E-Unification by Means of Tree Tuple Synchronized Grammars[1]

Sébastien Limet and Pierre Réty
LIFO - Université d'Orléans
B.P. 6759, 45067 Orléans cedex 2, France
e-mail : {limet, rety}@lifo.univ-orleans.fr

Abstract: The goal of this paper is both to give a E-unification procedure that always terminates, and to decide unifiability. For this, we assume that the equational theory is specified by a confluent and constructor-based rewrite system, and that four additional restrictions are satisfied. We give a procedure that represents the (possibly infinite) set of solutions thanks to a new kind of grammar, called tree tuple synchronized grammar, and that can decide unifiability thanks to an emptiness test. Moreover we show that if only three of the four additional restrictions are satisfied then unifiability is undecidable.

1 Introduction

First order E-unification is a tool that plays an important role in automated deduction, in particular in functional logic programming and for solving symbolic constraints. It consists in finding instances to variables that make two terms equal modulo an equational theory given by a set of equalities, i.e. it amounts to solve an equation (called goal). General E-unification is undecidable and may have infinitely many solutions. This is why E-unification procedures, like narrowing, often loop, enumerating infinite set of unifiers or computing unproductive branches.

When solving equations in a computation (of a functional logic program for instance), most of the time, it is not interesting to enumerate the solutions. It is more important to test whether the equation has at least one solution (unifiability test) and to have a finite representation of the solutions. The first point allows to cut unproductive branches, and the second avoids generation of infinite sets of solutions.

We have several aims in this paper. First of all, we want to define restrictions on the unification problem that insure decidability of unifiability. In addition of confluence and constructor-based property of the rewrite system that represents the equational theory, we need four other restrictions that are shown necessary to decide unifiability (i.e. if any of them is not satisfied unifiability is undecidable). Thus these restrictions define a limit between decidability and undecidability of unifiability. Our second goal is to give a E-unification procedure that never loops when our restrictions are verified, and that decides unifiability. The problem is that theories defined in this framework may be infinitary, i.e. for some goals the set of solutions cannot be described by a finite complete set of unifiers. So we need a way to represent infinite sets of substitutions.

A solution being defined by the instances of the variables of the goal, i.e. by a tuple of terms, and terms being trees, the set of solutions can be viewed as a tree tuple

[1] Missing formal definitions and proofs are given in [10].

language. To describe this language, we introduce a new kind of grammar, the tree tuple synchronized grammars (TTSG). Their particularity is the notion of synchronization, i.e. the fact that some productions must be applied at the same time. For this reason TTSG's can define context-sensitive languages like $\{d(a^i(0), b^i(0), c^i(0))\}$. The class of languages defined by TTSG's is larger than we need and does not have nice properties. Fortunately the TTSG's we build from a unification problem are not any, and the recognized languages have particular properties :

- their intersection is a language recognized by a TTSG,
- emptiness is decidable.

Some authors have already used tree languages to represent infinite sets of solutions. For example in [5], they are used to solve set constraints, but without any notion of synchronization. The TTSG's are not identical to the coupled context-free grammars of [6] because we need a finer control of synchronizations which is achieved thanks to a tuple of integers. The following example explains the principle of our procedure.

Example 1.1 Consider the TRS that defines the functions f and g

$$f(s(s(x))) \xrightarrow{1} f(x),\ f(p(x)) \xrightarrow{2} f(x),\ f(0) \xrightarrow{3} 0,$$
$$g(s(x)) \xrightarrow{4} s(g(x)),\ g(0) \xrightarrow{5} 0$$

and the goal $f(g(x)) \stackrel{?}{=} 0$.

Step 1. The goal $f(g(x)) \stackrel{?}{=} 0$ is decomposed into three parts, $g(x) \stackrel{?}{=} y_1$, $f(y_2) \stackrel{?}{=} y_3$ and $0 \stackrel{?}{=} y_4$, where y_1, y_2, y_3, y_4 are new variables. The set of ground data-solutions of $g(x) \stackrel{?}{=} y_1$ can be considered as an infinite set of pairs of terms defined by $\{(t_1, t_2) | g(t_2) \rightarrow^* t_1\}$. This set is considered as a language (says $\mathcal{L}_1$) of pairs of trees where the two components are not independent. In the same way, the set of ground data-solutions of $f(y_2) \stackrel{?}{=} y_3$ can be viewed as the language (says $\mathcal{L}_2$) of pairs of trees that describes the set $\{(t_1, t_2) | f(t_2) \rightarrow^* t_1\}$ and 0 can be viewed as the language (says $\mathcal{L}_3$) of 1-uple reduced to $\{(0)\}$. These languages can be described by TTSG's. The grammars are computed from the rewrite system and the goal.

Step 2. Once these three TTSG's are constructed, the initial goal is re-composed by two steps. First the languages $\mathcal{L}_1$ and $\mathcal{L}_2$ are combined to get the language $\mathcal{L}_4$ of the ground data-solutions of $f(g(x)) \stackrel{?}{=} y_3$. This is done by computing a special kind of intersection between two TTSG's that corresponds to the join operation in relational data-bases. The result is a TTSG that describes the language of triples of trees defined by $\{(t_1, t_2, t_3) | (t_2, t_3) \in \mathcal{L}_1 \text{ and } (t_1, t_2) \in \mathcal{L}_2\}$. In other words, t_2 is the result of $g(x)$ when instantiating x by t_3, moreover t_2 belong to the definition domain of the function f, and t_1 is the result of $f(t_2)$, i.e. of $f(g(t_3))$. Second the TTSG of $\mathcal{L}_4$ is combined with the TTSG of $\mathcal{L}_3$ in the same way. We get a TTSG that describes the language of triples of trees $\mathcal{L}_5$ defined by $\{(t_1, t_2, t_3) | t_1 = 0 \text{ and } (t_1, t_2, t_3) \in \mathcal{L}_4\}$. As t_3 is an instance of x, t_1 is the result of $f(g(t_3))$ and $t_1 = 0$, we get a finite description of the ground data-substitutions σ such that $\sigma f(g(x)) \rightarrow^* 0$. Moreover it is decidable to know whether the language $\mathcal{L}_5$ is empty or not. Therefore we can decide the unifiability of $f(g(x)) \stackrel{?}{=} 0$.

After basic definitions given in section 2, the four additional restrictions as well as the undecidability results are given in section 3. The first step of our method is presented in section 4 and the second step in section 5. An overview of related work and the conclusion are given in section 6.

2 Preliminaries

We assume that the reader is familiar with standard definitions of one-sorted terms, substitutions, equations, rewrite systems (see [3]). We just recall here the main definitions and notations used in the paper.

Let Σ be a finite set of symbols and V be an infinite set of variables, $T_{\Sigma \cup V}$ is the term algebra over Σ and V. Σ is partitioned in two parts: the set $\mathcal{F}$ of **function symbols**, and the set $\mathcal{C}$ of **constructors**. The terms of $T_{\mathcal{C} \cup V}$ are called **data-terms**. A term is said **linear** if it does not contain several occurrences of the same variable. In the following x, y, z denote variables, s, t, l, r denote terms, f, g, h function symbols, c a constructor symbol, and u, v, w occurrences.

Let t be a term, $\boldsymbol{D(t)}$ is the set of occurrences of t, $\boldsymbol{t|_u}$ is the subterm of t at occurrence u and $\boldsymbol{t(u)}$ is the symbol that labels the occurrence u of t. $\boldsymbol{t[u \leftarrow s]}$ is the term obtained by replacing in t the subterm at occurrence u by s. We generalize the occurrences (as well as the above notations) to tuples in the following way: let $p = (p_1, \ldots, p_n)$ a tuple, $\forall i \in [1, n]\ p|_i = p_i$, and when the p_i's are terms, $p|_{i.u} = p_i|_u$. Moreover we define the **concatenation** of two tuples by $(t_1, \ldots, t_n) * (t'_1, \ldots, t'_{n'}) = (t_1, \ldots, t_n, t'_1, \ldots, t'_{n'})$ and the **component elimination** by $(t_1, \ldots, t_i, \ldots, t_n) \backslash_i = (t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n)$. A term rewrite system (**TRS**) is a finite set of oriented equations called rewrite rules or rules. For a TRS R, the rewrite relation is denoted by $\rightarrow_R$ and is defined by $t \rightarrow_R s$ if there exists a rule $l \rightarrow r$ in r and a non-variable occurrence u in t such that $t|_u = \sigma l$ and $s = t[u \leftarrow \sigma r]$. The transitive closure of $\rightarrow_R$ is denoted by $\rightarrow_R^*$. **lhs** means left-hand-side and **rhs** means right-hand-side. A TRS is said **confluent** if $t \rightarrow_R^* t_1$ and $t \rightarrow_R^* t_2$ implies $t_1 \rightarrow_R^* t_3$ and $t_2 \rightarrow_R^* t_3$ for some t_3. If the lhs (resp. rhs) of every rule is linear the TRS is said **left-(resp. right-)linear**. If it is both left and right-linear the TRS is said **linear**. A TRS is **constructor based** if every rule is of the form $f(t_1, \ldots, t_n) \rightarrow r$ where the t_i's are data-terms.

t **narrows** into s, written $\boldsymbol{t \rightsquigarrow s}$, if there exists a rule $l \rightarrow r$ in R, a non-variable occurrence u of t, such that $\sigma t|_u = \sigma l$ where $\sigma = mgu(t|_u, l)$ and $s = (\sigma t)[u \leftarrow \sigma r]$. We write $\boldsymbol{t \rightsquigarrow_{[u, l \rightarrow r, \sigma]} s}$.

3 Undecidability Results

The considered rewrite systems are supposed to be constructor-based and confluent. Our four additional restrictions are :

1. **Linearity of rewrite rules**: every rewrite rule side is linear.
2. **No σ_{in}** : if a subterm r of some rhs unifies with some lhs l (after variable renaming to avoid conflicts) then the mgu σ does not modify the variables of l[2].
3. **No nested functions in rhs's**: the function symbols in the rhs's may not appear at comparable occurrences. For example f and g are nested in $f(g(x))$ but not in $c(f(x), g(y))$.
4. **Linearity of the goal**: the goal does not contain several occurrences of the same variable.

[2] In other words, if σ is split into $\sigma = \sigma_{in} \cup \sigma_{out}$ where $\sigma_{in} = \sigma\,[var(l)]$ and $\sigma_{out} = \sigma\,[var(r)]$ then σ_{in} must be the identity mapping.

These four restrictions together allow non-finitary theories[3]. The (even minimal) complete set of solutions and then also the narrowing search space may be infinite.

Theorem 3.1 *If any of the four above restrictions is not satisfied, unifiability is undecidable.*

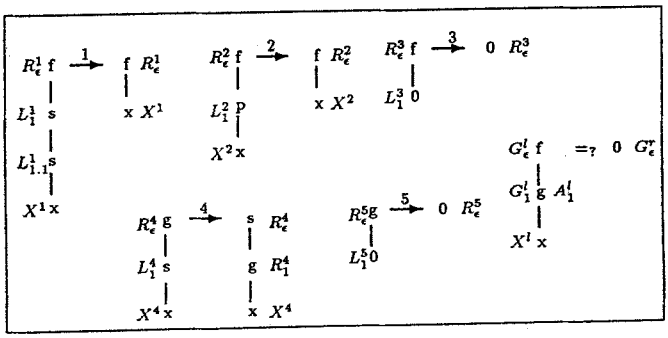
To prove this result, we show that for each restriction, there exists a rewrite system satisfying the three others, that encodes a well-known undecidable problem, the Post correspondence problem.

4 Step 1 : Transformation of a TRS into TTSG's

Here is the first step of our method. Recall that the TRS is assumed to be confluent and constructor-based, and satisfies restrictions 1 to 4. The aim is to convert the TRS and the goal into several TTSG's. This step is illustrated by example 1.1. For this example, three TTSG's will be constructed, one for $g(x)$, one for $f(y_2)$ and one for 0. The terminals of grammars are the constructors.

4.1 Non-Terminals

To each occurrence of each term of the TRS and the goal we associate a non-terminal, next the productions will be deduced from subterms relations and syntactic unifications. To each non variable occurrence u of the lhs (resp. rhs) of each rule i is associated the non terminal L_u^i (resp. R_u^i), except when $u = \epsilon$, we associate R_u^i even to the lhs. To the occurrences of the variable x is associated X^i (see figure below). In the same way, the non-terminal G_u^l (resp G_u^r) is associated to each occurrence u of the lhs (resp rhs) of the goal. $NT(t, u)$ denotes the non-terminal associated to the occurrence u of t. An additional non-terminal A_u^l (resp A_u^r) is associated to the arguments of function of the goal (here occurrence 1 of $f(g(x))$ to encode the variable y_2). t being a side of the goal, $ANT(t, u)$ denotes the additional non-terminal associated to the occurrence u of t.



[3]for example the rewrite system $\{f(s(x)) \to f(x), f(0) \to 0\}$.

4.2 Productions

Two kinds of productions are deduced from the TRS. The **free productions** that are similar to the productions of regular tree grammars. These productions generate constructor symbols and are deduced from subterm relations. The second kind of productions are called **synchronized productions** and come from syntactically unifiable terms. These productions are empty (they do not produce any constructor).

The way the productions are deduced is motivated by narrowing techniques. From the correspondence between rewriting and narrowing (lifting lemma [7]), the languages $\mathcal{L}_1$, $\mathcal{L}_2$ of example 1.1 are the ground instances of the data-solutions computed by narrowing. This is why we look for narrowing possibilities. For instance, the rhs of rule 4 in example 1.1, unifies with the lhs of the same rule. Therefore the narrowing step $g(x) \leadsto_{[\epsilon, r_4, x \mapsto s(x')]} s(g(x'))$ is possible. This step achieves two operations: it maps the variable x to $s(x')$ and it sets the result of the narrowing step to $s(g(x'))$.

From TTSG point of view, this narrowing step is simulated as follows. The term $g(x)$ is represented by the non-terminal R_1^4 (see figure) and the variable x by X^4 Therefore the pair (R_1^4, X^4) encodes $(g(x), x)$. The fact that $g(x)$ unifies with $g(s(x'))$ (the renamed version of the lhs of rule 4) is encoded by the empty production $R_1^4 \Rightarrow R_\epsilon^4$. The fact that the previous unification instantiates x is encoded by the empty production $X^4 \Rightarrow L_1^4$. In order to force these two operations to be achieved at the same time, the two productions are synchronized in the pack of productions $\{R_1^4 \Rightarrow R_\epsilon^4,\ X^4 \Rightarrow L_1^4\}$. Thus when it is applied on (R_1^4, X^4), we get (R_ϵ^4, L_1^4) which means that the unification is about to be done and therefore the narrowing step too, but the new constructors produced by the unification and the narrowing step have not appeared yet.

This is the aim of the free productions deduced from subterm relationships. On our example, we just have narrowed $g(x)$ on top with rule 4 and we get $s(g(x'))$. So the narrowing step generates a term with the constructor s on top whose argument is the function call $g(x')$. This is encoded by the free production $R_\epsilon^4 \Rightarrow s(R_1^4)$. In the same way, x is instantiated by $s(x')$, which is encoded by the free production $L_1^4 \Rightarrow s(X^4)$. The narrowing step is completely achieved by the derivation $(R_\epsilon^4, L_1^4) \Rightarrow (s(R_1^4), L_1^4) \Rightarrow (s(R_1^4), s(X^4))$. One can easily see that a second application of rule 4 on $s(g(x'))$ can be simulated by applying again the pack of productions and next the two free productions. Now, let us define more formally all the productions deduced from the unification problem.

First the free production: For any term t in the TRS or in the goal and any constructor position u in t (i.e. $t(u)$ is a constructor), we create the free production $NT(t,u) \Rightarrow t(u)(NT(t,u.1), \ldots, NT(t,u.n))$ where n is the arity of $t(u)$. In our example, we get :

$L_1^1 \Rightarrow s(L_{1.1}^1)$, $L_{1.1}^1 \Rightarrow s(X^1)$, $L_1^2 \Rightarrow p(X^2)$, $L_1^3 \Rightarrow 0$, $R_\epsilon^3 \Rightarrow 0$,
$L_1^4 \Rightarrow s(X^4)$, $R_\epsilon^4 \Rightarrow s(R_1^4)$, $L_1^5 \Rightarrow 0$, $R_\epsilon^5 \Rightarrow 0$, $G_\epsilon^r \Rightarrow 0$

Second the synchronized productions: for all $r^i|_u$ and l^j syntactically unifiable, we create the **pack of productions** (i.e. The set of synchronized productions)

$$\{NT(r^i,u) \Rightarrow NT(l^j,\epsilon),\quad NT(r^i,u.v_1) \Rightarrow NT(l^j,v_1), \ldots,\quad NT(r^i,u.v_n) \Rightarrow NT(l^j,v_n)\}$$

where $v_1, \ldots v_n$ are the variable occurrences of $r^i|u$ (let $\theta = mgu(r^i|_u, l^j)$, from the σ_{in} restriction we know that $\theta r^i|_u = l^j$ therefore $v_1, \ldots, v_n$ are also occurrences of l^j). For our example, r^1 unifies with l^1, l^2 and l^3 which gives the synchronized productions

$\{R^1_\epsilon \Rightarrow R^1_\epsilon, X^1 \Rightarrow L^1_1\}$, $\{R^1_\epsilon \Rightarrow R^2_\epsilon, X^1 \Rightarrow L^2_1\}$ and $\{R^1_\epsilon \Rightarrow R^3_\epsilon, X^1 \Rightarrow L^3_1\}$. r^2 unifies with l^1, l^2 and l^3 too, so we get $\{R^2_\epsilon \Rightarrow R^1_\epsilon, X^2 \Rightarrow L^1_1\}$, $\{R^2_\epsilon \Rightarrow R^2_\epsilon, X^2 \Rightarrow L^2_1\}$ and $\{R^2_\epsilon \Rightarrow R^3_\epsilon, X^2 \Rightarrow L^3_1\}$. Finally $r^4|_1$ unifies with l^4 and l^5 so we get $\{R^4_1 \Rightarrow R^4_\epsilon, X^4 \Rightarrow L^4_1\}$ and $\{R^4_1 \Rightarrow R^5_\epsilon, X^4 \Rightarrow L^5_1\}$.

To generate the synchronized productions coming from the goal, remember that we consider in fact $f(y_2)$, $g(x)$ and 0. For each function occurrence u of the goal t such that $t(u) = l^j(\epsilon)$ (i.e. $t(u)(x_1, \ldots, x_n)$ unifies with l^j), we create the synchronized productions:

$$\{NT(t,u) \Rightarrow NT(l^j, \epsilon),\quad ANT(t,u.1) \Rightarrow NT(l^j, 1), \ldots,\\ ANT(t,u.n) \Rightarrow NT(l^j, n)\}$$

The language derived from $NT(t,u)$ expresses the terms issued by narrowing from $t(u)(x_1, \ldots, x_n)$ while the languages derived from $ANT(t,u.i)$ expresses the instances of the fictitious variables x_i. In example 1.1, $f(y_2)$ unifies with l^1, l^2 and l^3, this gives the synchronized productions $\{G^l_\epsilon \Rightarrow R^1_\epsilon, A^l_1 \Rightarrow L^1_1\}$, $\{G^l_\epsilon \Rightarrow R^2_\epsilon, A^l_1 \Rightarrow L^2_1\}$ and $\{G^l_\epsilon \Rightarrow R^3_\epsilon, A^l_1 \Rightarrow L^3_1\}$. $g(x)$ unifies with l^4 and l^5 so we get $\{G^l_1 \Rightarrow R^4_\epsilon, X^l \Rightarrow L^4_1\}$ and $\{G^l_1 \Rightarrow R^5_\epsilon, X^l \Rightarrow L^5_1\}$[4].

The languages we want express, are the ground data-instances of the solutions provided by narrowing. The productions described so far express the solutions provided by narrowing. To get ground data-instances we introduce the non-terminal ANY and for each constructor c we create the free production $ANY \Rightarrow c(ANY, \ldots, ANY)$. Because of linearity, any variable X^j that appears in a rhs under only constructors, will not be instantiated anymore by narrowing. So to generate the ground data-instances of these variables we create the production $X^j \Rightarrow ANY$ for each X^j.

4.3 Grammars

Many productions have been deduced from the TRS and the goal, let us now define the grammars that are constructed with them. All the considered grammars have the same terminals (the constructors), the same non-terminals, and the same productions, as defined before. Just the axioms (tuples of non-terminals) are different[5]. For example 1.1, we get the grammars

- Gr^l_ϵ defined by the axiom (G^l_ϵ, A^l_1), which generates the language $\mathcal{L}_2$,
- Gr^l_1 defined by the axiom (G^l_1, X^l), which generates the language $\mathcal{L}_1$,
- Gr^r_ϵ defined by the axiom (G^r_ϵ), which generates the language $\mathcal{L}_3$.

Here is an example of derivation for Gr^l_ϵ.

$$\begin{array}{ll}(G^l_\epsilon,\ A^l_1) \Rightarrow (R^1_\epsilon, L^1_1) \Rightarrow (R^1_\epsilon, s(L^1_{1.1}))\\ \quad \Rightarrow (R^1_\epsilon, s(s(X^1))) & \Rightarrow (R^2_\epsilon, s(s(L^2_1)))\\ \quad \Rightarrow (R^2_\epsilon, s(s(p(X^2)))) & \Rightarrow (R^3_\epsilon, s(s(p(L^3_1))))\\ \quad \Rightarrow (0, s(s(p(L^3_1)))) & \Rightarrow (0, s(s(p(0))))\end{array}$$

This encodes the narrowing derivation

$$f(y_2) \leadsto_{[\epsilon,1,y_2 \mapsto s(s(x_1))]} f(x_1) \leadsto_{[\epsilon,2,x_1 \mapsto p(x_2)]} f(x_2) \leadsto_{[\epsilon,3,x_2 \mapsto 0]} 0$$

where the resulting term is 0 and y_2 is instantiated by $s(s(p(0)))$.

In the general case, the definition of the grammars (i.e. of theirs axioms) is a bit technical because of the constructors that may appear in the goal. See [10].

---

[4]Within the goal, the argument of the function symbol g is a variable, therefore we do not need an additional non-terminal $A^l_{1.1}$ for it.

[5]The grammars could be optimized by removing non reachable non-terminals and non usable productions.

4.4 Control

Synchronized grammars, as defined previously, are close to regular tree grammars (and very close to coupled grammars of [6]) and are easy to use, but unfortunately they do not work in every case because they do not take into account variable renamings. Indeed, consider the rewrite system $\{f(c(x,y)) \xrightarrow{1} c(f(x),f(y))\}$ and the goal $f(x) = t$ where t is an arbitrary term. The tree grammar Gr^l_ϵ contains the productions $L^1_1 \Rightarrow c(X^1,Y^1), R^1_\epsilon \Rightarrow c(R^1_1,R^1_2), \{R^1_1 \Rightarrow R^1_\epsilon, X^1 \Rightarrow L^1_1\}, \{R^1_2 \Rightarrow R^1_\epsilon, Y^1 \Rightarrow L^1_1\}, \{G^l_\epsilon \Rightarrow R^1_\epsilon, X^l \Rightarrow L^1_1\}$ and the axiom is (G^l_ϵ, X^l). A possible derivation of Gr^l_ϵ is:

$$\begin{aligned}(G^l_\epsilon, X^l) &\Rightarrow (R^1_\epsilon, L^1_1)\\ &\Rightarrow (c(R^1_1,R^1_2), L^1_1)\\ &\Rightarrow (c(R^1_1,R^1_2), c(X^1,Y^1))\\ &\Rightarrow (c(R^1_\epsilon,R^1_2), c(L^1_1,Y^1))\\ &\Rightarrow (c(c(R^1_1,R^1_2),R^1_2), c(L^1_1,Y^1))\\ &\Rightarrow (c(c(R^1_1,R^1_2),R^1_2), c(c(X^1,Y^1),Y^1))\end{aligned}$$

This encodes the narrowing derivation:

$$f(x) \rightsquigarrow_{[x \mapsto c(x_1,y_1)]} c(f(x_1),f(y_1)) \rightsquigarrow_{[x_1 \mapsto c(x_2,y_2)]} c(c(f(x_2),f(y_2)),f(y_1))$$

The problem now is that both R^1_2 and Y^1 occur twice. One occurrence of R^1_2 corresponds to the term $f(y_2)$ and the other to $f(y_1)$. In the same way one occurrence of Y^1 corresponds to y_2 and the other to y_1. Obviously if $f(y_1)$ is narrowed, y_1 is instantiated whereas if $f(y_2)$ is narrowed, y_2 is instantiated. But using the grammar, the synchronized productions $\{R^1_2 \Rightarrow R^1_\epsilon, Y^1 \Rightarrow L^1_1\}$ can be applied on the first occurrence of R^1_2 and the second occurrence of Y^1. This means that $f(y_2)$ is narrowed while y_1 is instantiated.

The solution of this problem consists in using an integer number, called **control**, to encode variable renamings. In a grammar computation, each non-terminal is coupled with an integer of control, which is incremented into a not yet used value when a synchronized production is applied on it. When a free production is applied, the control number is preserved. Moreover a pack of productions will be applied only on non-terminals that have the same control number. For example the previous derivation is transformed into:

$$\begin{aligned}((G^l_\epsilon,0),(X^l,0)) &\Rightarrow ((R^1_\epsilon,1),(L^1_1,1))\\ &\Rightarrow (c((R^1_1,1),(R^1_2,1)),(L^1_1,1))\\ &\Rightarrow (c((R^1_1,1),(R^1_2,1)),c((X^1,1),(Y^1,1)))\\ &\Rightarrow (c((R^1_\epsilon,2),(R^1_2,1)),c((L^1_1,2),(Y^1,1)))\\ &\Rightarrow (c(c((R^1_1,2),(R^1_2,2)),(R^1_2,1)),c((L^1_1,2),(Y^1,1)))\\ &\Rightarrow (c(c((R^1_1,2),(R^1_2,2)),(R^1_2,1)),c(c((X^1,2),(Y^1,2)),(Y^1,1)))\end{aligned}$$

Now the pack $\{R^1_2 \Rightarrow R^1_\epsilon, Y^1 \Rightarrow L^1_1\}$ cannot be applied in the wrong way.

Thus we can prove the following result, which insures soundness of step 1.

Theorem 4.1 *The tree tuple language recognized by a TTSG gives exactly the ground data-instances of the data-terms computed by narrowing, thanks to the first field of tuples, as well as the corresponding instances of variables thanks to other fields.*

4.5 General Definition of TTSG's

In the following, NT is a finite set of non-terminal symbols and recall that C is the set of constructor symbols. Upper-case letters denote elements of NT.

Actually a tuple of integers instead of one integer is needed to control synchronizations after intersections of grammars (see section 5). In the following definition, k is the rank of field (also called level) in the control tuple that is incremented when applying the pack of productions.

Definition 4.2 A **production** is a rule of the form $X \Rightarrow t$ where $X \in NT$ and $t \in t_{CONT}$. A **pack of productions** is a set of productions coupled with a non negative integer and denoted $\{X_1 \Rightarrow t_1, \ldots, X_n \Rightarrow t_n\}_k$.

When $k = 0$ the pack is a singleton and it is of the form $\{X_1 \Rightarrow c(Y_1, \ldots, Y_n)\}_0$ where c is a constructor and $Y_1, \ldots, Y_n$ non-terminals. The production is said **free**, and is written more simply $X_1 \Rightarrow c(Y_1, \ldots, Y_n)$.

When $k > 0$ the pack is of the form $\{X_1 \Rightarrow Y_1, \ldots, X_n \Rightarrow Y_n\}_k$ where $Y_1, \ldots, Y_n$ are non-terminals. The productions of the pack are said **synchronized.** $\diamond$

Definition 4.3 A **TTSG** is defined by a 5-uple $(Sz, \mathcal{C}, NT, PP, TI)$ where

- Sz is a positive integer that defines the size of the tuple of control,
- $\mathcal{C}$ is the set of constructors (terminals in the terminology of grammars),
- NT is the finite set of non-terminals,
- PP is a finite set of packs of productions,
- TI is the axiom of the TTSG. It is a tuple $((I_1, ct_1), \ldots, (I_n, ct_n))$ where every I_i is a non-terminal, and every ct_i is a Sz-uple of control containing 0's and $\perp$'s.

$\diamond$

$\perp$ means that this field of the control is not used. In fact Sz is the number of intersection + 1 done to build the grammar. Intuitively a free production $X \Rightarrow c(Y_1, \ldots, Y_n)$ can be applied as soon as X appears in a computation of the grammar, and then $Y_1, \ldots, Y_n$ preserves the same control as X. On the other hand a pack of productions $\{X_1 \Rightarrow Y_1, \ldots, X_n \Rightarrow Y_n\}_k$ can be applied iff $X_1, \ldots X_n$ appear at the same time in a derivation and the k^{th} components of their controls are identical (and are not $\perp$). The x_i's are then replaced by the y_i's and the k^{th} component of control is set to a new fresh value.

5 Step 2: Intersection of TTSG's over one Component

This section describes the second step of our method. Let us consider again example 1.1. Recall that we have decomposed the problem into three parts $g(x) \stackrel{?}{=} y_1$, $f(y_2) \stackrel{?}{=} y_3$ and $0 \stackrel{?}{=} y_4$. In subsection 4.3, three TTSG's have been deduced from the problem to solve each of the three parts. The point now is to reconstruct the initial problem thanks to the intersection over one component of sets of tuples. This operation corresponds to the join operation in the relational algebra (relational databases).

Definition 5.1 Let E_1 be a set of n_1-uples and E_2 be a set of n_2-uples. The **one component k_1, k_2 intersection** of E_1 and E_2 is the set of $n_1 + n_2 - 1$-uples defined by $\{tp_1 * (tp_2 \backslash_{k_2}) \mid tp_1 \in E_1 \text{ and } tp_2 \in E_2 \text{ and } tp_1|_{k_1} = tp_2|_{k_2}\}$. $\diamond$

For example the one component 2, 1 intersection of the sets $E_1 = \{(0, s(0)), (s(0), 0)\}$ and $E_2 = \{(s(0), s(s(0)), (s(s(0)), 0)\}$ is the set of triples $E_3 = \{(0, s(0), s(s(0)))\}$.

To get the solutions of the initial goal, we have to compute incrementally the one component k_1, k_2 intersection for each pair of grammars Gr_1, Gr_2 such that the k_1^{th} component of the axiom of G_1 is G_u^l and the k_2^{th} component of the axiom of G_2 is A_u^l with the same u (resp. G_u^r and A_u^r). At the end, we have also to compute the intersection for components G_ϵ^l and G_ϵ^r.

When considering any TTSG's, we have the following result.

Lemma 5.2 Emptiness of intersection of languages recognized by TTSG's is undecidable.

Moreover the intersection of languages recognized by TTSG's is not always a language recognized by a TTSG. Fortunately, we do not consider any TTSG's, but only the ones coming from a unification problem, and in this case the problem is decidable.

Emptiness of intersection becomes decidable if the component k_1 or k_2 has the property of **external synchronization**. This means that at most one production can be applied on this component when using a pack of synchronized productions. So, an externally synchronized component of a TTSG behaves as a regular tree language in the sense that any branch of this component can be generated independently from the others.

Lemma 5.3 The first component of every TTSG produced from the unification problem[6] has the external synchronization property.

Let us give thanks to an example the idea of the intersection algorithm.

Example 5.4 This example does not come from a unification problem, but it is easier to understand, and every component is nevertheless externally synchronized. Let
$G_1 = (1, \{s, 0\}, \{X, X', Y, Y', Y''\}, \{X' \Rightarrow 0, Y' \Rightarrow 0, X' \Rightarrow s(X), Y' \Rightarrow s(Y''), Y'' \Rightarrow s(Y), \{X \Rightarrow X', Y \Rightarrow Y'\}_1\}, ((X, 0), (Y, 0)))$ and
$G_2 = ((1, \{s, 0\}, \{Z, Z', T, T', T''\}, \{Z' \Rightarrow 0, T' \Rightarrow 0, Z' = s(Z), T' \Rightarrow s(T''), T'' \Rightarrow s(T), \{Z \Rightarrow Z', T \Rightarrow T'\}_1\}, ((Z, 0), (T, 0)))$.
G_1 and G_2 generate the same language i.e. the pairs $(s^n(0), s^{2n}(0))$. The 2,1 intersection of G_1 and G_2 is then the language of triples $L_3 = \{(s^n(0), s^{2n}(0), s^{4n}(0))\}$. The question is how building from G_1 and G_2, a new TTSG G_3 that generates L_3? The idea is that the first component of L_3 will be generated by the productions of G_1, the last component of L_3 will be generated by the productions of G_2, therefore G_3 contains the non-terminals and the productions of both G_1 and G_2. Thanks to synchronizations, the links (between components) coming from G_1 and G_2 are preserved. The second component of L_3 is the intersection of the second component of G_1 with the first component of G_2. The productions that generate it are built using the same idea as for the intersection of regular languages, i.e. by computing the Cartesian product of the grammars. More precisely, we note at first that only the non-terminals Y, Y', Y'' (resp. Z, Z') may appear in the second (resp. the first) component of G_1 (resp. G_2). Thus, for the intersection the set of non-terminals is the Cartesian product $\{YZ, Y'Z, Y''Z, YZ', Y'Z', Y''Z'\}$, and the free productions

[6]Recall that the first component computes the ground instances of terms obtained by narrowing.

are $\{Y'Z' \Rightarrow s(Y''Z), Y'Z' \Rightarrow 0, Y''Z' \Rightarrow s(YZ)\}$. The packs of productions are constructed such that when a synchronization were possible in the initial grammars, it is still possible in the intersection. More precisely, for each pack of productions of G_1 (resp G_2) that deals with Y or Y' or Y'' (resp Z or Z'), we create a new pack in G_3. We get $\{X \Rightarrow X', YZ \Rightarrow Y'Z\}_1, \{X \Rightarrow X', YZ' \Rightarrow Y'Z'\}_1$ from the packs of G_1 and $\{YZ \Rightarrow YZ', T \Rightarrow T'\}_1, \{Y'Z \Rightarrow Y'Z', T \Rightarrow T'\}_1, \{Y''Z \Rightarrow Y''Z', T \Rightarrow T'\}_1$ from the packs of G_2. The axiom of G_3 is $((X,0),(YZ,0),(T,0))$.

Using the pack $\{X \Rightarrow X', YZ \Rightarrow Y'Z\}_1$, the axiom is derived into $((X',1),(Y'Z,1),(T,0))$. Now the pack $\{Y'Z \Rightarrow Y'Z', T \Rightarrow T'\}_1$ cannot be applied because the control numbers of $Y'Z$ and T are not equal, and none other production can derive $Y'Z$. The axiom can also be derived using the pack $\{YZ \Rightarrow YZ', T \Rightarrow T'\}_1$, but we get the same conclusion. Thus the language recognized by G_3 is empty. This problem is solved by considering pairs of integers as control in G_3, the first (resp. second) field being incremented when applying a pack that comes from G_1 (resp. G_2). So the packs of productions coming from G_2 must have 2 as rank (and are then $\{YZ \Rightarrow YZ', T \Rightarrow T'\}_2, \{Y'Z \Rightarrow Y'Z', T \Rightarrow T'\}_2, \{Y''Z \Rightarrow Y''Z', T \Rightarrow T'\}_2$. The axiom is now $((X,(0,\bot)),(YZ,(0,0)),(T,(\bot,0)))$. $\bot$ means that this field of the control is not used by the non-terminal. A possible derivation for G_3 is
$((X,(0,\bot)),(YZ,(0,0)),(T,(\bot,0))) \Rightarrow ((X',(1,\bot)),(Y'Z,(1,0)),(T,(\bot,0)))$
Now $\{Y'Z \Rightarrow Y'Z', T \Rightarrow T'\}_2$ is applicable and we get

$$
\begin{aligned}
&((X',(1,\bot)),(Y'Z',(1,1)),(T',(\bot,1)))\\
&\quad \Rightarrow^*_{[free-prods.]} (s((X,(1,\bot))), s((Y''Z,(1,1))), s(s((T,(\bot,1)))))\\
&\quad \Rightarrow (s((X,(1,\bot))), s((Y''Z',(1,2))), s(s((T',(\bot,2)))))\\
&\quad \Rightarrow^*_{[free-prods.]} (s((X,(1,\bot))), s(s((YZ,(1,2)))), s(s(s(s((T,(\bot,2)))))))\\
&\quad \Rightarrow (s((X',(2,\bot))), s(s((Y'Z,(2,2)))), s(s(s((T,(\bot,2)))))))\\
&\quad \Rightarrow (s((X',(2,\bot))), s(s((Y'Z',(2,3)))), s(s(s((T',(\bot,3)))))))\\
&\quad \Rightarrow^*_{[free-prods.]} (s(0), s(s(0)), s(s(s(s(0)))))
\end{aligned}
$$

For lack of space, the general algorithm to compute intersection is not given here. See [10]. Since the external synchronization property is preserved when computing intersection, we can do it incrementally, and next we can prove:

Lemma 5.5 Emptiness of languages recognized by TTSG's built from unification problems is decidable.

Thus we get the decidability result:

Theorem 5.6 *The satisfiability of linear equations in theories given as confluent, constructor based, linear, without σ_{in}, without nested functions in rhs's, rewrite systems is decidable. Moreover the set of solutions can be expressed by a tree tuple synchronized grammar.*

Example 5.7 Let $R = \{0+x \to x,\ x+0 \to x,\ s(x)+s(y) \to s(s(x+y)),\ s(x)+p(y) \to x+y,\ p(x)+s(y) \to x+y,\ p(x)+p(y) \to p(p(x+y))\}$ that defines the addition in positive and negative integers. This rewrite system does not satisfy the restrictions given in the previous works [7, 11, 2, 12, 9, 1, 8, 4] but satisfies ours. Therefore we are able to solve linear equations modulo this theory.

Example 5.8 Let $\begin{array}{lll} r_1 & f(c(x,x'),c(y,y')) & \rightarrow c(f(x,y'),f(x',y)) \\ r_2 & f(0,0) & \rightarrow 0 \end{array}$

This system provides an idea of the expressiveness of TTSG's because when solving the equation $f(x,y) = z$, the set of possible instantiations of x and y are the binary trees such that the instance of x is the symmetric of that of y. For example if we consider the following narrowing derivation issued from $f(x,y)$:

$$\begin{array}{ll} f(x,y) & \\ \rightsquigarrow_{[\epsilon, r_1, x \mapsto c(x_1,x_1'), y \mapsto c(y_1,y_1')]} & c(f(x_1,y_1'),f(x_1',y_1)) \\ \rightsquigarrow_{[1, r_1, x_1 \mapsto c(x_2,x_2'), y_1' \mapsto c(y_2,y_2')]} & c(c(f(x_2,y_2'),f(x_2',y_2)),f(x_1',y_1)) \\ \rightsquigarrow^*_{[r_2]} & c(c(0,0),0) \end{array}$$

the generated substitution is $x \mapsto c(c(0,0),0)$, $y \mapsto c(0,c(0,0))$. Since this rewrite system satisfies all our restrictions, our method will be able to compute a TTSG that recognized the solutions, i.e. the symmetric trees.

6 Related Decidability Results and Conclusion

In the rewrite domain, some authors have already established decidability results for unifiability, assuming some restrictions on the TRS. The first result imposed that the rewrite system is ground. J.-M. Hullot has extended it [7] to rewrite systems whose rhs's are either variables or ground terms (S. Mitra in [11] allows that the rhs's are data-terms). Actually these results are very restrictive because they forbid recursivity. In [1], J. Christian defines a new criterion: every rewrite rule lhs is flat ($f(s_1,\ldots,s_n)$ is flat if $\forall i \in [1,n]$, s_i is either a variable or a ground data-term) and the rewrite rules are oriented by a well founded ordering. H. Comon, M. Haberstrau and J.-P. Jouannaud in [2] show that decidability also holds for shallow rewrite systems (the sides of rewrite rules have variables occurring at depth at most one). R. Nieuwenhuis in [12] extends the shallow theories to standard theories that allow non-shallow variables. The restriction of D. Kapur and P. Narendran in [9], extended in [11] imposes that for every rule, every subterm of the rhs having a function symbol on top, is a strict subterm of the lhs. For all these restrictions the theory is finitary i.e. there always exists a finite complete set of unifiers. Most decidability proofs are thus based on the fact that there exists a complete narrowing strategy whose search space is always finite.

As concerns non finitary theories, a decidability result is given by Mitra in [11] for constructor-based rewrite systems, assuming that for every function symbol f there is at most one rewrite rule among the rules defining f, that does not have a data-term as rhs. Moreover this rhs must contain only one function symbol and the subterm rooted by this function is flat in the sense of [1]. Thanks to the notion of iterated-substitution, he is able to represent finitely the infinite set of unifiers and decide unifiability. In [8], Y. Kaji, T. Fujiwara and T. Kasami give a procedure that, when it terminates, decides unifiability by means of tree automata. They assume linearity for the goal, right linearity and (nearly) left linearity for the TRS. Unfortunately, their procedure does not represent the set of solutions, and does not terminate for an example like $\{s(x)+y \rightarrow s(x+y), 0+y \rightarrow y\}$ because of the superposition of $s(x)$ with $s(x+y)$. In [4] H. Faßbender and S. Maneth give a decision procedure for unifiability, without representing the set of solutions. But they need very strong restrictions : only one function can be defined and every constructor (as well as the function) is monadic.

In opposite to these results, we can solve only linear goals, but our procedure can decide unifiability for an example like example 5.7 whereas no other work can.

In the future it would be nice to use TTSG's to deal with disunification problem i.e. finding the substitutions that are not solution of a given equation. This may be achieved if it is possible to compute the set minus between two languages recognized by TTSG's. Another way may consist in studying the place of TTSG's in the known hierarchies of tree grammars. Thus, we would know more precisely which kind of problems can be treated with TTSG's.

Acknowledgements

We would like to thank A. Bockmayr, H. Comon, A. Despland and E. Domenjoud for helpful discussions.

References

[1] J. Christian. Some Termination Criteria for Narrowing and E-Unification. In Saragota Springs, editor, *CADE, Albany (NY, USA)*, volume 607 of *LNAI*, pages 582–588. Springer-Verlag, 1992.

[2] H. Comon, M. Haberstrau, and J.-P. Jouannaud. Syntacticness, Cycle-Syntacticness and Shallow Theories. *Information and Computation*, 111(1):154–191, 1994.

[3] N. Dershowitz and J.-P. Jouannaud. Rewrite Systems. In J. Van Leuven, editor, *Handbook of Theoretical Computer Science*. Elsevier Science Publishers, 1990.

[4] H. Faßbender and S. Maneth. A strict border for the decidability of E-unification for recursive functions. In *proceedings of the intern. Conf. on Algebraic and Logic Programming. To appear.*, 1996.

[5] R. Gilleron, S. Tison, and M. Tommasi. Some new decidability results on positive and negative set constraints. In *LNCS*, volume 845, pages 336–351, 1994. First International Conference on Constraints in Computational Logics.

[6] Y. Guan, G. Hotz, and A. Reichert. Tree Grammars with Multilinear Interpretation. Technical Report FB14-S2-01, Fachbereich 14, 1992.

[7] J.-M. Hullot. Canonical Forms and Unification. In W. Bibel and R. Kowalski, editors, *CADE, Les Arcs (France)*, volume 87 of *LNCS*, pages 318–334. Springer-Verlag, 1980.

[8] Y. Kaji, T. Fujiwara, and T. Kasami. Solving a Unification Problem under Constrained Substitutions Using Tree Automata. In *Proc. Fourteenth Conference on FST & TCS, Madras, India*, volume 880 of *LNCS*, pages 276–287. Springer-Verlag, 1994.

[9] D. Kapur and P. Narendran. Matching, unification and complexity. *Sigsam Bulletin*, 21(4):6–9, November 1987.

[10] S. Limet and P. Réty. E-Unification by Means of Tree Tuple Synchronized Grammars. Technical Report 96-16, Laboratoire d'Informatique Fondamentale d'Orléans, 1996. Available by anonymous ftp at ftp-lifo.univ-orleans.fr.

[11] S. Mitra. *Semantic Unification for Convergent Rewrite Systems*. Phd thesis, Univ. Illinois at Urbana-Champaign, 1994.

[12] R. Nieuwenhuis. Basic Paramodulation and Decidable Theories. In *procedings of the 11th Annual IEEE Symposium on Logic in Computer Science, to appear*, 1996.