Solving disequations

Claude Kirchner, Pierre Lescanne

INRIA

UNITÉ DE RECHERCHE
INRIA-LORRAINE

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

N° 686

SOLVING DISEQUATIONS

Claude KIRCHNER
Pierre LESCANNE

JUIN 1987

SOLVING DISEQUATIONS

RESOUDRE DES DISEQUATIONS

PAR

Claude KIRCHNER

ET

Pierre LESCANNE

ABSTRACT :

We present a general study of equations (objects of form s=t and disequations (objects of form $s \neq t$) solving. The problem is approached from its fully general mathematical definition clearly separating universally and existentially quantified variables. In addition it is showed to have many connections with unification in equational theories like associativity commutativity, in particular methods similar to those used to solve equational unification problem works in solving disequations. This abstract framework is then applied to study the sufficient completeness of a rewrite rule based definition of a function.

RESUME :

Nous présentons une étude générale des équations (objets de la forme s=t) et disequations (objets de la forme $s \neq t$). Le problème est présenté dans un cadre mathématique très général en séparant clairement les variables quantifiées universellement et existentiellement. De plus, de nombreux liens avec l'unification dans les théories équationnelles notamment associative et commutative sont mis en évidence. En particulier des méthodes similaires à celle utilisées pour résoudre les problèmes d'unification équationnelle sont facilement adaptables à la résolution de diséquations. Ce formalisme est appliqué à l'étude de la complétude suffisante d'une définition de fonction fondé sur un ensemble de règles de réécriture.

SOLVING DISEQUATIONS

Claude Kirchner Pierre Lescanne

Centre de Recherche en Informatique de Nancy,
CNRS UA 262 and INRIA-Lorraine,
Campus Scientifique, BP 239,
54506 VANDŒUVRE-LES-NANCY, FRANCE
E-mail: mcvax!inria!crin!(ckirchner lescanne)

March 30, 1987

Abstract

We present a general study of equations (objects of form $s = t$ and disequations (objects of form $s \neq t$) solving. The problem is approached from its fully general mathematical definition clearly separating universally and existentially quantified variables. In addition it is showed to have many connections with unification in equational theories like associativity commutativity, in particular methods similar to those used to solve equational unification problem works in solving disequations. This abstract framework is then applied to study the sufficient completeness of a rewrite rule based definition of a function.

1 The Problem

Solving equations in the algebra of terms is well-known in the computer science community as unification. In this paper, we study the dual problem of solving disequations which is finding solutions to problems of the form $s \neq t$. As we will see this cannot be dissociated from solving equations, thus we study the general question of computing solutions for a combination of equations and disequations. This problem has many connections with the now classical one of unification in theories, like unification in associative and commutative theories. First, the basis of the set of solutions which is a set of substitutions contains usually more than one element. Second, methods proposed for unification, namely decomposition , merging and mutation [12].

A first explicit reference to this problem is due to A. Colmerauer [3,2]. He indeed introduces disequations in Prolog II and proposes algorithms to solve them. But he studies solutions in infinite trees, as our method solves a combination of equations and disequations in the empty theory, i.e., in the finite terms algebra without axioms. Moreover, in some problems we are looking for solutions in the initial algebra only, i.e., the theory of ground terms. As another fundamental difference, we solve disequations containing free variables and Bürckert [1] has shown in the case of unification this is a deeply different problem, same thing happens here. Mention to this problem exists also in [14].

In addition to the theorical interest of solving disequation, a main motivation of this work was to propose tools for proving the sufficient completeness of an algebraic specification [4] or for checking inductive reducibility [11]. The sufficient completeness is the fact that the definition of a function by a set of rewrite rules is complete in the sense that this function is defined on any possible pattern. This is useful for checking the correctness of a specification. We show that our framework for solving a combination of equations and disequations applies precisely to check sufficient completeness and, in the case of an incomplete definition, to give the set of patterns where the function is undefined. The inductive reducibility is the reducibility of all the ground instances of a term. It plays an important

*This research was partly supported by the GRECO of programmation (CNRS).

role in the so-called inductionless induction method. In the next section, we define the kind of problem we address in this work : equational problems. Section 3 deals with the rules we are using in order to decompose an equational problem into a simpler one. In Section 4, the main result is stated : the rules of Section 3 are terminating and the normal forms of an equational problem is a equational problems in solved form. In Section 5, we show how to get rid of disequalities when they are solved in the initial algebra. Section 6 shows how the problems of sufficient completeness and inductive reducibility can be reduced to our notion of equational problems. Examples of applications are given in Section 7 and we conclude in Section 8.

2 Equational Problems

In the remainder of this work, we assume the reader familiar with equational logic and equational unification. References can be found respectively in [10] and [13,12]

A **system** is a conjunction of pairs of terms written $s = t$ and called **equations** or written $s \neq t$ and called **disequations**. Note that a disequation is not an *inequation* (usualy denoted $x < y$ in arithmetic or in algebraic semantics [6]) and that we intentionaly choose a different name, following a terminology proposed by J.A.Goguen and J.Meseguer [7]. An **equational problem** P in the unknowns $y_1, \ldots, y_n$ is a **disjunction of systems**. We denote by $Evar(P)$, the set of Existentially quantified variables $\{y_1, y_2, \ldots, y_n\}$. Outside the unknowns, an equational problem P contains a set $Uvar(P)$ of Universally quantified variables. Actually the problem should be written,

$$\exists y_1, y_2, ..., y_n \in T(F,X), \forall x_1, x_2, ..., x_m \in T(F,X) : \\ P(y_1, y_2, ..., y_n, x_1, x_2, ..., x_m)$$

In order to make clearer the difference between the two kinds of variables we use x's for universally quantified variables and y's for existentially quantified variables or unknowns.

A classical unification problem is of the form:

$$\exists y_1, \ldots, y_n,\ t(y_1, y_2, \ldots, y_n) = t'(y_1, y_2, \ldots, y_n),$$

and recently Bückert [1] has studied the notion of *restricted unification*, that is a problem of the form (note the quantification order):

$$\forall x_1, \ldots, x_m, \exists y_1, \ldots, y_n \\ t(y_1, \ldots, y_n, x_1, \ldots, x_m) = t'(y_1, \ldots, y_n, x_1, \ldots, x_m),$$

He shows that the universally quantified variables cannot be equivalently replaced by new constants. In particular, if one knows a unification algorithm for a theory T, this does not imply necessarily the existence of a matching algorithm for T.

Actually, only the existentially quantified variables can be instantiated and one says that a substitution σ, s.t. $D(\sigma) \subseteq \{y_1, y_2, \ldots, y_n\}$, is a **solution** of a problem $P = S_1 \vee \ldots \vee S_p$, if there exists a system S_i such that, for all equations $s = t$ in S_i, $\sigma(s) = \sigma(t)$ and, for all disequations $s \neq t$ in S_i, $\sigma(s) \neq \sigma(t)$.

3 Rules

We are now going to describe a set of rules for solving a mixture of equations and disequations. By solving we mean to reduce an equational problem to a simpler form, from which the set of solutions can be easily deduced. So the purpose of these rules is to simplify an equational problem in such a way that the set of solutions is preserved. For a precise inside into the problem of equivalence of equational problems see [12].

For readibility, the set of rules is divided into different classes, one of them deals with **decomposition** and **clash**. There are two types of decomposition and clash according to the kind of pairs which is decomposed. The decomposition of an equation or $=$-decomposition is the same as the

decomposition used in unification, it transforms an equation into a conjunction of equations. The decomposition of a disequation or $\neq$-decomposition transforms a disequation into a disjunction of conjunctions of disequations. An operation called **sliding** transforms a pair of an equation of the form $y = s$ and a disequation of the form $y \neq t$ into the equation $y = s$ and the disequation $s \neq t$. The rules are presented as a set of premisses and a conclusion. Both part are separated by an arrow $\mapsto$. An inference rule presentation is also possible.

Warning: The x's are quantified variables, the y's are the unknowns of the problem and, the ξ's are indifferently universally and existentially quantified variables. To emphasize this convention, universal quantifiers are explicitely written.

3.1 Basic rules

The disequation $\xi \neq \xi$ is trivially never satisfied:

$$\{\xi \neq \xi\} \quad \mapsto \quad false \tag{1}$$

Similarly the equation $\xi = \xi$ is always satisfied:

$$\{\xi = \xi\} \quad \mapsto \quad true \tag{2}$$

3.2 Anteposition

In what follows we transform each equation of the form $s = \xi$ into an equation of the form $\xi = s$. The same thing is done for disequations. Colmerauer has called this rule anteposition.

$$s = \xi \quad \mapsto \quad \xi = s \text{ if } s \text{ is not a variable} \tag{3}$$

The same rule for disequation is also interesting,

$$s \neq \xi \quad \mapsto \quad \xi \neq s \text{ if } s \text{ is not a variable} \tag{4}$$

3.3 Quantifiers rules

The following rules are classical and useful:

$$\forall x(P \vee Q) \quad \mapsto \quad (\forall x P) \vee Q \text{ if } x \notin Var(Q) \tag{5}$$
$$\forall x(Q \wedge R) \quad \mapsto \quad (\forall x Q \wedge \forall x R) \tag{6}$$

3.4 Occur checks

We define first an ordering on the equations of a system S [9,16] by $(\xi = t) \prec_S (\xi' = t')$ if and only if t and t' are non variable terms such that $\xi \in Var(t')$, $(\xi = t) \in S$ and $(\xi' = t') \in S$.

If there is a cycle in the system S for the transitive closure $\prec_S^+$, then S is never satisfied:

$$S \quad \mapsto \quad false \text{ if } S \ni (\xi = s) \text{ s.t. } (\xi = s) \prec_S^+ (\xi = s) \tag{7}$$

Something similar can happen with equations and disequations. If $\xi \in Var(s)$ and $\xi \neq s$ or there is a path in the system for the transitive and reflexive closure $\prec_S^+$ that can be prolonged into a cycle by the disequation working like an equation, then $\xi \neq s$ is always satisfied:

$$S \wedge \xi \neq s \quad \mapsto \quad S \text{ if } C \tag{8}$$

where C is the following condition: $\xi \in Var(s)$ and s is not a variable or there exists $(\xi' = s') \prec_S^+ (\xi'' = s'')$, with $\xi'' \in Var(s)$ and $\xi \in Var(s')$.

3.5 Clashes and Decompositions

If the top operators are different in a disequation, then this disequation is always satisfied.
[≠-clash]

$$f(u_1, ..., u_m) \neq g(v_1, ..., v_m) \quad \mapsto \quad true \text{ if } f \neq g \tag{9}$$

If the the top operators are different in an equation, then this equation is never satisfied.
[=-clash]

$$f(u_1, ..., u_m) = g(v_1, ..., v_m) \quad \mapsto \quad false \text{ if } f \neq g \tag{10}$$

If the top operators are similar in a disequation, this disequation can be transformed into a disjunction of disequations.
[≠-decomposition]

$$f(u_1, ..., u_m) \neq f(v_1, ..., v_m) \quad \mapsto \quad \bigvee_{1 \leq i \leq m} u_i \neq v_i \tag{11}$$

If the top operators are similar in an equation, this equation can be transformed in a conjunction of equations.
[=-decomposition]

$$f(u_1, ..., u_m) = f(v_1, ..., v_m) \quad \mapsto \quad \bigwedge_{1 \leq i \leq m} (u_i = v_i) \tag{12}$$

By using the basic rule and these rules, one gets $s = s \stackrel{*}{\mapsto} true$ and $s \neq s \stackrel{*}{\mapsto} false$.

3.6 Sliding

If $\xi = s$ and $\xi \neq t$, obviously $s \neq t$, similary if $\xi = s$ and $\xi = t$, then $s = t$. So doing we may continue to solve the problem by a clash or a decomposition. This corresponds to the merging in [12].

$$\xi = s \wedge \xi \neq t \quad \mapsto \quad \xi = s \wedge s \neq t \text{ if } s, t \text{ are not } \xi \tag{13}$$

For instance, using this rule and $s \neq s \stackrel{*}{\mapsto} false$ one gets $\xi = s \wedge \xi \neq s \stackrel{*}{\mapsto} false$.

$$\xi = s \wedge \xi = t \quad \mapsto \quad \xi = s \wedge s = t \text{ if } s, t \text{ are not } \xi \tag{14}$$

For instance, using this rule and $s = s \stackrel{*}{\mapsto} true$ one gets $\xi = s \wedge \xi = s \stackrel{*}{\mapsto} \xi = s$.

3.7 Boolean rules

The following rules are classical when dealing with boolean expressions. We give only some of them. In addition, it is assumed that the operators $\vee$ and $\wedge$ are associative and commutative.

$p \wedge false \mapsto p$	
$p \wedge p \mapsto p$	*Idempotence*
$q \vee (s \wedge q) \mapsto q$	*Absorption*
$p \vee (s \wedge q) \mapsto (p \wedge s) \vee (p \wedge q)$	*Distributivity*
$\ldots \mapsto \ldots$	

3.8 Substitution.

Consider the formula $\forall x((P \wedge \{x \neq s\}) \vee Q(x))$ where x does not occur in s. We have the following sequence of equivalences: (note that $\forall x\{x \neq s\}$ is always false if $x \notin Var(s)$)

$$\begin{array}{ll} & \forall x((P \wedge \{x \neq s\}) \vee Q(x)) \\ \equiv & \forall x[(P \vee Q(x)) \wedge (x \neq s \vee Q(x))] \\ \equiv & \forall x(P \vee Q(x)) \wedge \forall x(x \neq s \vee Q(x)) \\ \equiv & \forall x(P \vee Q(x)) \wedge Q(s) \\ \equiv & \forall x[(P \wedge Q(s)) \vee (Q(x) \wedge Q(s))] \end{array}$$

Therefore we state the following rule,

$$\begin{array}{c} \forall x((P \wedge \{x \neq s\}) \vee Q(x)) \mapsto \\ \forall x[(P \wedge Q(s)) \vee (Q(x) \wedge Q(s))] \text{ if } x \notin Var(s) \end{array}$$

The goal of this rule is to get rid of disequations of the form $x \neq s$. It has specific variants when P or Q is empty.

$$\begin{array}{lll} \forall x(\{x \neq s\} \vee Q(x)) & \mapsto & Q(s) \text{ if } x \notin Var(s) \\ \forall x\{x \neq s\} & \mapsto & false \text{ if } x \notin Var(s) \end{array}$$

Example: we may apply a previous rule to derive $s \neq t$ from $\forall x(x \neq s \vee x \neq t)$, when $x \notin Var(s)$. Indeed in this case, P is empty and Q is $x \neq t$.

4 Problems in Normal Form

A problem is said to be in normal form, if all the systems it contains are made of disequations of the form $y \neq s$ and of equations of the form $y = s$. In addition, if there is an equation of the form $y = s$, there is no equation of the form $y = t$ or disequation of the form $y \neq u$. Problems in normal form determine the solutions of the problem by values to attribute to existential variables or sets of values to forbid to these variables. Disequations of the form $y \neq s$ are called *constraints* by H.Comon. The previous rules are powerful enough to allow reducing each problem into a problem in normal form as stated by the following theorem.

Theorem 1 *Given an equational problem P, there exists always an equational problem P' in normal form such that*

$$P \overset{*}{\mapsto} P'.$$

Proof: (sketch) Equations and disequations of the from $s = t$ or $s \neq t$ disappear by decompositions or clashes. Equations of the form $x = s$ disappear by sliding or by a basic rule, if they occur with equations or disequations with the same left-hand side. Disequations of the form $x \neq s$ disappear by substitution. If there are two equations $y = s$ and $y = t$, one of them disappear by sliding, the same if the second is $x \neq t$ instead. Equations or disequations of the form $s = \xi$ or $s \neq \xi$ are transformed by anteposition. These processes produce always an expression which is smaller w.r.t. the following well-founded ordering,

$$\begin{array}{c} P \gg P' \\ \Leftrightarrow \\ (\phi(P) > \phi(P')) \text{ or } (\phi(P) = \phi(P') \text{ and } P >_{rpo} P') \end{array}$$

where $\phi(P)$ is the number of variables ξ in position $\xi = s$ or $\xi \neq s$ plus twice the number of variables in position $s = \xi$ or $s \neq \xi$ and $>_{rpo}$ is the recursive path ordering [5] described by the precedence

$$\{=, \neq\} > \wedge > \vee > \{true, false\}$$

$\diamond$

Note that the normalization proposed in the previous proof does not use the occur check and that the following example issued from [3] is in normal form since the substitution rule does not apply:

$$\exists y_1, y_2, (y_1 = f(y_1)) \wedge (y_2 = f(f(y_2))) \wedge (y_1 \neq y_2).$$

The following theorem says that restricted to problems that contain only equations, the rules of the previous section provide a unification algorithm .

Theorem 2 *If P contains only equalities, then P' contains only equalities. If P contains only disequalities, then P' contains only disequalities.*

5 Getting rid of disequalities

The following rule is really important since is allows getting rid of disequalities in some cases and to transform them into equalities. Usually this fully determines the substitutions that are solutions. This rule differs from the others in the sense that it refers to a specific model, namely the initial algebra. This means that when one uses it, solutions are not anymore true in any algebra, but only in the initial algebra $T(C)$ where C is the set of constructors or generators of the initial algebra. This also means that the quantifications, essentially the universal quantifications, are taken over the initial algebra. Another interesting aspect of this rule is that it introduces new existential variables. This fact is not very surprising, since a similar property appears for example in associative commutative unification. The rule means that if y does not match some patterns of the form $f_i(...)$ for $i \in I$, there exists a pattern of the form $f_j(...)$ for $i \in C - I$, that y has to match.
[Disequations to equations]

$$\begin{array}{c} \exists y'_1, \ldots, \exists y'_m, \forall x'_1, \ldots, \forall x'_n \\ (R \vee \forall x_{1,1}, \ldots, \forall x_{i,k} \in T(C)(Q \wedge \bigwedge_{i \in I} y \neq f_i(\underline{x}_i))) \\ \mapsto \\ \exists y'_1, \ldots, \exists y'_m, \exists y_{1,1}, \ldots, \exists y_{j,h}, \forall x'_1, \ldots, \forall x'_n \\ (R \vee (Q \wedge \bigvee_{j \notin C-I} y = f_j(\underline{y}_j))) \end{array}$$

A more general form of this rule could be given using the complementary, as described in [15].

6 Disequations, Sufficient Completeness, Inductive Reducibility

H.Comon [4] has shown how disequations can be used to solve the problem of sufficient completeness [8] and recently Jouannaud and Kounalis [11] introduced the concept of inductive reducibility.

A term is said to be **inductively reducible** by a term rewriting system R if all its ground instances are reducible. If R terminates the **sufficient completeness** of the definition of an operator f is just the inductive reducibility by normalized ground instances of the term $f(x_1, \ldots, x_n)$. So we introduce the following property equivalent to inductive reducibility: a term t is **inductively internally reducible** by a term rewriting system R if all its ground instances by normalized substitutions are reducible.

Let t be a term and $Sub(t)$ the set of its subterms. If $T^N(C)$ is the set of normalized ground terms, inductive internal reducibility of t can be written:

$$\begin{array}{l} \forall y_{1,1} \in T^N(C) \ldots \forall y_{j,h} \in T^N(C) \\ \exists x_{1,1} \in T(C) \ldots \exists x_{i,k} \in T(C) \ldots \\ \bigvee_{s \in Sub(t)} \bigvee_{(l \to g \in R)} s(\underline{y}_j) = l(\underline{x}_i) \end{array}$$

Indeed, the internal reducibility means there exists a non variable subterm (disjunction on the subterms) that matches (existence of a values $x_{i,k} \in T(C)$) the left-hand-side of a rule (disjunction on the rules). The inductive internal reducibility means that this has to be satisfied for each ground

instance by normalised substitutions (universal quantification over $y_{j,h} \in T^N(C)$). This is not a unification problem because of the quantifiers: in a unification the existential quantifier $\exists$ is in first position. Its negation is,

$$\begin{array}{l}\exists y_{1,1} \in T^N(C) \ldots \exists y_{j,h} \in T^N(C) \\ \forall x_{1,1} \in T(C) \ldots \forall x_{i,k} \in T(C) \ldots \\ \bigwedge_{s \in Sub(t)} \bigwedge_{(l \to g \in R)} s(\underline{y_j}) \neq l(\underline{x_i})\end{array}$$

This has now a flavor of unification, since the existential quantifier is in first position. It can be decomposed into a two steps process: (1) find the solutions (or better, a representation of the set of solutions) of the problem above where $\exists y_{1,1} \in T^N(C) \ldots \exists y_{j,h} \in T^N(C)$ has been replaced by $\exists y_{1,1} \in T(C) \ldots \exists y_{j,h} \in T(C)$ and then (2) check that the solutions are in normal form. This last check is of course trivial if there is no solution.

This states the inductive internal reducibility partly as an equational problem without providing a specific strategy to get the solution. A brute force strategy is then provided by the application of the inference rules, but it can be very inefficient because of the size of the equational problem to solve. This could be a starting point for the study of efficient algorithm testing the inductive reducibility.

7 Examples

In these examples we are not going to write the existential quantifiers systematically

Example 1

Let us first solve a problem posed by H.Comon [4]. This is a problem in y and z,

$$\forall x_1, \forall x_2 \; a(x_1, x_1) \neq a(y, z) \wedge a(f(x_1, x_2), x_1) \neq a(y, z).$$

By decomposing the first disequation we get,

$$\begin{array}{rl}\forall x_1, \forall x_2 & [x_1 \neq y \wedge a(f(x_1, x_2), x_1) \neq a(y, z))] \\ \vee & [x_1 \neq z \wedge a(f(x_1, x_2), x_1) \neq a(y, z))]\end{array}$$

By decomposing the second disequation we get,

$$\begin{array}{rl}\forall x_1, \forall x_2 & [x_1 \neq y \wedge f(x_1, x_2) \neq y] \\ \vee & [x_1 \neq y \wedge x_1 \neq z] \\ \vee & [x_1 \neq z \wedge f(x_1, x_2) \neq y] \\ \vee & [x_1 \neq z \wedge x_1 \neq z]\end{array}$$

which simplifies to

$$\forall x_1, \forall x_2 \; [x_1 \neq y \wedge f(x_1, x_2) \neq y] \vee x_1 \neq z.$$

By using the substitution rule, this gives,

$$\begin{array}{l}[z \neq y \wedge \forall x_2 f(z, x_2) \neq y] \\ \quad \vee \forall x_1, \forall x_2 [x_1 \neq y \wedge f(x_1, x_2) \neq y],\end{array}$$

and using the second quantifier rule and the last basic rule,

$$(z \neq y) \wedge (\forall x_2, f(z, x_2) \neq y).$$

Example 2

Let us solve a problem of sufficient completeness. Suppose one wants to prove that the definition

$$\{ \quad eq(0,s(x_1))=false \quad eq(s(x_2),0)=false \quad eq(s(x_3),s(x_4))=eq(x_3,x_4) \quad eq(x_5,x_5)=true \quad \}$$

is sufficiently complete. This can be stated as proving,

$$\begin{array}{l} \forall y,z \in T(\{s,0\})\ \exists x_1,x_2,x_3,x_4,x_5 \in T(\{s,0\}) \\ \quad eq(0,s(x_1)) = eq(y,z) \\ \vee \quad eq(s(x_2),0) = eq(y,z) \\ \vee \quad eq(s(x_3),s(x_4)) = eq(y,z) \\ \vee \quad eq(x_5,x_5) = eq(y,z). \end{array}$$

whose negation is,

$$\begin{array}{l} \exists y,z \in T(\{s,0\})\ \forall x_1,x_2,x_3,x_4,x_5 \in T(\{s,0\}) \\ \quad eq(0,s(x_1)) \neq eq(y,z) \\ \wedge \quad eq(s(x_2),0) \neq eq(y,z) \\ \wedge \quad eq(s(x_3),s(x_4)) \neq eq(y,z) \\ \wedge \quad eq(x_5,x_5) \neq eq(y,z). \end{array}$$

This leads to look for a possible assignment of values to y and z in the following problem.

$$\begin{array}{l} \forall x_1,x_2,x_3,x_4,x_5 \in T(\{s,0\}) \\ \quad eq(0,s(x_1)) \neq eq(y,z) \\ \wedge \quad eq(s(x_2),0) \neq eq(y,z) \\ \wedge \quad eq(s(x_3),s(x_4)) \neq eq(y,z) \\ \wedge \quad eq(x_5,x_5) \neq eq(y,z). \end{array}$$

The definition of eq is sufficiently complete if there is no such assignment, which means the negation of the sufficient completeness problem is identically *false*. This demonstrates the important difference between *free variables* and *unknows* in an equational problem. In the following we assume quantification are taken over $T(\{s,0\})$. By decompositions the initial equational problem is equivalent to

$$\begin{array}{c} \forall x_1,x_2,x_3,x_4,x_5, \\ (0 \neq y \vee s(x_1) \neq z) \wedge \\ (s(x_2) \neq y \vee 0 \neq z) \wedge \\ (s(x_3) \neq y \vee s(x_4) \neq z) \wedge \\ (x_5 \neq y \vee x_5 \neq z). \end{array}$$

Using first the quantifier rules one gets

$$\begin{array}{l} (0 \neq y \vee \forall x_1 s(x_1) \neq z) \wedge \\ (\forall x_2 s(x_2) \neq y \vee 0 \neq z) \wedge \\ (\forall x_3 s(x_3) \neq y \vee \forall x_4 s(x_4) \neq z) \wedge \\ (\forall x_5 x_5 \neq y \vee x_5 \neq z). \end{array}$$

By the rule called *disequations to equations*, $\forall x_1 s(x_1) \neq z$ and $\forall\ x_4\ s(x_4)\ \neq\ z$ are $0 = z$, $\forall x_2 s(x_2) \neq y$ and $\forall\ x_3\ s(x_3)\ \neq\ y$ are $0 =\ y$, and $\forall x_5 (x_5 \neq y \vee x_5 \neq z)$ is $y \neq z$. So we get,

$$\begin{array}{l} (0 \neq y \vee 0 = z) \wedge (0 = y \vee 0 \neq z) \wedge \\ (0 = y \vee 0 = z) \wedge y \neq z \end{array}$$

which normalizes to the empty disjunction.

Example 3

Let us show now that the resolution of an equational problem can also be used to look for the patterns in which a given function is not defined. If there is no such patterns, the function is

completely defined or sufficiently complete. Let us go back to the function eq and suppose we look at the (actually incomplete) definition,

$$\{eq(0, s(x_1)) = false, eq(s(x_2), 0) = false, eq(x_5, x_5) = true\}.$$

Like previously, this can be stated as solving,

$$\exists y, z \forall x_1, x_2, x_5 (eq(0, s(x_1)) \neq eq(y, z) \wedge \\ eq(s(x_2), 0) \neq eq(y, z) \wedge eq(x_5, x_5) \neq eq(y, z)).$$

Let us drop the $\exists$ and apply decompositions first,

$$\forall x_1, x_2, x_5 (0 \neq y \vee s(x_1) \neq z) \wedge \\ (s(x_2) \neq y \vee 0 \neq z) \wedge (x_5 \neq y \vee x_5 \neq z).$$

By applying the rule on quantifiers one gets,

$$(0 \neq y \vee \forall x_1 s(x_1) \neq z) \wedge \\ (0 \neq z \vee \forall x_2 s(x_2) \neq y) \wedge \\ (\forall x_5 (x_5 \neq y \vee x_5 \neq z)),$$

by the *disequation to equation* rule,

$$(0 \neq y \vee 0 = z) \wedge (0 = y \vee 0 \neq z) \wedge \\ \forall x_5 (x_5 \neq y \vee x_5 \neq z).$$

Using the result of the example given after the substitution rule,

$$(0 \neq y \vee 0 = z) \wedge (0 = y \vee 0 \neq z) \wedge (y \neq z) \\ \stackrel{*}{\mapsto} (0 \neq y \wedge 0 \neq z \wedge y \neq z) \vee (0 = y \wedge 0 = z \wedge y \neq z)$$

Systems of the form $(0 = y \wedge 0 \neq y \wedge y \neq z)$ disappear by sliding. And by sliding on the last expression, one gets,

$$(0 \neq y \wedge 0 \neq z \wedge y \neq z) \vee (0 = y \wedge 0 = z \wedge 0 \neq z) \\ \mapsto (0 \neq y \wedge 0 \neq z \wedge y \neq z)$$

By *disequation to equation* (with the convention on existential quantifiers),

$$y = s(x_1) \wedge y = (s_2) \wedge s(y_1) \neq s(y_2) \\ \stackrel{*}{\mapsto} y = s(x_1) \wedge y = (s_2) \wedge y_1 \neq y_2$$

This means the operator eq is not sufficiently complete and has to be completed on patterns of the form $y = s(y_1)$ and $y = s(y_2)$ with $y_1 \neq y_2$.

8 Conclusion

Attempts to address the problem of solving inequations in term algebra are not new since we cited already A. Colmerauer [3] and H. Comon [4]. However our approach is original in two respects, we make a clear distinction between roles plaid by variables (existential and universal), and we exhibit a set of inference rules that highlight the basic operations to perform in order to solve disequations. In some sense, this improves the readibility as Martelli and Montanari [16] or G.Huet [9] did for unification or C.Kirchner [12] for equational unification. Our hope is that this axiomatic approach will provide tools for proposing a simple algorithm for solving the decision of inductive reducibility. Next this very general framework has to be extended to equational theories.

Acknowledgements: We are very grateful to Hubert Comon and Jean Pierre Jouannaüd for many constructive discussions and remarks on early versions of this paper.

References

[1] H. Bürckert. Some relationships between unification, restricted unification and matching. In J. Siekmann, editor, *Proceedings 8th Conference on Automated Deduction, Oxford*, pages 514–524, Springer Verlag, Oxford (England), 1986.

[2] A. Colmerauer. Equations and inequations on finite and infinite trees. In *FGCS'84Proceedings*, pages 85–99, November 1984.

[3] A. Colmerauer. *PROLOG II, Manuel de Référence et Modèle théorique.* Technical Report, GIA, Université Aix-Marseille II, 1982.

[4] H. Comon. Sufficient completeness, term rewriting system and anti-unification. In J. Siekmann, editor, *Proceedings 8th Conference on Automated Deduction, Oxford*, pages 128–140, Springer Verlag, Oxford (England), 1986.

[5] N. Dershowitz. Termination. In *Proceedings 1st Conference on Rewriting Techniques and Applications*, pages 180–224, Springer Verlag, Dijon (France), May 1985.

[6] J. Gallier. The semantics of recursive programs with functions parameters of finite types: n-rational algebras and logic of inequalities. In M. Nivat and Reynolds J.C., editors, *Algebraic Methods in Semantics*, Cambridge University Press, 1985.

[7] J.A. Goguen and J. Meseguer. EQLOG: equality, types and generic modules for logic programming. In D. DeGroot and G. Lindstrom, editors, *Logic Programming. Funstions, relations and equations*, Prentice Hall, 1986.

[8] J.V. Guttag and J.J. Horning. The algebraic specification of abstract data types. *Acta Informatica*, 10:27–52, 1978.

[9] G. Huet. Resolution d'equations dans les langages d'ordre 1,2, ...,ω. Thèse d'état de l'Université de Paris 7, 1976.

[10] G. Huet and D. Oppen. Equations and rewrite rules: a survey. In R. Book, editor, *Formal Language Theory: Perspectives and Open Problems*, Academic Press, 1980.

[11] J.P. Jouannaud and E. Kounalis. Proof by induction in equational theories without constructors. In *Proceeding of the first symposium on Logic In Computer Science, Boston (USA)*, pages 358–366, 1986.

[12] C. Kirchner. Computing unification algorithms. In *Proceeding of the first symposium on Logic In Computer Science, Boston (USA)*, pages 206–216, 1986.

[13] C. Kirchner. Méthodes et outils de conception systématique d'algorithmes d'unification dans les théories équationnelles. Thèse d'état de l'Université de NancyI, 1985.

[14] J-L. Lassez, M.J. Maher, and K. Marriot. *Unification Revisited.* Technical Report, IBM Thomas J. Watson Research Center, 1986.

[15] A. Lazrek, P. Lescanne, and J-J. Thiel. *Proving inductive equalities, Algorithms and Implementation.* Technical Report, Internal Report CRIN 86-R-087, to be published in Information and Control, 1986.

[16] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions On Programming Languages And Systems*, 4(2):258–282, 1982.

Contents

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique