABADAB$, $o = ABA$, $r = BADABADABABADABADAB$, $u = 1$: (43 ms).

References

Abdulrab H. (1987). Résolution d'équations sur les mots: étude et implémentation LISP de l'algorithme de Makanin. (Thèse). University of Rouen. And Rapport LITP 87-25, University of Paris-7.

Abdulrab H. (1987). Implementation of Makanin's algorithm. Rapport LITP 87-72, University of Paris-7.

Abdulrab H., Pécuchet J.P. (1989). Solving word equations. to appear in JSC.

Abdulrab H. (1989). Equations in words. to appear in RAIRO d'Info. th..

Albert M.H, Lawrence J. (1985). A proof of Ehrenfeucht's conjecture. Theor. Com. Sci. 41 p. 121-123.

Colmeraur A. (1987). Une Introduction à PROLOG-3. Faculte des sciences de Luminy.

Fages F., Huet G. (1986). Complete sets of unifiers and matchers in equational theories. Theor. Com. Sci., 43, p. 189-200.

Hmelevskii Yu. I. (1971). Equations in free Semigroups. Trudy Mat. Inst. Steklov, 107.

Jaffar J. (1989). Minimal and complete word unification. To appear in JACM.

Kirchner C. (1987). From unification in combination of equational theories to a new AC-Unification algorithm. Proc. of the CREAS, Austin, Texas.

Lentin A. (1972). Équations dans le monoïde libre, Gauthier – Villars, Paris.

Lentin A., Schutzenberger M.P. (1967). A combinatorial problem in the theory of free monoids. Proc. of the University of northCarolina, p. 128-144.

Lyndon R.C. (1960). Equations in free groups, Trans. Amer. Math. Soc. 96, p. 445-457.

Makanin G.S. (1977). On the rank of equations in four unknowns in a free semigroup. English transl. in Math. USSR Sb. 29, p. 257-280.

Makanin G.S. (1977). The problem of solvability of equations in a free semigroup. Mat. Sb. 103(145) (1977) p. 147-236 English transl. in Math. USSR Sb. 32.

Makanin G.S. (1978). Algorithmic decidability of the rank of constant free equations in a free semigroup. Dokl. Akad. Nauk. SSSR 243.

Makanin G.S. (1983). Equations in a free group. Math. USSR Izvestiya Vol. 21, No 3.

Nielsen J. (1918). Die Isomorphismen der allgemeinen, unendlichen Gruppe mit zwei Erzeugenden, Math. Ann. 78, p. 385-397.

Pécuchet J.P. (1981). Équations avec constantes et algorithme de Makanin. (Thèse). University of Rouen.

Pécuchet J.P. (1984). Solutions principales et rang d'un système d'équations avec constantes dans le monoïde libre. Discrete Mathematics, 48, p. 253-274.

G. Plotkin. (1972). Building-in equational theories. Machine Intelligence 7, p. 73-90.

A. Robinson (1965). A Machine-Oriented Logic Based on The Resolution Principle. JACM.

Rouaix F. (1987). Une implémentation de l'algorithme de Makanin en CAML. Mémoire de DEA d'Informatique, University of Paris7.

Solving systems of linear diophantine equations

Evelyne Contejean and Hervé Devie
LRI, Université Paris-Sud, Bât 490
91405 ORSAY Cedex, France

March 30, 1989

We discribe an algorithm for solving systems of linear diophantine equations which extends Fortenbacher's algorithm [For Clau 87], the best one currently known for solving a single equation. Such systems naturally arise in automated theorem proving on one hand (AC unification and matching) and in Petri nets on the other hand (for deciding reachability).
Let us introduce some terminology, starting with a linear homogenous system S with p equations and q unknowns denoted:

$$\begin{cases} d_{1,1}x_1 + \ldots + d_{1,q}x_q = 0 \\ \vdots \\ d_{p,1}x_1 + \ldots + d_{p,q}x_q = 0 \end{cases}$$

where the coefficients $d_{i,j}$ $1 \leq i \leq p$, $1 \leq j \leq q$ are in Z, and the unknowns x_j $1 \leq j \leq q$ range over N.
We denote by $\vec{e_i}$ the j-th canonical q-tuple in N^q:

$$\vec{e_j} = (\underbrace{0,\ldots,0}_{j-1\ times},1,0,\ldots,0)$$

Let $\vec{x} = (x_1,\ldots,x_q)$ be in N^q. The p-tuple $\vec{d^S}(\vec{x})$ of N^p such that $d_i^S(\vec{x}) = (\sum_{j=1}^{q} d_{i,j}x_j)$ for $1 \leq i \leq p$ is called the *defect* of $\vec{x}$ for S. So, the defect of $\vec{x}$ is a vector whose components are the value of the p linear forms defining S. Hence, a q-tuple $\vec{x}$ is a solution for S if and only if $\vec{d^S}(\vec{x}) = \vec{0}$.
Note that $\vec{d^S}(\vec{x}) = \sum_{j=1}^{q} x_j \vec{d^s}(\vec{e_j})$ and let

$$D = \{\vec{d^S}(\vec{e_1}),\ldots,\vec{d^S}(\vec{e_q})\}$$

As usual, we are not interested in generating all solutions, a basis will suffice to represent them all.

Let $\leq^q$ be the ordering on tuples in N^q extending the ordering on N:

$$(s_1,\ldots,s_q) \leq^q (t_1,\ldots,t_q) \ \ iff \ \ s_j \leq t_j \ for \ j \in [1..q]$$

Fact 1 *A non-null solution of S is a linear combination of minimal (with respect to $\leq^q$) solutions with coefficients in N.*

Our algorithm therefore searches N^q for minimal solutions. The basic idea is to compute these minimal solutions by incrementing one by one the components of the canonical q-tuples until the defect becomes null. This can be seen as summing vectors taken from D until the resulting sum is $\vec{0}$. The only subtlety in the algorithm comes from the policy for choosing the vectors from D. Since there are several possibilities, this choice is non-deterministic, hence can be easily expressed by a tree construction. Let us see N^q as a labelled tree.

Definition 1 *Let $\mathcal{N}^q$ be the infinite tree whose nodes are labelled by vectors of N^q as follows:*

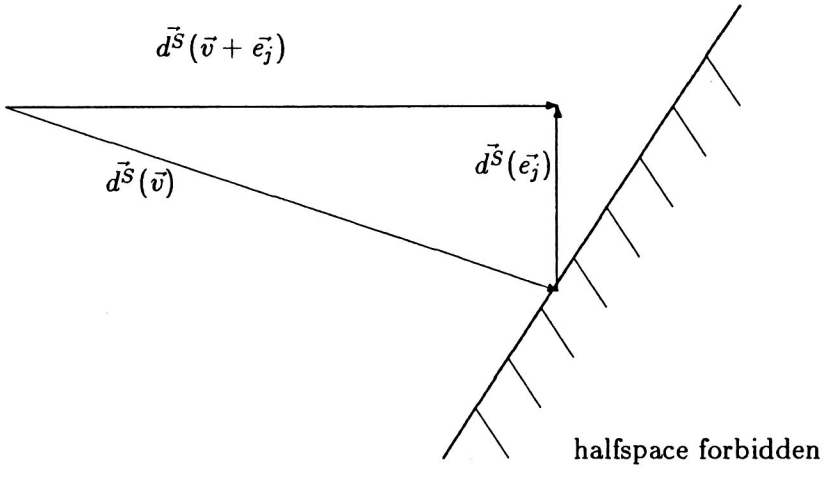
- *the root is labelled by $\vec{0}$.*
- *if $\vec{v}$ is the label of a node, its successors from left to right are labelled by*

$$\vec{v}+\vec{e^1},\ldots,\vec{v}+\vec{e^q}$$

Note that the deepth of a node is exactly the sum of the components of its label. Hence if a tuple is greater than another tuple for $\leq^q$, its depth is greater.
We search $\mathcal{N}^q$ bread-first. If a node is labelled by a vector $\vec{v}$ which is a solution for S, there is no need to visit the infinite branches of $\mathcal{N}^q$ starting at that node,since they would not yield minimal solutions. Since we are searching the tree bread-first, we know for each node whether its label is greater than a solution. Again,there is no need to visit the branches issued from such a node.
The heart of the algorithm is in the following restriction: if a node is labelled by a vector $\vec{v}$, no solution is omitted by keeping only its successors labelled by $\vec{v}+\vec{e^j}$ such that

$$\vec{d^S}(\vec{v}).\vec{d^S}(\vec{e_j}) < 0$$

where . denotes the scalar product in R^p. This means that the vector $\vec{d^S}(\vec{e_j})$ is in a halfspace delimited by an hyperplan perpendicular to the vector $\vec{d^S}(\vec{v})$. Intuitively, the new defect



$\vec{d^S}(\vec{v}+\vec{e_j})$ should not go too far from $\vec{0}$.

Now, the following theorem holds:

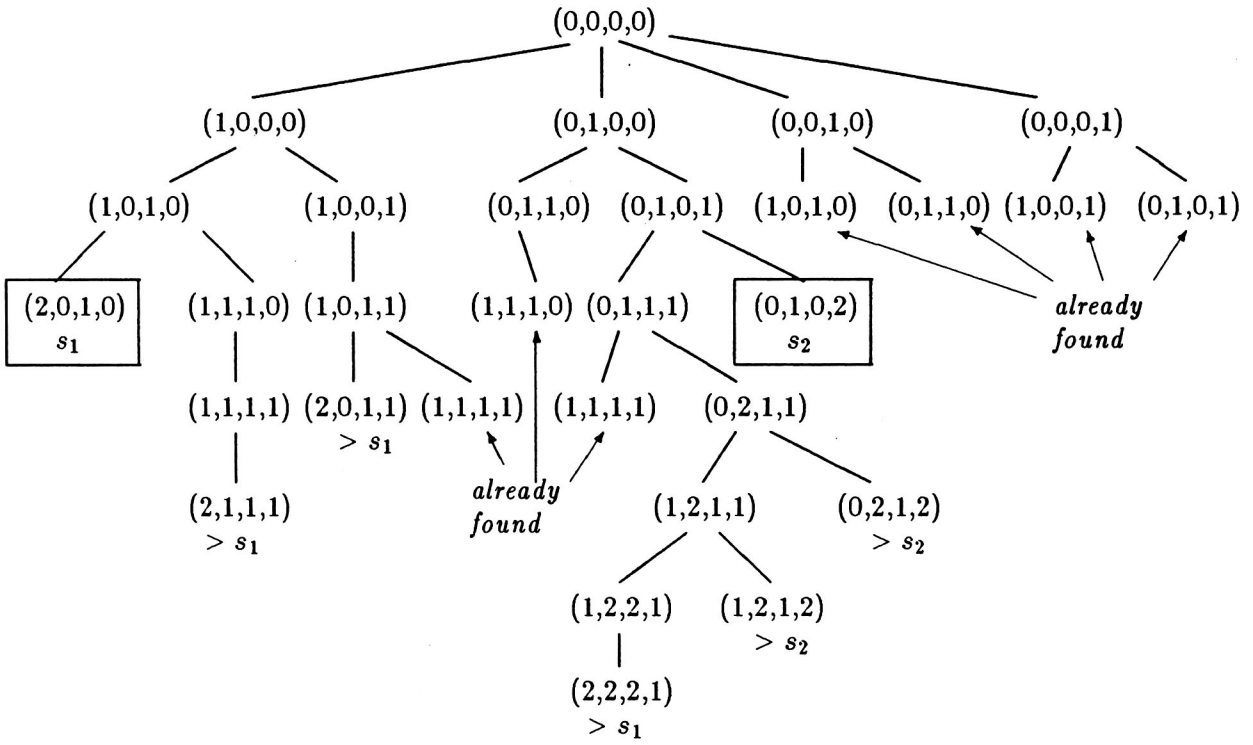
Theorem 1 *The algorithm sketched above*

- *is correct*
- *is complete*
- *terminates (there is no infinite branch in the search tree).*

Correctness is trivial. Completness is by induction on the depth of a solution in the tree. The hard point is to prove termination. The proof is not constructive, hence we do not know a bound for the depth of the tree (in the case of a single equation, Fortenbacher's algorithm yields such a bound).
Let us now show our algorithm at work on the following example:

$$\begin{cases} x + 2y - 2z - t = 0 \\ x - 2z = 0 \end{cases}$$

It generates the following tree:



One can also solve non homogenous systems with this algorithm by turning them into homogenous ones. Solving

$$\begin{cases} d_{1,1}x_1 + \ldots + d_{1,q}x_q + b_1 = 0 \\ \vdots \\ d_{p,1}x_1 + \ldots + d_{p,q}x_q + b_p = 0 \end{cases}$$

is equivalent to solve

$$\begin{cases} d_{1,1}x_1 + \ldots + d_{1,q}x_q + b_1x_{q+1} = 0 \\ \vdots \\ d_{p,1}x_1 + \ldots + d_{p,q}x_q + b_px_{q+1} = 0 \end{cases}$$

where x_{q+1} is bound to the value 1, which can be easily checked on the tree.

References

[For Clau 87]
A FORTENBACHER & M CLAUSEN
Efficient Solution of Linear Diophantine Equations
Universität Karslruhe
Fakultät für Informatik
Interner Bericht Nr.32/87 (nov 1987)

Solutions of a linear diophantine system

Jean-François Romeuf
L.I.R. & L.I.T.P.

Laboratoire d'Informatique de Rouen
Université de Rouen – Faculté des Sciences
Place Émile Blondel – B.P. 118
76134 Mont-Saint-Aignan CEDEX, FRANCE

e.m.: mcvax!inria!litp!jfr

May 4, 1989

The study of linear diophantine systems arises in many domains, such as integer optimization, associative-commutative unification, and even pure associative unification, since the resolution of such systems appears in Makanin's algorithm [Mak].

Let n, k be two positive integers. A system of n linear inhomogeneous diophantine equations with k unknowns may be written $Ax = b$, where:

$$A = (a_{ij})_{1 \leq i \leq n,\, 1 \leq j \leq k} \in \boldsymbol{Z}^{n \times k}, \text{ and } b = (b_i)_{1 \leq i \leq n} \in \boldsymbol{Z}^n$$

The set of *solutions* of such a system is then: $S = \{x \in \boldsymbol{N}^k : Ax = b\}$.

We recall that one can decide whether S is empty or not, and that this problem is NP-complete [BT]. [GS] indicates various upper bounds of a minimal height element of S, and some algorithms may be found in [GN] which find a minimal length element of S.

We are interested here in computing a representation of all solutions of a given system. When the system is homogeneous, S is a substractive submonoid of $\boldsymbol{N}^k$. This submonoid may be represented by its finite basis Y, which is the subset of minimal elements of $S \setminus \{0\}$ for the natural partial order on $\boldsymbol{N}^k$. In the general case, S is a rational part of $\boldsymbol{N}^k$ [GSp,ES], of the form $X + Y^{\oplus}$, where X is the finite set of minimal elements of S, and Y is the basis of solutions of $Ax = 0$. For $n = 1$, the problem has already been treated by Huet [Hu], improved by Lambert [Lam]. The more recent algorithm are by Fortenbacher [CF] and Lankford [Lan].

Let $\Sigma = \{\sigma_i : 1 \leq i \leq k\}$ be the basis of $\boldsymbol{N}^k$. S being a rational part of $\boldsymbol{N}^k$, there exists a finite Σ-automaton which recognizes S. For $n = 1, 2$, we give here a method for constructing explicitly such an automaton.

Let k be a positive integer, $A = {}^t(a_i)_{1\leq i\leq k} \in \boldsymbol{Z}^{1\times k}$, $b \in \boldsymbol{Z}$, and $S = \{x \in \boldsymbol{N}^k : Ax = b\}$.

We assume that $b \leq 0$, and we set: $a_{k+1} = -b$. Let:

- $l_1 = \max\limits_{a_i\leq 0} |a_i|$, and $l_2 = \max\limits_{a_i\geq 0} a_i$
- $Q =]-l_1, l_2]$ if $l_1 > 0$, and $[0, l_2]$ if $l_1 = 0$ (integer intervals)
- $F = \big\{(q, \sigma_i, q + a_i) : q \in Q,\ 1 \leq i \leq k,\ q + a_i \in Q\big\}$

We show that S is recognized by the Σ-automaton: $\mathcal{A} = (Q, \{-b\}, \{0\}, F)$

This result and its proof are similar to Fortenbachers' [CF], only the terminology changes.

Let k be a positive integer, $A = (a_{ij})_{1\leq i\leq 2,\, 1\leq j\leq k} \in \boldsymbol{Z}^{2\times k}$, $b = (b_1, b_2) \in \boldsymbol{Z}^2$, and $S = \{x \in \boldsymbol{N}^k : Ax = b\}$.

We set: $a_{1,k+1} = -b_1$, $a_{2,k+1} = -b_2$, and for all i, $1 \leq i \leq k$, $a_i = (a_{1i}, a_{2i})$. Let:

- $l_1 = \max\limits_{a_{1i}\leq 0} |a_{1i}|$, $l_2 = \max\limits_{a_{1i}\geq 0} a_{1i}$, $p_1 = \max\limits_{a_{2i}\leq 0} |a_{2i}|$, and $p_2 = \max\limits_{a_{2i}\geq 0} a_{2i}$
- $Q = \big([-l_1, l_2] \times [-p_1, p_2]\big) \cup \big(]-l_1, l_2]\times]p_2, 2p_2]\big) \cup \big([-2l_1, -l_1[\times]-p_1, p_2]\big) \cup \big([-l_1, l_2[\times[-2p_1, -p_1[\big) \cup \big(]l_2, 2l_2] \times [-p_1, p_2[\big)$ (integer intervals), and
- $F = \big\{(q, \sigma_i, q + a_i) : q \in Q,\ 1 \leq i \leq k,\ q + a_i \in Q\big\}$

Then S is recognized by the Σ-automaton: $\mathcal{A} = (Q, \{-b\}, \{0\}, F)$.

Let $Ax = b$ be a system of n diophantine equations with k unknowns, $n = 1, 2$, S the set of its solutions, and $X + Y^{\oplus}$ the rational expression of S. Let $\mathcal{A}$ be the Σ-automaton defined above. Then the following results hold:

Proposition 1 *x is a minimal length element of S iff it is the label of a shortest path from $-b$ to 0 in $\mathcal{A}$.*

Proposition 2 *For all x in S, $x \in X$ iff no path from $-b$ to 0 in $\mathcal{A}$ with label x passes twice by the same state.*

Proposition 3 *For all x in $S \setminus \{0\}$, $x \in Y$ iff no path from 0 to 0 in $\mathcal{A}$ with label x passes twice by the same state after leaving 0.*

We use these results to conceive, in the cases $n = 1, 2$, new efficient algorithms to compute a minimal length solution, the set X of minimal solutions, or the basis Y of the solutions of $Ax = 0$. So, we get an efficient algorithm to compute, for $n = 1, 2$, the rational expression of the solutions of an inhomogeneous diophantine system.

For $n = 1$, our algorithm has the same theoretical basis than Fortenbacher's algorithm [CF], but the methods for computing minimal elements differ notably.

As a corollary of propositions 2 and 3, in the case $n = 1$, we get:

Proposition 4 $\max_{x\in X}|x| \leq l_1 + l_2 - 1$, *and* $\max_{y\in Y}|y| \leq l_1 + l_2$.

Moreover, as in [CF], if we give a closer look at the form of some paths recognizing the elements of Y, we get a trivial proof of Lambert's result [Lam]:

Proposition 5 (Lambert) *For all y in Y,* $\sum_{a_i>0} y_i \leq l_1$ *and* $\sum_{a_i<0} y_i \leq l_2$.

In the case $n = 2$, we get:

Proposition 6 *Let* $L = (l_1 + l_2 + 1)(p_1 + p_2 + 1) + 2(l_1 + l_2)(p_1 + p_2)$. *Then:* $\max_{x\in X}|x| \leq L - 1$, *and* $\max_{y\in Y}|y| \leq L$.

Let $Ax = b$ be a system of two equations. We set: $l = \max(l_1, l_2, p_1, p_2)$. Using a slightly different automaton, we get the bound $12l^2$.

We give methods to compute the rational expression of the solutions of any linear diophantine system using the algorithms for systems of one or two equations. Let $Ax = b$ be a diophantine system. We set: $a_{i,k+1} = -b_i$ for all i, $1 \leq i \leq n$. Let: $l = \max_{1\leq i\leq n,\, 1\leq j\leq k+1} |a_{ij}|$. By iterating over one equation, we get the following bound:

Proposition 7 $\max_{x\in X}|x| \leq (2l)^{2^{n-1}} - 1$, *and* $\max_{y\in Y}|y| \leq (2l)^{2^{n-1}}$.

[GS] indicates, for a minimal height element of S, bounds in height of the order of $(k+1)\,n!\,l^n$ and l^{n^2}, which are smaller by one exponantial. However, our bound concerns all the terms of the rational expression of S. Moreover, we obtain here a bound in length which is independent from k.

If we iterate over two equations, we get a more efficient algorithm, and the bound lowers to $\left(\sqrt{12}\,l\right)^{3^{\frac{n}{2}}-1}$, n even.

References

[GSp] S. Ginsburg, E. H. Spanier: *Bounded Algol-like languages.*
Trans. Am. Math. Soc. 113 (1964), 333–368.

[ES] S. Eilenberg, M. P. Schützenberger: *Rational sets in commutative monoids.*
Journal of Algebra, vol. 13, n° 2, 1969.

[BT] I. Borosh, L. B. Treybig: *Bounds on positive solutions of linear diophantine equations.*
Proc. Amer. Math. Soc. 55 (1976), 299–304.

[GS] J. von zur Gathen, M. Sieveking: *A bound on solutions of linear equalities and inequalities.*
Proc. Amer. Math. Soc. 72 (1976), 155–158.

[Mak] G. S. Makanin: *The problem of solvability of equations in a free semigroup.*
Math USSR Sb. 32, 1977.

[GN] R. S. Garfinkel, G. L. Nemhauser: Integer Programming.
John Wiley, 1972.

[Hu] G. Huet: *An algorithm to generate the basis of solutions to homogeneous linear diophantine equations.*
Information Processing Letters, vol. 3, n° 7, 1978.

[Lam] J.-L. Lambert: *Une borne pour les générateurs des solutions entières positives d'une équation diophantienne linéaire.*
C. R. Acad. Sci. Paris, t. 305, Série I, 39–40, 1987.

[CF] M. Clausen, A. Fortenbacher: *Efficient solution of linear diophantine equations.*
Interner Bericht, Universität Karlsruhe, 32/87.
To appear in Journal of Symbolic Computation.

[Lan] D. Lankford: *New non-negative integer basis algorithms for linear equations with integer coefficients.*
Unpublished manuscript, 1987.

[Rom] J.-F. Romeuf: *Solutions of a linear diophantine system.*
Rapport L.I.T.P. 88-76, Université Paris 7.

Abstraction Tree Indexing for Terms

Hans Jürgen Ohlbach
FB. Informatik, University, Postf. 3049
D-675 Kaiserslautern, West Germany
email: ohlbach@uklirb.uucp

Introduction

In many deduction systems large databases of first-order predicate logic terms and literals have to be maintained. Typical queries to such a database are:

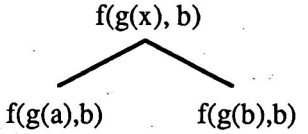
- For a given term t, find out all terms which are unifiable with t.
  This is necessary for example for finding all resolution partners for a clause.
- Backward subsumption tests and demodulation (for a new demodulator find all terms to rewrite) rely on fast access to all instances of a term.
- Forward subsumption tests and demodulation again (for a new literal find all applicable demodulators) rely on fast access to all generalizations of a term.

Since the state of the art in implementation techniques for deduction systems allows to generate millions of terms during a run, the efficiency of term indexing contributes considerably to the overall efficiency of the deduction system. Various techniques have been developed for this purpose - discrimination tree indexing, FPA/Path indexing, bit pattern, hashing etc. [HD82, LO80, BLC86, WP84]. In general these techniques work only for free terms, i.e. no theory unification algorithms are supported, they need large and complex datastructures, and a query evaluation has to touch at least all "answer terms".

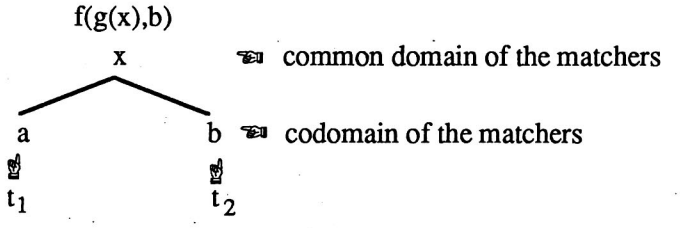
Abstraction tree (AT) indexing as it is proposed in this paper has none of these shortcomings. It works for all finitary unification theories, it uses a small and simple datastructure, and in the best case a few unification and matching operations are sufficient to access thousands of answer terms.
In the sequel x,y,z,u,v,w denote variables, a,b,c,d,e constants and f,g,h functions.

The Abstraction Tree Datastructure

Terms together with the usual instance relation form a partial ordering. For example the three terms t_1 = f(g(x),b), t_2 = f(g(a),b) and t_3 = f(g(b),b) can be ordered according to their instance relation:



In principle this structure is already an abstraction tree. Since both terms t_2 and t_3 are instances of t_1, however, this representation is somewhat redundant. It suffices to represent the two matchers which reproduce t_2 and t_3 from t_1. A more compact representation is therefore:

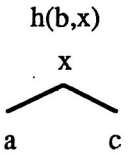


Now t_1 and t_2 are uniquely determined by the term at the root node and the matchers obtained from the variable

list attached to the root node and the corresponding termlists of the leaf nodes. When we adopt the convention that leaf nodes only represent terms, the tree is a representation of the two terms t_2 and t_3 only. That means, in order to represent t_2 and t_3, the term t_1 has to be generated as a common abstraction of t_2 and t_3. For free terms there is always a unique (up to renaming) most specific common abstraction and there are algorithms to compute it [Re69], [Pl69]. For interpreted terms there may be more than one most specific common abstraction. If f is associative, for example, the two terms f(a,b,a,b,a) and f(a,c,a,c,a) have at least the two different most specific common abstractions f(a,x,x) and f(x,a,y). For our purpose, however, the existence of at least one is sufficient and it even needs not necessarily be most specific. The selection of the "wrong" one may cause the abstraction tree to grow more than necessary, but this is no principle problem.

Let us consider some more examples:
For the three terms f(g(a),b), f(g(c),b) and f(g(c),d) there are two different abstraction trees:

If h is a commutative function symbol, an abstraction tree for the two terms h(a,b) and h(b,c) looks as follows:



That means the syntactic structure of the original terms needs no longer be represented in the tree. It is, however, still guaranteed that the represented terms are equivalent to the original ones in the underlying equational theory.

In general, an AT is a tree where the nodes are labelled with termlists such that the free variables of the termlist at node N and the termlists of N´s subnodes form the domain and codomain of a substitution, or more precisely, a matcher. In particular the root node can also be labelled with a termlist. In the examples below, however, only singleton termlists, i.e. terms, will be used. In order to have for the matchers represented by the free variables of a node N and the termlists of N´s subnodes a precise assignment of a domain term to a codomain term, we assume the variables to be an ordered set. In an implementation this variable list should be attached as a second component to the node.

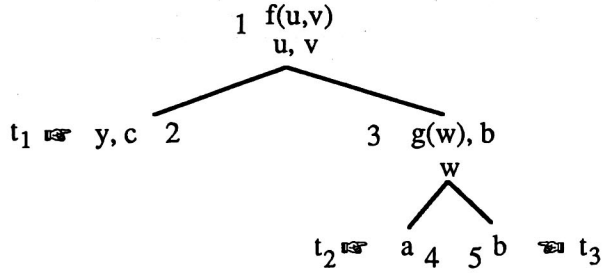
As we have seen, a leaf node represents a term, or a termlist respectively when the root node has also a termlist. All intermediate nodes, however, represent also terms, namely the term obtained by instantiating the root node´s term with all substitutions on the path to that particular node. To distinguish between this term, the term represented by the whole path down to the node, and the termlist attached to the node itself, we shall call the represented term a node´s r-term (or r-termlist respectively) and the termlist attached to the node directly the node´s a-termlist. Furthermore, when we speak of a node´s variables the free variables of a node´s a-termlist are meant.

Finding Unifiable Terms

The algorithm for finding all terms in an AT which are unifiable with a given term is a straightforward recursive procedure for traversing the tree. We introduce it with a few examples.

Example: ATs with free terms only.

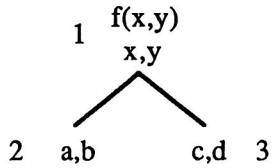
Consider again the tree for the three terms $t_1 = f(y,c)$, $t_2 = f(g(a),b)$ and $t_3 = f(g(b),b)$:



In order to find all terms in the tree which are unifiable with f(g(x),x) we start with the root node, unifying f(u,v) with f(g(x),x). The unifier, $\sigma_1 = \{u \mapsto g(v), v \mapsto x\}$, is the applied to the variable list (u,v) yielding the termlist s = (g(x),x). Taking this termlist as a new query termlist we map over all subnodes of the current node and ask recursively for all unifiable terms. Unification of s with node 2's a-termlist, (y,c), yields $\{y \mapsto g(c), x \mapsto c\}$, and this is actually the unifier for f(y,c) and f(g(x),c), i.e. the first unifiable term together with the unifier has been found. Continuing with node 3 we get $\sigma_2 = \{w \mapsto b, x \mapsto b\}$ as a unifier for the termlists (g(x),x) and (g(w), b). Application of σ_2 to the variable w yields b which has to be unified with the a-terms a and b at the nodes 4 and 5. Node 5 is the only successful one therefore the term t_3 is the second answer term and the unifier is $\sigma_{2|\{x\}} = \{x \mapsto b\}$.

Example: ATs with interpreted terms.

Assume now the function f is commutative. The AT for the two terms f(a,b) and f(c,d) is



To retrieve the terms unifiable with s = f(d,z), we unify s with f(x,y). Commutative unification yields the two unifiers $\{x \mapsto d, y \mapsto z\}$ and $\{x \mapsto z, y \mapsto d\}$. Both unifiers have to be applied to the variable list (x,y). We obtain (d,z) and (z,d). The main difference to the case with free terms only is that we enter the recursion not with a single termlist, but with a set of termlists, in this case {(d,z), (z,d)} and the recursive query is "give all termlists which are unifiable with *at least* one of the query termlists". In this case node 2 fails completely while node 3 succeeds with (z,d), i.e. f(c,d) is unifiable with f(d,z) and the unifier is $\{z \mapsto d\}$.

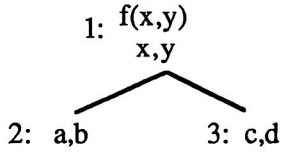
The general procedure for accessing unifiable terms now takes a node N and a set of termlists. It computes a new set of termlists by unifying every termlist in the set with the label of N and applying each unifier to N's variables. If the set is not empty, the search goes down into every subnode of N until the leaf nodes are reached. In this case one positive unification with one query termlist is sufficient to determine unifiability itself. In order to find also a complete set of unifiers, all unifiers with all query termlists have to be computed. If the unifier happens to be a matcher which instantiates into the query termlist only, the recursion may stop before reaching a leaf node because all subnodes of the actual node represent unifiable terms. This is easy to check and may considerably improve the performance of the query mechanism.

Finding Generalizations of Terms (Forward Subsumption)

The algorithm for finding generalizations of a term in an AT is the same as the unification query algorithm except that instead of unification, matching has to be used in order to prevent instantiation in the query terms. We must, however, follow the tree down to the leaf nodes because only the leaf nodes contain the full information about the variable bindings. For deciding whether this term is more general than a given query term the complete binding is needed.

Finding Instances of Terms (Backward Subsumption)

The instance query algorithm is again almost exactly like the unification query algorithm, except that matching instead of unification is used at the leaf nodes. It might be a little bit surprising that that unification instead of matching has to be used at the intermediate nodes, but the following example shows why. The AT for the two terms f(a,b) and f(c,d) is



The first term, f(a,b), is an instance of the query term f(a,z). Matching f(a,z) against the root node term, f(x,y), fails because x cannot be instantiated. In the "real" term, f(a,b), as represented by node 2, however, x is instantiated. Therefore preventing variables at intermediate nodes from becoming instantiated is not adequate in this case. Except for the leaf nodes we have to apply full unification.

If, however, the unifier happens to be a matcher which instantiates in the query term only, we can apply the same optimization as in the query unification algorithm and take all subnodes as instances without further unification or matching. This is in particular useful when the instances of a given term are to be removed from the index (backward subsumption). Simply removing the node with all its subnodes does the job in this case.

Completeness of the Query Evaluation Algorithms

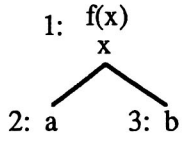
An important issue is the question whether the algorithms really find all possible answer terms and all unifiers or matchers respectively. In the case of interpreted functions the answer is not at all obvious. What we do, however, is nothing else than unification with variable abstraction. The abstracted terms are unified first and the unifiers are then merged with the variable bindings. Whenever this technique is complete, our query evaluation is also complete. This view shows also where the efficiency of the procedure comes from: A considerable part of the unification of the terms represented by all the subnodes of a node has already been done when the common abstracted term has been unified.

Insertion of Terms into ATs

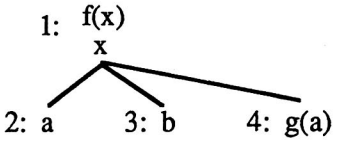
The price we have to pay for efficient query evaluation is a relative complex and, compared with other methods, more expensive insertion procedure. In most applications, however, insertion is combined with forward subsumption (insert unless subsumed) and backward subsumption (remove all instances from the tree). In this case, insertion together with the two subsumption tests can be combined to a quite efficient procedure.

The basic idea for the insertion procedure is to search the node N in the tree representing the most specific generalization of the term t to be inserted. If for some subnode M of N, there exists an "allowed" common generalization with t then M is replaced by a new node M´ with a modification of M and the new term as subnodes. If no allowed common generalization exists, t is made an additional subnode of N. The trivial generalization x,y for the two termlists a,b and c,d, for example, is not considered as an allowed generalization. For free terms, generalization of this kind are the only ones which are not allowed. For interpreted terms there may be more.

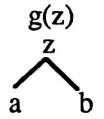
An Example: The AT for the two terms f(a) and f(b) is:



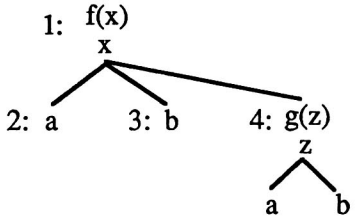
To insert the term t = f(g(a)), we find f(x) as the most specific generalization of t. The matcher, $\{x \mapsto g(a)\}$ is applied to x yielding g(a). Since there is no allowed common generalization, neither for a and g(a) nor for b and g(a), g(a) is made a new subnode of 1, i.e. we obtain



If the next term, s = f(g(b)), is to be inserted, we find again f(x) as most specific generalization. Now there is an allowed common generalization between g(a) (node 4) and g(b). Therefore node 4 is replaced by



and the whole tree is now



This procedure contains two nondeterminisms whose solution influences the growth of the tree. First of all the node representing the most specific generalization of the term to be inserted is not necessarily unique. One node has to be selected heuristically. If the insertion procedure is combined with forward subsumption where all nodes have to be checked whether they represent a generalization of the term to be inserted, all interesting nodes have to be visited anyway. In this case an optimal node can be selected according to some heuristical criteria, depth in the tree for example, or number of subnodes etc. The second nondeterminism comes from the problem to find a term with an optimal allowed common generalization. For this purpose, an heuristic assessment of the generalization possibilities which prefers stronger instantiated terms is necessary. There is a third kind of nondeterminism when interpreted terms are present. Since there need not be a unique most specific common generalization for two interpreted terms, one possibility has to be selected. Again, heuristical assessment is necessary. The heuristical solution of the nondeterminisms influence the structure of the generated AT and therefore its discriminative power - in the worst case the whole tree consists of a variable x as root node and a flat list of terms as subnodes. Complex heuristics may produce a well balanced tree, but slow down the insertion algorithm. As always, a good compromise has to be found.

AT-Indexing has not yet been implemented. Therefore no empirical experience about its performance is available at the time being.

References

BLC86 Butler, R., Lusk, E., McCune, W., Overbeek, R., *Paths to high-performance automated theorem proving.* CADE-8 proceedings, LNCS 230, Springer (1986).

HD82 Hoffman, C.M., O'Donnel, M.J., *Pattern matching in trees,* J.ACM 29,1 (1982), 68-95.

LO80 Lusk, E, Overbeek R., *Data structures and control architectures for the implementation of theorem proving programs.* CADE-5 proceedings, LNCS 87, Springer (1980), 232-249.

Pl69 Plotkin, G.D.: *A Note on Inductive Generalization,* Machine Intelligence 5, edited by Bernard Meltzer and Donald Michie, Edinburgh University Press (1969), 153-163.

Re69 Reynolds, J.C.: *Transformational Systems and the Algebraic Structure of Atomic Formulas* Machine Intelligence 5, edited by Bernard Meltzer and Donald Michie, Edinburgh University Press (1969), 135-151.

WP84 Wise, M., Powers, D. *Indexing Prolog clauses via superimposed codewords and field encoded words.* Proc. of the IEEE Conference on Logic Programming, Atlantic City, 203-210 (1984).

Section 6:
Special Algorithms II

S. Hölldobler: Unification over Rational Trees

L. Pottier: Term Generalizations Modulo Equations

H. Leiß: A Semi-Unification Algorithm?

B. Gramlich: Unification of Term Schemes

Unification over Rational Trees

Steffen Hölldobler
FG Intellektik, FB Informatik
Technische Hochschule Darmstadt
Alexanderstraße 10
6100 Darmstadt, Germany (W)
E-mail: xiisshoe@ddathd21.bitnet

It was Robinson's (1965) celebrated idea to put the unification process at the heart of the deduction mechanism. By that time unification was considered as the process of determining bindings for the variables occurring in two or more expressions such that their respective instances become equal within the Herbrand universe (or the domain of finite trees). In the meantime we have achieved a much broader view under which the original unification algorithm can be regarded as a special case of a more general *constraint solver*. Such a constraint solver computes solutions for a given set of constraints (i.e. a set of atoms and equations) over domains different from the Herbrand universe.

The development of such procedures was motivated, for example, by the need to solve linear equations or to evaluate boolean expressions within logic languages (see e.g. [Jaffar and Lassez, 1987; Büttner and Simonis, 1987]) or by the observation that the propagation of constraints may reduce the search space considerably [van Hentenryck and Dincbas, 1986]. But there was yet another reason.

Many implementations of logic languages such as Prolog have the following bug. Though a unification algorithm linear in time and space is known for the traditional unification problem [Paterson and Wegman, 1978], most existing Prolog systems incorrectly dispense with the so–called occur check for efficiency reasons. The *occur check* tests whether, for the unification of the variable x with the term t, x occurs in t. Techniques have been developed to determine for programs whether the occur check may safely be avoided or not (e.g. [Plaisted, 1984]). Though these methods reduce the costs connected with the occur check, it still has to be carried out it in certain cases.

Prolog programmers have argued that the need for the occur check arises in unnatural cases only and, thus, it is not worth to perform it. Hence, it is up to the programmer to write programs that still work in its absence. However, if an occur check problem occurs, then the user often cannot predict how the Prolog system will react. Marriott and Sondergaard [1988] have shown that the unification procedures, which are actually used in various Prolog implementations, do not deal uniformly with the occur check problem.

Even worse, in Poplog[1] – for example – a query causing an occur check problem will succeed, fail, or run forever depending on the query.

Colmerauer [1982] fixed this problem by computing unifiers in Prolog II no longer within the domain of finite trees, but within the domain of rational trees. His unification algorithm is based on the following transformations on a (multi-) set S of equations:

Compaction: Eliminate any equation of the form $x \doteq x$ from S, where x is a variable.

Merging: If $x \doteq y$ is in S, x and y are distinct variables, and x has further occurrences in S, then replace these further occurrences of x by occurrences of y.

Variable Anteposition: Replace $t \doteq x$ by $x \doteq t$ in S if x is a variable, and t is not a variable.

Confrontation: Replace $x \doteq s, x \doteq t$ by $x \doteq s, s \doteq t$ in S if x is a variable, s and t are not variables and $|s| \leq |t|$ where $|s|$ denotes the number of occurrences in s.

Splitting: Replace $f(s_1, ..., s_n) \doteq f(t_1, ..., t_n)$ by $s_1 \doteq t_1, ..., s_n \doteq t_n$ in S.

These transformations preserve the solutions of a set of equations. They are terminating, i.e. there is no infinite sequence of transformations which can be applied to a set of equations. If no transformation can be applied, then the set S of equations either contains an equation of the form $f(s_1, ..., s_n) \doteq g(t_1, ..., t_m)$, where f and g are different function symbols, and is said to be *unsolvable*, or S is in the form $\{x_1 \doteq t_1, ..., x_n \doteq t_n\}$, where the x_i's are different variables, and is said to be *solved* over the domain of rational trees.

Several problems remain. The unification procedures actually used in various Prolog implementations do not perform an occur check and are not based on Colmerauer's transformations either. In many cases we simply do not know how a Prolog system deals with an occur check problem (see [Marriott and Søndergaard, 1988]). This is unfortunate, especially because any solved set of equations over the domain of rational trees either inherits an occur check problem or it can be transformed into a solved set over the domain of *finite* trees. This transformation can be achieved by applying

Variable Elimination: If $x \doteq t$ is an element of S, x is a variable which does not occur in the term t, and x has further occurrences in S, then replace these further occurrences of x by occurrences of t.

It is easy to see that an application of variable elimination to a set S of equations, which is solved over the domain of rational trees, yields again a set solved over the domain of rational trees and preserves the solutions for S. Furthermore, since S is finite and each application eliminates a variable in the right–hand sides of the equations in S, it can be applied only finitely many times (see also [Martelli and Montanari, 1982]).

So why do we not answer queries posed to Prolog programs over the domain of rational trees first and, if the questioner is not satisfied, apply then variable elimination to obtain an answer over the domain of finite trees provided that such an answer exists?

There are other problems as well. The semantics of Prolog II is still unclear. Van Emden and Lloyd [1984] proved the soundness of Prolog II by viewing it as a logic language

[1]Poplog 9.3 on Vax 750 under UNIX BSD 4.3; see [Marriott and Sondergaard, 1988]

augmented with a special equational theory. Jaffar et al. [1985] showed that Prolog II is an instance of their logic programming language scheme. But neither do we have a strong completeness result for Prolog II so far nor is the question answered how the negation–as–failure rule can be added to Prolog II (see [Clark, 1988]).

However, in this talk, we concentrate on unification over rational trees. What happens if we consider unification problems under special equational theories such as associativity, commutativity, associativity and commutativity, etc. over the domain of rational trees? Are these problems decidable? Does there exist a set of most general unifiers? Is the set of solutions enumerable? Can we find a minimal or type conformal unification procedure? Can we increase the efficiency of special unification procedures over the Herbrand universe?

Looking at universal unification procedures over the Herbrand universe we almost always find that the search space contains far too many redundant or irrelevant inferences. As pointed out by Gallier and Snyder [1988] and by Hölldobler [1988], many of these inferences are linked to the way of how equations of the form $x \doteq t$ are treated. Can we eagerly eliminate the variable x if it does not occur in t? If so, what happens with equations of the form $x \doteq t$, where x occurs in t? Much of the inefficiency of complete sets of transformations for equational theories has its cause in the transformations additionally to be applied to equations of the form $x \doteq t$ in order to ensure the completeness of the calculus.

A typical example is an equational theory which contains a non–permutative axiom like

$$f(x, a) \doteq a.$$

Now, if we want to solve an equation like

$$y \doteq c(f(y, a))$$

over the Herbrand universe, then narrowing or some similar inference rule have to be applied upon the proper subterm $f(y, a)$ of the left–hand side of the equation and we obtain

$$y \doteq c(a),$$

which is in solved form over the Herbrand universe. Though this example terminates after one additional step, in general, the need to apply rules like narrowing upon proper subterms of the elements of an equation may lead to infinite derivations. But observe that the initial equation is already in solved form over the domain of rational trees.

Now look at complete sets of transformations for non–trivial (conditional) equational theories as developed by Gallier and Snyder [1988] and Hölldobler [1988]. Besides the rules to handle equations like $x \doteq t$ and to unify equations syntactically there is only one transformation rule:

Lazy Narrowing: Replace $f(t_1, ..., t_n) \doteq t_{n+1}$ by $s_1 \doteq t_1, ..., s_{n+1} \doteq t_{n+1}$ in S, if $f(s_1, ..., s_n) \doteq s_{n+1}$ is a new variant of an equational axiom and t_{n+1} is a not a variable[2] (Similarily for $t_{n+1} \doteq f(t_1, ..., t_n)$).

Together with Colmerauer's transformations lazy narrowing can be used to solve problems like the following. Let : be a binary function symbol written in infix notation and denoting the list constructor. Consider the axiom

$$int(x) \doteq x : int(s(x)),$$

which can be used to generate an infinite list of integers. Now, if we want to compare a partially instantiated list with a list of integers, we may ask whether there exists a solution for

$$int(0) \doteq 0 : y.$$

Applying lazy narrowing we obtain

$$x \doteq 0,\ x : int(s(x)) \doteq 0 : y.$$

Applying splitting and anteposing y yields

$$x \doteq 0,\ x \doteq 0,\ y \doteq int(s(0)).$$

Finally, applying confrontation followed by compaction we obtain the solved set

$$x \doteq 0,\ y \doteq int(s(x))$$

and the answer "y is bound to $int(1)$".

It should be observed that the search space generated is finite, whereas the search space generated by - for example - Gallier and Snyder's transformations is infinite. This gives rise to several questions as follows.

For which class of equational theories are Colmerauer's transformations together with the lazy narrowing rule a complete unification procedure over the domain of rational trees? Can we solve interesting problems with such a unification procedure? Can we implement these transformations efficiently? Can we use these transformations to compute solutions for universal unification problems over the domain of finite trees by first solving these problems over the domain of rational trees and, afterwards, checking whether these solution can be extended to solutionsb over finite trees?

In this talk we focus on these problems and show that Colmerauer's transformations and lazy narrowing are complete over the domain of rational trees for (conditional) canonical theories. This result is proven by transforming refutations with respect to the inference rules defined in [Hölldobler, 1988] into sequences of Colmerauer's transformations and lazy narrowing yielding solved forms. Moreover, we demonstrate that from these solved forms all solutions for unification problems over the domain of finite trees can be computed.

[2]In [Gallier and Snyder, 1988] and [Hölldobler, 1988] t_{n+1} may also be a variable.

References

[Büttner and Simonis, 1987] W. Büttner and H. Simonis. Embedding boolean expressions into logic programming. *Journal of Symbolic Computation*, 4:191–205, 1987.

[Clark, 1988] K. L. Clark. Logic programming schemes. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 120–139, 1988.

[Colmerauer, 1982] A. Colmerauer. Prolog and infinite trees. In Clark and Tarnlund, editors, *Logic Programming*, pages 231–251. Academic Press, 1982.

[Gallier and Snyder, 1988] J. H. Gallier and W. Snyder. Complete sets of transformations for general E–unification. Technical report, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, 1988.

[Hölldobler, 1988] S. Hölldobler. Horn equality theories and complete sets of transformations. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 405–412, 1988.

[Jaffar and Lassez, 1987] J. Jaffar and J-L. Lassez. Constraint logic programming. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, pages 111–119, 1987.

[Jaffar *et al.*, 1985] J. Jaffar, J.-L. Lassez, and M. J. Maher. Prolog II as an instance of the logic programming language scheme. Technical Report RC 11701, IBM T. J. Watson Research Center, 1985.

[Marriott and Sondergaard, 1988] K. Marriott and H. Sondergaard. On Prolog and the occur check problem. Technical Report 88/21, Department of Computer Science, University of Melbourne, 1988.

[Martelli and Montanari, 1982] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4:258–282, 1982.

[Paterson and Wegman, 1978] M. S. Paterson and M. N. Wegman. Linear unification. *Journal of Computer Systems Sciences*, 16:158–167, 1978.

[Plaisted, 1984] D. A. Plaisted. The occur-check problem in Prolog. *New Generation Computing*, 2:309–322, 1984.

[van Emden and Lloyd, 1984] M. H. van Emden and J. W. Lloyd. A logical reconstruction of Prolog II. *Journal of Logic Programming*, 1:143–149, 1984.

[van Hentenryck and Dincbas, 1986] P. van Hentenryck and M. Dincbas. Domains in logic programming. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 759–765, 1986.

TERM GENERALIZATION MODULO EQUATIONS

Loïc Pottier

July 26, 1989

Institut National de Recherche en Informatique et Automatique
2004 route des Lucioles
Sophia Antipolis
06565 Valbonne CEDEX
FRANCE

email: pottier@mirsa.inria.fr

Extended abstract

1 Introduction.

Generalization is the dual notion of unification. The problem is to study and compute the upper lower bounds of two terms, relative to the preorder of substitutions modulo an equationnal theory. We recall results in the initial algebra of terms, and present new results in the general case of quotient algebras, in the syntactic case, and in the associative, commutative, associative-commutative cases.

We use the notation of [G.Huet 80], and the following:

$T(F,V)$, T in short, is the algebra of terms built with a graduated set of functions F and a denumerable infinite set of variables V.

$E = \{e_1, \ldots, e_n\}$ is a set of equations theory, which generates a equivalence relation $=_E$ (= in short), the smallest equivalence relation compatible with functions of F and substitutions of V. When name E or $=_E$ indifferently *equational theory.*

$t \equiv t'$ is the syntactic equality of terms.

$E(t)$ is the equivalence class of t modulo $=_E$.

The substitution preorder modulo E is $\leq_E$, defined by $t \leq_E t'$ if and only if there exists a substitution σ such that $\sigma t =_E t'$ or equivalently $t' \in E(\sigma t)$.

This preorder define naturally a partial order $\preceq_E$, $\preceq$ in short, on the quotient set $\tilde{T}$ of T by the relation $\sim$, with $t \sim t'$ if and only if $t \leq_E t'$ and $t' \leq_E t$.
$\tilde{T}$ is named the set of *quotient terms*.

We define now the *principal generalizations* of $\tilde{t}$ and $\tilde{t'}$, noted $\tilde{t} \wedge \tilde{t'}$, as the set of upper lower bounds of $\tilde{t}$ and $\tilde{t'}$:

$$\tilde{t} \wedge \tilde{t'} = MAX\{\tilde{t_1} \mid \tilde{t_1} \preceq \tilde{t} \; and \; \tilde{t_1} \preceq \tilde{t'}\}$$

Then generalization can be viewed as the dual notion of unification.
The problem we address now is to determine properties and algorithms for $\tilde{t} \wedge \tilde{t'}$. We will give results in general, trivial, finite , associative, commutative, associative-commutative, and syntactic cases, without proofs (which can be found in [L.Pottier 89]). In the following we will use the notation $t \wedge t'$ to represent a set of elements of each class of $\tilde{t} \wedge \tilde{t'}$.

2 General case

In general we don't know if $\tilde{t} \wedge \tilde{t'}$ is empty or not. In some cases it can be infinite, as in the following example:
$E = \{\wedge(X,X) = X, \vee(X,X) = X\}$ (idempotent theory),
$t = \wedge(a,b), t' = \vee(a,b)$ (lower case letters are constants, others are variables)
$a'_0 = \wedge(\vee(a,X), \vee(Y,b))$
$a_0 = \vee(\wedge(a,Y), \wedge(X,b))$
...
$a_{n+1} = \vee(a'_0, \wedge(a'_0, a_n))$

Then the $\tilde{a_0}, \tilde{a_1}, \ldots, \tilde{a_n}, \ldots$ are an infinity of principal generalization of $\tilde{t}$ and $\tilde{t'}$.
Some work is needed to find properties of $\tilde{t} \wedge \tilde{t'}$ in the general case.

3 Initial algebra: $E = \emptyset$.

In this case we have no equation, and then $\tilde{t} = \tilde{t'} \Leftrightarrow \exists \sigma bijective, \sigma t = t'$ (equality up to variables' renaming).
We have then ($\#S$ denoting the cardinal of S) :

Property 1 *[G.Plotkin 70],[Reynolds 70]*
$\forall t, t', \#\tilde{t} \wedge \tilde{t'} = 1$

We describe now two algorithms to compute $\tilde{t} \wedge \tilde{t'}$.

3.1 Huet's algorithm

From [G.Huet 76].
Let $\Phi : \tilde{T} \times \tilde{T} \rightarrow V$ bijective, we define inductively $\tilde{t} \wedge \tilde{t'}$ by:

$$f(t_1, \ldots, t_n) \wedge f(t'_1, \ldots, t'_n) = f(t_1 \wedge t'_1, \ldots, t_n \wedge t'_n)$$
$$\text{else } t \wedge t' = \phi(t, t').$$

This gives a recursive algorithm with linear complexity (on the size of terms).

3.2 Jouannaud's algorithm

From [J.P.Jouannaud 89].
We use inference rules on *generalization problems*. A generalization problem is $G \mid S$ where G is a term, S is a list of $t_i =_{X_i} t'_i$, the X_i being variables of G.
The inference system is $\mathcal{SG}$ with two inference rules :

$\mathcal{R}_1$:

$$\frac{G \mid f(t_1,\ldots,t_n) = X f(t'_1,\ldots,t'_n), S}{[X \to f(Y_1,\ldots,Y_n)]G \mid t_1 = Y_1 t'_1,\ldots,t_n = Y_n t'_n, S}$$

where Y_i are new variables.

$\mathcal{R}_2$:

$$\frac{G \mid u = X v, u = Y y, S}{[X \to Y]G \mid u = Y y, S}$$

We have then :

Theorem 1 • *$\mathcal{SG}$ is noetherian.*

• *$G \mid S$ is a normal form of $X \mid t =_X t'$ implies that $\tilde{G}$ is the principal generalization of $\tilde{t}$ and $\tilde{t'}$.*

When have then a $O(nlog(n))$ complexity algorithm, but the rule $\mathcal{R}_2$ allows factorization of work, and can make the work of the bijection Φ of Huet.

4 Finite classes theories.

When all the classes $E(t)$ are finite, it is easy to show that $\tilde{t} \wedge \tilde{t'}$ is non empty and finite. By enumerating the classes and the terms below a given term, one can compute the principal generalization. However, this algorithm is intractable in cases like associativity or commutativity, in which classes are of exponential size. Some improvements can be made in these cases, which we detail now.

5 Associative, Commutative, Associative-commutative theories.

In these cases, classes $E(t)$ are finite, so we have always a non zero and finite number of principal generalizations.

5.1 Associativity

We use the canonical form of a A-term : fA where f is associative, A is a non-empty list of A-terms which have not the root f.

Like in the trivial case, we define a system of inference rules on generalization problems, $\mathcal{SGA} = \{\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_A\}$, with the new rule:

$\mathcal{R}_A$:

$$\frac{G \mid f\,A\,A' =_X f\,B\,B', S}{[X \to f(Y,Z)]G \mid f\,A =_Y f\,B, f\,A' =_Z f\,B', S}$$

where f is associative, A, A', B, B' are non empty, one of A and B have only one element , Y and Z are new variables.

We define $FN_A(t,t') = \{\tilde{G} \mid G \mid S$ is a normal form of $X \mid t =_X t'\}$. Then :

Theorem 2 • *$\mathcal{SGA}$ is noetherian.*

- $\tilde{t} \wedge \tilde{t'} = MAX(FN_A(t,t'))$.
- $\tilde{t} \wedge \tilde{t'} = FN_A(t,t')$ *is possible.*
- $\#\tilde{t} \wedge \tilde{t'}$ *is in general exponential in the size of* t *and* t'.

5.2 Commutativity

We have the same results as with associativity, with the system $\mathcal{SGC} = \{\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_C\}$, where :

$\mathcal{R}_C$:

$$\frac{G \mid f(t_0,t_1) =_X f(t'_0,t'_1), S}{[X \to f(Y,Z)]G \mid t_0 =_Y t'_i, t_1 =_Z t'_{1-i}, S}$$

where f is commutative , Y and Z are new variables.

5.3 Associativity-Commutativity

We use the canonical form of a AC-term : fA where f is associative-commutative, A is a non-empty multi-set of AC-terms which have not the root f.

We have the same results as with associativity, with the system $\mathcal{SGAC} = \{\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_{AC}\}$, where :

$\mathcal{R}_{AC}$:

$$\frac{G \mid f\,A\,A' =_X f\,B\,B', S}{[X \to f(Y,Z)]G \mid f\,A =_Y f\,B, f\,A' =_Z f\,B', S}$$

where f is associative-commutative, AA' and BB' are partitions of multisets, A, A', B, B' are non empty, one of A and B has only one element, Y and Z are new variables.

5.4 A + C + AC

We can use the system $\{\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_A, \mathcal{R}_C, \mathcal{R}_{AC}\}$ in mixed theories, and we obtain the same results as previously.

Remark : if we only want *linear* principal generalizations, it suffices to omit the rule $\mathcal{R}_2$; in this case the matching is much faster when computing the function MAX on the normal forms of generalization problems.

6 Syntactic theories.

Let $E = \{e_1, \ldots, e_n\}$ a syntactic theory [C.Kirchner 85]. Let $\mathcal{SG}_{synt} = \{\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_{synt}\}$, where : $\mathcal{R}_{synt}$:

$$\frac{G \,|\, t =_X t', S}{G \,|\, t'' =_X t', S}$$

where t'' is obtained by using an equation of E at the root of t.

Then $\tilde{t} \wedge \tilde{t'}$ is inclued in the set of first parts of generalization problems derived from $X \mid t =_X t'$ by $\mathcal{SG}_{synt}$.

7 Extensions, applications

We can extend this research in the following ways :

- Find *one* principal generalization.
- Find non trivial, but non principal generalizations.
- Find more results on the structure of $\tilde{t} \wedge \tilde{t'}$ in the general case.
- ...

We have applications of term generalization modulo equations in these domains :

- Compiling terms rewrite systems.
- Automated induction, rules learning [L.Pottier 89].
- Infinite Knuth-Bendix completion [H.Kirchner 87].
- ...

References

[G.Huet 76] "Résolution d'équations dans les langages d'ordre 1 ... ω" Thèse d'Etat, Univ. Paris 7, 1976.

[G.Huet 80] "Confluent reduction: abstract properties and applications to term rewrite systems", J.ACM 27 4 pp 787-821, oct.80.

[J.P.Jouannaud 89] private communication.

[C.Kirchner 85] "Méthodes et Outils de Conception systématique d'Algorithmes d'Unification dans les Théories équationnelles" , Thèse d'Etat, Univ. Nancy, France, 1985.

[H.Kirchner 87] "Shematization of infinite sets of rewrite rules. Application to the divergence of completion process" in P.Lescanne (editor) Proceedings of the second conference on Rewriting Techniques and Applications, Springer Verlag, Bordeaux (France) May 1987.

[G.Plotkin 70] "A note on inductive generalization" Machine Intelligence 5 pp 153-163, Edinburgh Univ. Press ed. 1970.

[L.Pottier 89] "Algorithmes de complétion et généralisation en logique du premier ordre", Thèse de Doctorat, Université de Nice, février 1989.

[Reynolds 70] "Transformational systems and the algebraic structure of atomic formulas" Machine Intelligence 5 pp 135-152, Edinburgh Univ. Press ed. 1970.

A Semi-Unification Algorithm ? Extended Abstract

Hans Leiß
Siemens AG, ZFE F2 INF 2
Otto-Hahn-Ring 6, 8000 München 83

July 26, 1989

Abstract

Semi-unification is a common generalization of unification and matching. It seems to be a new concept introduced independently by several authors to overcome a limitation of unification-based type inference algorithms (cf. [1], [2], [5]). We present a new reduction calculus for semi-unification problems and discuss partial results on termination.

Motivation. Milner's unification-based type-inference algorithm, used in functional languages like ML or Miranda, forces the recursion operator to be *monomorphic*: a recursive definition for a function f is considered well-typed only if the type of the defining term equals the type of f used in the definition. A more powerful typing scheme would be *polymorphic* recursion, where the types of f inside the definition may be *instances* of the type of the defining term (cf.[4]). To infer the type of a polymorphic recursive function f, one first infers a type of the defining term from type assumptions for f's occurences in this term, and then uses semi-unification to refine these types in order to make the refined types of the occurences of f be instances of the refined type of the defining term.

In general, semi-unification can be formulated as follows.

Definition 1 Let S be a finite set of equations and inequalities between first-order terms, using different inequality symbols $\leq_1, \ldots, \leq_q$. A substitution R is a *semi-unifier* for S if

- $R(s) \equiv R(t)$ for each equation $s = t \in S$, and
- there are *residual* substitutions $T_1, \ldots, T_q$ such that for $1 \leq i \leq q$ $T_i R(s) \equiv R(t)$ for each inequation $s \leq_i t \in S$.

S is called a *uniform* semi-unification problem, if $q \leq 1$, and *non-uniform* otherwise. A semi-unifier R for S is *most general*, if for any semi-unifier U for S there is a substitution $\tilde{U}$ such that $U = \tilde{U} \circ R$.

It is obvious that matching and unification are special cases of semi-unification where R and $T_1, \ldots, T_q$ are the identity, respectively. Semi-unification is unsymmetric: x semi-unifies with $f(x)$ as it matches with $f(x)$, but $f(x)$ neither matches, unifies nor semi-unifies with x.

Due to the existence condition for residual substitutions, semi-unification is much more difficult than unification. We elaborate on

Conjecture 1 ([5], [1]) There is an algorithm *semi-unify* that, given a finite set S of equalities and inequalities, decides whether there is a semi-unifier for S, and in case there is, returns a most general one.

It is necessary to construct the residual substitutions along with the most general unifier. During the construction, the residual substitutions are represented by unknowns that operate as homomorphisms on terms, and any combination of these homomorphisms may occur. Therefore, we extend the original language L, say, by non-atomic variables $x^{i_1 \cdots i_n}$ representing the term $T_{i_n} \ldots T_{i_1} R(x)$, where T_i and R are the substitutions to be constructed. Inequalities between L-terms are translated into equalities between L^*-terms. A reduction calculus is used to solve *generalized* semi-unification problems, i.e. finite sets of L^*-equations.

Definition 2 Let $I = \{1, \ldots, q\}$, and let $\{\leq_i | \ i \in I\}$ be the binary relation symbols of L. Let I^* be the set of finite words over I, and define a new set of variables by $Var(I^*) = \{x^w \mid x \in Var, w \in I^*\}$. *$L^*$-terms* are build like L-terms, but with variables taken from $Var(I^*)$. The set of variables free in an L^*-term t is defined using $free(x^w) = \{x^u \mid u \in I^*, uv = w \text{ for some } v \in I^*\}$. For each $u \in I^*$ and L^*-term t define an L^*-term t^u via $(x^v)^u = x^{vu}$ and $f(s_1, ..., s_n)^u = f(s_1^u, ..., s_n^u)$.

Definition 3 Let $T = \{T_i \mid i \in I\}$ be a sequence of substitutions T_i : Var $\rightarrow$ L^*-term, and R : $Var \rightarrow$ L-term. For any L^*-term s, define $T(s)$ and $R(s)$ by

$$T(x) = x, \quad T(x^{vi}) = T_i(T(x^v)), \quad T(f(s_1, ..., s_n)) = f(T(s_1), ..., T(s_n)),$$

$$\text{and} \quad R(x^v) = R(x)^v, \quad R(f(s_1, ..., s_n)) = f(R(s_1), ..., R(s_n)).$$

For any set S of equations between L^*-terms, we say (T, R) *solves* S, if

$$TR(s) \equiv TR(t) \quad \text{for each } s = t \in S.$$

We say (T, R) is a *most general solution* (*mgs*) of S, if whenever $(\tilde{T}, \tilde{R})$ solves S there is R' : $Var \rightarrow$ L-term such that $\tilde{R} = R'R$.

Lemma 1 Let S be a set of atomic L-formulas. R is a most general semi-unifier (*mgsu*) for S iff there is a sequence $T = \{T_i \mid i \in I\}$, with T_i : $Var \rightarrow$ L-term, such that (T, R) is a most general solution of

$$S^* := \{s = t \mid s = t \in S\} \cup \{s^i = t \mid i \in I, s \leq_i t \ \in S\}.$$

Example 1 Let S be $\{x \leq_i y, f(z) \leq_i x\}$. Then $S^* = \{x^i = y, f(z^i) = x\}$ has most general solution (T, R), where

$$R = [f(z_1)/x, f(z_2)/y], \quad T = [z_2/z_1, z_1/z],$$

with fresh variables z_1 and z_2. So R is a mgsu of S. The simpler $(\tilde{T}, \tilde{R}) = (Id\ , [f(z)/x, f(z)/y])$ also solves S^*, but $\tilde{R}$ is not a mgsu of S.

The following reduction calculus has been designed so that at most one reduction rule applies to a selected equation of S. Hence the conditions which make the rules look complicated. If we would drop the equations used in substitution rules (6) and (10), the calculus would certainly be incomplete.

Definition 4 A *basic reduction* is any triple $S_1 \xrightarrow{R} S_2$ between sets S_1, S_2 of sets of L^*-equations and a substitution R : $Var \to L$-term of the following form:

1. Elimination of structure:

$$S \cup \{f(s_1, ..., s_n) = f(t_1, ..., t_n)\} \xrightarrow{Id} S \cup \{t_1 = s_1, ..., t_n = s_n\} \quad (1)$$

$$S \cup \{f(s_1, ..., s_n) = g(t_1, ..., t_m)\} \xrightarrow{Id} \text{fail} \quad (2)$$

2. Normalization of basic equations:

$$S \cup \{t = x^v\} \xrightarrow{Id} S \cup \{x^v = t\}, \quad (3)$$
$$\text{if } t \notin \text{Var}(I^*)$$

$$S \cup \{y^w = x^v\} \xrightarrow{Id} S \cup \{x^v = y^w\}, \quad (4)$$
$$\text{if length}(w) < \text{length}(v)$$

3. Elimination of variables:

$$S \cup \{x^v = x^v\} \xrightarrow{Id} S \quad (5)$$

$$S \cup \{x^v = y^w\} \xrightarrow{Id} S[y^w/x^v] \cup \{x^v = y^w\}, \quad (6)$$
$$\text{if } 0 \neq \text{length}(v) \geq \text{length}(w),\ S[y^w/x^v] \neq S$$

$$S \cup \{x^v = s\} \xrightarrow{Id} \text{fail}, \quad (7)$$
$$\text{if } x^v \in \text{free}(s), s \notin \text{Var}(I^*)$$

$$S \cup \{x = s\} \xrightarrow{[s/x]} S[s/x], \quad (8)$$
$$\text{if } x \notin \text{free}(s),\ s \in L\text{-term} \quad (9)$$

$$S \cup \{x^v = s\} \xrightarrow{Id} S[s/x^v] \cup \{x^v = s\}, \quad (10)$$
$$\text{if } x^v \notin \text{free}(s),\ s \notin \text{Var}(I^*), S[s/x^v] \neq S,$$
$$\text{and } v \not\equiv \epsilon \text{ or } s \notin L\text{-term}$$

Define a relation $S_1 \xrightarrow{R} * S_2$ by finite sequencing of basic reductions, where R is the composition of the substitutions involved. A set S of L^*-equations is called *reduced* if none of the basic reductions can be applied to S.

Theorem 1 1. If $S_a \xrightarrow{R_a} S_b$, then (T, R_b) solves S_b iff $(T, R_b R_a)$ solves S_a.

2. Let (T, R_e) be a mgs of S_e, and $S_a \xrightarrow{R} * S_e$. Then $(T, R_e R)$ is a mgs of S_a.

3. If $S \neq \mathit{fail}$ is reduced, there is a mgs (T, R) of S.

Thus every set of L^*-equations that can be reduced has a most general semi-unifier. We conjecture that *any* reduction sequence is finite. It is not hard to see that any reduction sequence not using rule (10) is finite. Some termination results for restricted classes of semi-unification problems and proofs of the above lemma and theorem are given in [6]. In particular, we there give a termination proof of reductions for uniform semi-unification, which was solved earlier by [1] and [2]. Our reduction method is an extension of the method of Kapur e.a.[2], but has been developped independently. However, the (proof of the) polynomial time bound of [2] does not extend to non-uniform problems, as it rests on a cancellation reduction rule that is unsound in the nonuniform case.

In addition, we have a proof of the following special case, which will be published elsewhere.

Theorem 2 The semi-unification problem restricted to instances with at most two variables is decidable.

The proof constructs a most general semi-unifier for solvable instances, but does not extend to the *generalized* semi-unification problem in two variables in an obvious way.

Kfoury e.a.[3] give a solution of the semi-unification problem with *linear* left-hand sides, i.e. where variables may occur at most once in s of each $s \leq_i t$ in S. (They do not allow equations in S, so the linearity condition applies to the set obtained after solving the equations by unification.) Pudlák [7] shows that semi-unification with many inequality relations can be reduced to semi-unification with two inequality relations.

References

[1] F. Henglein. Type inference and semi-unification. In *Proceedings of the 1988 ACM Conference on LISP and Functional Programming. Snowbird, Utah, July 25-27*, pages 184–197, 1988.

[2] D. Kapur, D. Musser, P. Narendran, and J. Stillman. Semi-unification. In *Proceedings of the 8th Conference on Foundations of Software Technology and Theoretical Computer Science. Pune, India, December 21 - 23, 1988*, pages 435 – 454. Springer LNCS 338.

[3] A. J. Kfoury, J. Tiuryn, and P. Urzyczyn. Computational consequences and partial solutions of a generalized unification problem. In *Fourth IEEE Symposium on Logic in Computer Science. Asilomar, California, June 5-8, 1989.*

[4] A. J. Kfoury, J. Tiuryn, and P. Urzyczyn. On the computational power of universally polymorphic recursion. In *Third Annual Symposium on Logic in Computer Science. Edinburgh, Scotland, July 5–8, 1988*, pages 72–83.

[5] H. Leiß. On type inference for object-oriented programming languages. In E. Börger, H. Kleine-Büning, and M. M. Richter, editors, *CSL '87. 1st Workshop on Computer Science Logic. Karlsruhe, FRG, October 12–16, 1987*, pages 151–172. Springer LNCS 329, 1988.

[6] H. Leiß. Semi-unification and type inference for polymorphic recursion. Bericht INF2-ASE-5-89, Siemens AG, München, May 1989.

[7] P. Pudlák. On a unification problem related to Kreisel's conjecture. *Commentationes Mathematicae Universitatis Carolinae*, 29(3):551–556, 1988.

Unification of Term Schemes

Bernhard Gramlich, FB Informatik, Universität Kaiserslautern,
Postfach 3049, D-6750 Kaiserslautern, FR Germany
E-mail: gramlich@uklirb.uucp

Extended Abstract

We present a new approach for solving certain infinite sets of first order unification problems represented by term schemes. Within the framework of second order equational logic solving such scheme unification problems amounts exactly to solving (variable-) restricted unification problems. A method known for solving first order restricted unification problems (cf. [Bü87]) is generalized to the second order case. Essentially this is achieved by transforming a restricted unification problem into an unrestricted one, solving the latter and retransforming the solutions obtained. The results on second order restricted unification are then used to solve the original problem, namely to decide the solvability of scheme unification problems and - in the positive case - to compute the corresponding most general unifiers.

The motivation for this work stems from the analysis of the divergence phenomenon in Knuth-Bendix like completion procedures for term rewriting systems (cf. [KnBe70]). Consider for example the equations A: f(h(u),v) = f(u,h(v)) and B_0: f(w,g(w)) = w. Completion with input $\{A, B_0\}$ and an appropiate reduction ordering diverges, i.e. does not terminate, producing the following infinite sequence of rewrite rules:

A $\quad f(h(u),v) \rightarrow f(u,h(v))$

B_0 $\quad f(w,g(w)) \rightarrow w$

B_1 $\quad f(w,h(g(h(w)))) \rightarrow h(w)$

B_2 $\quad f(w,h(h(g(h(h(w)))))) \rightarrow h(h(w))$

.....

B_n $\quad f(w,h^n(g(h^n(w)))) \rightarrow h^n(w)$

.....

The reason for divergence obviously comes from the fact that any left hand side l_n of B_n is unifiable with the left hand side l of A yielding via critical pair construction $l_{n+1} \rightarrow r_{n+1}$. One might conjecture that this "repeated unifiablity" of l with every l_n is due to the fact that l is unifiable with every "instance of the term scheme" f(w,Y(w)) producing as new left hand side l_{n+1} again a term of this form.

A formal definition of this kind of unification problems as well as solution methods for them are developed within the framework of second order logic. The "scheme variable" Y above will

become a unary second order (function) variable. Let us give a rough sketch of our approach and its main results assuming familiarity with the basic notions and results of λ-calculus (cf. [Hu76], [GaSn88]). A detailed presentation of this work including proofs and various examples can be found in [Gr88].

Scheme Unification and Restricted Unification

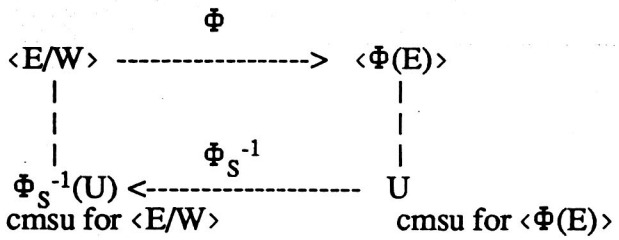
Let $\mathcal{T} = \mathcal{T}(\mathcal{C},\mathcal{V})$ be the set of restricted second order terms built as usual over some set $\mathcal{C}$ of function constants and some family $\mathcal{V} = \cup_{n\geq 0}\mathcal{V}_n$ of sets $\mathcal{V}_n$ of n-ary (function) variables. General second order terms are obtained by completing $\mathcal{T}$ into $\underline{\mathcal{T}}$ using λ-abstraction (cf. [Hu76]). $\underline{\mathcal{T}}$ may be regarded as the restriction of the language of extensional normal forms of λ-calculus to second order terms. Terms are compared via their extensional normal forms and modulo α-conversion, i.e. renaming of bound variables. For instance $X \in \mathcal{V}_2$, $\lambda uv.X(u,v)$ and $\lambda yz.X(y,z)$ are considered to be equal ($u, v, y, z \in \mathcal{V}_0$). The set of free variables of a term $t \in \underline{\mathcal{T}}$ is denoted by $\mathbf{V(t)}$. In a straightforward way we then define second order substitutions σ with domain $\mathbf{D(\sigma)}$ and set of introduced variables $\mathbf{I(\sigma)}$. The restriction of a substitution σ to a set W of variables is denoted by $\sigma|_W$. σ is said to be **strict** if for all $X \in D(\sigma)$ with $\sigma(X) = \lambda u_1 \ldots u_{arity(X)}.t$ we have $\{u_1,\ldots,u_{arity(X)}\} \subseteq V(t)$.

A **scheme unification problem** (SUP) is represented by a finite set E of (unordered) term pairs with its set V(E) of free variables partitioned into a set $W \subseteq \mathcal{V}_0$ of ordinary first order variables and a set $W^c := V(E)\backslash W$ of (possibly second order) scheme variables. Solving such an SUP denoted by ⟨E/W⟩ consists in deciding whether every first order unification problem (UP) ⟨ψ(E)⟩, where ψ is a substitution with $D(\psi) \subseteq W^c$, $I(\psi) \cap W = \emptyset$ and $V(\psi(W^c)) \subseteq \mathcal{V}_0$, is solvable, and if so, in computing the unifiers for all these problems.

In order to be able to handle SUP's we consider the following generalized form of (second-order) unification problems.

A **(variable-) restricted unification problem (RUP)** is given by ⟨E/W⟩, where E is a finite set of (unordered) term pairs and $W \subseteq V(E)$. A solution (or variable-restricted unifier) of the RUP ⟨E/W⟩ is any substitution σ that solves the equations from E but leaves the variables from W^c untouched, i.e. $\sigma(s) = \sigma(t)$ for all $(s,t) \in E$ and $D(\sigma) \cap W^c = \emptyset$.

Note that second order unification is in general undecidable ([Go81]) and - unlike the first order case - most general unifiers (mgu's) do not necessarily exist any more for solvable problems. Instead appropiate notions of complete and minimal sets of unifiers (csu/cmsu) can be defined (cf. [Hu76], [SnGa88]). We generalize these definitions as well as that of solved forms to the case of RUP's. Then we show how to transform bijectively an arbitrary RUP ⟨E/W⟩ into an equivalent unrestricted UP ⟨Φ(E)⟩ by interpreting the forbidden variables from W^c as distinct new funcion constants of corresponding arity. It is proved that the properties of substitutions to be (most general) solutions and of solution sets to be complete (and minimal) are preserved under this transformation (see figure, where Φ_S is the induced transformation function for substitutions σ with $D(\sigma) \cap W^c = \emptyset$ and Φ_S^{-1} its inverse).



If only the forbidden variables from W^c are allowed to be of second order, the transformed problem <Φ(E)> essentially is a first order UP (over a different signature). And for such problems it is well-known that they are either unsolvable or possess an mgu which may be computed by any of the well-known first order unification algorithms.

Since for SUP's we are interested in unrestricted UP's (obtained by instantiating the original UP) we next investigate the connection between RUP's and their unrestricted versions. Clearly every solution of the restricted problem also solves the unrestricted one, but not vice versa in general. For the solution sets we have the following results. An mgu for a first order RUP <E/W>, i.e. $V(E) \subseteq \mathcal{V}_0$, also is an mgu for the unrestricted UP <E>. But this result cannot be generalized to the case that E contains second order variables. To be more precise: If a set S of substitutions is a csu for <E/W> with $V(E) \not\subseteq \mathcal{V}_0$ then S is not necessarily a csu for <E>. This is shown by a simple counterexample.

From now on we consider only RUP's <E/W> with $W \subseteq \mathcal{V}_0$, i.e. the variables allowed for substitution are first order ones. For the purpose of solving the corresponding SUP <E/W> we have to consider the unrestricted versions <ψ(E)> of RUP-instances <ψ(E)/W> with $D(\psi) \subseteq V(E)\backslash W$, $I(\psi) \cap W = \emptyset$ and $V(\psi(V(E)\backslash W)) \subseteq \mathcal{V}_0$. We proceed incrementally and investigate first the restricted versions of RUP-instances <ψ(E)/W> still allowing second order variables to be introduced by ψ.

Theorem (solving instantiated RUP's)

Let <E/W> with $E \subseteq \mathcal{T}^2$, $W \subseteq V(E)$, $W \subseteq \mathcal{V}_0$ be a solvable RUP and σ be a mgu for it (w.l.o.g. let $I(\sigma) \subseteq V(E)$). Assume further that ψ is a substitution with $D(\psi) \subseteq V(E)\backslash W$, $I(\psi) \cap W = \emptyset$. Then $(\psi\sigma)|_{D(\sigma)}$ is a solution for the RUP <ψ(E)/W> which is most general for strict ψ.

This first main result establishes a uniform kind of computing the solutions for instantiated RUP's (for strict ψ) in the following sense. Provided that <E/W> is solvable, say with mgu σ, we need not explicitely solve <ψ(E)/W> again but can directly compute an mgu for it from ψ and σ as described. As an immediate consequence of this result we obtain the following

Corollary

For $E \subseteq \mathcal{T}^2$, $W \subseteq V(E)$, $W \subseteq \mathcal{V}_0$ the RUP <E/W> is solvable iff <ψ(E)/W> is solvable for every ψ with $D(\psi) \subseteq V(E)\backslash W$ and $I(\psi) \cap W = \emptyset$, and if so, for any such ψ which is strict the mgu for <ψ(E)/W> may be computed from the mgu for <E/W> as described by the above theorem.

In order to solve our original SUP $\langle E/W \rangle$ we have to restrict the substitutions ψ with $D(\psi) \subseteq V(E) \setminus W$ and $I(\psi) \cap W = \emptyset$ to the case $V(\psi(V(E) \setminus W)) \subseteq \mathcal{V}_0$ and to investigate the resulting unrestricted UP's $\langle \psi(E) \rangle$.

Theorem (solving scheme unification problems)

Let $E \subseteq \mathcal{T}(\mathcal{C},\mathcal{V})^2$, $W \subseteq V(E)$, $W \subseteq \mathcal{V}_0$ be given. Assume further that the underlying signature $\mathcal{C}$ contains at least one function constant of arity ≥ 1 and another one of arity ≥ 2. Then the RUP $\langle E/W \rangle$ is solvable iff the SUP $\langle E/W \rangle$ is solvable, i.e. $\langle \psi(E) \rangle$ is solvable for every ψ with $D(\psi) \subseteq V(E) \setminus W$, $I(\psi) \cap W = \emptyset$ and $V(\psi(V(E) \setminus W)) \subseteq \mathcal{V}_0$, and if so, for any such ψ which is strict the mgu for $\langle \psi(E) \rangle$ may be computed from the mgu σ for $\langle E/W \rangle$ as $(\psi\sigma)|_{D(\sigma)}$.

Thus it turns out that solvability of the (infinitely many) ordinary first order UP's defined by an SUP $\langle E/W \rangle$ is equivalent to solvability of the corresponding RUP $\langle E/W \rangle$ provided that the underlying signature is rich enough. And moreover, in the case of solvability, we get a uniform way of computing the respective mgu's of the instantiated (unrestricted) UP's.

Based on these general results we finally show how to get sufficient conditions for a property of "repeated unifiability" which may be applied for instance to divergence analysis of completion. In the introductory example we can thus explain and describe divergence for every starting set of rules consisting of A: $f(h(u),v) \rightarrow f(u,h(v))$ and B_0: $\psi_0(f(w,Y(w))) \rightarrow w$, where ψ_0 is any (strict) substitution with $D(\psi_0) = \{Y\}$, $I(\psi_0) \subseteq \mathcal{V}_0$ and $I(\psi_0) \cap \{w\} = \emptyset$, e.g. $\psi_0 = \{Y \leftarrow \lambda x.g(x)\}$. By the same technique we can explain and describe divergence even for more general starting situations, e.g. for A of the form $f(h(u),v) \rightarrow \varphi_0(f(u,Z(v)))$ such that φ_0 is any (strict) substitution with $D(\varphi_0) = \{Z\}$, $I(\varphi_0) \subseteq \mathcal{V}_0$ and $I(\varphi_0) \cap \{u,v\} = \emptyset$ (indeed, here we should require that φ_0 does not introduce any new variable, otherwise A would contain extra variables on the right hand side).

A more detailed presentation of this kind of applications of the theoretical framework can be found in [Gr88].

References

[Bü87] Bürckert, H.-J.: Matching - A Special Case of Unification, SEKI-Report SR-87-08,University of Kaiserslautern, 1987, to appear also in a Special Issue on Unification of JSC

[Go81] Goldfarb, W.: The Undecidability of the Second-Order Unification Problem, TCS 13/2, 1981

[Gr88] Gramlich, B.:Unification of Term Schemes - Theory and Applications, SEKI-Report SR-88-18, University of Kaiserslautern, 1988

[Hu76] Huet, G.: Resolution d'Equations dans les Languages d'Ordre 1,2,...,ω, These d'Etat, Universite de Paris VII, 1976

[KnBe70] Knuth, D.E., Bendix,P.B.: Simple word problems in universal algebras, in "Computational Problems in Abstract Algebra", Pergamon Press, 1970

[SnGa87] Snyder, W., Gallier, J.H.: A General Complete E-Unification Procedure, Proc. of 2nd RTA, Bordeaux, France,1987

[SnGa88] Snyder, W., Gallier, J.H.: Higher Order Unification Revisited: Complete Sets of Transformations, research report, University of Pennsylvania, USA, March 1988

Section 7:
Equational Problems & Constraint Solving

C. Kirchner, H. Kirchner: Constrained Equational Reasoning for Completion

H. Comon: Solving Equational Problems in an Order-Sorted Algebra

H. Aït-Kaci: Disjunctive ψ-Term Unification

L. Puel, A. Suarez: Unification of Restricted Terms

Constrained Equational Reasoning for Completion

Claude Kirchner
LORIA & INRIA

Hélène Kirchner
LORIA & CRIN

BP239, 54506 Vandoeuvre les Nancy Cedex, France.
E-mail: kirchner@loria.crin.fr

Abstract

Equational reasoning is based on replacement of equal by equal and these replacements use substitutions determined as solutions of equations. Constrained equational reasoning takes advantage from the information contained into the equation itself to develop equational reasoning, avoids instantiations as much as possible and solve equations as late as possible. For theories like associativity-commutativity this can lead to a considerable reduction of the substitution computation overhead, bringing some problems, untractable in practice, to the realm of computational reality.

1 Introduction

Equational reasoning and term rewriting, its operational realization, are quite important in today computer science. They play a central role in theorem proving [27,30] and in programming languages like OBJ [11]. Extensions of standard term rewriting like rewriting modulo associativity-commutativity [29] (AC in short) are quite powerful in theory but limitated in application because of the computational cost involved in computing (e.g. associative-commutative) unifiers and matchers (both problems are NP-complete for AC [20]) or in checking equality. We focuss here on completion processes modulo a theory T and on their two basic operations involving equational reasoning, namely rewriting and computing critical pairs by unification.

This work relies for one part on the remark that rewriting or computing critical pairs amounts to solve equations; however, in some cases, not the whole information contained in the solution is needed. It may happen that the equation, as it stands, contains enough information to continue the reduction or the completion process. Especially when an equation has an infinite, or at least a huge set of solutions, one can keep the equation and try to reason formally with it, instead of effectively computing the solutions.

Assuming that we also express the orientation problem of an equation into a rule as an inequation between two terms in a given ordering, such as the recursive path ordering for instance, we shall get as a by-product the expression of completion modulo a theory as a pure equational problem.

Beyond equations and inequations, another type of constraints considered here are disequations. They can be introduced for termination problem of class rewriting as in [33]. Disequations can also be used to translate equational problems into another equational theory for which the problem is simpler to solve, or for which the expression of the solutions is simpler. A typical case is for associativity, commutativity and identity ($AC1$ in short) in which an AC problem can be expressed and solved more simply [9].

Moreover the accumulation of equational constraints in a given problem decreases the size of the search space quite a lot. This is quite similar to what has been done for logic programming with constraints [14,8].

These ideas are exploited here, using constrained equational reasoning, for giving another method for completion modulo a theory T.

2 A simple example

Let us first develop a simple example that illustrates constrained equational reasoning.

Let a be a constant, let the operator $*$ be associative and commutative i.e. satisfying:

$$x * (y * z) = (x * y) * z$$
$$x * y = y * x$$

and let R be the following term rewriting system:

$$(1) \quad x * x * x * x \rightarrow x$$
$$(2) \quad u * v * w * a \rightarrow a$$

In an equational completion process [2,16], aimed to build from R a canonical term rewriting system modulo AC, the critical pairs of the rule (1) on rule (2) have to be computed. For that, the equation

$$x * x * x * x =_{AC} u * v * w * a$$

has to be solved modulo AC. Unfortunately, this equation has a huge number of AC-unifiers, which makes the completion process untractable.

A first solution, developped in [9], consists in using the fact that $AC1$-unification is simpler than AC-unification [4] and to solve the previous equation in a protected enrichment in which it has considerably less solutions.

But here we shall rather consider the equation $(x*x*x*x =_{AC} u*v*w*a)$ as a constraint that the variables x, u, v, w must satisfy and we do not solve it for now. Instead, we only test its satisfiability and build a constrained critical pair:

$$([x, a], \{x*x*x*x =_{AC} u*v*w*a\}).$$

This pair is directed into a constrained rule

$$(x \to a, \{x*x*x*x =_{AC} u*v*w*a,\ x \neq a\}).$$

where the new constraint $(x \neq a)$ is added in order to avoid a trivial rewrite rule $(a \to a)$ as a possible instantiation. This constrained rule can then be used to simplify other rules. Checking if a term t is reducible on top amounts to solve a match equation $(x \ll_{AC} t)$ together with the contraints already associated to x. During the completion process, it is needed to assume that rewrite rules do not have common variables. For that, the second rule is renamed into $(u'*v'*w'*a \to a)$. Its left-hand side is reducible at top occurrence by the constrained rule because the set of constraints $\mathcal{E}$:

$$\begin{gathered} x*x*x*x =_{AC} u*v*w*a \\ x \neq a \\ x \ll_{AC} u'*v'*w'*a \end{gathered}$$

is satisfiable, for instance by the substitution

$$\begin{gathered} (x \mapsto u'*v'*w'*a) \\ (u \mapsto u'*v'*w'*a) \\ (v \mapsto u'*v'*w'*a) \\ (w \mapsto u'*u'*v'*v'*w'*w'*a). \end{gathered}$$

The left-hand side reduces to the constrained term $(a, \mathcal{E})$ and the constrained pair resulting from this simplification $(a = a, \mathcal{E})$ can then be deleted.

On the contrary the first rule $(x'*x'*x'*x' \to x')$ is not reducible by the constrained rule, because the set of constraints

$$\begin{gathered} x*x*x*x =_{AC} u*v*w*a \\ x \neq a \\ x \ll_{AC} x'*x'*x'*x' \end{gathered}$$

is not satisfiable. The (constrained) rule system is now

$$\begin{gathered} x \to a, \{x*x*x*x =_{AC} u*v*w*a,\ x \neq a\} \\ x'*x'*x'*x' \to x'. \end{gathered}$$

However obviously some critical pairs have not been computed: precisely those coming from superposition of the second rule into a possible instantiation of x. At this point, two directions are possible: either try to go on with formal reasoning on constraints and express conditions stating that such critical pairs are or are not convergent, or solve constraints. This is the last solution that we propose here. A midterm is to solve constraint lazily, that is to transform the set of constraints into another equivalent one and try to reason on this more solved form. For instance, the set of constraints

$$\begin{gathered} x*x*x*x =_{AC} u*v*w*a \\ x \neq a \end{gathered}$$

is equivalent modulo AC to

$$\begin{gathered} x = a*z \\ a*a*a*z*z*z*z =_{AC} u*v*w. \end{gathered}$$

Using this new set of constraints, one can find the superpositions of the rule $(x'*x'*x'*x' \to x')$ into the constrained solved form of x at top occurrence, because the set of constraints

$$\begin{gathered} x = a*z \\ a*a*a*z*z*z*z =_{AC} u*v*w \\ x =_{AC} x'*x'*x'*x' \end{gathered}$$

is satisfiable.

3 Foundations

We assume the reader familiar with equational logic [32], class rewriting systems, also called equational term rewriting systems [13,1,16], and equational problems [21,5,6,3,23].

We use the following usual notations: given a set T of equalities and a set $\mathcal{R}$ of rewrite rules, $\longleftrightarrow^T$ denotes one step of replacement of equal by equal in T, $\overset{*}{\longleftrightarrow}^T$ or $=_T$ denotes the reflexive symmetric transitive closure of the previous relation, and $\to^{\mathcal{R},T}$ denotes equational rewriting modulo T with rules in $\mathcal{R}$ [29].

3.1 Equational problems

Given an equational theory T' and an arbitrary subtheory T of T', an **equational problem** is built from disjunctions ($\vee$) and conjunctions ($\wedge$) of elementary problems of the form:

1. $u =_T v$, called an equation,
2. $u \neq_T v$, called a disequation,
3. $u \ll_T v$, called a match-equation,

where u and v are terms.

The set of T-solutions of the above elementary problems are defined as follows: a substitution σ is solution of

1. $u =_T v$, iff $\sigma(u) =_T \sigma(v)$,

2. $u \neq_T v$, iff $\sigma(u) \neq_T \sigma(v)$,

3. $u \ll_T v$, iff $\sigma(u) =_T v$,

If $\mathcal{E}_1$ and $\mathcal{E}_2$ are equational problems then $\mathcal{E}_1 \vee \mathcal{E}_2$ and $\mathcal{E}_1 \wedge \mathcal{E}_2$ are also equational problems. A substitution σ is solution of an equational problem $\mathcal{E}_1 \vee \mathcal{E}_2$ iff σ is solution of $\mathcal{E}_1$ or $\mathcal{E}_2$. σ is solution of an equational problem $\mathcal{E}_1 \wedge \mathcal{E}_2$ iff σ is solution of $\mathcal{E}_1$ and $\mathcal{E}_2$. The set of solutions of an equational problem $\mathcal{E}$ is denoted by $Sol(\mathcal{E})$. Substitutions in $Sol(\mathcal{E})$ are assumed to be idempotent.

The **trivial equational problem** for which any substitution is T'-solution is denoted by $\top$. Any equational problem $\mathcal{E}$ such that $Sol(\mathcal{E}) = \emptyset$ is equivalent to the **unsatisfiable equational problem** denoted by $\bot$.

For example, if T' is the associative-commutative theory ($T' = AC$),

$$f(x,a) =_C f(y,g(c)) \vee (g(u,v) =_{AC} g(x,a) \wedge x =_\emptyset a + b)$$

is an equational problem.

Whenever T' is the empty theory, we allow omitting the subscript $\emptyset$ in an equational problem.

Two important concerns on equational problems are first their satisfiability with respect to the considered class of models, second their solving, that is the computation of (minimal) complete sets of solutions or of solved forms of these problems [5,6], when the set of solutions is not stable by instantiation. For equational problems composed only of equations and disequations in the empty theory ($T = \emptyset$), these questions are fully solved by [5,6], in the general case of equational problems including parameters (i.e. universally quantified variables). For equational problems containing only equations and disequations without parameters in AC theories, these questions are also solved in [5]. However it is a challenging problem to get satisfiability and solving decision procedures for general equational problems in other equational theories.

Definition 1 *A* **canonical solved system** *is a conjunction of equations and match-equations of the form* $(x_i == t_i)$ *and* $(x_j \ll t_j)$ *such that* $(\bigcup_i \{x_i\}) \cap (\bigcup_i V(t_i)) = \emptyset$, *where* x_i *are variables and* $V(t_i)$ *denotes the set of variables occurring in* t_i.

An equational problem $\mathcal{E}$ *is said in* **canonical solved form** *iff* $\mathcal{E}$ *is* $\top$, $\bot$ *or a disjunction of canonical solved systems.*

$V(\mathcal{E})$ *denotes the set of variables* $(\bigcup_i \{x_i\})$.

Note that this definition forbids systems containing for instance $(x == f(y))$ and $(y == g(z))$, that are often said in (quasi) solved form. The canonical solved form would be $(x == f(g(z))) \wedge (y == g(z))$.

We assume given in the following a process that computes, for any equational problem $\mathcal{E}$, its canonical solved form, denoted $\mathcal{E}\downarrow$. Such a process does not exists in general, since for example equational unification is undecidable, but it exists for several theories of interest like associativity-commutativity.

3.2 Schematization

The schematization by an equational problem of a set of substitutions allows delaying as much as possible the computation of the substitutions themselves, so that only the equational information about them is known and used.

Let U be a multiset of terms. U is said **constrained by the equational problem** $\mathcal{E}$ iff the substitutions allowed for the variables of U are only the solutions of $\mathcal{E}$. The pair $(U, \mathcal{E})$ is called a **constrained multiset.**

The multiset of terms schematized by a constrained multiset is thus by definition:

$$\mathcal{S}(U,\mathcal{E}) = \{\{\sigma(t) |\ t \in U\} \mid \forall \sigma \in Sol(\mathcal{E})\}.$$

In practice it is often sufficient to consider only a complete (if possible minimal) set of solutions for the constraints, denoted $CSS(\mathcal{E})$:

$$\mathcal{S}c(U,\mathcal{E}) = \{\{\sigma(t) |\ t \in U\} \mid \forall \sigma \in CSS(\mathcal{E})\}.$$

The definition of constrained multiset of terms captures the notions of:

1. constrained term, when U is reduced to a single element;

2. constrained rewrite rule, when U consists of two ordered terms,

3. constrained critical pair, when U consists in two elements,

4. constrained equational proof with n steps of constrained equational replacement, when U consists of n elements.

3.3 Constrained rewriting

Let us see what kind of constraints are introduced by different processes such as reduction, superposition and completion and how to handle them.

A term t is reducible modulo T by a rewrite rule $(g \to d)$ at occurrence m iff there exists a match σ modulo T from g to $t_{|m}$, that is there exists a solution to the equational problem $(g \ll_T t_{|m})$. So match-equations are introduced by applying a rewrite rule.

Example 1 *For the rule* $x*1 \to x$, *the term* $(a*(1*a))$ *rewrites modulo AC into the constrained term*

$$(a*x, x*1 \ll_{AC} 1*a).$$

More generally, the notion of constrained rewrite rule is needed.

Definition 2 *A* **constrained rewrite rule** *is given by two ordered terms $l \to r$ and an equational problem $\mathcal{E}$. A constrained rewrite rule $(l \to r, \mathcal{E})$ schematizes the following set of rewrite rules:*

$$\mathcal{R}(l \to r, \mathcal{E}) = \{\sigma(l) \to \sigma(r) | \sigma \in Sol(\mathcal{E})\}.$$

A constrained term $(t_1, \mathcal{E}_1)$ is reducible modulo T by a constrained rule $(l \to r, \mathcal{E})$ if there exists a solution μ of $\mathcal{E} \wedge \mathcal{E}_1$ such that $\mu(l) =_T t_{1|m}$. So μ is actually a solution of $\mathcal{E} \wedge \mathcal{E}_1 \wedge (l \ll_T t_{1|m})$ that must be satisfiable. This is the set of constraints associated to the new term t_2 resulting from the constrained rewriting process.

Definition 3 *Let $(l \to r, \mathcal{E})$ be a constrained rewrite rule. A* **constrained** *term $(t_1, \mathcal{E}_1)$* **rewrites** *modulo the theory T, at occurrence m into $(t_2, \mathcal{E}_2)$, and we denote $(t_1, \mathcal{E}_1) \rightarrowtail (t_2, \mathcal{E}_2)$, iff $t_2 = t_1[m \leftarrow r]$ and $\mathcal{E}_2 = (\mathcal{E} \wedge \mathcal{E}_1 \wedge (l \ll_T t_{1|m}))$ is satisfiable.*

Note that constrained rewriting needs to check first that the set of constraints $\mathcal{E}_2$ is satisfiable. Otherwise obviously termination problems arise. To check satisfiability of an equational problem is in general much simpler than the problem of finding a complete set of solutions or a solved form, especially in equational theories. For instance, a criteria to check the satisfiability of a system of equations modulo associativity-commutativity is given in [34]. Finding complete sets of AC-solutions of equation systems is much more difficult.

The notion of constrained rewriting is powerful enough to strictly include the notion of conditional rewriting. In order to show this, let us consider the theory T' generated by a set of conditional rewrite rules $(c \Rightarrow l \to r)$ where c is a conjonction of equalities $\wedge_{i=1,\ldots,n} u_i = v_i$. In order to apply such a rule on the term t at occurence m, one has to check that there exists a redex, i.e. that the match-equation $l \ll t_{|m}$ is satisfiable, and that there exists a solution of the previous match-equation that satisfies (in the whole theory T') the conditions in c. Thus one looks for solutions of the system

$$\begin{cases} l & \ll_\emptyset & t|m \\ u_1 & =_{T'} & v_1 \\ \vdots & & \\ u_n & =_{T'} & v_n \end{cases}$$

This is obviously an equational problem and thus the notion of constrained rewriting captures the notion of conditional rewriting: the constrained rewrite rule $(l \to r, \wedge_{i=1,\ldots,n} u_i =_{T'} v_i)$ acts as the conditional rule $(\wedge_{i=1,\ldots,n} u_i = v_i \Rightarrow l \to r)$. Constrained rewriting is strictly more general, since it allows using arbitrary equational problems and not only those consisting in equations in the empty theory or the whole theory T' but also in any subtheory of T'. Note by the way that the notion of constraint introduced in this paper can easily be enlarged e.g. by using also type constraints.

After a discussion with M. Okada, we have introduced in this paper a more general notion of equational problem than in [22], in order to be able to cover also the notion of conditional rewriting. But from now on, we shall consider the case where T' is a set of unconditional identities with a subset of (usually) unorientable identities E, and we shall deal only with constraints consisting of equational problems built on equations in the theory $T = E$. This specific case rules out the case of conditional rewriting.

The following results give the semantics of constrained rewriting. Given a set R of constrained rewrite rules, let $\mathcal{R} = \{\mathcal{R}(l \to r, \mathcal{E}) | (l \to r, \mathcal{E}) \in R\}$ be the set of schematized rules. The associated rewriting relation modulo T is denoted by $\to_{\mathcal{R},T}$.

Proposition 1 *If $(t_1, \mathcal{E}_1) \rightarrowtail (t_2, \mathcal{E}_2)$ with the constrained rewrite rule $((l \to r), \mathcal{E})$, then for any solution σ of $\mathcal{E}_2$, $\sigma(t_1) \to_{\mathcal{R},T} \sigma(t_2)$ using $(\sigma(l) \to \sigma(r))$.*

Proof: Since σ is solution of $\mathcal{E}_2$, $\sigma(l) =_T t_{1|m}$ and $\sigma(t_{1|m}) =_T \sigma(\sigma(l))$. So $\sigma(t_1) \to_{\mathcal{R},T} \sigma(t_1)[m \leftarrow \sigma(\sigma(r))] = \sigma(t_1)[m \leftarrow \sigma(r)] = \sigma(t_2)$, because σ is idempotent. □

Note that, according to the definition of $\mathcal{E}_2$, $\sigma(t_1) \in \mathcal{S}(t_1, \mathcal{E}_1)$, $(\sigma(l) \to \sigma(r)) \in \mathcal{S}((l \to r), \mathcal{E})$ and $\sigma(t_2) \in \mathcal{S}(t_2, \mathcal{E}_2)$.

Considering complete sets of solutions, instead of sets of solutions, leads to refine this result. We write $\sigma =_T \sigma'[X]$ (resp. $\sigma \leq_T \sigma'[X]$) to say that for any variable x in the set of variables X, $\sigma(x) =_T \sigma'(x)$ (resp. $\beta(\sigma(x)) =_T \sigma'(x)$ for some substitution β).

Proposition 2 *If $(t_1, \mathcal{E}_1) \rightarrowtail (t_2, \mathcal{E}_2)$ with the constrained rewrite rule $((l \to r), \mathcal{E})$, then for any solution $\sigma \in CSS(\mathcal{E}_2)$, there exists $\alpha \in CSS(\mathcal{E})$ such that $\sigma(t_1) \to_{\mathcal{R},T} \sigma'(t_2) =_T \sigma(t_2)$ using $(\alpha(l) \to \alpha(r))$, with $\sigma' =_T \sigma[V(l) \cup V(r)]$.*

Proof: Since the set of variables of t_1 and l can always be chosen disjoint, any solution σ of $\mathcal{E}_2$ can be decomposed into $\beta \circ \gamma$ with γ being the restriction of σ to the variables of l and r, denoted by $\gamma = \sigma[V(l) \cup V(r)]$, and $\beta = \sigma[V(t_1)]$. Since $\gamma \in Sol(\mathcal{E})$, there exists $\alpha \in CSS(\mathcal{E})$ such that $\gamma =_T \mu \circ \alpha[V(l) \cup V(r)]$.

Since σ is solution of $\mathcal{E}_2$, $\sigma(l) =_T \mu(\alpha(l)) =_T t_{1|m}$ and $\beta(\mu(\alpha(l))) =_T \beta(t_{1|m})$. So $\beta(t_1) \to_{\mathcal{R},T} \beta(t_1)[m \leftarrow \beta(\mu(\alpha(r)))] = \beta(t_1[m \leftarrow \mu(\alpha(r))]) = \beta \circ \mu \circ \alpha(t_2) =_T \sigma(t_2)$. Indeed $\beta \circ \mu \circ \alpha =_T \sigma[V(l) \cup V(r)]$. □

Example 2 *Consider the constrained rewrite rule*

$$(x * x * x * x \to x, \{x \neq a\}).$$

The constrained term $(b*b*b*b*y*y*y*y, \top)$ *with the trivial constraint* $\top$ *rewrites to another constrained term*

$$(b*b*b*b*x, \{x*x*x*x \ll y*y*y*y, x \neq a\})$$

then to $(x'*x, \mathcal{E})$ *with* $\mathcal{E}$ *being*

$$\begin{gathered} x*x*x*x \ll y*y*y*y \\ x \neq a \\ x'*x'*x'*x' \ll b*b*b*b \\ x' \neq a. \end{gathered}$$

Note that the substitution $(x \mapsto y)(x' \mapsto b)$ *is solution of the last constraint. By using this instantiation , we get the usual rewriting derivation:*

$$b*b*b*b*y*y*y*y \to b*b*b*b*y \to b*y.$$

A constrained equational replacement step is the symmetric closure of $\twoheadrightarrow$. Its reflexive symmetric transitive closure is called constrained equational replacement.

Definition 4 *A* **constrained equational proof** *using a constrained set of rewrite rules* R *modulo a theory* T *is a sequence of constrained terms* $\{(t_0, \mathcal{E}_0), ..., (t_n, \mathcal{E}_n)\}$ *such that for* $0 < i \leq n$, *either* $(t_{i-1}, \mathcal{E}_{i-1}) \longleftrightarrow^T (t_i, \mathcal{E}_i)$, *or* $(t_{i-1}, \mathcal{E}_{i-1}) \twoheadrightarrow (t_i, \mathcal{E}_i)$, *or* $(t_{i-1}, \mathcal{E}_{i-1}) \twoheadleftarrow (t_i, \mathcal{E}_i)$.

A **constrained rewrite proof** *using a constrained set of rewrite rules* R *modulo a theory* T *is a constrained equational proof*

$$(t_0, \mathcal{E}_0) \twoheadrightarrow ... \twoheadrightarrow (t_i, \mathcal{E}_i) \overset{*}{\longleftrightarrow}^T (t_j, \mathcal{E}_j) \twoheadleftarrow ... \twoheadleftarrow (t_n, \mathcal{E}_n).$$

When it is more convenient, we also denote a constrained equational proof by $(\{t_0, ..., t_n\}, \mathcal{E}_0 \wedge ... \wedge \mathcal{E}_n)$.

An equational proof can be considered as a constrained one in which all the constraints are the trivial equational problem. A constrained equational proof schematizes a family of equational proofs.

Proposition 3 *Let* $\{(t_0, \mathcal{E}_0), ..., (t_n, \mathcal{E}_n)\}$ *be a constrained equational proof. Then for any substitution* $\sigma \in Sol(\mathcal{E}_0 \wedge ... \wedge \mathcal{E}_n)$, $\{\sigma(t_0), ..., \sigma(t_n)\}$ *is an equational proof with standard rules.*

Proof: By induction on the number of elementary steps in the constrained equational proof and applying Proposition 1. □

A constrained rewrite proof schematizes a family of rewrite proofs with standard equational rewriting.

Remind that, given a class rewriting system $(\mathcal{R}, T)$, a proof $\{t_0 = t, ..., t_n = t'\}$ of $(t = t')$ is a **rewrite proof** for $\to^{\mathcal{R},T}$ iff there exist s, s' such that

$$t \overset{*}{\longrightarrow}^{\mathcal{R},T} s \overset{*}{\longleftrightarrow}^T s' \overset{*}{\longleftarrow}^{\mathcal{R},T} t'.$$

Proposition 4 *Let* $\{(t_0, \mathcal{E}_0), ..., (t_n, \mathcal{E}_n)\}$ *be a constrained rewrite proof. Then for any substitution* $\sigma \in Sol(\mathcal{E}_0 \wedge ... \wedge \mathcal{E}_n)$, $\{\sigma(t_0), ..., \sigma(t_n)\}$ *is a rewrite proof with standard rules.*

Proof: Again by induction on the number of elementary steps in the constrained rewrite proof and applying Proposition 1. □

3.4 Constrained superposition

The computation of critical pairs introduces equations or unification problems.

Two given constrained rewrite rules $(l \to r, \mathcal{E})$ and $(g \to d, \mathcal{E}')$ superpose at some occurrence m in g if there exists a solution μ of $\mathcal{E} \wedge \mathcal{E}'$ such that $\mu(g_{|m}) =_T \mu(l)$. So there exists at least a solution for the new set of constraints $(\mathcal{E} \wedge \mathcal{E}' \wedge (g_{|m} =_T l))$, associated to the pair $[g[m \leftarrow r], d]$.

Definition 5 *The constrained rewrite rule* $(l \to r, \mathcal{E})$ *overlaps* $(g \to d, \mathcal{E}')$ *at occurrence* m *in* g *iff the equational problem* $(\mathcal{E} \wedge \mathcal{E}' \wedge (g_{|m} =_T l))$ *is satisfiable. The* **constrained critical pair** *obtained by superposition of* $(l \to r, \mathcal{E})$ *into* $(g \to d, \mathcal{E}')$ *at occurrence* m *is by definition:* $([g[m \leftarrow r], d], \mathcal{E} \wedge \mathcal{E}' \wedge (g_{|m} =_T l))$.

A constrained critical pair of the form

$$([g[m \leftarrow r], d], (\mathcal{E} \wedge \mathcal{E}' \wedge (g_{|m} =_T l)))$$

schematizes the following set of pairs:

$$\mathcal{CP} = \{[\sigma(g[m \leftarrow r]), \sigma(d)] | \sigma \in Sol(\mathcal{E} \wedge \mathcal{E}' \wedge (g_{|m} =_T l))\}.$$

The semantics of constrained superposition is given by the following results:

Proposition 5 *Let* $([g[m \leftarrow r], d], \mathcal{E} \wedge \mathcal{E}' \wedge (g_{|m} =_T l))$ *be the constrained critical pair obtained by superposition of* $(l \to r, \mathcal{E})$ *into* $(g \to d, \mathcal{E}')$ *at occurrence* m. *Then for any solution* σ *of* $(\mathcal{E} \wedge \mathcal{E}' \wedge (g_{|m} =_T l))$, $\sigma(g[m \leftarrow r]) \leftarrow^{\mathcal{R},T} \sigma(g) \to^{\mathcal{R}} \sigma(d)$ *is a critical peak of* $(\sigma(l) \to \sigma(r))$ *into* $(\sigma(g) \to \sigma(d))$ *at* m.

Proof: Since σ is solution of $(\mathcal{E} \wedge \mathcal{E}' \wedge (g_{|m} =_T l))$, $\sigma(g_{|m}) =_T \sigma(l)$. So there is a superposition of $(\sigma(l) \to \sigma(r))$ into $(\sigma(g) \to \sigma(d))$ at occurrence m. The corresponding critical peak is $\sigma(g[m \leftarrow r]) \leftarrow^{\mathcal{R},T} \sigma(g) \to^{\mathcal{R}} \sigma(d)$. □

Note that $(\sigma(l) \to \sigma(r)) \in \mathcal{S}(l \to r, \mathcal{E})$ and $(\sigma(g) \to \sigma(d)) \in \mathcal{S}(g \to d, \mathcal{E}')$.

Considering now complete sets of solutions, we get that for any critical peak, obtained as in Proposition 5, there exists a critical pair between two schematized rules, in the set of pairs schematized by the constrained critical pair.

Proposition 6 *Let* $([g[m \leftarrow r], d], \mathcal{E} \wedge \mathcal{E}' \wedge (g_{|m} =_T l))$ *be the constrained critical pair obtained by superposition of* $(l \rightarrow r, \mathcal{E})$ *into* $(g \rightarrow d, \mathcal{E}')$ *at occurrence* m. *Then for any solution* $\sigma \in CSS(\mathcal{E} \wedge \mathcal{E}' \wedge (g_{|m} =_T l))$, *there exist* $\alpha \in CSS(\mathcal{E})$, $\beta \in CSS(\mathcal{E}')$ *and* $\sigma' \leq_T \sigma[V(l) \cup V(r) \cup V(g) \cup V(d)]$ *such that* $[\sigma'(g[m \leftarrow r]), \sigma'(d)]$ *is a critical pair of* $(\alpha(l) \rightarrow \alpha(r))$ *into* $(\beta(g) \rightarrow \beta(d))$ *at occurrence* m.

Proof: Assuming that the variables of the two constrained rules are disjoint, and using the facts that σ is solution of $\mathcal{E}$ and $\mathcal{E}'$, we get $\alpha, \mu_1, \beta, \mu_2$ such that $\sigma =_T \mu_1 \circ \alpha[V(l) \cup V(r)]$ and $\sigma =_T \mu_2 \circ \beta[V(g) \cup V(d)]$. Since $\sigma(g_{|m}) =_T \sigma(l)$, $\mu_1 \circ \mu_2$ T-unifies $\alpha(l)$ and $\beta(g)_{|m}$. So there exist μ is in a complete set of T-unifiers of these two terms, and some substitution γ such that $\sigma =_T \gamma \circ \mu \circ \alpha \circ \beta$. Note that $\mu \circ \alpha \circ \beta$ is a solution of $(\mathcal{E} \wedge \mathcal{E}' \wedge (g_{|m} =_T l))$. So there is a critical pair $[\mu(\beta(g)[m \leftarrow \alpha(r)]), \mu(\beta(d))]$ obtained by superposition of $(\alpha(l) \rightarrow \alpha(r))$ into $(\beta(g) \rightarrow \beta(d))$ at occurrence m. This critical pair can be written as $[\sigma'(g[m \leftarrow r]), \sigma'(d)]$, with $\sigma' = \mu \circ \alpha \circ \beta$, and $\sigma' \leq_T \sigma[V(l) \cup V(r) \cup V(g) \cup V(d)]$. □

Example 3 *Consider again the two rewrite rules*

$$x * x * x * x \rightarrow x$$
$$u * v * w * a \rightarrow a.$$

By constrained superposition at top occurrence, the constrained critical pair $([x, a], \{x * x * x * x =_{AC} u * v * w * a\})$ *is computed. It schematizes for instance the trivial critical pair* $[a, a]$ *by using the solution of the equational problem* $(x \mapsto a)(u \mapsto a)(v \mapsto a)(w \mapsto a)$; *it also schematizes the critical pair* $[a * x', a]$, *by considering the solution* $(x \mapsto a * x')(u \mapsto a * x' * x')(v \mapsto a * x')(w \mapsto a * x')$.

Since we would like to deal with completion modulo a theory T, the superposition of a constrained rule into an axiom of T has to be considered too.

Definition 6 *The constrained rewrite rule* $(l \rightarrow r, \mathcal{E})$ *overlaps the axiom* $(g = d) \in T$ *at occurrence* m *in* g *iff the equational problem* $(\mathcal{E} \wedge (g_{|m} =_T l))$ *is satisfiable. The* **constrained extended rule** *obtained by superposition of* $(l \rightarrow r, \mathcal{E})$ *into* $(g = d)$ *at occurrence* m *is by definition:* $(g[m \leftarrow l] \rightarrow g[m \leftarrow r], \mathcal{E} \wedge (g_{|m} =_T l))$.

The semantics of constrained extended rules is given in a similar way:

Proposition 7 *Let* $(g[m \leftarrow l], g[m \leftarrow r], \mathcal{E}'')$ *be the constrained extended rule obtained by superposition of* $(l \rightarrow r, \mathcal{E})$ *into* $(g = d)$ *at occurrence* m. *Then for any solution* σ *of* $\mathcal{E}'' = (\mathcal{E} \wedge (g_{|m} =_T l))$, *the critical cliff* $\sigma(g[m \leftarrow r]) \leftarrow^{\mathcal{R},T} \sigma(g) \longleftrightarrow^{T} \sigma(d)$ *satisfies* $\sigma(d) \rightarrow^{\mathcal{R},T} \sigma(g[m \leftarrow r])$ *using an instance* $\sigma(g[m \leftarrow l]) \rightarrow \sigma(g[m \leftarrow r])$ *of the constrained extended rule.*

Proof: Since σ is solution of $(\mathcal{E} \wedge (g_{|m} =_T l))$, $\sigma(g_{|m}) =_T \sigma(l)$. So there is a cliff $\sigma(g[m \leftarrow r]) \leftarrow^{\mathcal{R},T} \sigma(g) \longleftrightarrow^{T} \sigma(d)$. Since $\sigma(d) =_T \sigma(g) =_T \sigma(g[m \leftarrow l])$, the rule $\sigma(g[m \leftarrow l]) \rightarrow \sigma(g[m \leftarrow r])$ reduces $\sigma(d)$ on top into $\sigma(g[m \leftarrow r])$. □

Considering now complete sets of solutions, we get that for any critical cliff obtained as in Proposition 7, there exists an extended rule of a schematized rule, in the set of pairs schematized by the constrained extended rule.

Proposition 8 *Let* $(g[m \leftarrow l], g[m \leftarrow r], \mathcal{E}'')$ *be the constrained extended rule obtained by superposition of* $(l \rightarrow r, \mathcal{E})$ *into* $(g = d)$ *at occurrence* m. *Then for any solution* $\sigma \in CSS(\mathcal{E} \wedge (g_{|m} =_T l))$, *there exist* $\alpha \in CSS(\mathcal{E})$, $\alpha \leq_T \sigma[V(l) \cup V(r)]$ *such that* $(\alpha(g[m \leftarrow l]) \rightarrow \alpha(g[m \leftarrow r]))$ *is an extended rule for* $(\alpha(l) \rightarrow \alpha(r))$.

Proof: Assuming that the variables of the contrained rule and the axiom are disjoint, and using the facts that σ is solution of $\mathcal{E}$, we get α, μ, β such that $\sigma =_T \mu \circ \alpha[V(l) \cup V(r)]$ and $\sigma = \beta[V(g) \cup V(d)]$. Since $\sigma(g_{|m}) =_T \sigma(l)$, $\beta \circ \mu$ T-unifies $\alpha(l)$ and $g_{|m}$. So there exist μ' in a complete set of T-unifiers of these two terms, and some substitution γ such that $\sigma =_T \gamma \circ \mu' \circ \alpha$. Note that $\mu' \circ \alpha$ is a solution of $(\mathcal{E} \wedge (g_{|m} =_T l))$. So there is an extended rule for $(\alpha(l) \rightarrow \alpha(r))$, namely $(g[m \leftarrow \alpha(l)] \rightarrow g[m \leftarrow \alpha(r)])$. It can also be written as $(\alpha(g[m \leftarrow l]) \rightarrow \alpha(g[m \leftarrow r]))$. □

Example 4 *In the considered AC-theory, the constrained rule* $(x \rightarrow a, \{x*x*x*x =_{AC} u*v*w*a\})$ *needs an extension* $(x*z \rightarrow a*z, \{x*x*x*x =_{AC} u*v*w*a\})$.

However all critical pairs computed by a (non-constrained) completion procedure modulo T are not yet taken into account: other superpositions need to be considered, namely superpositions of a constrained rewrite rule $(l \rightarrow r, \mathcal{E})$ into the canonical solved form of the constraints $\mathcal{E}'\downarrow$ associated to the constrained rule $(g \rightarrow d, \mathcal{E}')$.

Example 5 *Consider again the two (constrained) rules:*

$$x \rightarrow a, \{x * x * x * x =_{AC} u * v * w * a,\ x \neq a\}$$
$$x' * x' * x' * x' \rightarrow x'.$$

A possible instantion of the first rule is given by the substitution α *defined as* $(x \mapsto a * b * b * b * z)$ *and the subterm* $b * b * b * z$ *is unifiable with* $x' * x' * x' * x'$ *using* $(x' \mapsto b)(z \mapsto b)$ *This superposition yields a critical pair* $[\sigma(x), a]$ *where* $\sigma(x) = a * b$.

Definition 7 *The constrained rewrite rule* $(l \to r, \mathcal{E})$ *overlaps in constraints* $(g \to d, \mathcal{E}'\downarrow)$ *at occurrence* m *in* $(x = t) \in \mathcal{E}'\downarrow$ *iff the equational problem*

$$(\mathcal{E} \wedge \mathcal{E}'\downarrow \wedge (t_{|m} =_T l))$$

is satisfiable. The **critical pair in constraints** *obtained by superposition of* $(l \to r, \mathcal{E})$ *into* $(g \to d, \mathcal{E}'\downarrow)$ *at occurrence* m *in* $(x = t) \in \mathcal{E}'\downarrow$ *is by definition:*

$$([g, d], \mathcal{E} \wedge (\mathcal{E}'\downarrow - (x = t)) \wedge (x = t[m \leftarrow r]) \wedge (t_{|m} =_T l)).$$

The semantics of superposition in constraints is given by the following result:

Proposition 9 *Let* $([g, d], \mathcal{E}'')$ *be the critical pair in constraints obtained by superposition of* $(l \to r, \mathcal{E})$ *into* $(g \to d, \mathcal{E}'\downarrow)$ *at occurrence* m *in* $(x = t) \in \mathcal{E}'\downarrow$. *Let* n *be an occurrence of* x *in* g. *Then for any solution* σ *of* $\mathcal{E}'' = \mathcal{E} \wedge (\mathcal{E}'\downarrow - (x = t)) \wedge (x = t[m \leftarrow r]) \wedge (t_{|m} =_T l)$, *there exist a substitution* α *solution of* $(\mathcal{E} \wedge \mathcal{E}'\downarrow \wedge (t_{|m} =_T l))$ *and a critical peak* $\alpha(g)[n.m \to \alpha(r)] \leftarrow^{\mathcal{R},T} \alpha(g) \to^{\mathcal{R}} \alpha(d)$ *such that*

$$\alpha(g)[n.m \to \alpha(r)]) \xrightarrow{*}^{\mathcal{R},T} \sigma(g) \text{ and } \alpha(d) \xrightarrow{*}^{\mathcal{R},T} \sigma(d).$$

Proof: From σ, let deduce α such that $\alpha(y) = \sigma(y)$ for $y \neq x$ and $\alpha(x) = t$. Since α is solution of $(\mathcal{E} \wedge \mathcal{E}'\downarrow \wedge (t_{|m} =_T l))$, $\alpha(t_{|m}) =_T \alpha(l)$. Let n be one of the (possibly many) occurrences of the variable x in g. Then $\alpha(g)_{|n.m} =_T \alpha(l)$. Thus $\alpha(g) \to \alpha(d)$ and $\alpha(l) \to \alpha(r)$ superpose at occurrence $n.m$ in $\alpha(g)$ with the identity substitution. This yields the critical peak $\alpha(g)[n.m \to \alpha(r)] \leftarrow^{\mathcal{R},T} \alpha(g) \to^{\mathcal{R}} \alpha(d)$. Because of the possible non-linearity of g, it possibly remains some terms $\alpha(x)$ which are yet $\mathcal{R},T$-reducible into $\sigma(x)$. So $\alpha(g)[n.m \leftarrow \alpha(r)] \xrightarrow{*}^{\mathcal{R},T} \sigma(g)$ and $\alpha(d) \xrightarrow{*}^{\mathcal{R},T} \sigma(d)$. □

Considering now complete sets of solutions, we get that for any critical peak, obtained as in Proposition 9, there exists a critical pair between two schematized rules.

Proposition 10 *Let* $([g, d], \mathcal{E} \wedge (\mathcal{E}'\downarrow - (x = t)) \wedge (x = t[m \leftarrow r]) \wedge (t_{|m} =_T l))$ *be the critical pair in constraints obtained by superposition of* $(l \to r, \mathcal{E})$ *into* $(g \to d, \mathcal{E}'\downarrow)$ *at occurrence* m *in* $(x = t) \in \mathcal{E}'\downarrow$. *Let* n *be an occurrence of* x *in* g. *Then for any solution* $\sigma \in CSS(\mathcal{E} \wedge \mathcal{E}'\downarrow \wedge (t_{|m} =_T l))$, *there exist* $\alpha' \in CSS(\mathcal{E})$, $\beta' \in CSS(\mathcal{E}'\downarrow)$ *and* $\sigma' \leq_T \sigma[V(l) \cup V(r) \cup V(g) \cup V(d)]$ *such that* $[\sigma'(g)[n.m \leftarrow \sigma'(r)], \sigma'(d)]$ *is a critical pair of* $(\alpha'(l) \to \alpha'(r))$ *into* $(\beta'(g) \to \beta'(d))$ *at occurrence* $n.m$.

Proof: As in the proof of Proposition 9, from σ, let deduce α such that $\alpha(y) = \sigma(y)$ for $y \neq x$ and $\alpha(x) = t$. Since α is solution of $(\mathcal{E} \wedge \mathcal{E}' \wedge (t_{|m} =_T l))$, $\alpha(t_{|m}) =_T \alpha(l)$. Let n be one of the (possibly many) occurrences of the variable x in g. Then $\alpha(g)_{|n.m} =_T \alpha(l)$. Assuming that the variables of the two constrained rules are disjoint, and using the facts that α is solution of $(\mathcal{E} \wedge \mathcal{E}'\downarrow \wedge (t_{|m} = =_T l))$, we get $\alpha', \mu_1, \beta', \mu_2$ such that $\alpha =_T \mu_1 \circ \alpha'[V(l) \cup V(r)]$ and $\alpha =_T \mu_2 \circ \beta'[V(g) \cup V(d)]$. Since $\alpha(g)_{|n.m} =_T \alpha(l)$, $\mu_1 \circ \mu_2$ T-unifies $\alpha'(l)$ and $\beta'(g)_{|n.m}$. So there exist μ in a complete set of T-unifiers of these two terms, and some substitution γ such that $\alpha =_T \gamma \circ \mu \circ \alpha' \circ \beta'$. Note that $\mu \circ \alpha' \circ \beta'$ is a solution of $(\mathcal{E} \wedge \mathcal{E}'\downarrow \wedge (t_{|m} =_T l))$. So there is a critical pair $[\mu(\beta'(g)[n.m \leftarrow \alpha'(r)]), \mu(\beta'(d))]$ obtained by superposition of $(\alpha'(l) \to \alpha'(r))$ into $(\beta'(g) \to \beta'(d))$ at occurrence $n.m$. This critical pair can be written as $[\sigma'(g)[n.m \leftarrow \sigma'(r)]), \sigma'(d)]$, with $\sigma' = \mu \circ \alpha' \circ \beta'$, and $\sigma' \leq_T \alpha[V(l) \cup V(r) \cup V(g) \cup V(d)]$. □

3.5 Constrained critical pair lemma

Computation of constrained critical pairs and of critical pairs in constraints allows turning constrained peaks and cliffs into rewrite proofs. Note that we are interested only in constrained peaks and cliffs that schematize peaks and cliffs relevant for completion modulo T (cf. for instance [1]). This justifies the following definition.

Definition 8 *The constrained proof*

$$(t'', \mathcal{E} \wedge \mathcal{E}_1 \wedge (l \ll_T t_{|m})) \twoheadleftarrow (t, \mathcal{E}) \twoheadrightarrow (t', \mathcal{E} \wedge \mathcal{E}_0 \wedge (g \ll t_{|n}))$$

is a **constrained peak** *of the constrained rewrite rules* $(l \to r, \mathcal{E}_1)$ *and* $(g \to d, \mathcal{E}_0)$, *iff*

$$\forall \sigma \in Sol(\mathcal{E} \wedge \mathcal{E}_0 \wedge (g \ll t_{|n}) \wedge \mathcal{E}_1 \wedge (l \ll_T t_{|m})),$$
$$\sigma(t'') \longleftarrow^{\sigma(l) \to \sigma(r), T} \sigma(t) \to^{\sigma(g) \to \sigma(d)} \sigma(t').$$

where the $\longleftarrow$*-rewrite step modulo* T *occurs below the* $\to$*-rewrite step.*

The constrained proof

$$(t'', \mathcal{E} \wedge \mathcal{E}_1 \wedge (l \ll_T t_{|m})) \twoheadleftarrow (t, \mathcal{E}) \longleftrightarrow^{T} (t', \mathcal{E} \wedge (g \ll t_{|n}))$$

is a **constrained cliff** *of the constrained rewrite rule* $(l \to r, \mathcal{E}_1)$ *and the axiom* $(g = d)$, *iff*

$$\forall \sigma \in Sol(\mathcal{E} \wedge (g \ll t_{|n}) \wedge \mathcal{E}_1 \wedge (l \ll_T t_{|m})),$$
$$\sigma(t'') \longleftarrow^{\sigma(l) \to \sigma(r), T} \sigma(t) \longleftrightarrow^{\sigma(g) = \sigma(d)} \sigma(t').$$

where the $\longleftarrow$*-rewrite step modulo* T *occurs below the* $\longleftrightarrow$*-equality step.*

Lemma 1 *Consider any constrained peak*

$$(t'', \mathcal{E} \wedge \mathcal{E}_1 \wedge (l \ll_T t_{|m})) \twoheadleftarrow (t, \mathcal{E}) \twoheadrightarrow (t', \mathcal{E} \wedge \mathcal{E}_0 \wedge (g \ll t_{|n}))$$

of the constrained rewrite rules $(l \to r, \mathcal{E}_1)$ *and* $(g \to d, \mathcal{E}_0)$. *Then* $\forall \sigma \in Sol(\mathcal{E} \wedge \mathcal{E}_0 \wedge (g \ll t_{|n}) \wedge \mathcal{E}_1 \wedge (l \ll_T t_{|m}))$,

- *either* $\sigma(t'') \xrightarrow{*}^{\mathcal{R},T} \xleftrightarrow{*}^{T} \xleftarrow{*}^{\mathcal{R},T} \sigma(t')$,
- *or there exists a constrained critical pair or a critical pair in constraints of the constrained rewrite rules* $(l \to r, \mathcal{E}_1)$ *into* $(g \to d, \mathcal{E}_0)$, *say* $([p,q], \mathcal{E}')$, *such that there are substitutions* $\alpha \in CSS(\mathcal{E}')$ *and* β *satisfying* $\sigma(t'') \xrightarrow{*}^{\mathcal{R},T} \xleftrightarrow{*}^{T} \beta(\alpha(p))$ *and* $\sigma(t') \xrightarrow{*}^{\mathcal{R},T} \xleftrightarrow{*}^{T} \beta(\alpha(q))$.

Consider any constrained cliff

$$(t'', \mathcal{E} \wedge \mathcal{E}_1 \wedge (l \ll_T t_{|m})) \leftarrow\!- (t, \mathcal{E}) \longleftrightarrow^{T} (t', \mathcal{E} \wedge (g \ll t_{|n}))$$

of the constrained rewrite rule $(l \to r, \mathcal{E}_1)$ *and the axiom* $(g = d)$. *Then* $\forall \sigma \in Sol(\mathcal{E} \wedge (g \ll t_{|n}) \wedge \mathcal{E}_1 \wedge (l \ll_T t_{|m}))$,

- *either* $\sigma(t'') \xrightarrow{*}^{\mathcal{R},T} \xleftrightarrow{*}^{T} \xleftarrow{*}^{\mathcal{R},T} \sigma(t')$,
- *or there exists a constrained extended rule* $(g[m \to l] \to g[m \to r], \mathcal{E}'')$ *of the constrained rewrite rule* $(l \to r, \mathcal{E}_1)$ *into* $(g = d)$, *such that there are substitutions* $\alpha \in CSS(\mathcal{E}'')$ *and* β *satisfying* $\sigma(t') \xleftrightarrow{*}^{T} \beta(\alpha(g[m \to l]))$ *and* $\sigma(t'') \xleftrightarrow{*}^{T} \beta(\alpha(g[m \to r]))$.

Proof: Consider first the case of a constrained peak. If the constrained rewritings apply on disjoint subterms of t, then they simply commute. The other case to be considered is when one constrained rewriting applies above the other. Without lost of generality, we can assume that $(g \to d, \mathcal{E}_0)$ applies at occurrence ϵ in t, thus $t' = d$, and $(l \to r, \mathcal{E}_1)$ applies at some occurrence m below. The proof is by case on the position of m.

1. m is a non-variable occurrence in g:
   Since σ is solution of $\mathcal{E} \wedge \mathcal{E}_0 \wedge \mathcal{E}_1 \wedge (g \ll t) \wedge (l <<_T t_{|m})$, $\sigma(g) = t$, $\sigma(g)_{|m} = t_{|m} =_T \sigma(l)$. So the equational problem $\mathcal{E}_0 \wedge \mathcal{E}_1 \wedge (g_{|m} =_T l)$ is satisfiable and there exists a constrained critical pair between the two rules, namely $([g[m \leftarrow r], d], \mathcal{E}_0 \wedge \mathcal{E}_1 \wedge (g_{|m} =_T l))$. Now there exists $\alpha \in CSS(\mathcal{E}_0 \wedge \mathcal{E}_1 \wedge (g_{|m} =_T l)$ such that $\alpha \leq_T \sigma[V(g) \cup V(d) \cup V(l) \cup V(r)]$. This yields: $\sigma(t'') \xleftrightarrow{*}^{T} \beta(\alpha(p))$ and $\sigma(t') \xleftrightarrow{*}^{T} \beta(\alpha(q))$.
2. $m = n.p$ with n being an occurrence of a variable x in g, $(x = t_0) \in \mathcal{E}_0\!\downarrow$ and p being an occurrence of t_0: since σ is solution of $\mathcal{E} \wedge \mathcal{E}_0\!\downarrow \wedge \mathcal{E}_1 \wedge (g \ll t) \wedge (l \ll_T t_{|m})$, $\sigma(g) = t$, $\sigma(g)_{|m} = t_{|m} = t_{0|p} =_T \sigma(l)$. So the equational problem $\mathcal{E}_0\!\downarrow \wedge \mathcal{E}_1 \wedge (t_{0|p} =_T l)$ is satisfiable and there exists a critical pair in constraints between the two rules, namely $([g, d], \mathcal{E}')$ with $\mathcal{E}' = \mathcal{E}_1 \wedge (\mathcal{E}_0\!\downarrow - (x = t_0)) \wedge (x = t_0[p \leftarrow r]) \wedge (t_{0|p} =_T l))$. Now $\sigma(t') \xrightarrow{*}^{\mathcal{R},T} \sigma'(t')$ and $\sigma(t'') \xrightarrow{*}^{\mathcal{R},T} \sigma'(t'')$ where σ' is a solution of $\mathcal{E}'$. Then there exists $\alpha \in CSS(\mathcal{E}')$ such that $\alpha \leq_T \sigma'[V(\mathcal{E}')]$. Thus we get: $\sigma(t'') \xrightarrow{*}^{\mathcal{R},T} \xleftrightarrow{*}^{T} \beta(\alpha(p))$ and $\sigma(t') \xrightarrow{*}^{\mathcal{R},T} \xleftrightarrow{*}^{T} \beta(\alpha(q))$.
3. m does not belong to $\sigma(g)$:
   then there exists a variable y at some occurrence n of $\sigma(g)$ and a constraint $(y = u_0) \in \mathcal{E}'\!\downarrow$ where $\mathcal{E}' = \mathcal{E} \wedge \mathcal{E}_0 \wedge (g \ll t) \wedge \mathcal{E}_1 \wedge (l \ll_T t_{|m})$, such that $u_0 \to^{\mathcal{R},T} u_0'$.
   Let σ' be the substitution defined by $\sigma'(z) = \sigma(z)$ if $z \neq y$ and $\sigma'(y) = u_0'$. Then $\sigma(t'') \xrightarrow{*}^{\mathcal{R},T} \sigma'(t'') = \sigma'(t)$ and $\sigma(t') \xrightarrow{*}^{\mathcal{R},T} \sigma'(t') = \sigma'(d)$.
   Let also define $\mathcal{E}'' = \mathcal{E}'\!\downarrow - (y = u_0) \wedge (y = u_0')$. Then σ' is a solution of $\mathcal{E}''$ and $(t, \mathcal{E}'') \rightarrowtail (d, \mathcal{E}'')$ using $(g \to d, \mathcal{E}_0)$. This implies that $\sigma'(t'') = \sigma'(t) \xrightarrow{*}^{\mathcal{R},T} \sigma'(d) = \sigma'(t')$.

Let consider now the case of a constrained cliff. If the constrained rewriting and replacement apply on disjoint subterms of t, then they simply commute. The other cases, assuming that $(g \to d, \mathcal{E}_0)$ applies at occurrence ϵ in t, depend on the occurrence m of the rewriting.

1. m is a non-variable occurrence in g:
   Since σ is solution of $\mathcal{E} \wedge (g \ll t) \wedge \mathcal{E}_1 \wedge (l << _T t_{|m})$, $\sigma(g) = t$, $\sigma(g)_{|m} = t_{|m} =_T \sigma(l)$. So there exists a constrained extended rule $g[m \to l] \to g[m \to r], \mathcal{E}_1 \wedge (l \ll_T g_{|m})$. Since σ is solution of $\mathcal{E}_1 \wedge (l \ll_T g_{|m})$, there exists $\alpha \in CSS(\mathcal{E}_1 \wedge (g_{|m} =_T l)$ such that $\alpha \leq_T \sigma[V(g) \cup V(d) \cup V(l) \cup V(r)]$. This yields: $\sigma(t'') \xleftrightarrow{*}^{T} \beta(\alpha(g[m \to r]))$ and $\sigma(t') \xleftrightarrow{*}^{T} \beta(\alpha(g[m \to l]))$.
2. m does not belong to $\sigma(g)$:
   then there exists a variable y at some occurrence n of $\sigma(g)$ and a constraint $(y = u_0) \in \mathcal{E}'\!\downarrow$ where $\mathcal{E}' = \mathcal{E} \wedge (g \ll t) \wedge \mathcal{E}_1 \wedge (l \ll_T t_{|m})$, such that $u_0 \to^{\mathcal{R},T} u_0'$.
   Let σ' be the substitution defined by $\sigma'(z) = \sigma(z)$ if $z \neq y$ and $\sigma'(y) = u_0'$. Then $\sigma(t'') \xrightarrow{*}^{\mathcal{R},T} \sigma'(t'') = \sigma'(t)$ and $\sigma(t') \xrightarrow{*}^{\mathcal{R},T} \sigma'(t') = \sigma'(d)$.
   Let also define $\mathcal{E}'' = \mathcal{E}'\!\downarrow - (y = u_0) \wedge (y = u_0')$. Then σ' is a solution of $\mathcal{E}''$ and $(t, \mathcal{E}'') \longleftrightarrow^{T} (d, \mathcal{E}'')$ using $(g = d)$. This implies that $\sigma'(t'') = \sigma'(t) \longleftrightarrow^{T} \sigma'(d) = \sigma'(t')$.

□

Corollary 1 *If all constrained critical pairs and all critical pairs in constraints have constrained rewrite proofs, then any critical peak schematize rewrite proofs.*

If all constrained extended rules are added, then any critical peak schematize rewrite proofs.

Proof: This is easily deduced from Lemma 1 just by noticing that constrained rewritings will occur in p or q and thus, after instantiation, the T-equality steps around are emboddied into the corresponding rewriting steps modulo T.

A similar reasoning holds for constrained cliffs and constrained extended rules. □

3.6 Constrained completion

Constrained completion is quite similar to standard completion, except that instead of terms, equalities and rules, it manipulates constrained terms, constrained equalities and constrained rewrite rules. Informally the activities of completion are turning constrained equalities into constrained rewrite rules, simplifying constrained terms and computing constrained critical pairs. The differences detailed below are due to the fact that the substitutions involved in the constraints must also be taken care of.

1. A first difference is that a new deletion rule is added. A constrained equality $(p = q, \mathcal{E})$ can be deleted if for any solution σ of $\mathcal{E}$, $\sigma(p) =_T \sigma(q)$. Which is negated by: there exists σ solution of $\mathcal{E}$ such that $\sigma(p) \neq_T \sigma(q)$. So the constrained equality $(p = q, \mathcal{E})$ is deleted whenever the equational problem $\mathcal{E} \wedge (p \neq_T q)$ has no solution (as we already pointed out, this problem is decidable for the unparametrized AC problems for instance).

2. A second difference is introduced in the orientation rule: turning a constrained equality $(p = q, \mathcal{E})$ into a constrained rule must be precised. Assume given a reduction ordering $\succ$ on terms. Since it is closed by substitutions, if $p \succ q$ then $\forall \sigma \in Sol(\mathcal{E}), \sigma(p) \succ \sigma(q)$. The constrained rule $(p \to q, \mathcal{E})$ can be added. A similar case holds if $q \succ p$. However it may happen that some instances of $p = q$ satisfy $\sigma(p) \succ \sigma(q)$, while others satisfy $\sigma(q) \succ \sigma(p)$ or $\sigma(p) =_T \sigma(q)$. The last case can be ruled out by considering the new set of constraints $\mathcal{E}' = \mathcal{E} \wedge (p \neq_T q)$, restricting thus to non-trivial instances. If it remains conflicts, then constraints have to be lazily solved and a family of rewrite rules may be introduced in that step of constrained completion.

3. The third difference concerns the completeness of critical pair computation. Superpositions in the constrained part of rules must not be forgotten. This is why a new deduction inference rule that adds critical pairs in constraints is introduced. Because this inference rule asks solving constraints, it should be applied lazily, when all the constrained critical pairs have been computed.

4. This leads to the last difference that concerns the completion strategy. Since constraints are solved lazily, this induces a superposition strategy that is mainly top-down.

Let P be a set of constrained equalities denoted $(p = q, \mathcal{E})$, C the current set of constrained rules, E the current set of constrained extended rules, and $\succ$ a T-commuting reduction ordering [18]. R will denote $C \cup E$. Assume that P contains $(q = p, \mathcal{E})$ whenever it contains $(p = q, \mathcal{E})$. Since we are considering equational completion through the constraint formalism, we must take care of the application of the collapse inference rule [1] and we need a well-founded ordering on constrained terms denoted hereafter by $\rhd$. It can be deduced from any well-founded ordering on terms $\rhd_0$ (for instance the specialization ordering used in [1]) by defining: $(t, \mathcal{E}) \rhd (t', \mathcal{E}')$ iff $\forall \sigma \in Sol(\mathcal{E}), \exists \sigma' \in Sol(\mathcal{E}')$ such that $\sigma(t) \rhd_0 \sigma'(t')$.

A *constrained completion procedure* is expressed by the following inference rules:

1. Deduce
$$\frac{P, C, E}{P \cup \{(p = q, \mathcal{E})\}, C, E}$$
if $([p, q], \mathcal{E})$ *is a constrained critical pair of* R

2. Deduce in constraints
$$\frac{P, C, E}{P \cup \{(p = q, \mathcal{E})\}, C, E}$$
if $([p, q], \mathcal{E})$ *is a critical pair in constraints of* R

3. Delete
$$\frac{P \cup \{(p = q, \mathcal{E})\}, C, E}{P, C, E}$$
if $p =_T q$ *or* $\mathcal{E} \wedge (p \neq_T q)$ *unsatisfiable*

4. Simplify
$$\frac{P \cup \{(p = q, \mathcal{E})\}, C, E}{P \cup \{(p' = q, \mathcal{E}')\}, C, E}$$
if $(p, \mathcal{E}) \twoheadrightarrow (p', \mathcal{E}')$

5. Orienting a pair
$$\frac{P \cup \{(p = q, \mathcal{E})\}, C, E}{P, C \cup \{(p \to q), \mathcal{E}\}, E}$$
if $p \succ q$

6. Add a constrained extended rule
$$\frac{P, C, E}{P, C, E \cup \{(l \to r, \mathcal{E})\}}$$
if $(l \to r, \mathcal{E})$ *is a constrained extended rule of* R

7. Compose rule
$$\frac{P, C \cup \{(l \to r, \mathcal{E})\}, E}{P, C \cup \{(l \to r', \mathcal{E}')\}, E}$$
if $(r, \mathcal{E}) \twoheadrightarrow (r', \mathcal{E}')$

8. Compose extension
$$\frac{P, C, E \cup \{(l \to r, \mathcal{E})\}}{P, C, E \cup \{(l \to r', \mathcal{E}')\}}$$
if $(r, \mathcal{E}) \twoheadrightarrow (r', \mathcal{E}')$

9. Collapse
$$\frac{P, C \cup \{(l \to r, \mathcal{E})\}, E}{P \cup \{(l' = r, \mathcal{E}')\}, C, E}$$
if $(l, \mathcal{E}) \twoheadrightarrow (l', \mathcal{E}')$ *with* $(g \to d, \mathcal{E}'')$ *s.t.* $(l, \mathcal{E}) \rhd (g, \mathcal{E}'')$

10. Develop an unorientable pair

$$\frac{P \cup \{(p = q, \mathcal{E})\}, C, E}{P \cup \{(\sigma(p) = \sigma(q), \top) | \sigma \in CSS(\mathcal{E})\}, C, E}$$

if $\mathcal{E} \neq \top, p \not\succ q$ and $q \not\succ p$

The correctness of these inference rules comes from the fact that each of them (except the last one) represents a family of inference rules of the completion modulo T. This is a consequence of previous propositions 2, 6, 8, 10. The correctness of the last inference rule is obvious.

Each inference rule allows computing $(P_{i+1}, C_{i+1}, E_{i+1})$ from (P_i, C_i, E_i), which is denoted by

$$(P_i, C_i, E_i) \vdash (P_{i+1}, C_{i+1}, E_{i+1}).$$

Let C^* be the set of all generated constrained rules, E^* the set of all generated constrained extended rules, and P^* the set of all generated constrained pairs.

Let C^∞, E^∞ and P^∞ be respectively the set of persisting constrained rules, extended rules and pairs, i.e. the sets effectively generated by completion starting from (P_0, C_0, E_0). Formally

$$P^\infty = \bigcup_i \bigcap_{j>i} P_j, \quad C^\infty = \bigcup_i \bigcap_{j>i} C_j$$
$$E^\infty = \bigcup_i \bigcap_{j>i} E_j, \quad R^\infty = C^\infty \cup E^\infty.$$

The set of rules schematized by R^∞ is denoted by $\mathcal{R}^\infty$.

Definition 9 *A derivation*

$$(P_0, C_0, E_0) \vdash (P_1, C_1, E_1) \vdash \ldots$$

is **fair** *if*

- *the set of all constrained critical pairs and critical pairs in constraints of R^∞, denoted $CCP(R^\infty)$, is a subset of P^*,*
- *and the set of all constrained extended rules of R^∞, denoted $EXT(R^\infty)$, is a subset of E^*.*

A derivation $(P_0, R_0) \vdash (P_1, R_1) \vdash$ .... does not fail if P^∞ is empty.

The following result states the completeness of constrained completion for equational logic.

Theorem 1 *Let $\succ$ be an T-commuting reduction ordering and P_0 be a set of equalities with the trivial equational problem $\top$ as constraints. If the derivation $(P_0, C_0, E_0) \vdash (P_1, C_1, E_1) \vdash$ .... is fair and does not fail for inputs $(P_0, \emptyset, \emptyset)$ and $\succ$, then any equational theorem $(t = t')$ using P_0 has a rewrite proof using $\mathcal{R}^\infty$ modulo the theory T.*

Proof: (Sketch) Let us consider any provable equalities $(t = t')$ with the initial unconstrained set of equalities $P_0 \cup T$. Such a proof can be considered as a constrained proof using the same equalities with the trivial equational problem $\top$ as constraints. To each (P_i, C_i, E_i) is associated a proof of $(t = t')$ using constrained rules in $R_i = C_i \cup E_i$, constrained equalities in P_i and axioms in T. Following [15,1], to each inference rule is associated a transformation rule on constrained proofs using (P^*, C^*, E^*, T). Proving that the relation $\Longrightarrow$ induced by these transformation rules is terminating is achieved by finding a complexity measure $c((\mathcal{P}, \mathcal{E}))$ for each constrained proof $(\mathcal{P}, \mathcal{E})$ and a well-founded ordering $>$ on these measures, that satisfy the following property: $(\mathcal{P}, \mathcal{E}) \Longrightarrow (\mathcal{P}', \mathcal{E}')$ implies $c((\mathcal{P}, \mathcal{E})) > c((\mathcal{P}', \mathcal{E}'))$, for each proof transformation rule. Let c_0 be a complexity measure that allows proving the termination of proof ordering for completion modulo T. Then the complexity of a constrained proof is defined as the multiset of the complexities of its intances:

$$c((\mathcal{P}, \mathcal{E})) = \{c_0(\sigma(\mathcal{P})) | \forall \sigma \in CSS(\mathcal{E})\}.$$

The next step is to show that each constrained proof has a constrained rewrite proof for (R^∞, T), by noetherian induction on $\Longrightarrow$. Assume that the constrained proof is not a constrained rewrite proof:

- If a non-persisting equality or a non-persisting rule is used, by definition, there exists a step i such that it belongs to $P_i \cup R_i$ and not to $P_{i+1} \cup R_{i+1}$. Since $\Longrightarrow$ reflects $\vdash$, the constrained proof is reducible by $\Longrightarrow$ to a constrained proof that does not use it any more.
- If the proof contains a constrained peak $(t'', \mathcal{E}'') \leftarrowtail (t, \mathcal{E}) \rightarrowtail (t', \mathcal{E}')$ or a constrained cliff $(t'', \mathcal{E}) \longleftrightarrow^T (t, \mathcal{E}) \rightarrowtail (t', \mathcal{E}')$, it is also reducible, due to Lemma 1, the fairness hypothesis, and the induction hypothesis.
- it contains no constrained equality step using P^∞, since the corresponding derivation does not fail.

□

Thus, from the result of a fair derivation, it is possible to deduce a set of rewrite rules $\mathcal{R}^\infty$ with the Church-Rosser property.

4 Conclusion

The theory of constrained equational reasoning is presented in this paper. Of course many questions and prolongations of this work arise. Let us mention:

1. The general notion of equational problem has to be more deeply investigated, in particular in order to get satisfiability and solving decision procedures for a larger class of theories with respect to the class of models considered. The initial model will be in particular useful for inductive completion [24,17,1].

2. Constrained equational reasoning allows dealing with finite representations of infinite complete sets of unifiers or matchers and it will be interesting to consider the specific problems of equational completion modulo a theory that has infinite sets of unifiers, like associativity [28]. In this respect, the notion of rewriting with meta-rules [25,26] is strongly connected to constrained rewriting.

3. We have seen that constrained equational reasoning covers the notion of conditional and contextual rewriting [10,7,19]. The relation between conditional completion and constrained completion has to be invastigated now.

4. The notion of constrained equational reasoning uses the notion of symbolic constraint in contrast with constrained logic programming [14] where numerical constraints are mainly used. We are now investigating the combination of the two concepts as a component of a language of constraints [31] and its use for the design and the implementation of functional, logic constrained and safe programming language environments.

5. To check the real power and efficiency of this process, an implementation of constrained completion is needed. More generaly, the application of the concept of constrained reasoning has to be investigated in the context of theorem provers.

Finally it must be emphasized that the concepts developed here are general enough to go beyond equational logic and can be adapted to typed Horn clauses logic with equality.

Acknowledgements: This work has been partly supported by the GRECO de programmation du CNRS. We thank H.-J. Bürckert, H. Comon, E. Domenjoud, J.-P. Jouannaud, J.-L. Lassez, P. Lescanne, M. Okada and M. Stickel for fruitful discussions at early stage of this work.

References

[1] L. Bachmair. *Proof methods for equational theories.* PhD thesis, University of Illinois, Urbana-Champaign, 1987. Revised version, August 1988.

[2] L. Bachmair and N. Dershowitz. Completion for rewriting modulo a congruence. In *Proceedings Second Conference on Rewriting Techniques and Applications*, Springer Verlag, Bordeaux (France), May 1987.

[3] H-J. Bürckert. *Solving Disequations in Equational Theories.* Technical Report SEKI-Report SR-87-15, Kaiserslautern, 1987.

[4] H-J. Bürckert, A. Herold, D. Kapur, J. Siekmann, M. Stickel, M. Tepp, and H. Zhang. Opening the AC-unification race. *Journal of Automated Reasoning*, 1988.

[5] H. Comon. Unification et disunification. Théories et applications. Thèse de l'Institut Polytechnique de Grenoble, 1988.

[6] H. Comon and P. Lescanne. Equational problems and disunification. *Journal of Symbolic Computation*, 3-4(7):371–426, 1989. Special issue on unification. Part one.

[7] N. Dershowitz and D.A. Plaisted. Equational programming. In J. E. Hayes, D. Michie, and J. Richards, editors, *Machine Intelligence 11: The logic and acquisition of knowledge*, chapter 2, pages 21–56, Oxford Press, Oxford, 1988.

[8] M. Dincbas, H. Simonis, and P. Van Hentenryck. Extending equation solving and constraint handling in logic programming. In *Proceedings of the Colloquium on Resolution of Equations in Algebraic Structures*, Austin (Texas), May 1987.

[9] E. Domenjoud. Ac unification through order-sorted acl unification. In H-J. Bürckert and W. Nutt, editors, *Proceedings of UNIF'89, third international workshop on unification*, Lambrecht (FR Germany), June 1989. Also in CRIN Reseach Report 89-R-67.

[10] H. Ganzinger. A completion procedure for conditional equations. In *Proceedings 1st International Workshop on Conditional Term Rewriting Systems*, pages 62–83, Springer-Verlag, 1987.

[11] J. Goguen, C. Kirchner, H. Kirchner, A. Megrelis, J. Meseguer, and T. Winkler. An introduction to OBJ-3. In J-P. Jouannaud and S. Kaplan, editors, *Proceedings 1st International Workshop on Conditional Term Rewriting Systems*, Springer-Verlag, June 1988. Also as internal report CRIN: 88-R-001.

[12] G. Grätzer. *Universal Algebra.* Springer-Verlag, 1979. Second Edition.

[13] G. Huet and D. Oppen. Equations and rewrite rules: a survey. In R. Book, editor, *Formal Language Theory: Perspectives and Open Problems*, pages 349–405, Academic Press, New-York, 1980.

[14] J. Jaffar and J-L. Lassez. *Constraint Logic Programming.* Technical Report, IBM TJ Watson Research Center, June 1986.

[15] J-P. Jouannaud. Proof algebras. *Invited conference to the second Rewriting Techniques and Applications conference, Bordeaux (France)*, 1987.

[16] J.P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal of Computing*, 15(4):1155–1194, 1986. Preliminary version in Proceedings 11th ACM Symposium on Principles of Programming Languages, Salt Lake City, 1984.

[17] J.P. Jouannaud and E. Kounalis. Proof by induction in equational theories without constructors. In *Proceedings 1st Symp. on Logic In Computer Science*, pages 358–366, Boston (USA), 1986.

[18] J.P. Jouannaud and M. Muñoz. Termination of a set of rules modulo a set of equations. In *Proceedings of the 7th Conference on Automated Deduction*, Springer-Verlag, 1984.

[19] S. Kaplan and J-L. Rémy. Completion algorithms for conditional rewriting systems. In H. Ait-Kaci and M. Nivat, editors, *Colloquium on the Resolution of Equations in Algebraic Structures*, MCC and INRIA, Austin, (USA), 1987.

[20] D. Kapur and P. Narendran. NP-completeness of the set unification and matching problems. In J. Siekmann, editor, *Proceedings 8th Conference on Automated Deduction*, Springer-Verlag, 1986.

[21] C. Kirchner. Methods and tools for equational unification. In *Proceedings of the Colloquium on Resolution of Equations in Algebraic Structures*, Austin (Texas), May 1987.

[22] C. Kirchner and H. Kirchner. Constrained equational reasoning. In *Proceeding of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation*, pages 382–389, ACM Press, July 1989.

[23] C. Kirchner and P. Lescanne. Solving disequations. In D. Gries, editor, *Proceeding of the Second Symposium on Logic In Computer Science*, pages 347–352, IEEE, 1987.

[24] H. Kirchner. Preuves par complétion dans les variétés d'algèbres. Thèse d'état de l'Université de Nancy I, 1985.

[25] H. Kirchner. Schematization of infinite sets of rewrite rules. Application to the divergence of completion processes. In P. Lescanne, editor, *Proceedings Second Conference on Rewriting Techniques and Applications*, pages 180–191, Springer-Verlag, Bordeaux (France), May 1987.

[26] H. Kirchner. Schematization of infinite sets of rewrite rules generated by divergent completion processes. 1988. to appear in Theoretical Computer Science.

[27] D. Knuth and P. Bendix. *Simple Word Problems in Universal Algebra*, pages 263–297. Pergamon Press, 1970.

[28] G.S. Makanin. The problem of solvability of equations in a free semigroup. *Akad. Nauk. SSSR*, 233(2), 1977.

[29] G. Peterson and M. Stickel. Complete sets of reductions for some equational theories. *Journal of the Association for Computing Machinery*, 28:233–264, 1981.

[30] M. Rusinowitch. Démonstration automatique par des techniques de réécriture. Thèse d'Etat de l'Université de Nancy I, 1987.

[31] G. Smolka. *Logic Programming with Polymorphically Order-Sorted Types.* Technical Report, IBM Deutschland, 1988. To appear in the Proceedings of the First International Workshop on Algebric and Logic Programming.

[32] W. Taylor. Equational logic. *Houston Journal of Mathematics*, 1979. Appears also in [12], Appendix 4.

[33] Ralph W. Wilkerson Timothy B. Baird, Gerald E. Peterson. Complete sets of reductions modulo associativity, commutativity and identity. In N. Dershowitz, editor, *Proceedings of the third RTA conference*, Springer-Verlag, Chapell Hill (North Carolina), April 1989.

[34] J. Von Zur Gathen and M. Sieveking. A bound on solutions of linear integer equalities and inequalities. *Proceedings of the American Mathematical Society*, 72(1), 1978.

Solving Equational Formulas in an Order-Sorted Algebra (Preliminary Notes)

Hubert COMON*

1 Introduction

Equational formulas are first order formulas which only involve one predicate symbol: the equality. Solving equational formulas in the algebra of terms leads to many applications (see [CL89,Com88,Mah88]). This has been studied in [CL89,Mah88] where it is shown (in particular) that it is possible to turn any equational formula into a *solved form* which either is $\bot$ or possesses at least one solution in $T(F)$, the Herbrand Universe, leading to a complete axiomatization of finite trees under a finite alphabet.

In this paper, we investigate transformation rules for equational formulas in *order sorted algebras* as they are defined in [KKM88,SNMG87]. Actually, we show how to extend the results of [CL89,Mah88] to the order-sorted case. As we will see, an order-sorted signature is just a finite tree automaton. Therefore, solving equational formulas in order-sorted algebras provides a complete axiomatization of any recognizable subset of $T(F)$.

Such an extension allows to extend the applications of equational formulas simplification to the order-sorted case. For example, specification transformations as in [Com89a] can be applied to order-sorted specifications.

What is the problem ?

We know how to solve equational formulas in the many-sorted case. The first idea for solving equational formulas in the order-sorted case is therefore to reduce it to the many-sorted one. This is, roughly, the method used in Schmidt-Schauss unification algorithm [Sch86] which proceeds in the following way: in order to unify s and t, first unify the many-sorted terms, then postprocess the unifiers, only keeping the well-sorted solutions. However, such a method does not work for equational formulas since forgetting the sorts enlarge the set of solutions of an equation but restricts the set of solutions of a disequation $s \neq t$. After forgetting some solutions of $s \neq t$, it is no longer possible to retrieve them.

That is the reason why we have to take into account the sort structure during the transformation. Therefore, we merely use the rules given in [Kir88] for unification. Then, some rules for solving equations have to introduce "new" variables which are implicitly existentially quantified. Thus, turning them by negation into rules for solving disequations, leads to the introduction of universally quantified variables. On the other hand, the control in [CL89] mainly

*Laboratoire d'Informatique fondamentale et d'Intelligence Artificielle, Institut IMAG, 46 Ave. Félix Viallet, 38031 Grenoble cedex, France. E-mail : comon@lifia.imag.fr

states that we have first to eliminate the universally quantified variables (called *parameters*). We now come to the first difficulty: there are termination problems with the cycle elimination-introduction of parameters.

The second difficulty comes from problems such as: $\forall y : \underline{s} : x : \underline{s}' \neq y$ whose solutions in $T(F)$ are all the ground terms which have sort $\underline{s}'$ and not $\underline{s}$. Such a problem cannot occur in the many sorted case neither in unification problems. We have to propose a rule for solving such "sort complement" problems.

2 Syntax

2.1 Equational Formulas

An *equation* is an unordered pair of terms denoted $s = t$. An *equational formula* (as defined in [Com89b]) is an arbitrary first order formula whose atoms are equations. An *elementary formula* is an equational formula that may be written

$$\bigvee_{j \in J} \exists \vec{w}, \forall \vec{y} : \ P$$

where P is a quantifier free equational formula. As shown in e.g. [Com89b], we have only to consider such formulas and it is possible to assume P in conjunctive normal form.

In this paper, we add one more syntactic construction: each variable occurring in an elementary formula is followed by an expression "$\in \underline{s}$" where $\underline{s}$ is a symbol from a given set of *sort symbols* S. Moreover, we assume here that two occurrences of x in a same formula : $x \in \underline{s}$ and $x \in \underline{s}'$ satisfy the condition $\underline{s} = \underline{s}'$.

2.2 Order-Sorted Signatures and Tree Automata

(Finite) Order-Sorted signatures are defined in e.g. [SNMG87]. Such signatures are actually tree automata and we will adopt both point of view for some reasons that will appear later.

A *signature* is a triple (S, D_s, D_f) where S is a finite set of *sort symbols* (or *states*), D_s is a finite set of *sort declarations* $\underline{s} \geq \underline{s}'$ (or ϵ-transitions $\underline{s}'(t) \rightarrow \underline{s}(t)$), $D_{\cdot\ \cdot}$ is a graduate alphabet F together with *function declarations* $f : \underline{s}_1 \times \ldots \times \underline{s}_n \rightarrow \underline{s}$ (or transitions $f(\underline{s}_1(t_1), \ldots, \underline{s}_n(t_n)) \rightarrow \underline{s}(f(t_1, \ldots, t_n))$).

For every sort $\underline{s}$, $T(F)_{\underline{s}}$ (also denoted $T_{\underline{s}}$) is the set of ground terms of sort $\underline{s}$ (equivalently, this is the subset of $T(F)$ recognized by the finite ascending tree automaton, with final state $\underline{s}$). T is the union of all $T_{\underline{s}}$ for $\underline{s} \in S$ (equivalently, this is the subset of $T(F)$ of all terms that are recognized by the automaton in which all states are final states).

Example 1 Let us illustrate both point of view on a simple example. This is a specification

of integers:

$$S = \{\underline{pos}, \underline{neg}, \underline{zero}, \underline{int}\} \qquad S = \{q_p, q_n, q_z, q_r\}$$

$$D_s = \begin{cases} \underline{pos} \geq \underline{zero} \\ \underline{neg} \geq \underline{zero} \\ \underline{int} \geq \underline{pos} \\ \underline{int} \geq \underline{neg} \end{cases} \qquad P_0 = \begin{cases} q_z(t) \to q_p(t) \\ q_z(t) \to q_n(t) \\ q_p(t) \to q_r(t) \\ q_n(t) \to q_r(t) \end{cases}$$

$$F = \{0, s, p\} \qquad F = \{0, s, p\}$$

$$D_f = \begin{cases} 0: & \to & \underline{zero} \\ s: & \underline{zero} & \to & \underline{pos} \\ s: & \underline{pos} & \to & \underline{pos} \\ p: & \underline{zero} & \to & \underline{neg} \\ p: & \underline{neg} & \to & \underline{neg} \end{cases} \qquad P_1 = \begin{cases} 0 & \to & q_z(0) \\ s(q_z(t)) & \to & q_p(s(t)) \\ s(q_p(t)) & \to & q_p(s(t)) \\ p(q_z(t)) & \to & q_n(p(t)) \\ p(q_n(t)) & \to & q_n(p(t)) \end{cases}$$

The automaton point of view is useful for some problems, for example:

- For comparing the expressiveness of specifications. For example, the sets $T(F)/=_{\mathcal{R}}$ that can be defined in the many sorted case by a canonical left-linear term rewriting system $\mathcal{R}$ are a strict subclass of the sets T that can be defined by an order-sorted signature (without equations). [1] In the same way, every term algebra $T/=_{\mathcal{R}}$ where $\mathcal{R}$ is a left linear convergent term rewriting system, is an algebra T' for some (other) order-sorted signature...(see [Com89a] for more information on specifications transformations)
- For performing some operations on sets of ground terms that are well-known in the automata field. For example, it is possible to compute the set of all ground terms that have the sort $\underline{s}$ and not the sort $\underline{s'}$ (this is just the computation of the complement of a regular tree language into another regular tree language).

3 Semantics

Given an order-sorted signature, we define $[\![\cdot]\!]$ by $[\![x \in \underline{s}]\!] = T_{\underline{s}}$ and $[\![f(t_1, \ldots, t_n)]\!] = f([\![t_1]\!], \ldots, [\![t_n]\!])$. A term t is *well-sorted* if $[\![t]\!] \subseteq T$.

> In this paper, we will assume that all terms occurring in an equational formula are well-sorted

The semantics of an equational formula is the set of its solutions $\mathcal{S}(\phi, \mathcal{U})$ defined in the following way:

$$\begin{aligned} \mathcal{S}(t = u, \mathcal{U}) &= \{\sigma \mid \sigma \text{is an assignment from } \mathcal{U} \text{ to } T(F) \text{ s.t. } \sigma(x \in \underline{s}) \in T_{\underline{s}} \text{ and } \sigma(t) \equiv \sigma(u)\} \\ \mathcal{S}(\top, \mathcal{U}) &= \{\sigma \mid \sigma \text{is an assignment from } \mathcal{U} \text{ to } T(F) \text{ s.t. } \sigma(x \in \underline{s}) \in T_{\underline{s}}\} \\ \mathcal{S}(d_1 \wedge d_2, \mathcal{U}) &= \mathcal{S}(d_1, \mathcal{U}) \cap \mathcal{S}(d_2, \mathcal{U}) \\ \mathcal{S}(\neg d, \mathcal{U}) &= \mathcal{S}(\top, \mathcal{U}) - \mathcal{S}(d, \mathcal{U}) \\ \mathcal{S}(\exists x, \phi, \mathcal{U}) &= \{\sigma_{|\mathcal{U}} \mid \sigma \in \mathcal{S}(\phi, \mathcal{U} \cup \{x\})\} \end{aligned}$$

[1] Because the class of languages that are the set of irreducible ground trees for some left linear term rewriting system is a strict subclass of recognizable tree languages (see [GB85,FV88]).

The definition for other logical connectives can be obtained from the above ones in a standard way.

4 Solved Forms

As in the many-sorted case described in [CL89,Com89b,Mah88], we are interested in transforming any elementary formula into a simpler one called *solved form.* Such solved forms are, in the case we consider here, either $\top$, $\bot$ or formulas

$$\bigvee_{j \in J} \exists \vec{w_j} : x_1 = t_1 \wedge \ldots \wedge x_n = t_n \wedge z_1 \neq u_1 \wedge \ldots \wedge z_m \neq u_m$$

(For convenience, we omit sometimes the expressions "$\in \underline{s}$") with the following properties:

- J is finite
- $\forall i, [\![t_i]\!] \subseteq [\![x_i]\!]$
- $x_1, \ldots, x_m$ are variables and occur only once
- $\forall i, z_i \notin u_i$
- $\forall z \in Var(z_1, u_1, \ldots, z_m, u_m)$, $[\![z]\!]$ is infinite
- $Var(z_1, u_1, \ldots, z_m, u_m) \subseteq Var(t_1, \ldots, t_m)$

The main property of solved form is their solvability:

Proposition 1 *Any solved form distinct from $\bot$ has at least one solution.*

5 A set of rules

We now design a set of transformation rules for elementary formulas that are correct (i.e. they transform a formula ϕ into a formula ψ that has the same set of solutions), terminating and complete (i.e. any formula is eventually transformed into a solved form). We use the same formalism than in [CL89,Com89b]. (We do not recall all definitions in this abstract).

We mainly use the same rules than in the many-sorted case (see [CL89]) except that replacements are correct (in our semantics) only if x is replaced with a term t such that $[\![t]\!] \subseteq [\![x]\!]$. Adding this restriction, all rules given in the many-sorted framework hold in the order-sorted case. However, because of this restriction, the set of rules is no longer complete. We have to add some new rules for solving e.g. problems $\forall y \in \underline{s} : x \in \underline{s'} \neq y$ when $\underline{s} \neq \underline{s'}$.

The rules we add are given in figures 1 and 2. In figure 1 we recall the main rules for unification (similar to those given in [Kir88]).

In figure 2 we give other rules (for quantifiers elimination) mainly obtained by negating the rules in figure 1 or by some operation on automata. Such operations on automata may introduce new sorts and new function declarations (and thus change the signature) but do not change the languages $T_{\underline{s}}$ and therefore do not alter the set of solutions. Also note, that each sort (introduced during the transformation or not) may be assumed to have a non-empty carrier (i.e. $T_{\underline{s}} \neq \emptyset$) and even to have an infinite carrier, since finiteness of the generated language

Solving equations $x = t$ (SE)

$$(SE_1) \quad \mathcal{P}[z_1 \in \underline{s}_1 = z_2 \in \underline{s}_2] \quad \mapsto \quad \exists w_3 \in \underline{s}_3, \mathcal{P}[z_1 = w_3 \wedge z_2 = w_3]$$

If z_1 et z_2 are unknowns, $w_3 \notin Var(\mathcal{P})$, $T_{\underline{s}_1} \not\subseteq T_{\underline{s}_2}$, $T_{\underline{s}_2} \not\subseteq T_{\underline{s}_1}$ and $T_{\underline{s}_3} = T_{\underline{s}_1} \cap T_{\underline{s}_2}$.

$$(SE_2) \quad \mathcal{P}[z = f(t_1, \ldots, t_n)] \quad \mapsto \quad \bigvee_{\underline{s}_1 \ldots \underline{s}_n \in E(z)} \exists w_1 \in \underline{s}_1, \ldots, w_n \in \underline{s}_n, \mathcal{P}[z = f(w_1, \ldots, w_n) \wedge w_1 = t_1 \wedge \ldots \wedge w_n = t_n]$$

If

1. z is an unknown
2. $[\![f(t_1, \ldots, t_n)]\!] \not\subseteq [\![z]\!]$
3. $$E(z) = \max\{\underline{s}'_1 \ldots \underline{s}'_n \mid f : \underline{s}'_1 \times \ldots \times \underline{s}'_n \to \underline{s} \text{ and } T_{\underline{s}} \subseteq [\![z]\!]\}$$
4. $t_1, \ldots, t_n$ do not contain any parameter's occurrence

$$(SE_3) \quad z_1 : \underline{s}_1 = z_2 : \underline{s}_2 \quad \mapsto \quad \bot$$

If $T_{\underline{s}_1} \cap T_{\underline{s}_2} = \emptyset$

Figure 1: Rules for solving equations $x = t$ in OSA

can be checked in a standard way for finite tree automata and, if the language is finite, it is easy to replace any variable of the corresponding sort by all the possible values it may take. Finally, all boolean operations on sorts are easy to perform as they are well known for regular tree languages. We use them without any more comment.

Proposition 2 *All rules given in figures 1 and 2 are correct.*

6 Termination and Completeness

As mentioned in the introduction, we must be careful when applying the rules if we want to obtain termination. We have no room for giving the control here (see [Com88]).

Theorem 1 *There is a set of rules (together with a control), which does terminate and such that irreducible equational problems are in solved form.*

This proves the following result:

Corollary 1 *The theory of T is decidable.*

Actually, it is possible to give a complete and recursive axiomatization of any such algebra. Also, this result can be reformulated:

Corollary 2 *Any regular tree language has a decidable first order theory.*

Parameters elimination

$$(UP_2') \quad \forall \vec{y} : P \wedge (y_1 \in \underline{s}_1 \neq y_2 \in \underline{s}_2 \vee d) \quad \mapsto \forall \vec{y}, y_3 \in \underline{s}_3 : P \wedge d\{y_1 \rightarrow y_3;\ y_2 \rightarrow y_3\}$$

If

1. $T_{\underline{s}_3} = T_{\underline{s}_1} \cap T_{\underline{s}_2}$ and $T_{\underline{s}_3}$ is neither equal to $T_{\underline{s}_1}$ nor equal to $T_{\underline{s}_2}$ nor empty.
2. $y_3 \notin Var(P, y_1, y_2, d)$

$$(UP_2'') \quad \forall \vec{y} : P \wedge (y \in \underline{s} \neq f(t_1, \ldots, t_n) \vee d) \quad \mapsto$$
$$\forall \vec{y}, \vec{y'} : P \bigwedge_{\phi} (y_1 \in \underline{s}_1 \neq t_1 \vee \ldots \vee y_n \in \underline{s}_n \neq t_n \vee d\{y \rightarrow f(y_1, \ldots, y_n)\})$$

If

1. ϕ stands for $f : \underline{s}_1 \times \ldots \times \underline{s}_n \rightarrow \underline{s}'$, $T_{\underline{s}'} \subseteq T_{\underline{s}}$, $\underline{s}_1 \ldots \underline{s}_n$maximal
2. $y \in \vec{y}$ and $y_1, \ldots, y_n \in \vec{y'}$
3. $\vec{y} \cap \vec{y'} = \emptyset$
4. $[\![y]\!] \not\subseteq [\![f(t_1, \ldots, t_n)]\!]$

"Sort Complement" rules (SC)

$$(SC_1) \quad \mathcal{P}[y \in \underline{s} \neq z \in \underline{s}'] \quad \mapsto \quad \exists w \in \underline{s}'', \mathcal{P}[y \neq w \in \underline{s}'' \wedge z = w]$$

If

1. y is a parameter and z is an unknown
2. $T_{\underline{s}'} - T_{\underline{s}} = T_{\underline{s}''} \neq \emptyset$

$$(CS_2) \quad y \in \underline{s} \neq z \in \underline{s}' \quad \mapsto \quad \top$$

If y is a parameter, z is a variable and $T_{\underline{s}'} - T_{\underline{s}} = \emptyset$.

Figure 2: Parameter elimination in OSA : complementary rules

References

[CL89] H. Comon and P. Lescanne. Equational problems and disunification. *J. Symbolic Computation*, 7:371–425, 1989.

[Com88] H. Comon. *Unification et Disunification: Théorie et Applications.* Thèse de Doctorat, I.N.P. de Grenoble, France, 1988.

[Com89a] H. Comon. Inductive proofs by specifications transformation. In *Proc. Rewriting Techniques and Applications 89, Chapel Hill, LNCS 355*, pages 76–91, Springer-Verlag, April 1989.

[Com89b] H. Comon. *Disunification.* Submitted, 1989.

[FV88] Z. Fülöp and S. Vàgvölgyi. *A characterization of irreducible sets modulo left-linear term rewiting systems by tree automata.* Research Report, Research Group on Theory of Automata, Hungarian Academy of Sciences, H-6720 Szeged, Somogyi u. 7. Hungary, 1988.

[GB85] J. H. Gallier and R. V. Book. Reductions in tree replacement systems. *Theoretical Computer Science*, 37:123–150, 1985.

[Kir88] C. Kirchner. Order-sorted equational unification. In *Proc. 5th Int. Conference on Logic Programming, Seattle*, August 1988.

[KKM88] C. Kirchner, H. Kirchner, and J. Meseguer. *Operational Semantics of OBJ-3.* Technical Report, CRIN, 1988.

[Mah88] M. J. Maher. Complete axiomatizations of the algebras of finite, rational and infinite trees. In *Proc. 3rd IEEE Symp. Logic in Computer Science, Edinburgh*, pages 348–357, July 1988.

[Sch86] M. Schmidt-Schauss. Unification in many-sorted equational theory. In *Proc. 8th Conf. on Automated Deduction, Oxford, LNCS 230*, pages 538–552, Springer-Verlag, July 1986.

[SNMG87] G. Smolka, W. Nutt, J. Meseguer, and J. Goguen. Order-sorted equational computation. In *Proc. Coll. on the Resolution of Equations in Algebraic Structures, Lakeway*, May 1987.

Disjunctive ψ-Term Unification
(Summary)

Hassan Aït-Kaci*

Digital, Paris Research Lab.
85, avenue Victor Hugo
92563 Rueil Malmaison Cedex
France

March 24, 1989

1 Introduction

I first briefly recall basic notions of the ψ-calculus—a calculus of structured type inheritance. In the talk, I shall then detail unification algorithms in the ψ-calculus, one without disjunction, and one with disjunction.

The ψ-calculus consists of a syntax of structured types called ψ-terms together with subtyping and type intersection operations. Intuitively, as expounded in [3], the ψ-calculus is an attempt at obtaining a convenience for representing record-like data structures in logic and functional programming more adequate than first-order terms without loss of the well-appreciated instantiation ordering and unification operation.

The natural interpretation of a ψ-term is that of a data structure built out of constructors, access functions, and subject possibly to equational constraints which reflect access coreference—sharing of structure. Thus, the syntactic operations on ψ-terms which stand analogous to instantiation and unification for first-order terms simply denote, respectively, sub-algebra ordering and algebra intersection, modulo type and equational constraints. This scheme even accommodates type constructors which are known to be partially-ordered with a given subtyping relation. As a result, a powerful operational calculus of structured subtypes is achieved formally without resorting to complex translation trickery. In essence, the ψ-calculus formalizes and operationalizes *data structure inheritance*, all in a way which is quite faithful to a programmer's perception.

Let us take an example to illustrate. Let us say that one has in mind to express syntactically a type structure for a *person* with the property, as expressed for the underlined symbol in Figure 1, that a certain functional diagram commutes.

One way to specify this information algebraically would be to specify it as a *sorted equational theory* consisting of a *functional signature* giving the sorts of the functions involved, and an *equational presentation*. Namely,

X : *person* *with*

functions

$$\begin{array}{lll} name & : person & \rightarrow id \\ first & : id & \rightarrow string \\ last & : id & \rightarrow string \\ spouse & : person & \rightarrow person \end{array}$$

equations

$$\begin{aligned} last(name(X)) &= last(name(spouse(X))) \\ spouse(spouse(X)) &= X \end{aligned}$$

The syntax of ψ-terms is one simply tailored to express as a term this specific kind of sorted monadic algebraic equational presentations. Thus, in the ψ-calculus, this information of Figure 1 is unambiguously encoded into a formula, perspicuously expressed as the ψ-term:

$$\begin{array}{l} X : person(name \Rightarrow id(first \Rightarrow string, \\ \qquad\qquad last \Rightarrow S : string), \\ \quad spouse \Rightarrow person(name \Rightarrow id(last \Rightarrow S), \\ \qquad\qquad spouse \Rightarrow X)). \end{array}$$

Since it is beyond the informal scope of this summary, we shall abstain from giving a complete formal definition of ψ-term syntax. (Such may be found elsewhere [4,3].) Nevertheless, it is important to distinguish among the three kinds of symbols which participate in a ψ-term expression. Thus we assume given a signature Σ of *type constructor symbols*, a set $\mathcal{A}$ of *access function symbols* (also called *attribute* symbols), and a set $\mathcal{R}$ of *reference tag symbols*. In the ψ-term above, for example, the symbols *person*, *id*, *string* are drawn from Σ, the symbols *name*, *first*, *last*, *spouse* from $\mathcal{A}$, and the symbols X, S from $\mathcal{R}$.[1]

A ψ-term is either *tagged* or *untagged*. A tagged ψ-term is either a reference tag in $\mathcal{R}$ or an expression of the form $X : t$ where $X \in \mathcal{R}$ and t is an untagged ψ-term. An untagged ψ-term is either *atomic* or *attributed*. An atomic ψ-term is a type symbol in Σ. An attributed ψ-term is an expression of the form $s(l_1 \Rightarrow t_1, \ldots, l_n \Rightarrow t_n)$ where $s \in \Sigma$ and the ψ-term principal type, the l_i's are mutually distinct attribute symbols in $\mathcal{A}$, and the t_i's are ψ-terms ($n \geq 1$).

Reference tags may be viewed as typed variables where the type expressions are untagged ψ-terms. Hence, as a condition to be well-formed, a ψ-term must have all occurrences of reference tags consistently refer to the same structure. For example, the reference tag X in

$$\begin{array}{l} person(id \Rightarrow name(first \Rightarrow string, \\ \qquad\qquad last \Rightarrow X : string), \\ \quad father \Rightarrow person(id \Rightarrow name(last \Rightarrow X : string))) \end{array}$$

refers consistently to the atomic ψ-term *string*. To simplify matters and avoid redundancy, we shall obey a simple convention of specifying the type of a reference tag at most once as in

*hak@decprl.dec.com

[1] We shall use the lexical convention of using capitalized identifiers for reference tags.

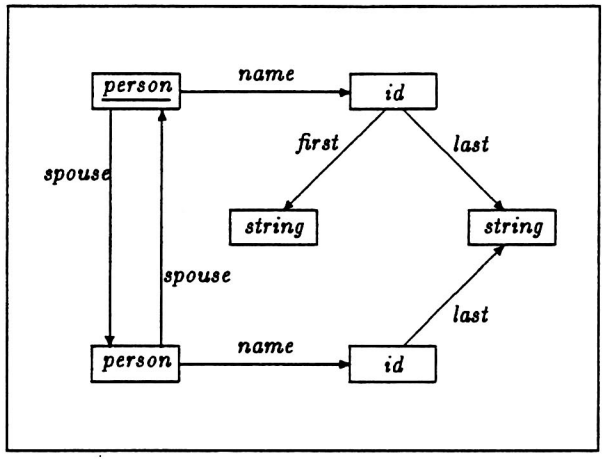


Figure 1: A Functional Diagram

$$person(id \Rightarrow name(first \Rightarrow string, \\ last \Rightarrow X : string), \\ father \Rightarrow person(id \Rightarrow name(last \Rightarrow X)))$$

and understand that other occurrences are equally referring to the same structure. In fact, this convention is necessary if we have circular references as in $X : person(spouse \Rightarrow person(spouse \Rightarrow X))$. Finally, a reference tag appearing nowhere typed, as in $junk(kind \Rightarrow X)$ is implicitly typed by a special universal type symbol $\top$ always present in Σ. This symbol will be left invisible and not written explicitly as in $(age \Rightarrow integer, name \Rightarrow string)$. In the sequel, by ψ-term we shall always mean *well-formed* ψ-term.

Similarly to first-order terms, a subsumption preorder can be defined on ψ-terms which is an ordering up to reference tag renaming. Given that the signature Σ is partially-ordered (with a greatest element $\top$), its partial ordering is extended to the set of attributed ψ-terms. Informally, a ψ-term t_1 is subsumed by a ψ-term t_2 if (1) the principal type of t_1 is a subtype in Σ of the principal type of t_2; (2) all attributes of t_2 are also attributes of t_1 with ψ-terms which subsume their homologues in t_1; and, (2) all coreference constraints binding in t_2 must also be binding in t_1.

For example, if $student < person$ and $austin < cityname$ in Σ then the ψ-term

$$student(id \Rightarrow name(first \Rightarrow string, \\ last \Rightarrow X : string), \\ lives\_at \Rightarrow Y : address(city \Rightarrow austin), \\ father \Rightarrow person(id \Rightarrow name(last \Rightarrow X), \\ lives\_at \Rightarrow Y))$$

is subsumed by the ψ-term

$$person(id \Rightarrow name(last \Rightarrow X : string), \\ lives\_at \Rightarrow address(city \Rightarrow cityname), \\ father \Rightarrow person(id \Rightarrow name(last \Rightarrow X))).$$

In fact, if the signature Σ is such that *greatest lower bounds* (GLB's) exist for any pair of type symbols, then the subsumption ordering on ψ-term is also such that GLB's exist. Such are defined as the *unification* of two ψ-terms. Consider for example the signature displayed in Figure 2 and the two ψ-terms

$$X : student(advisor \Rightarrow faculty(secretary \Rightarrow Y : staff, \\ assistant \Rightarrow X), \\ roommate \Rightarrow employee(representative \Rightarrow Y))$$

and

$$employee(advisor \Rightarrow f_1(secretary \Rightarrow employee, \\ assistant \Rightarrow U : person), \\ roommate \Rightarrow V : student(representative \Rightarrow V), \\ helper \Rightarrow w_1(spouse \Rightarrow U)).$$

Their unification (up to tag renaming) yields the term[2]

$$W : workstudy(advisor \Rightarrow f_1(secretary \Rightarrow Z : workstudy(repres \\ assistant \Rightarrow W, \\ roommate \Rightarrow Z, \\ helper \Rightarrow w_1(spouse \Rightarrow W)).$$

Thus,

Theorem 1 *If the type signature is a lattice, then so is the set of ψ-terms.*

I shall detail unification algorithm for ψ-terms in my talk. This algorithm is an adaptation of an efficient unification algorithm based on a rooted labelled (directed) graph representation of ψ-terms, such as is illustrated in Figure 1. The nodes are labelled with type symbols from Σ, and the arcs are labelled with attribute symbols. The root node is one from which every other is reachable and is labelled with the principal type of the ψ-term (underlined in Figure 1). Nodes which are shared in the graph correspond to tagged subterms. Such graphs are quite like finite-state automata

[2]Incidentally, if *least upper bounds* (LUBs) are defined as well in Σ, so are they for ψ-terms. For example for these two ψ-terms, their LUB (most specific generalization) is

$$person(advisor \Rightarrow faculty(secretary \Rightarrow employee, \\ assistant \Rightarrow person), \\ roommate \Rightarrow person)).$$

Thus, a lattice structure can be extended from Σ to ψ-terms [2,4]. Although it may turn out useful in other contexts, we shall ignore this generalization operation here.

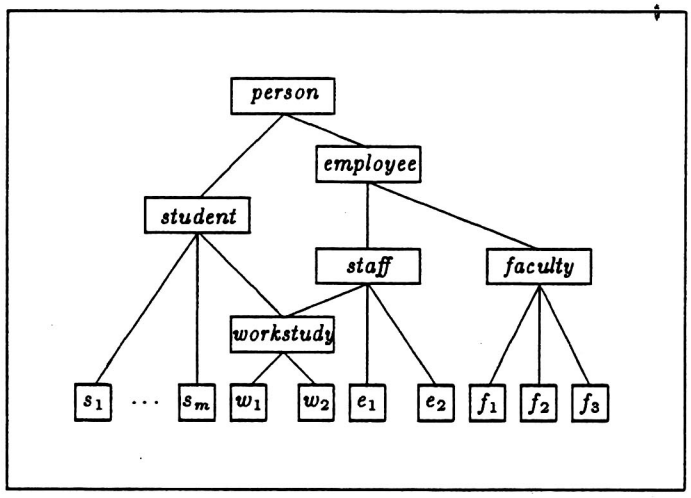


Figure 2: A Signature with Well-Defined GLB's

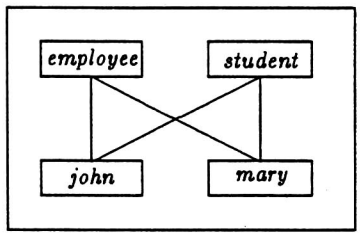


Figure 3: A non-lattice

with Σ-sorted nodes (Moore machines) and where the transitions are attribute symbols. In fact, the ψ-term unification algorithm is an immediate adaptation of the algorithm deciding equivalence of finite-state automata [1]. This algorithm merges nodes which are reached by equal transition paths into coreference classes, starting from the roots and following all reachable strings of attributes from them.

[3] Each merged class is assigned the type symbol in Σ which is the GLB of the types of all nodes in the class. The inconsistent type $\bot$ (the least element in Σ) may result which makes the whole unification fail.

A technicality arises if Σ is not a lower semi-lattice. For example, given the (non-lattice) type signature of Figure 3 the GLB of *student* and *employee* is not uniquely defined, in that it could be *john or mary*. That is, the set of their common lower bounds does not admit *one* greatest element. However, the set of their *maximal* common lower bounds offers the most general choice of candidates. Clearly, the *disjunctive* type $\{john; mary\}$ is an adequate interpretation.[4] Thus the ψ-term syntax may be enriched with disjunction denoting type union.

For a more complete formal treatment of disjunctive ψ-terms, the reader is referred to [4] and to [3]. It will suffice to indicate here that a *disjunctive* ψ-term is a set of incomparable ψ-terms, written $\{t_1; \ldots; t_n\}$ where the t_i's are basic ψ-terms. A *basic* ψ-term is one which is non-disjunctive. The subsumption ordering is extended to disjunctive (sets of) ψ-terms such that $D_1 \leq D_2$ *iff* $\forall t_1 \in D_1, \exists t_2 \in D_2$ such that $t_1 \leq t_2$. This justifies the convention that a singleton $\{t\}$ is the same as t, and that the empty set is identified with $\bot$. Unification of two disjunctive ψ-terms consists in the enumeration of the set of all maximal ψ-terms obtained from unification of all elements of one with all elements of the other. For example, limiting ourselves to disjunctions of atomic ψ-terms in the context of signature in Figure 2, the unification of $\{employee; student\}$ with $\{faculty; staff\}$ is $\{faculty; staff\}$. It is the set of maximal elements of the set $\{faculty; staff; \bot; workstudy\}$ of pairwise GLB's.

Therefore,

Theorem 2 *Given any partially-ordered type signature, the set of ϵ-terms is always a distributive lattice.*

In practice, it is convenient to allow nesting disjunctions in the structure of ψ-terms. For instance, to denote a type of person whose friend may be an astronaut with same first name, or a businessman with same last name, or a charlatan with first and last names inverted, we may write such expressions as:

$$\begin{aligned} person(id \Rightarrow name(&first \Rightarrow X : string,\\ &last \Rightarrow Y : string),\\ friend \Rightarrow \{&astronaut(id \Rightarrow name(first \Rightarrow X))\\ &; businessman(id \Rightarrow name(last \Rightarrow Y))\\ &; charlatan(id \Rightarrow name(first \Rightarrow Y,\\ &\qquad last \Rightarrow X))\}) \end{aligned}$$

[3] It is not quite true that ψ-term graphs are *finite*-state since an essential difference is that the attribute set is unbound. Indeed, transitions which do not appear in the graph are implicitly present leading to infinitely many distinct states, each of which is labelled with $\top$. Merging two nodes which *both* possess out-going attributes will proceed by materializing such invisible transitions and states as needed should a transition out of one node not be present out of the other node. Nevertheless, the process is always terminating since this materialization does not happen if at least one of the two nodes has no out-going arcs. This subtle point is essential as finite-state automata *inequivalence* is decidable by guessing non-deterministically a string of transitions and following it in both graphs while this is not possible for (even first-order term) unification as transitions out of "variable" nodes are not *a priori* known [6].

[4] See [5] for a description of an efficient method for computing such GLB's.

Tagging may even be chained or circular within disjunctions as in:

$$P : \{charlatan \\ ; person(id \Rightarrow name(first \Rightarrow X : 'john', \\ last \Rightarrow Y : \{ 'doe'; X\}), \\ friend \Rightarrow \{P;\ person(id \Rightarrow name(first \Rightarrow Y, \\ last \Rightarrow X))\})\}$$

which expresses the type of either a charlatan, or a person named either "John Doe" or "John John" and whose friend may be either a charlatan, or himself, or a person with his first and last names inverted. These are no longer graphs but hypergraphs.

Of course, one can always expand out all nested disjunctions in such an expression, reducing it to a canonical form consisting of a set of non-disjunctive ψ-terms. The process is described in [2], and is akin to converting a non-deterministic finite-state automaton to its deterministic form, or a first-order logic formula to its disjunctive normal form. However, more for pragmatic efficiency than just notational convenience, it is both desirable to keep ψ-terms in their non-canonical form. In my talk, I shall describe an and/or graph unification algorithm with lazy expansion (saving expansions in case of failure or unification against $\top$).

References

[1] Aho, V.A., Hopcroft, J.E., and Ullman, J.D. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA. 1974.

[2] Aït-Kaci, H. *A Lattice-Theoretic Approach to Computation Based on a Calculus of Partially-Ordered Type Structures.* Ph.D. Thesis. Computer and Information Science, University of Pennsylvania. Philadelphia, PA. 1984.

[3] Aït-Kaci, H. and R. Nasr, "LOGIN: A Logic Programming Language with Built-in Inheritance." *Journal of Logic Programming* **3**(3), pp. 187–215. 1986.

[4] Aït-Kaci, H., "An Algebraic Semantics Approach to the Effective Resolution of Type Equations." *Journal of Theoretical Computer Science* **45**, pp. 293–351. 1986.

[5] Aït-Kaci, H., Boyer, R., Lincoln, P., and R. Nasr, Efficient Implementation of Lattice Operations. *ACM Transactions on Programming Languages and Systems***11**(1), pp. 115-146. January 1989.

[6] Paris Kanellakis, *Personal communication.* 1988.

[7] Smolka, G., and H. Aït-Kaci, *Inheritance Hierarchies: Semantics and Unification.* MCC Technical Report AI-057-86, AI/ISA Project. Microelectronics and Computer Technology Corporation, Austin, TX. July, 1986. (To appear in *Journal of Symbolic Computation*, special issue on unification.)

Unification on Restricted Terms

Laurence Puel
LIENS URA CNRS 1327
Ecole Normale Supérieure
45 rue d'Ulm
75005 Paris FRANCE

Ascánder Suárez
Digital Equipment Corporation
Paris Research Laboratory
85 Av Victor Hugo
92563 Rueil Malmaison, FRANCE

Introduction

A term t with variables, is a representation of all ground terms got by replacing its variables by ground terms. A subset of a given set can be defined either by a description of its elements or as the complement of another subset. For example, the set $\{a, f(a), f(f(a)), \ldots\}$ of ground terms over the alphabet containning the constant a and the unary function f can be represented by the variable x. A representation of $\{f(a), f(f(a)), \ldots\}$ is either $f(x)$ or x such that $x \neq a$. Following this idea, we define the concept of restricted terms where structural constraints are associated to variables and we use it to define partitions on a set of ground terms. Thus, in order to compute with these terms, we describe two operations on restricted terms: unification and decomposition with respect to an ordered set of patterns.

1 Restricted Term

The term $t = f(x, y)$ where f belongs to Σ and x , y are variables stands for the set $T = \{f(\sigma(x), \sigma(y)) | \forall \sigma : X \rightarrow T(\Sigma)\}$. We want to deal with terms that represent an arbitrary set of ground terms. For example, let $\Sigma = \{f, a, b\}$. Let us consider the subset of $f(x, y)$ of all terms different from $f(a, b)$ (i.e. $\{f(\sigma(x), \sigma(y)) | \forall \sigma$ such that $\sigma(x) \neq a$ or $\sigma(y) \neq b\}$). This subset can be represented by the union of $f(b, a)$, $f(b, f(x, y))$, $f(a, a)$, $f(b, b)$, $f(f(z, r), a)$ and $f(f(u, v), f(w, s))$. On the other hand, the union of $f(f(x, y), a)$ and $f(f(u, v), f(w, s))$ stands for the subset of $f(x, y)$ where x is different from a and b and y from b. These examples suggest us to represent such a set by a term with variables on which there are some structural constraints. This can be illustrated as following:

1. $\{t = f(x, y)$ such that $t \neq f(a, b)\} = \{f(x \neq a, y), f(x, y \neq b)\}$
2. $\{f(b, a), f(b, f(w, s)), f(f(u, v), a), f(f(u, v), f(w, s))\} = \{f(x \neq a, y \neq b)\}$
3. $\{f(f(u, v), a), f(f(u, v), f(w, s))\} = \{f(x \neq (a, b), y \neq b)\}$

Let us notice that even in case of an infinite alphabet term representation with constraints can be finite where it is not with the classical representation. We formalize this notion of restricted term.

Definition 1 *Let Ω be a symbol of arity 0 not belonging to Σ. An Ω-term is a term belonging to $T(\Sigma, \Omega)$.*

In such an Ω-term variables' name is not relevant and we call this kind of term a pattern. When we compute with an Ω-term, for example in order to unify it with a term, we replace each occurrence of Ω by a new variable.

Definition 2 (Restricted term) *A restricted variable, denoted Rvariable, is defined as a variable with a list of Ω-terms associated to. A restricted term, denoted Rterm, is recursively defined as either a Rvariable or $f(t_1, \ldots, t_n)$ where $f \in \Sigma$ and $t_1, \ldots, t_n$ are Rtems.*

$$\begin{array}{lcl} \mathit{Rvariable} & := & (\mathit{variable}, \mathit{List\ of}\ \Omega\text{−terms}) \\ \mathit{Rterm} & := & \mathit{Rvariable} \\ & & \mid \mathit{symbol}\ (\mathit{Rterm}, \ldots, \mathit{Rterm}) \end{array}$$

In fact in our representation, we associate to variables a list of patterns to characterize some unallowed substitutions. More generally we define constraints on terms. Notice that we deal with three different kinds of terms: those that belong to $T(\Sigma, X)$, the restricted terms and the Ω-terms in which variables' names are not relevant.

2 Constraints

Definition 3 (Atomic Constraint, Variable Atomic Constraint) *An atomic constraint (resp. a variable atomic constraint) is a pair (term, list of* $\Omega-$terms*) (resp. (variable, list of* $\Omega-$terms*))*

Definition 4 (Constraint) *A constraint is recursively defined as either an atomic constraint or the disjunction of two constraints or the conjunction of two constraints.*

$$\begin{array}{rcl} Constraint & := & (term, list\ of\ \Omega-\text{terms}) \\ & & \mid (Constraint \vee Constraint) \\ & & \mid (Constraint \wedge Constraint) \end{array}$$

Using these connectors *or* ($\vee$) and *and* ($\wedge$), a constraint can always be written as a disjunction of conjunction of atomic constraints named a normal form. The notation $x \neq l$ is often more convenient than the pair (x, l) and will be used instead of that last one if necessary in the examples.

Definition 5 (Substitution over a constraint) *Let σ be a substitution, $P = (t, \mathcal{L})$ an atomic constraint, P_1, P_2 two constraints. By definition $\sigma(P) = (\sigma(t), \mathcal{L})$, $\sigma(P_1 \vee P_2) = \sigma(P_1) \vee \sigma(P_2)$ and $\sigma(P_1 \wedge P_2) = \sigma(P_1) \wedge \sigma(P_2)$*

Note that there is no variables belonging to X in the right part of a constraint. Such constraints are said to be structural.

Constraint Valuation

We define a function from the set of constraints to a two-elements set, called a valuation, in order to formalize the fact that an atomic constraint $(t, \mathcal{L})$ is satisfied if and only if there is no term $l \in \mathcal{L}$ such that t is an instance of l.

Definition 6 (Filter on Terms) *The relation $\leq$ over terms is defined by $t \leq t'$ if and only if there exists a substitution σ such that $\sigma(t) = t'$.*

Definition 7 (Filter on Substitutions) *The relation $\leq$ over substitutions is defined by $\sigma \leq \sigma'$ if and only if there exists a substitution η such that $\sigma' = \eta \circ \sigma$. σ is said to be more general than σ'.*

Definition 8 (Valuation) *Let $\mathcal{P}$ be a set of constraints and $\{\mathcal{T}, \mathcal{F}\}$ a two-elements set. The valuation $v : \mathcal{P} \rightarrow \{\mathcal{T}, \mathcal{F}\}$ is defined as following: Let $(t, \mathcal{L})$ be an atomic constraint.*

$$\begin{array}{rcl} v((t, \mathcal{L})) & = & \mathcal{F}\ if\ \exists l \in \mathcal{L}\ such\ that\ l \leq t \\ & = & \mathcal{T}\ if\ not \end{array}$$

Connectors $\vee$ and $\wedge$ are interpreted as usually.

Notation We write $\vdash P$ when $v(P) = \mathcal{T}$. The empty list [] is written instead of $(t, [])$ when $\mathcal{L}$ is empty.
Remarks For every term t, $v((t, [\Omega])) = \mathcal{F}$ and $v((t, [])) = v(()) = \mathcal{T}$.

Definition 9 *A substitution σ satisfies a constraint P if and only if $\vdash \sigma(P)$.*

Lemma 1 *Let P be a constraint and σ a substitution such that $\vdash \sigma(P)$. For every substitution ρ more general than σ, $\vdash \rho(P)$.*

3 Restricted Terms and Unification

Definition 10 (Restricted Term, Structurally Restricted Term) *A restricted term is defined as a pair (term, constraint) where the atomic constraints are variable atomic constraints.*

When $\vdash P$ the restricted term $(t,\ P)$ is denoted by $T\ =\ t\ s.t.\ P$, otherwise T is a representation of the empty set and thus $T = \emptyset$. When $P = (,[])$, the restricted term $(t,\ P)$ is in fact unrestricted and is written t.

Definition 11 (Allowed Substitution) *Let $T\ =\ t\ s.t.\ P$ be a restricted term in which by definition $\vdash P$. Let σ be a substitution. If $\vdash \sigma(P)$, the substitution σ is said to be allowed for T and $\sigma(T) = \sigma(t)\ s.t.\ simpl(\sigma(P))$. Otherwise the substitution σ is not allowed for T and $\sigma(T) = \emptyset$.*

Definition 12 *Let T be a restricted term and σ a substitution allowed for T. The restriction of T by σ is the restricted term $T|_\sigma = t\ s.t.\ P \wedge \bigvee_{x \in Dom(\sigma) \cap Var(t)}(x, \sigma(x))$ where $Var(t)$ means the set of t's variables.*

Let Id be the identical substitution. Note that $T|_{Id} = \emptyset$.

Proposition 1 *For every restricted term T and every allowed substitution σ for T, $T = \sigma(T) \cup T|_\sigma$.*

Definition 13 (Unification on restricted terms) *Let $T_1 = t_1\ s.t.\ P$ and $T_2 = t_2\ s.t.\ Q$ be two restricted terms. A substitution σ is a unifier of T_1 and T_2 if and only if σ unifies t_1 and t_2 and σ satisfies both constraints P and Q (i.e. $\sigma(t_1) = \sigma(t_2)$ and $\vdash \sigma(P)$, $\vdash \sigma(Q)$). The unified restricted term is $\sigma(t_1)(= \sigma(t_2))$ s.t. $\sigma(P) \wedge \sigma(Q)$.*

Proposition 2 (Most general unifier) *The most general unifier of two restricted terms $t\ s.t.\ P$ and $t'\ s.t.\ Q$ is the most general unifier of the unrestricted term t and t' if this unifier exists and satisfies the constraint $P \wedge Q$.*

Algorithm

$Unify(T_1, T_2) =$ case of

$(X = x\ s.t.\ P_x, Y = y\ s.t.\ P_y)\ \rightarrow$
  If $x = y$ then () else let $\sigma_0 = mgu(x, y)$, $(X \leftarrow x\ s.t,\ P_x\ \wedge\ \sigma_0 P_y)$

$(X = x\ s.t.\ P_x, f(t_1, \ldots, t_n)\ s.t.\ P_f)\ \rightarrow$
  Let $\sigma_0 = mgu(x, f(t_1, \ldots, t_n))$, if $\vdash \sigma_0 P_x$
    then $(X \leftarrow f(t_1, \ldots, t_n)\ s.t.\ simpl(\sigma_0 P_x \wedge P_f))$
    else fail

$(f(t_1, \ldots, t_n)\ s.t.\ P_f, X = x\ s.t.\ P_x,)\ \rightarrow$
  $Unify(X = x\ s.t.\ P_x, f(t_1, \ldots, t_n)\ s.t.\ P_f)$

$(f(t_1, \ldots, t_n)\ s.t.\ P_f, f(t'_1, \ldots, t'_n)\ s.t.\ P'_f)\ \rightarrow$
  $Unify(t_1\ s.t.\ P_f, t'_1\ s.t. P'_f) \circ \ldots \circ Unify(t_n\ s.t.\ P_f, t'_n\ s.t. P'_f)$

$(f(t_1, \ldots, t_n)\ s.t.\ P_f, f'(t'_1, \ldots, t'_p)\ s.t.\ P_{f'})\ \rightarrow$
  fail.

The function *simpl* represents the process that transforms a constraint P into a variable constraint or $(t, [\Omega])$ or $(t, [])$.

4 Term Decomposition

Let $S = (s_1, \ldots, s_n)$ be an ordered list of $\Omega - terms$ named patterns and $T = t\ s.t.\ P$ a restricted term. As already said T is a representation of a set of ground terms. To decompose T with respect to S is to split the set associated to T into disjoint subsets such that each subset contains instances of at most one element of S. For example, let $S = \{f(a, g(\Omega)), f(a, \Omega)\}$ and $T = f(x, y)\ s.t.\ []$. T is the disjoint union of T_1, T_2 and T_3 where $T_1 = f(a, g(z))\ s.t.\ []$, $T_2 = f(a, y)\ s.t.\ [y \neq g(\Omega)]$ and $T_3 = f(x, y)\ s.t.\ [x \neq a]$. We generalize this concept of decomposition looking for instances of patterns not only at the root of the tree

but also at an internal node (i.e. a node different from the root). For example, let $S = \{f(a,\Omega), h(\Omega)\}$ and $T = g(f(x,y),z)$ *s.t.* []. $T = T_1 \cup T_2 \cup T_3$ where $T_1 = g(f(a,y),z)$ *s.t.* [], $T_2 = g(f(x,y),h(k))$ *s.t.* $[x \neq a]$, $T_3 = g(f(x,y),z)$ *s.t.* $[x \neq a, z \neq h(\Omega)]$.

4.1 Decomposition at a Vertex

Definition 14 (Decomposition w.r.t. a Pattern at a vertex) *Let $T = t$ s.t. P be a restricted term, u a vertex of t and s a pattern. If T/u and s are unifiable with σ as their most general unifier, the decomposition of T w.r.t. s at u, denoted $decomp(T,s,u)$, is equal to $\sigma(T)$.*

With this definition, $decomp(T,\Omega,u) = T$.

Definition 15 (Decomposition w.r.t. an Ordered Set of Patterns at a Vertex) *Let $T = t$ s.t. P be a restricted term, u a vertex of t and $S = \{s_1,\ldots,s_n\}$ an ordered set of patterns. The decomposition of T w.r.t. S at u, $Decomp(T,S,u)$, is recursively defined as*

$$\begin{array}{lcl} Decomp(\emptyset,S,\_) & = & \emptyset \\ Decomp(T,\emptyset,u) & = & \emptyset \\ Decomp(T,S,u) & = & Decomp(T,\{s_2,\ldots,s_n\},u) \text{ if } T/u \text{ and } s_1 \text{ are not unifiable} \\ & & decomp(T,s_1,u) \cup Decomp(T|_{\sigma_1},S,u) \\ & & \text{where } \sigma_1 \text{ is the m.g.u. of } T/u \text{ and } s_1 \text{otherwise} \end{array}$$

Proposition 3 *Let $T = t$ s.t. P be a restricted term, u a vertex of t and $S = \{s_1,\ldots,s_n\}$ a set of patterns. Then $T = Decomp(T,\{s_1,\ldots,s_n,\Omega\},u)$.*

4.2 Decomposition w.r.t. a Set of Patterns

More generally, we decompose a term with respect to a set of patterns not only at a given vertex u but successively at each element of an ordered set of vertices (by the hierarchical ordering for example). Let $T|_{S,u} = Decomp(T,\{s_1,\ldots,s_n,\Omega\},u) - Decomp(T,S,u)$.

Definition 16 (Decomposition w.r.t. a Set of Patterns) *Let $U = \{u_1,\ldots,u_p\}$ an ordered set of vertices. Let $DECOMP(T,S,U) = Decomp(T,S,u_1) \bigcup DECOMP(T|_{S,u_1},S,\{u_2,\ldots,u_p\})$.*

$$\mathbf{decomp}(T,S,U) =$$

$$DECOMP(T,S,U) \bigcup \bigcup_{(f_1,\ldots,f_p)\in\Sigma^p} \mathbf{decomp}(T|_{S,U}[u_i \leftarrow f_i(\ldots) | 1 \leq i \leq p], S, \{\ldots,u_i.1,\ldots,u_i.ar(f_i),\ldots\})$$

where $T|_{S,U}$ denotes the remainder of $DECOMP(T,S,U)$.

$\mathbf{decomp}(x,S,\epsilon)$ gives all the different decompositions of a the variable x. Let S be unavoidable (i.e. there exists an integer k such that for every tree $t \in T(\Sigma,X)$ with $depth(t) > k$ there exists $s \in S$ factor of t). $\mathbf{decomp}(x,S,\epsilon)$ stops and allows us to compute by unification the decompositions of any tree [Pue87],[Pue89].

References

[Pue87] Laurence Puel. *Bons préordres sur les arbres associés à des ensembles inévitables et preuves de terminaison de systèmes de réécriture.* Thèse d'Etat, Université Paris 7, 1987.

[Pue89] Laurence Puel. Embedding with patterns and associated recursive path ordering. In , editor, *RTA89*, page , April 1989.

Appendix

P. Scammel: Wine Tasting at Weingut Schönfeld

P. Lescanne: Life with Limited Vocabulary (banquet speech)

List of Participants

WINE TASTING AT WEINGUT SCHÖNFELD, JUNE 27, 1989

Philip Scammel

1. 1988er DIEDESFELDER REBSTÖCKEL
   RIESLING KABINETT TROCKEN
2. 1988er WOLLMESHEIMER MÜTTERLE
   KERNER SPÄTLESE TROCKEN
3. 1987er DIEDESFELDER REBSTÖCKEL
   SILVANER HALBTROCKEN
4. 1988er WOLLMESHEIMER MÜTTERLE
   RIESLING KABINET HALBTROCKEN
5. 1988er DIEDESFELDER PARADIES
   SCHEUREBE KABINETT
6. 1987er DIEDESFELDER BERG
   KERNER SPÄTLESE
7. 1988er DIEDESFELDER REBSTÖCKEL
   WEIßBURGUNDER SPÄTLESE
8. 1988er WOLLMESHEIMER MÜTTERLE
   HUXELREBE AUSLESE TROCKEN
9. 1988er DIEDESFELDER ÖLGÄSSEL
   GEWÜRZTRAMINER AUSLESE
10. 1983er WOLLMESHEIMER MÜTTERLE
    RIESLING EISWEIN
    Goldene Kammerpreismünze 1984

Life with a limited vocabulary*

Pierre LESCANNE
Centre de Recherche en Informatique de Nancy
LORIA
Campus Scientifique, BP 239,
54506 Vandœuvre-lès-Nancy, France
email: lescanne@loria.crin.fr

In this paper we show that a limited vocabulary can lead to strange and irregular situations. A complete analysis is given using five simple canonical case studies.

The referee report

I read this paper *completely*, I noticed it gets harder and harder as the complexity increases *monotonically*. But since the author adopts a *canonical* point of view and the style is *regular* and *simple*, I recommend its acceptance.

The love letter

My *simple* happiness is not *complete* since my love for you increases *regularly* and *monotonically* every day. Do the *canonical* thing, *simply* marry me!

The letter to the bank

I received your *complete* account. It is not *canonical* since there is something *irregular*. My credit increases *monotonically* then all of a sudden starts to decrease. Is there a *simple* explanation?

At the restaurant

She (he) takes a *complete* menu. She (he) orders a *simple* entree. As she (he) is in France, she (he) drinks a "*canon* de vin" and ends with a *regular* coffee. The tip increases *monotonically* with her (his) degree of satisfaction.

Title of the communication for the next workshop

A *simple* and *complete* procedure for *monotonically* generating *canonical* solved forms for *regular* theories.

*This paper was presented at banquet speech at the Third International Workshop on Unification UNIF'89.

Acknowledgement

I would simply like to completely, monotonically, canonically and thank Eric Domenjoud, Nachum Dershowitz, Harald Ganzinger and, Wayne Snyder for their regular help.

Habib ABDULRAB
Fac. de Sciences LRI
BP 118
F-76134 Mont St. Aignan
France
inria!geocub!abdulrab

Mohamed ADI
CRIN
BP 239
F-54506 Vandoeuvre Cedex
France
adi@crin.crin.fr

Hassan AIT-KACI
Digital, Paris Research Laboratory
85 Avenue Victor Hugo
F-92563 Rueil Malmaison Cedex
France
hak@decprl.dec.com

Jürgen AVENHAUS
FB Informatik, Univ. Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern
FR Germany
avenhaus@uklirb.uucp

Franz BAADER
Institut für Math. Maschinen
Universität Erlangen
D-8520 Erlangen
FR Germany
baader@informatik.uni-erlangen.de

Alexander BOCKMAYR
SFB 314, Univ. Karlsruhe
Postfach 6980
D-7500 Karlsruhe
FR Germany
bockmayr@ira.uka.de

Alexandre BOUDET
LRI Orsay, Universite Paris-Sud
Bat 490
F-91405 Orsay CEDEX
France
boudet@lri.lri.fr

Hans-Jürgen BÜRCKERT
DFKI
Postfach 2080
D-6750 Kaiserslautern
FR Germany
buerckert@uklirb.uucp

Hubert COMON
LIFIA
Rue Felix Viallet
F-38031 Grenoble Cedex
France
comon@lifia.fr

Evelyne CONTEJEAN
LRI Orsay, Universite Paris-Sud
Bat 490
F-91405 Orsay CEDEX
France
contejea@lri.lri.fr

Catherine DELOR
LIENS
45 rue d'ULM
F-75005 Paris
France

Herve DEVIE
LRI Orsay, Universite Paris-Sud
Bat 490
F-91405 Orsay CEDEX
France
devie@lri.lri.fr

Eric DOMENJOUD
CRIN
BP 239
F-54506 Vandoeuvre-les-Nancy
France
domen@crin.crin.fr

Luc DUPONCHEEL
ALCATEL F.
Wellesplein 1
B-2018 Antwerpen
Belgium
##e-mail

Jean GALLIER
Univ. of Pennsylvania
Dept. of Comp.& Inf. Sci.
200 South 33 ad Street
Philadephia, PA 19104-6389
USA
jean@central.cis.upenn.edu

Harald GANZINGER
Lehrstuhl Informatik V
Univ. Dortmund
Postfach 500500
D-4600 Dortmund
FR Germany
hg@informatik.uni-dortmund.de

Ewa GRACZYNSKA-WRONSKA
Inst. of Mathematics
Polish Academy of Sciences
ul. Kopernika 18
51-617 Wroclaw
Poland

Bernhard GRAMLICH
Universiät Kaiserslautern, FB Informatik
Postfach 3049
6750 Kaiserslautern
FR Germany
gramlich@uklirb.uucp

Alexander HEROLD
ECRC
Arabellastr. 17
D-8000 München 81
FR Germany
herold@ecrcvax.uucp

Steffen HÖLLDOBLER
FG Intellektik, FB Informatik, TH Darmstadt
Alexanderstr. 10
D-6100 Darmstadt
FR Germany
xiishoe@ddathd21.bitnet

Claude KIRCHNER
CRIN
BP 239
F-54506 Vandoeuvre Cedex
France
ckirchne@crin.crin.fr

Helen KIRCHNER
CRIN
BP 239
F-54506 Vandoeuvre Cedex
France
hkirchne@crin.crin.fr

Francis KLAY
CRIN
BP 239
F-54506 Vandoeuvre-les-Nancy
France
klay@crin.crin.fr

Mike LAI
Dept. of Comp. Science,
Royal Holloway and Bedford New College
University of London
Egham Hill
Egham, Surrey, PW20 0EX
England
ml@uk.ac.rhbnc.cs.ux

Hans LEIß
Siemens AG, ZFE F2 INF 2
Otto-Hahn-Ring 6
D-8000 München 83
FR Germany
leiss@ztivax.uucp

Pierre LESCANNE
CRIN
BP 239
F-54506 Vandoeuvre Cedex
France
lescanne@bourbaki.crin.fr

Ursula MARTIN
Comp. Science Department, Univ. of London
Egham Hill
Egham, Surrey, TW20 0EX
England
uhm@uk.ac.man.cs.ux

Brian MATTHEWS
Systems Engineering
Rutherford Appleton Laboratory
Didcot
Oxon OX11 0QX
England
bmm@inf.rl.ac.uk

Tobias NIPKOW
Computer Laboratory, Univ. of Cmabridge
Pembroke Street
Cambrigde CB2 3QG
England
tnn@computer-lab.cambridge.ac.uk

Werner NUTT
DFKI
Postfach 2080
D-6750 Kaiserslautern
FR Germany
nutt@uklirb.uucp

Hans Jürgen OHLBACH
FB Informatik, Univ. Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern
FR Germany
ohlbach@uklirb.uucp

Jean-Pierre PECUCHET
Fac. de Sciences LRI
BP 118
F-76134 Mont St. Aignan
France
inria!geocub!pecuchet

Loic POTTIER
INRIA
2004 route des Lucioles
F-06565 Valbonne Cedex
France
pottier@mirsa.inria.fr

Axel PRÄCKLEIN
Univ. Kaiserslautern, FB Informatik
Postfach 3049
D-6750 Kaiserslautern
Germany
prckln@uklirb.uucp

Laurence PUEL
LIENS
45 rue d'ULM
F-75005 Paris
France
puel@ens.ens.fr

Jean-Luc REMY
CRIN
BP 239
F-54506 Vandoeuvre Cedex
France
remy@crin.crin.fr

Jean-Francois ROMEUF
Fac. de Sciences LRI
BP 118
F-76134 Mont St. Aignan
France
inria!litp!jfr

William C. ROUNDS
Comp. Science Department
University of Michigan
Ann Arbor
Michigan 48109
USA

current address: W.C. Rounds
Universität Stuttgart
Inst. f. masch. Sprachverarbeitung
Kepler-Str. 17
7000 Stuttgart 80
FR Germany

Rosa RUGGERI
Dipartimento di Mathematica,
Universita di Catania
Viale A. Doria 6
I-95125 Catania
Italy
mi3ctg5i@icineca.bitnet *or*
alfredo@astrct.infn.it

current address: R. Ruggeri
FB Informatik, Univ. Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern
FR Germany
ruggeri@uklirb.uucp

Klaus SCHULZ
SNS, Univ. Tübingen
Biesingerstr. 10
d-7400 Tübingen
FR Germany

Jörg H. SIEKMANN
FB Informatik, Univ. Kaiserslautern
Postfack 3049
D-6750 Kaiserslautern
FR Germany
siekmann@uklirb.uucp

Gert SMOLKA
IWBS, IBM Deutschland
Postfach 80 08 80
7000 Stuttgart 80
FR Germany
smolka@ds0lilog.bitnet

Wayne SNYDER
Boston University, Dept. of Comp. Sci.
111 Cummington Street
Boston, MA 02215
USA
snyder@bu-cs.bu.edu

Michael TEPP
Univ. Kaiserslautern, FB Informatik
Postfach 3049
6750 Kaiserslautern
FR Germany
tepp@uklirb.uucp

John K. TRUSS
The University of Leeds
Pure Mathematics
Leeds LS2 9JT
England
jkt@dcs.leeds.ac.uk

Uwe WALDMANN
Lehrstuhl Informatik V
Univ. Dortmund
Postfach 500500
D-4600 Dortmund
FR Germany

Phil WATSON
Royal Holloway & Bedford New College
Egham Hill
Egham, Surrey, TW20 0EX
England
pw@uk.ac.rhbnc.cs.ux