Avoiding Infinite Loops in the Solving of Equations Involving Sequence Variables and Terms with Flexible Arity Function Symbols

Jorge Coelho and Mário Florido

Technical Report Series: DCC-2005-1

Departamento de Ciência de Computadores – Faculdade de Ciências

&

Laboratório de Inteligência Artificial e Ciência de Computadores

Universidade do Porto

Rua do Campo Alegre, 823 4150 Porto, Portugal

Tel: +351+2+6078830 – Fax: +351+2+6003654

http://www.ncc.up.pt/fcup/DCC/Pubs/treports.html

Avoiding Infinite Loops in the Solving of Equations Involving Sequence Variables and Terms with Flexible Arity Function Symbols

Jorge Coelho and Mário Florido
DCC-FC & LIACC
University of Porto, Portugal
{jcoelho,amf}@ncc.up.pt

Abstract

Solving equations involving terms with variables that can be instantiated to sequences of terms (sequence variables) and terms with function symbols of arbitrary arity may lead to infinite loops. This is particularly relevant when the implementations traverse the solution tree using a depth-first strategy, because some solutions become unreachable when they appear after a branch that loops. In this paper we present a simple method for checking if a branch will lead to a loop. This makes it possible to go on with the unification algorithm without trying to explore those branches. The technique we use is based on an abstraction of the original unification by a Diophantine equation on the sizes of the terms involved. We present this abstraction and show that it is correct with respect to the original unification.

1 Introduction

Unification of terms with arbitrary arity and sequence variables has been applied in several domains, such as knowledge management [8, 10], databases [11, 22, 12], programming languages [24, 20] and XML processing [5].

This unification was shown to be decidable in [17] by reducing the original problem to an unification problem whose signature consists only of fixed arity function symbols and one flat arbitrary arity function symbol and then proving the decidability of the reduced problem.

More pragmatically Kutsia presented in [15] an elegant procedure that solves this kind of unification problems with the additional restriction that no variable can occur more than twice in an equation. This algorithm generates a tree of solutions for the equation using a breath-first strategy.

In [5] we presented a depth-first version of the previous algorithm. The breath-first algorithm terminates if the set of solutions to a given unification is finite. The same does not happen with the depth-first algorithm because in this case some solutions may never be computed because they lie on a branch after another branch that leads to a failure, but where the algorithm loops. The following example (due to Kutsia) shows this situation (where =*= stands to unification of terms with arbitrary function symbols and sequence variables):

Example 1.1 *Given the unification problem,*

$$f(X,\ f(a,Y)) = * = f(a,\ f(Y), f(a,Y))$$

the algorithm outputs the solution, $X = a$ *and then it will continue trying to unify* $f(f(a,Y))$ *with* $f(f(Y), f(a,Y))$*. This will lead to* $f(a,Y) = * = f(Y)$ *and it will result in an infinite loop. Note that this unification problem has a solution that is* $X = a, f(Y)$ *although the algorithm never reaches it.*

In [18] Kutsia mentions a decision algorithm based on Makanin algorithm [19] and Baader-Schultz combination method [1], that for any unification problem can tell whether this problem has a solution or not. With this method one could stop expanding a branch if the decision procedure tells us that the problem at the branch does not have a solution and then going to the next branch. However this procedure is not part of the implementation and it is not an easy task to implement it.

In this paper we present a procedure for checking if a branch has not a solution by abstracting the original unification problem by a Diophantine Equation on the sizes of sequences. This method is rather simple when compared with previous methods and it solves the previously presented problem of the *depth first* implementation.

Let us now show how we solve the problem in the previous example:

Example 1.2 *Given the sequences, $f(X,\, f(a, Y))$ and $f(a,\, f(Y),\, f(a, Y))$. The associated Diophantine equation is:*

$$X + Y + 2 = 2Y + 4 \Rightarrow X = Y + 2$$

Where X and Y stand for the lengths of sequences X and Y and the constants 2 and 4 to the number of constants and function symbols in the sequences $X, f(a, Y)$ and $a, f(Y), f(a, Y)$, respectively. The algorithm proceeds unifying $X = a$ and will continue trying to unify $f(f(a, Y))$ with $f(f(Y), f(a, Y))$. Here the associated equation is:

$$Y + 3 = 2Y + 4 \Rightarrow Y = -1$$

Thus, there are no solutions in the positive integers, meaning that there is not any sequence as solution for Y and this branch fails. After backtracking the algorithm will try $X = a, f(Y)$ and proceed with the unification of $f(f(a, Y))$ with $f(f(a, Y))$ that will obviously succeed. So, it was possible to output a solution previously unreachable.

Contribution: In this work we present a simple method for identifying non-terminating branches corresponding to failure in the tree of solutions for a unification problem with arbitrary arity function symbols and sequence variables. This is particularly relevant in depth-first implementation of the algorithm because it avoids not to reach solutions due to loops in branches corresponding to failure.

Related Work: In [21], Kutsia mentioned a decision algorithm that, for any unification problem could tell us whether the problem had a solution or not. This decision algorithm was not implemented and its implementation does not seem to be an easy task (it is a more theoretical method based on Makanin algorithm and Baader-Schulz combination method).

Diophantine equations played a key role in unification theory. In [16] Kutsia used Diophantine equations and/or inequalities as constraints in the representation of minimal complete set of unifiers in an equational theory with individual and sequence variables and patterns (patterns abbreviate sequences of terms of unknown length, where the terms match a certain “common pattern”).

Solving systems of linear Diophantine equations was applied before in many unification problems: unification modulo associativity [19], modulo associativity and commutativity [23, 13, 3] and modulo distributivity [6].

We assume that the reader is familiar with basic theory of unification [2] and knows the general notions related to unification of terms with arbitrary arity function symbols and sequence variables [15].

The paper is organized as follows: In section 2 we present terms with flexible arity symbols and sequence variables. In section 3 we present the unification algorithm and in section 4 we present our extension with Diophantine Equations on the length of the terms involved in the unification along with some examples of problems previously unsolvable.

2 Terms with Flexible Arity Symbols and Sequence Variables

In [15], Kutsia presented a breath-first algorithm for solving unification problems for terms with flexible arity symbols and sequence variables. In [5] we present a depth-first implementation of Kutsia algorithm, where, we restricted unification to terms containing only sequence variables (standard variables are not included). In this previous work, we extended a standard semantic domain of trees over uninterpreted functors with finite sequences of trees.

Some definitions and examples (taken from [5]) follow:

Definition 2.1 *A* sequence $\tilde{t}$, *is defined as follows:*

- *ε is the empty sequence.*
- *$t_1, \tilde{t}$ is a sequence if t_1 is a term and $\tilde{t}$ is a sequence*

Example 2.1 *Given the terms $f(a)$, b and X, then $\tilde{t} = f(a), b, X$ is a sequence.*

Equality is the only relation between trees. Equality between trees is defined in the standard way: two trees are equal if and only if their root functor are the same and their corresponding subtrees, if any, are equal.

Syntactically we consider an alphabet consisting of the following sets: the set of sequence variables (variables are denoted by upper case letters), the set of constants (denoted by lower case letters) and the set of flexible arity function symbols.

Definition 2.2 *The set of terms over the previous alphabet is the smallest set that satisfies the following conditions:*

1. *Constants and sequence variables are terms.*
2. *If f is a flexible arity function symbol and $t_1, \ldots, t_n$ ($n \geq 0$) are terms, then $f(t_1, \ldots, t_n)$ is a term.*

Terms of the form $f(t_1, \ldots, t_n)$ where f is a function symbol and $t_1, \ldots, t_n$ are terms are called *compound terms.*

Definition 2.3 *If t_1 and t_2 are terms then $t_1 = * = t_2$ (unification of terms with flexible arity symbols) is a constraint.*

A constraint $t_1 = * = t_2$ is solvable if and only if there is an assignment of sequences or ground terms, respectively, to variables therein such that the constraint evaluates to *true*, i.e. such that after that assignment the terms become equal.

Example 2.2 *Given the sequence variable X, $f(a, X, c, d)$ is a flexible arity term. X can be instantiated by a sequence of terms, leading for instance to the terms $f(a, a, a, c, d)$ or $f(a, c, d)$ (corresponding respectively to $X = a, a$ and $X = \varepsilon$).*

Example 2.3 *Accordingly to the definitions presented in [15], the minimal complete set of unifiers of the equation*

$$f(g(a, X), g(Y, c)) = * = f(U, g(b, V))$$

is: $\{\{U = g(a, X),\ Y = b,\ V = c\}, \{X = \varepsilon,\ U = g(a),\ Y = b, V = c\}, \{U = g(a, X),\ Y = b, Y,\ V = Y, c\},\ \{X = \varepsilon,\ U = g(a),\ Y = b, Y,\ V = Y, c\}\}$

2.1 Constraint Solving

Constraints of the form $t_1 = * = t_2$ are solved by a non-standard unification that calculates the corresponding minimal complete set of unifiers. This non-standard unification is basically a depth-first implementation of Kutsia algorithm [15] restricted to sequence variables. As motivation we present some examples of unification:

Example 2.4 *Given the terms $a(X, b, Y)$ and $a(a, b, b, b)$ where X and Y are sequence variables, $a(X, b, Y) = * = a(a, b, b, b)$ gives three results:*

1. *$X = a$ and $Y = b, b$*
2. *$X = a, b$ and $Y = b$*
3. *$X = a, b, b$ and $Y = \varepsilon$*

Example 2.5 *Given the terms $a(b,X)$ and $a(Y,d)$ where X and Y are sequence variables, $a(b,X) = * = a(Y,d)$ gives two possible solutions:*

1. *$X = d$ and $Y = b$*
2. *$X = N, d$ and $Y = b, N$ where N is a new sequence variable.*

Note that this non-standard unification is conservative with respect to standard unification: in the last example the first solution corresponds to the use of standard unification.

3 The Unification Algorithm

The unification algorithm, as presented in [15], consists of two main steps, *Projection* and *Transformation*. The first step, *Projection* is where some variables are erased from the sequence. This is needed to obtain solutions where those variables are instantiated by the empty sequence. The second step, *Transformation* is defined by a set of rules where the non-standard unification is translated to standard unification.

Definition 3.1 *Given terms T_1 and T_2, let V be the set of variables of T_1 and T_2 and A be a subset of V.* Projection *eliminates all variables of A in T_1 and T_2.*

Example 3.1 *Let $T_1 = f(b,Y,f(X))$ and $T_2 = f(X,f(b,Y))$. In the projection step we obtain the following cases (corresponding to $A = \{\}$, $A = \{X\}$, $A = \{Y\}$ and $A = \{X,Y\}$):*

- $T_1 = f(b,Y,f(X)), T_2 = f(X,f(b,Y))$
- $T_1 = f(b,Y,f), T_2 = f(f(b,Y))$
- $T_1 = f(b,f(X)), T_2 = f(X,f(b))$
- $T_1 = f(b,f), T_2 = f(f(b))$

Our version of Kutsia algorithm uses a special kind of terms, here called, *sequence terms* for representing sequences of arguments.

Definition 3.2 *A* sequence term*, $\bar{t}$ is defined as follows:*

- *ε is a* sequence term.
- *$seq(t,\bar{s})$ is a* sequence term *if t is a term and $\bar{s}$ is a* sequence term.

Definition 3.3 *A sequence term in* normal form *is defined as:*

- *ε is in normal form*
- *$seq(t_1,t_2)$ is in normal form if t_1 is not of the form $seq(t_3,t_4)$ and t_2 is in normal form.*

Example 3.2 *Given the function symbol f, the variable X and the constants a and b:*

$$seq(f(seq(a,\varepsilon)), seq(b, seq(X,\varepsilon)))$$

is a sequence term *in normal form.*

Note that sequence terms are lists and sequence terms in normal form are flat lists. We introduced this different notation because sequence terms are going to play a key role in our implementation of the algorithm and it is important to distinguish them from standard lists. Sequence terms in normal form correspond trivially to the definition of sequence presented in definition 2.1. In fact sequence terms in normal form are an implementation of this definition. Thus, in our implementation, a term $f(t_1,t_2,\ldots,t_n)$, where f has flexible arity, is internally represented as $f(seq(t_1,seq(t_2,\ldots,seq(t_n,\varepsilon)\ldots)))$, that is, arguments of functions of flexible arity are always represented as elements of a sequence term.

We now define a normalization function to reduce sequence terms to their normal form.

Definition 3.4 *Given the sequence terms* $\bar{t}_1$ *and* $\bar{t}_2$*, we define sequence term* concatenation *as* $\bar{t}_1 + +\bar{t}_2$*, where the ++ operator is defined as follows:*

$$\begin{array}{cccc} \varepsilon & ++ & \bar{t} & = & \bar{t} \\ seq(t_1,\bar{t_2}) & ++ & \bar{t_3} & = & seq(t_1,\bar{t_2}++\bar{t_3}) \end{array}$$

Definition 3.5 *Given a sequence term, we define sequence term* normalization *as:*

$$\begin{array}{lcl} normalize(\varepsilon) & = & \varepsilon \\ normalize(t) & = & seq(t,\varepsilon),\ \textit{if } t \textit{ is a constant or variable.} \\ normalize(t) & = & seq(f(normalize(t_1)),\varepsilon),\ \textit{if } t = f(t_1). \\ normalize(seq(t_1,\bar{t})) & = & normalize(t_1)\ ++\ normalize(\bar{t}) \end{array}$$

Proposition 3.1 *The normalization procedure always terminates yielding a sequence in normal form.*

Transformation rules are defined by the rewrite system presented in figure 1. We consider that upper case letters (X,Y,...) stand for sequence variables, lower case letters (s,t,...) for terms and overlined lower case letters ($\bar{t}, \bar{s}$) for *sequence terms.* These rules implement Kutsia algorithm applied to sequence terms by using standard unification. Note that rules 6, 7, 8 and 9 are non-deterministic: for example rule 6 states that in order to solve $seq(X, \bar{t}) = * = seq(s_1, \bar{s})$ we can solve $\bar{t} = * = \bar{s}$ with $X = s_1$ or we can solve $normalize(seq(X_1, \bar{t})) = * = normalize(\bar{s})$ with $X = seq(s_1, seq(X_1, \varepsilon))$. At the end the solutions given by the algorithm are normalized by the *normalize* function. When none of the rules is applicable the algorithm fails. Please note that in the rules in figure 1 standard unification $t = s$ propagates the resulting substitution to the remaining is constraints (similar to the behavior of = in Prolog). Kutsia showed in [15] that this algorithm terminated if it had a cycle check, (i.e. it stopped with failure if a unification problem gave rise to a similar unification problem) and if each sequence variable does not occur more than twice in a given unification problem.

For the sake of simplicity, from now on examples are presented in sequence notation, alternatively to the *sequence term* notation.

Example 3.3 *Given* $t = f(X, b, Y)$ *and* $s = f(c, c, b, b, b, b)$ *the projection step leads to the following transformation cases:*

- $f(X, b, Y) = * = f(c, c, b, b, b, b)$
- $f(b, Y) = * = f(c, c, b, b, b, b)$
- $f(X, b) = * = f(c, c, b, b, b, b)$
- $f(b) = * = f(c, c, b, b, b, b)$

Using the transformation rules we can see that only the first and third unifications succeed. For $f(X, b, Y) = * = f(c, c, b, b, b, b)$ *we have the following answer substitutions:*

- $X = c, c$ *and* $Y = b, b, b$
- $X = c, c, b$ *and* $Y = b, b$
- $X = c, c, b, b$ *and* $Y = b$

And for $f(X, b) = * = f(c, c, b, b, b, b)$ *we have:*

- $X = c, c, b, b, b$
- $Y = \varepsilon$

In [5] we prove the correctness of our implementation of Kutsia algorithm. In [15] Kutsia proved the correctness of his algorithm with respect to a given semantics for the non-standard unification. In [5] we show that our implementation of Kutsia algorithm is correct, i.e, both give the same set of solutions for a given equation.

[1]== denotes syntactic equality (in opposite with = which denotes standard unification)

Success

$$\begin{array}{llcll}
(1) & t & =*= & s & \Longrightarrow \text{True, if } t == s \,^{1} \\
(2) & X & =*= & t & \Longrightarrow X = t \text{ if } X \text{ does not occur in } t. \\
(3) & t & =*= & X & \Longrightarrow X = t \text{ if } X \text{ does not occur in } t.
\end{array}$$

Eliminate

$$\begin{array}{llcll}
(4) & f(\bar{t}) & =*= & f(\bar{s}) & \Longrightarrow \bar{t} =*= \bar{s} \\
(5) & seq(t_1, \bar{t}) & =*= & seq(s_1, \bar{s}) & \Longrightarrow t_1 =*= s_1, \\
 & & & & \quad\; normalize(\bar{t}) =*= normalize(\bar{s}) \\
(6) & seq(X, \bar{t}) & =*= & seq(s_1, \bar{s}) & \Longrightarrow X = s_1, \text{ if } X \text{ does not occur in } s_1, \\
 & & & & \quad\; normalize(\bar{t}) =*= normalize(\bar{s}) \\
 & & & & \Longrightarrow X = seq(s_1, seq(X_1, \varepsilon)), \\
 & & & & \quad\; \text{if } X \text{ does not occur in } s_1, \\
 & & & & \quad\; normalize(seq(X_1, \bar{t})) =*= normalize(\bar{s}), \\
 & & & & \quad\; \text{where } X_1 \text{ is a new variable.} \\
(7) & seq(t_1, \bar{t}) & =*= & seq(X, \bar{s}) & \Longrightarrow X = t_1, \text{ if } X \text{ does not occur in } t_1, \\
 & & & & \quad\; normalize(\bar{t}) =*= normalize(\bar{s}) \\
 & & & & \Longrightarrow X = seq(t_1, seq(X_1, \varepsilon)), \\
 & & & & \quad\; \text{if } X \text{ does not occur in } t_1, \\
 & & & & \quad\; normalize(\bar{t}) =*= normalize(seq(X_1, \bar{s})), \\
 & & & & \quad\; \text{where } X_1 \text{ is a new variable.} \\
(8) & seq(X, \bar{t}) & =*= & seq(Y, \bar{s}) & \Longrightarrow X = Y \\
 & & & & \quad\; \bar{t} =*= \bar{s}. \\
 & & & & \Longrightarrow X = seq(Y, seq(X_1, \varepsilon), \\
 & & & & \quad\; normalize(seq(X_1, \bar{t})) =*= normalize(\bar{s}), \\
 & & & & \quad\; \text{where } X_1 \text{ is a new variable and } X, Y \text{ are} \\
 & & & & \quad\; \text{distinct.} \\
 & & & & \Longrightarrow Y = seq(X, seq(Y_1, \varepsilon)), \\
 & & & & \quad\; normalize(\bar{t}) =*= normalize(seq(Y_1, \bar{s})), \\
 & & & & \quad\; \text{where } Y_1 \text{ is a new variable and } X, Y \text{ are} \\
 & & & & \quad\; \text{distinct.}
\end{array}$$

Split

$$\begin{array}{llcll}
(9) & seq(t_1, \bar{t}) & =*= & seq(s_1, \bar{s}) & \Longrightarrow \text{if } t_1 =*= s_1 \Longrightarrow r_1 =*= q_1 \text{ then} \\
 & & & & \quad\; normalize(seq(r_1, \bar{t})) =*= normalize(seq(q_1, \bar{s})) \\
 & & & & \quad\; \vdots \\
 & & & & \Longrightarrow \text{if } t_1 =*= s_1 \Longrightarrow r_w =*= q_w, \\
 & & & & \quad\; normalize(seq(r_w, \bar{t}_n)) =*= normalize(seq(q_w, \bar{s})), \\
 & & & & \quad\; \text{where } t_1 \text{ and } s_1 \text{ are compound terms.}
\end{array}$$

Figure 1: Transformation rules

4 Extension with Diophantine Equations on the length of the solutions

The algorithm presented in the previous section does not work for some unification problems. In fact it has the same limitation of Kutsia original algorithm concerning a maximum of two occurrences per variable in each constraint. In particular, when a variable occurs more than twice can lead to situations where the constraint has no solution and the algorithm loops (instead of failing). In this paper we present a solution for this problem. From now on the examples presented are related only with the Transformation rules, since the solution we propose is applied to these steps. Consider the following example:

Example 4.1 *Given the constraint $f(X,a,X) = * = f(a,X,a)$, the first solution is $X = a$ and then will try to unify $f(X_1,a,a,X_1)$ with $f(a,X_1,a)$, where $X = a, X_1$. After one step of the algorithm, this will lead to the unification of $f(a,a,a)$ with $f(a,a)$ where $X_1 = a$, which fails. Then the algorithm follows trying to unify $f(X_2,a,a,a,X_2)$ with $f(a,X_2,a)$, where $X_1 = a, X_2$. It is easy to see that this process will continue forever.*

Obviously the algorithm doesn't stop although the unification is never possible. Things get worse when the algorithm enters an infinite loop, before presenting all the solutions. Consider the next example:

Example 4.2 *Given the sequences, $f(X,Y,f(Z,a,Z))$ and $f(a,b,c,f(a,Z,a))$, the algorithm starts with $X = a$ and continues the unification with $f(Y,f(Z,a,Z)) = * = f(b,c,f(a,Z,a))$. This will fail for $Y = b$ but will go on with $Y = b, Y_1$ and the next unification is $f(Y_1,f(Z,a,Z)) = * = f(c,f(a,Z,a))$. Now $Y_1 = c$ and the algorithm tries the unification $f(f(Z,a,Z)) = * = f(f(a,Z,a))$. For $Z = a$, we have the solution, $X = a$, $Y = b, c$ and $Z = a$. Then the algorithm will backtrack to $Z = a, Z_1$ and $f(f(Z_1,a,a,Z_1)) = * = f(f(a,Z_1,a))$ and it enters an infinite loop. However, there is another solution, $X = a, b$, $Y = c$ and $Z = a$ which will never be computed.*

4.1 Diophantine equations on the length of the solutions

We solve the previous problem by maintaining constraints on the length of the terms. Whenever a new unification problem arises we solve a Diophantine equation based on it. The solutions of the equation represent the length of the solutions of the unification problem. Based on this values we can predict if a unification problem cannot be solved. We now show how to obtain the equations and how to use them.

Definition 4.1 *Given the sequence t, length(t) is defined as follows:*

$$\begin{array}{lcll} \mathit{length}(c) & = & 1 & \text{if } c \text{ is a constant} \\ \mathit{length}(X) & = & 1 & \text{if } X \text{ is a variable} \\ \mathit{length}(f(t_1,\ldots,t_n)) & = & 1+\mathit{length}(t_1)+\ldots+\mathit{length}(t_n) & \end{array}$$

Definition 4.2 *Given the sequence term or term t, $\mathcal{A}(t)$ translates t to a polynomial of degree 1 (an affine linear equation):*

$$\begin{array}{lcll} \mathcal{A}(\epsilon) & = & 0 & \\ \mathcal{A}(c) & = & 1 & \text{if } c \text{ is a constant} \\ \mathcal{A}(X) & = & X & \text{if } X \text{ is a variable} \\ \mathcal{A}(seq(t_1,\bar{t})) & = & \mathcal{A}(t_1)+\mathcal{A}(\bar{t}) & \\ \mathcal{A}(f(t_1,\ldots,t_n)) & = & 1+\mathcal{A}(t_1)+\ldots+\mathcal{A}(t_n) & \end{array}$$

Note that, given the unification problem $\bar{t} = * = \bar{s}$ a Diophantine equation on the length of the solutions results form applying $\mathcal{A}$ to both sides of the unification problem and equalizing them:

$$\mathcal{A}(\bar{t}) = \mathcal{A}(\bar{s})$$

We assume the existence of an algorithm $solve(\mathcal{A}(\bar{t}) = \mathcal{A}(\bar{s}))$ that returns true if $\mathcal{A}(\bar{t}) = \mathcal{A}(\bar{s})$ has solutions in $\mathbb{Z}^+$ and false otherwise. Examples of such algorithm for solving linear Diophantine equation can be found in [14, 4, 7, 9].

In figure 2 we present a new version of our algorithm, where we use the Diophantine equations solver in order to avoid loops. We now present some examples:

Example 4.3 *Given the sequences, $f(X, b, Y)$ and $f(a, b, b, b)$, from $\mathcal{A}(f(X, b, Y))$ and $\mathcal{A}(f(a, b, b, b))$ we have the following equation:*

$$X + Y + 2 = 5 \Rightarrow X + Y = 3$$

This means that the length of the solution of X added to the length of the solution of Y must be 3. And the possible solutions for this unification problem are (recall that we are dealing with transformation rules only):

- *X = a and Y = b,b*
- *X = a,b and Y = b*

Example 4.4 *Given the sequences, $f(X, b)$ and $f(a, Y)$ we have the following equation:*

$$X + 2 = Y + 2 \Rightarrow X = Y$$

This means that the length of the solution of X is the same than the solution of Y. And the possible solutions for this unification problem are:

- *X = a and Y = b*
- *X = a,N and Y = N,b where N is a new variable.*

Now, let's go back to the problematic examples presented before.

Example 4.5 *Given the sequences $f(X, a, X)$ and $f(a, X, a)$, the associated equation is:*

$$2X + 2 = X + 3 \Rightarrow X = 1$$

This means that the length of the solutions associated with X is 1. The algorithm proceeds with this unification for $X = a$ and succeeds. Now the algorithm tries $X = a, X_1$ and the resulting unification problem is $f(X_1, a, a, X_1) = * = f(a, X_1, a)$ and the associated equation:

$$2X_1 + 3 = X_1 + 3 \Rightarrow X_1 = 0$$

Since the equation doesn't have solutions in the set of positive integers, the algorithm does not proceed, thus avoiding an infinite loop.

Example 4.6 *Given the sequences, $f(X, Y, f(Z, a, Z))$ and $f(a, b, c, f(a, Z, a))$. The associated equation is:*

$$X + Y + 2Z + 3 = Z + 7 \Rightarrow X + Y + Z = 4$$

The algorithm proceeds with $X = a$ and continues the unification with $f(Y, f(Z, a, Z)) = * = f(b, c, f(a, Z, a))$ that has the equation:

$$Y + 2Z + 3 = Z + 6 \Rightarrow Y + Z = 3$$

This branch fails for $Y = b$ and the algorithm will go on with $Y = b, Y_1$. The next unification is $f(Y_1, f(Z, a, Z)) = * = f(c, f(a, Z, a))$ with the associated equation:

$$Y_1 + 2Z + 3 = Z + 5 \Rightarrow Y_1 + Z = 2$$

[2]== denotes syntactic equality (in opposite with = which denotes standard unification)

Success					
(1)	t	$=*=$	s	$\Longrightarrow$	True, if $t == s$ [2]
(2)	X	$=*=$	t	$\Longrightarrow$	$X = t$ if X does not occur in t.
(3)	t	$=*=$	X	$\Longrightarrow$	$X = t$ if X does not occur in t.
Eliminate					
(4)	$f(\bar{t})$	$=*=$	$f(\bar{s})$	$\Longrightarrow$	if solve$(\mathcal{A}(\bar{t}) = \mathcal{A}(\bar{s})) = false$ then fail
					else $\bar{t} = * = \bar{s}$
(5)	$seq(t_1, \bar{t})$	$=*=$	$seq(s_1, \bar{s})$	$\Longrightarrow$	if solve$(\mathcal{A}(seq(t_1, \bar{t})) = \mathcal{A}(seq(s_1, \bar{s}))) = false$
					then fail else $t_1 = * = s_1$,
					$normalize(\bar{t}) = * = normalize(\bar{s})$
(6)	$seq(X, \bar{t})$	$=*=$	$seq(s_1, \bar{s})$	$\Longrightarrow$	if solve$(\mathcal{A}(seq(X, \bar{t})) = \mathcal{A}(seq(s_1, \bar{s}))) = false$
					or X occurs in s_1 then fail
					else $X = s_1$, $normalize(\bar{t}) = * = normalize(\bar{s})$
				$\Longrightarrow$	if solve$(\mathcal{A}(seq(X, \bar{t})) = \mathcal{A}(seq(s_1, \bar{s}))) = false$
					or X occurs in s_1 then fail
					else $X = seq(s_1, seq(X_1, \varepsilon))$,
					$normalize(seq(X_1, \bar{t})) = * = normalize(\bar{s})$,
					where X_1 is a new variable.
(7)	$seq(t_1, \bar{t})$	$=*=$	$seq(X, \bar{s})$	$\Longrightarrow$	if solve$(\mathcal{A}(seq(t_1, \bar{t})) = \mathcal{A}(seq(X, \bar{s}))) = false$
					or X occurs in t_1 then fail
					else $X = t_1$, $normalize(\bar{t}) = * = normalize(\bar{s})$
				$\Longrightarrow$	if solve$(\mathcal{A}(seq(t_1, \bar{t})) = \mathcal{A}(seq(X, \bar{s}))) = false$
					or X occurs in t_1 then fail
					else $X = seq(t_1, seq(X_1, \varepsilon))$,
					$normalize(\bar{t}) = * = normalize(seq(X_1, \bar{s}))$,
					where X_1 is a new variable.
(8)	$seq(X, \bar{t})$	$=*=$	$seq(Y, \bar{s})$	$\Longrightarrow$	if solve$(\mathcal{A}(seq(X, \bar{t})) = \mathcal{A}(seq(Y, \bar{s}))) = false$
					then fail else $X = Y$, $\bar{t} = * = \bar{s}$.
				$\Longrightarrow$	if solve$(\mathcal{A}(seq(X, \bar{t})) = \mathcal{A}(seq(Y, \bar{s}))) = false$
					then fail else $X = seq(Y, seq(X_1, \varepsilon)$,
					$normalize(seq(X_1, \bar{t})) = * = normalize(\bar{s})$,
					where X_1 is a new variable and X, Y are distinct.
				$\Longrightarrow$	if solve$(\mathcal{A}(seq(X, \bar{t})) = \mathcal{A}(seq(Y, \bar{s}))) = false$
					then fail else $Y = seq(X, seq(Y_1, \varepsilon))$,
					$normalize(\bar{t}) = * = normalize(seq(Y_1, \bar{s}))$,
					where Y_1 is a new variable and X, Y are distinct.
Split					
(9)	$seq(t_1, \bar{t})$	$=*=$	$seq(s_1, \bar{s})$	$\Longrightarrow$	if $t_1 = * = s_1 \Longrightarrow r_1 = * = q_1$ then
					$normalize(seq(r_1, \bar{t})) = * = normalize(seq(q_1, \bar{s}))$
					$\vdots$
				$\Longrightarrow$	if $t_1 = * = s_1 \Longrightarrow r_w = * = q_w$,
					$normalize(seq(r_w, \bar{t}_n)) = * = normalize(seq(q_w, \bar{s}))$,
					where t_1 and s_1 are compound terms.

Figure 2: Transformation rules

Now $Y_1 = c$ and will try to unify $f(f(Z,a,Z))$ with $f(f(a,Z,a))$. This problem is similar to the one presented in example 4.5 and the result will be $Z = a$, reaching the final solution, $X = a$, $Y = b, c$ and $Z = a$. Since the use of the equations prevents entering an infinite loop, the algorithm will backtrack and try $X = a, X_1$ and proceed with the unification of $f(X_1, Y, f(Z,a,Z))$ and $f(b,c,f(a,Z,a))$ that has the associated equation:

$$X_1 + Y + 2Z + 3 = Z + 6 \Rightarrow X_1 + Y + Z = 3$$

The next unification is, $X_1 = b$ and proceeding with $f(Y,\ f(Z,a,Z))$ and $f(c,\ f(a,Z,a))$ leads to the equation:

$$Y + 2Z + 3 = Z + 5 \Rightarrow Y + Z = 2$$

Now, $Y = c$ and the algorithm proceeds with $f(f(Z,a,Z))$ and $f(f(a,Z,a))$ which is similar to example 4.5 and will reach $Z = a$. Finally we have the solution $X = a, b$, $Y = c$ and $Z = a$ that was unreachable with the original algorithm.

We now prove that the new algorithm correctly prunes branches which lead to failure. We first present some auxiliary results.

Lemma 4.1 *Given sequences t_1, s_1, $\bar{t}$ and $\bar{s}$,*

$$\mathcal{A}(t_1) = \mathcal{A}(s_1) \wedge \mathcal{A}(\bar{t}) = \mathcal{A}(\bar{s}) \Rightarrow \mathcal{A}(seq(t_1,\bar{t})) = \mathcal{A}(seq(s_1,\bar{s}))$$

Proof 4.1 *$\mathcal{A}(t_1) = \mathcal{A}(s_1) \Rightarrow \mathcal{A}(t_1) + \mathcal{A}(\bar{t}) = \mathcal{A}(s_1) + \mathcal{A}(\bar{t})$ and, since $\mathcal{A}(\bar{t}) = \mathcal{A}(\bar{s})$, one has $\mathcal{A}(t_1) + \mathcal{A}(\bar{t}) = \mathcal{A}(s_1) + \mathcal{A}(\bar{s})$. Now by the definition of $\mathcal{A}$, $\mathcal{A}(t_1) + \mathcal{A}(\bar{t}) = \mathcal{A}(s_1) + \mathcal{A}(\bar{s}) \Rightarrow \mathcal{A}(seq(t_1,\bar{t})) = \mathcal{A}(seq(s_1,\bar{s}))$. $\square$*

Proposition 4.1 *Given a sequence term $\bar{t}$,*

$$\mathcal{A}(normalize(\bar{t})) = \mathcal{A}(\bar{t})$$

Theorem 4.1 *Given two sequences, $\bar{t}$ and $\bar{s}$,*

$$\bar{t} = * = \bar{s} \text{ has solutions} \quad \Longrightarrow \quad \mathcal{A}(\bar{t}) = \mathcal{A}(\bar{s}) \text{ has solutions in } \mathbb{Z}^+$$

Proof 4.2 *We prove this result by induction on the number of steps of the transformation rules. The base cases are steps 1 to 3. Our induction hypothesis states that $\bar{t} = * = \bar{s}$ has solutions then, $\mathcal{A}(\bar{t}) = \mathcal{A}(\bar{s})$ has solutions on the positive integers. Assuming this hypothesis we will show that, if $seq(t_1,\bar{t}) = * = seq(s_1,\bar{s})$ has solutions then, $\mathcal{A}(seq(t_1,\bar{t})) = \mathcal{A}(seq(s_1,\bar{s}))$ has solutions in the positive integers.*

- *(1) $t = * = s$ has solution since both t and s are exactly the same, $\mathcal{A}(t) = \mathcal{A}(s) \Rightarrow V_1 + \ldots + V_n + k = V_1 + \ldots + V_n + k$, where $V_1, \ldots, V_n$ are variables occurring in t and s, k is the sum of the constants and functors, and this equation has solutions in the positive integers.*
- *(2) $X = t$ has solution since $\mathcal{A}(X) = \mathcal{A}(t) \Rightarrow X = V_1 + \ldots + V_n + k$ where $V_1, \ldots, V_n$ are the variables occurring in t and k is the sum of the constants and functors in t, thus it is enough to assign positive values to each one of the variables $V_1 + \ldots + V_n$, to have a positive solution.*
- *(3) This case is analogous to the previous one.*
- *(4) In this case we can apply the induction hypothesis directly and the result holds.*
- *(5) By lemma 4.1 and applying the induction hypothesis to $t_1 = * = s_1$ and to $normalize(\bar{t}) = * = normalize(\bar{s})$ the result holds.*
- *(6) Here we have two possible cases:*
  - *By lemma 4.1 and applying the induction hypothesis to $X = s_1$ and $normalize(\bar{t}) = * = normalize(\bar{s})$ the result holds.*

– *Here we have that:*

$$\begin{aligned} \mathcal{A}(X) = \mathcal{A}(seq(s_1, seq(X_1, \epsilon))) &= \mathcal{A}(s_1) + \mathcal{A}(X_1), && \textit{by the definition of } \mathcal{A} \\ &= \mathcal{A}(s_1) + X_1, && \textit{since } X_1 \textit{ is a new free variable} \end{aligned}$$

By the induction hypothesis we have that,

$$\mathcal{A}(normalize(seq(X_1, \bar{t}))) = \mathcal{A}(normalize(\bar{s}))$$

holds and by proposition 4.1,

$$\mathcal{A}(seq(X_1, \bar{t})) = \mathcal{A}(\bar{s})$$

Now,

$$\begin{aligned} \mathcal{A}(seq(X_1, \bar{t})) = \mathcal{A}(\bar{s}) &\Leftrightarrow \mathcal{A}(X_1) + \mathcal{A}(\bar{t}) = \mathcal{A}(\bar{s}), && \textit{by the definition of } \mathcal{A} \\ &\Leftrightarrow X_1 = \mathcal{A}(\bar{s}) - \mathcal{A}(\bar{t}), && \textit{since } X_1 \textit{ is a new free variable.} \end{aligned}$$

Now replacing X_1 *by* $\mathcal{A}(\bar{s}) - \mathcal{A}(\bar{t})$ *in* $\mathcal{A}(X) = \mathcal{A}(s_1) + X_1$ *leads to:*

$$\mathcal{A}(X) = \mathcal{A}(s_1) + \mathcal{A}(\bar{s}) - \mathcal{A}(\bar{t}) \Leftrightarrow \mathcal{A}(X) + \mathcal{A}(\bar{t}) = \mathcal{A}(s_1) + \mathcal{A}(\bar{s})$$

and by the definition of $\mathcal{A}$ *this is equivalent to,*

$$\mathcal{A}(seq(X, \bar{t})) = \mathcal{A}(seq(s_1, \bar{s}))$$

and the result holds.

- *(7) This case is analogous to the previous one.*
- *(8) Here we have three possible cases:*
  – *By lemma 4.1 and applying the induction hypothesis to* $X = * = Y$ *and to* $\bar{t} = * = \bar{s}$ *the result holds.*
  – *This case is similar to the one presented in the second case of (6)*
- *(9) Here we can apply the induction hypothesis directly and the result holds.* □

Note that the previous theorem states that whenever the equation associated with an unification problem does not have a solution in $\mathbb{Z}^+$ then, the original unification problem does not have any solution.

The inverse is not true, there are problems with an associated equation having solutions on the positive integers and without any solution. Consider the following examples:

Example 4.7 *Given the following unification,* $f(a, b) = * = f(b, c)$*, the associated Diophantine equation is* $\mathcal{A}(f(a, b)) = * = \mathcal{A}(f(b, c))$ *which results in* $3 = 3$*, but the unification doesn't have any solution.*

Example 4.8 *Given the following unification,* $f(g(a, a, a)) = f(X, X)$*, the associated Diophantine equation is* $\mathcal{A}(f(g(a, a, a))) = * = \mathcal{A}(f(X, X))$ *which results in* $4 = 2X + 1$*, but the unification doesn't have any solution.*

Note that, this happens because $\mathcal{A}$ abstracts different constant symbols to the same number (the constant 1).

Acknowledgements We thank Temur Kutsia for all the help provided explaining several details of his algorithm. In particular the examples 1.1, 4.1 and 4.2 were kindly provided by Kutsia. The work presented in this paper has been partially supported by funds granted to *LIACC* through the *Programa de Financiamento Plurianual, Fundação para a Ciência e Tecnologia* and *Programa POSI.*

References

[1] Franz Baader and Klaus U. Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. *J. Symb. Comput.*, 21(2):211–243, 1996.

[2] Franz Baader and Wayne Snyder. Unification theory. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 445–532. Elsevier and MIT Press, 2001.

[3] Alexandre Boudet, Evelyne Contejean, and Hervé Devie. A new AC-unification algorithm with a new algorithm for solving diophantine equations. In *5th IEEE LICS*, pages 289–299. IEEE, June 1990.

[4] Michael Clausen and Albrecht Fortenbacher. Efficient solution of linear diophantine equations. *J. Symb. Comput.*, 8(1/2):201–216, 1989.

[5] Jorge Coelho and Mario Florido. CLP(Flex): Constraint Logic Programming Applied to XML Processing. In *Ontologies, Databases and Applications of SEmantics (ODBASE)*, LNCS, Agia Napa, Cyprus, 2004. Springer Verlag.

[6] Evelyne Contejean. Solving $*$-problems modulo distributivity by a reduction to $AC1$-unification. *Journal of Symbolic Computation*, 16:493–521, 1993.

[7] Evelyne Contejean and Hervé Devie. An efficient algorithm for solving systems of diophantine equations. *Information and Computation*, 113(1):143–172, 1994.

[8] A. Farquhar, R. Fikes, and J. Rice. The ontolingua server: A tool for collaborative ontology construction. *International Journal of Human-Computer Studies*, 46(6):707–727, 1997.

[9] Miguel Filgueiras and Ana Paula Tomás. A fast method for finding the basis of nonnegative solutions to a linear diophantine equation. *J. Symb. Comput.*, 19(6):507–526, 1995.

[10] M. R. Genesereth and R. E. Fikes. Knowledge Interchange Format, Version 3.0 Reference Manual TR Logic-92-1. Technical report, Stanford University, 1992.

[11] Seymour Ginsburg and Xiaoyang Sean Wang. Pattern matching by rs-operations: Toward a unified approach to querying sequenced data. In *PODS*, pages 293–300. ACM Press, 1992.

[12] Gösta Grahne and Emmanuel Waller. How to make sql stand for string query language. In Richard C. H. Connor and Alberto O. Mendelzon, editors, *DBPL*, volume 1949 of *Lecture Notes in Computer Science*, pages 61–79. Springer, 1999.

[13] Alexander Herold and Jörg H. Siekmann. Unification in abelian semigroups. *J. Autom. Reasoning*, 3(3):247–283, 1987.

[14] Gérard P. Huet. An algorithm to generate the basis of solutions to homogeneous linear diophantine equations. *Inf. Process. Lett.*, 7(3):144–147, 1978.

[15] T. Kutsia. Unification with sequence variables and flexible arity symbols and its extension with pattern-terms. In *Artificial Intelligence, Automated Reasoning and Symbolic Computation. Proceedings of Joint AICS'2002 - Calculemus'2002 conference*, volume 2385 of *Lecture Notes in Artificial Intelligence*, pages 290–304, Marseille, France, 2002. Springer Verlag.

[16] Temur Kutsia. Pattern unification with sequence variables and flexible arity symbols. *Electronic Notes on Theoretical Computer Science*, 66(5), 2002.

[17] Temur Kutsia. Solving equations involving sequence variables and sequence functions. In Bruno Buchberger and John A. Campbell, editors, *Proceedings of the 7th International Conference on Artificial Intelligence and Symbolic Computation, AISC'04*, volume 3249 of *Lecture Notes in Artificial Intelligence*, pages 157–170, Hagenberg, Austria, 22–24 September 2004. Springer Verlag.

[18] Temur Kutsia. Solving equations involving sequence variables and sequence functions. Technical Report 04-01, Linz, Austria, 2004.

[19] G. S. Makanin. The problem of solvability of equations in a free semigroup. *Math. Sbornik USSR*, 103:147–236, 1977.

[20] Mircea Marin. Functional Programming with Sequence Variables: The Sequentica Package. In Jordi Levy, Michael Kohlhase, Joachim Niehren, and Mateu Villaret, editors, *Proceedings of the 17th International Workshop on Unification (UNIF 2003)*, pages 65–78, Valencia, June 2003.

[21] Mircea Marin and Temur Kutsia. On the implementation of a rule-based programming system and some of its applications. In Boris Konev and Renate Schmidt, editors, *Proceedings of the 4th International Workshop on the Implementation of Logics (WIL'03)*, pages 55–68, Almaty, Kazakhstan, 2003.

[22] Giansalvatore Mecca and Anthony J. Bonner. Sequences, datalog and transducers. In *PODS*, pages 23–35. ACM Press, 1995.

[23] Mark E. Stickel. A unification algorithm for associative-commutative functions. *J. ACM*, 28(3):423–434, 1981.

[24] S. Wolfram, editor. *The Mathematica Book - fourth edition.* Cambridge University Press and Wolfram Research Inc., 1999.