On Decidability of Unifiability modulo Rewrite Systems[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 to study the decidability of unifiability in theories represented by confluent constructor-based rewrite systems. We propose four additional restrictions on the rewrite system and the goal to be unified, and show that if only three of them are satisfied then unifiability is undecidable. Then we give a decision algorithm applicable when the four restrictions are satisfied. This algorithm generates a new kind of tree grammar as finite representation for the (possibly infinite) set of solutions.

1 Introduction

Constraint Programming has known an intensive development in the last decade, in particular with Constraint Logic Programming [8] because it is a powerful paradigm to prune significantly the search space when solving a problem and to represent the solutions. In this kind of programming it is essential to decide whether the constraint is satisfiable or not and to give a representation of its set of solutions. Some work have been done to find efficient algorithms for specific theories where the problem is decidable, whereas some other work have provided good decidable approximations of the problem for undecidable theories or when the search space is huge. Moreover, the way to represent finitely some infinite sets of solutions has been investigated.

In a theory T, solving symbolic constraints that contain only equalities amounts to solving T-unification problems. The satisfiability problem is called here unifiability problem. A lot of work have been done in this field, when the theory is represented by a set of rewrite rules, using techniques like narrowing. In this paper, we deal with confluent and constructor-based rewrite systems, propose four additional restrictions (in section 4) and show that if only three of them are satisfied then unifiability is undecidable. Then we give a decision algorithm when the four restrictions are satisfied (in section 5). This algorithm has the nice property of giving a finite representation of the set of solutions (even infinite) by means of a tree tuple synchronized grammar. Some authors have already used tree languages to solve constraints, for example in [5] for solving set constraints, but our grammars use the notion of synchronization that allows the generation of non-context free languages like $d(a^i(0), b^i(0), c^i(0))$. The synchronized grammars are not identical to the coupled context-free grammars of [6] because we need a fine control of the synchronization which is achieved thanks to a tuple of integers.

2 Preliminaries

We assume that the reader is familiar with standard definitions of one-sorted terms, substitutions, equations, rewrite systems and complete sets of unifiers (see [3]) and 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

[1] This paper is an extended abstract. Missing formal definitions and proofs are in the full paper [10].

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 $\boldsymbol{\rightarrow_R}$ 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 $\boldsymbol{\rightarrow^*_R}$. **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 $s = (\sigma t)[u \leftarrow \sigma r]$.

3 Related decidability results

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.

With regard to 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.

4 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 a variable renaming) then the mgu σ does not modify the variables of l.

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 constraint** : the constraint (also called **goal**) does not contain several occurrences of the same variable.

These four restrictions together allow non-finitary theories. The (even minimal) complete set of solutions and then also the narrowing search space may be infinite. In the opposite of [11], we may have several recursive rewrite rules defining the same function, and the rhs's may contain several function symbols.

To show that the above restrictions are necessary all together to get the decidability of unifiability, we prove that if one of them is removed, then there exists a rewrite system satisfying the three others, and that encodes a well-known undecidable problem, the Post correspondence problem (Post 1946) which definition is the following. Let A and C be finite disjoint alphabets and ϕ and ϕ' be two morphisms from A^* to C^*. The Post correspondence problem for ϕ and ϕ' consist in finding a non-empty sequence $\alpha \in A^+$ such that $\phi(\alpha) = \phi'(\alpha)$. It is known that exists no uniform algorithm for solving the Post correspondence problem. The problem remains undecidable when ϕ and ϕ' are injective.

We use the following code: given $A = \{a_1, \ldots, a_n\}$ and $C = \{c_1, \ldots, c_m\}$, to each $a_i \in A$ (resp. $c_i \in C$) we associate the unary symbol a_i (resp. c_i). The constant $\perp$ represents the end of words. Thus to each word $\omega \in C^*$ is associated a monadic term. In the proofs $\perp$ will be sometimes omitted to simplify the expressions. For each $i \in [1, n]$, ω_i and $\omega' i$ denote respectively $\phi(a_i)$ and $\phi'(a_i)$.

Lemma 4.1 *Linearity of rewrite rules is necessary.*

Proof. Let $R_1 =$

$$\left\{ \begin{array}{lllll} f(a_i(x)) & \rightarrow & \omega_i(f(x)) & f'(a_i(x)) & \rightarrow \ \omega'_i(f(x)) \mid i = 1, \ldots, n \\ f(\perp) & \rightarrow & \perp & f'(\perp) & \rightarrow \ \perp \\ h(x) & \rightarrow & c(x, f(x)) & h'(x) & \rightarrow \ c(x, f'(x)) \end{array} \right\}$$

R_1 respect all the restrictions but linearity (more precisely right-linearity). Obviously f encodes ϕ and f' encodes ϕ'. Then for any words $\alpha, \beta \in A^*$, the value of $h(\alpha)$ (resp. $h'(\beta)$) is $c(\alpha, \phi(\alpha))$ (resp. $c(\beta, \phi'(\beta))$). Thus $h(\alpha) = h'(\beta)$ implies $c(\alpha, \phi(\alpha)) = c(\beta, \phi'(\beta))$ and so $\alpha = \beta$ and $\phi(\alpha) = \phi'(\alpha)$ because c is a constructor. Thus solving $h(\alpha) \stackrel{?}{=} h'(\beta)$ amounts to solve the Post correspondence problem, when forbidding the case of the empty word.

If unifiability under restrictions 2. 3. and 4. can be decided, it can be for any linear constraint, then for all the constraints $C_i = h(a_i(x)) \stackrel{?}{=} h'(a_i(y))$ $(i = 1, \ldots, n)$. This amounts to decide the unifiability of $h(\alpha) \stackrel{?}{=} h'(\beta)$ without the empty word case. This is therefore impossible. □

Among our four restrictions, linearity is necessary. For the moment, only the right-linearity is proved necessary.

Lemma 4.2 *Forbidding σ_{in} is necessary.*

Proof. The following rewrite system comes from [4]. Let $R_2 =$

$$\left\{ \begin{array}{lll} f(a_i(x), y) & \rightarrow & w_i(f(x, a_i(y))) \mid i = 1, \ldots, n \\ f'(a_i(x), y) & \rightarrow & w'_i(f'(x, a_i(y))) \mid i = 1, \ldots, n \\ f(\perp, y) & \rightarrow & h(y) \quad f'(\perp, y) \rightarrow h(y) \end{array} \right\}$$

The function f encodes ϕ, and the second argument save the reverse of the word for which ϕ is computed, i.e. for any word $\alpha \in A^*$ $f(\alpha, \perp) \rightarrow^* \phi(\alpha)(f(\perp, \overline{\alpha})) \rightarrow \phi(\alpha)(h(\overline{\alpha}))$ where $\overline{\alpha}$

is the reverse of α. For example
$f(a_1(a_2(\perp)),\perp) \rightarrow \omega_1(f(a_2(\perp),a_1(\perp))) \rightarrow \omega_1(\omega_2(f(\perp,a_2(a_1(\perp))))) \rightarrow \omega_1(\omega_2(h(a_2(a_1(\perp)))))$
For the same reasons $f'(\beta,\perp) \rightarrow^* \phi'(\beta)(h(\overline{\beta}))$. Therefore $f(\alpha,\perp) = f'(\beta,\perp)$ implies $\phi(\alpha)(h(\overline{\alpha})) = \phi'(\beta)(h(\overline{\beta}))$, then $\phi(\alpha) = \phi'(\beta)$ and $\alpha = \beta$. Thus $\phi(\alpha) = \phi'(\alpha)$, i.e. solving $f(\alpha,\perp) \stackrel{?}{=} f'(\beta,\perp)$ amounts to solve the Post correspondence problem, when forbidding the empty word case. The conclusion of this proof is similar to the previous one, by considering the linear constraints $C_i = f(a_i(x),\perp) \stackrel{?}{=} f'(a_i(y),\perp)$, $i = 1,\ldots,n$. $\Box$

Lemma 4.3 *Forbidding nested functions is necessary.*

Proof. Let $R_3 =$

$$\left\{ \begin{array}{lcl} f(a_i(x),h(y)) & \rightarrow & w_i(f(x,g_i(y)))\,|\, i = 1,\ldots,n \\ f'(a_i(x),h(y)) & \rightarrow & w'_i(f'(x,g_i(y)))\,|\, i = 1,\ldots,n \\ f(\perp,h(y)) & \rightarrow & h(y), \quad f'(\perp,h(y)) \rightarrow h(y) \\ g_i(y) & \rightarrow & h(a_i(y))\,|\, i = 1,\ldots,n \end{array} \right\}$$

The proof looks like the previous one. $\Box$

Lemma 4.4 *Linearity of the constraint is necessary.*

Proof. Let $R_4 =$

$$\left\{ \begin{array}{lclllcl} f(a_i(x)) & \rightarrow & w_i(f(x)) & f'(a_i(x)) & \rightarrow & w'_i(f'(x))\,|\, i = 1,\ldots,n \\ f(\perp) & \rightarrow & \perp & f'(\perp) & \rightarrow & \perp) \end{array} \right\}$$

We consider the non-linear constraints $C_i = f(a_i(x)) \stackrel{?}{=} f'(a_i(x))$ for $i = 1,\ldots,n$. $\Box$

5 A decidability result

This section presents an algorithm to decide unifiability for linear constraints in theories induced by a confluent constructor-based rewrite system that satisfies restrictions 1 to 4. Here a constraint is an equality. This may be trivially extended to a linear conjunction of equalities. Let us give an intuitive idea of our approach.

Example 5.1 Consider the following TRS $f(s(s(x))) \xrightarrow{1} f(x)$, $f(p(x)) \xrightarrow{2} f(x)$, $f(0) \xrightarrow{3} 0$, $g(s(x)) \xrightarrow{4} s(g(x))$, $g(0) \xrightarrow{5} 0$ and the goal $f(g(x)) \stackrel{?}{=} 0$.

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 $\{(t_1,t_2)|g(t_1) \rightarrow^* t_2\}$. 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 $\{(t_1,t_2)|f(t_1) \rightarrow^* t_2\}$ and 0 can be viewed as the language (says $\mathcal{L}_3$) of 1-uple reduced to $\{(0)\}$. These languages can be described by a special kind of grammar called **tree tuple synchronized grammar** and denoted **TTSG**. The grammars are computed from the rewrite system and the goal.

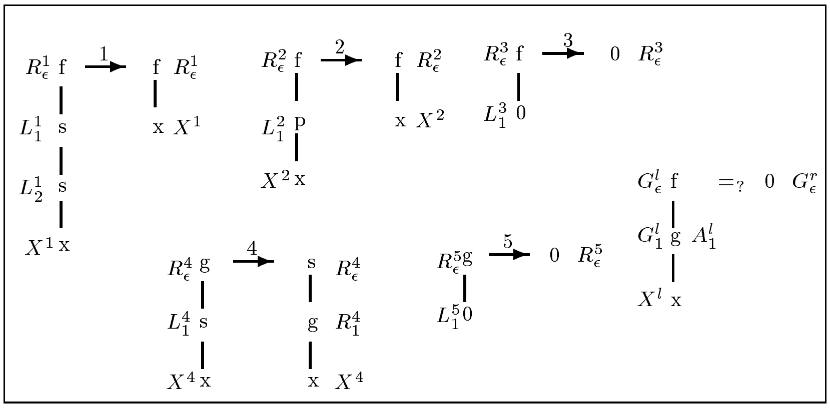
Once these three TTSG's are constructed, the initial goal is re-composed by two steps. First the languages $\mathcal{L}_1$ and $\mathcal{L}_2$ are combined to get the language $\mathcal{L}_4$ of the ground data-solutions of $f(g(x)) \stackrel{?}{=} y_3$. This is done by computing a special kind of intersection 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 $\{(t_1,t_2,t_3)|(t_1,t_2) \in \mathcal{L}_1$ and $(t_2,t_3) \in \mathcal{L}_2\}$. In other words, t_2 is the result of $g(x)$ when instantiating x by t_1, moreover t_2 belong to the definition domain of the function f, and t_3 is the result of $f(t_2)$, i.e. of $f(g(t_1))$. 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 $\{(t_1, t_2, t_3)|t_3 = 0$ and $(t_1, t_2, t_3) \in \mathcal{L}_4\}$. As t_1 is an instance of x, t_3 is the result of $f(g(t_1))$ and $t_3 = 0$, 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 $f(g(x)) \stackrel{?}{=} 0$.

5.1 Transformation of a TRS into TTSG's

Here is the first step of our method. The aim is to convert the TRS and the goal into several TTSG's. This step is illustrated by example 5.1. Recall that 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.

5.1.1 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 or syntactic unifications. To each non variable occurrence u of the lhs (resp. rhs) of each rule i is associated the non terminal L^i_u (resp. R^i_u), except when $u = \epsilon$, we associate R^i_u 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^l_u (resp G^r_u) 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^l_u (resp A^r_u) is associated to the arguments of function of the goal (here occurrence 1 of $f(g(x))$ to encode the variable y_2). g being a side of the goal, $ANT(g, u)$ denotes the additional non-terminal associated to the occurrence u of g.



5.1.2 Productions

There are two kinds of productions 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 the pairs of syntactically unifiable terms.

Recall that $t(u)$ denotes the symbol appearing in t at occurrence u. 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 $t(u)$. Thus in our example, we get :

$L_1^1 \Rightarrow s(L_2^1),\ L_2^1 \Rightarrow s(X^1),\ L_1^2 \Rightarrow p(X^2),\ L_1^3 \Rightarrow 0,\ R_\epsilon^3 \Rightarrow 0,$
$L_1^4 \Rightarrow s(X^4),\ R_\epsilon^4 \Rightarrow s(R_1^4),\ L_1^5 \Rightarrow 0,\ R_\epsilon^5 \Rightarrow 0,\ G_\epsilon^r \Rightarrow 0$

Consider again the example. From the correspondence result between rewriting and narrowing (lifting lemma [7]), the languages $\mathcal{L}_1$, $\mathcal{L}_2$ are the ground instances of the data-solutions computed by narrowing. For this reason we look for narrowing possibilities. A narrowing step can exist when a (sub)term of some rhs is unifiable with a lhs thanks to some mgu. In this case the resulting term and the instances of variables must be set at the same time. For that we need synchronized productions, i.e. productions that are applied together. For example let us study the narrowing steps issued from the rhs of rule 1 $f(x)$. We starts from the pair (R_ϵ^1, X^1) where R_ϵ^1 will express the terms issued by narrowing from $f(x)$, and X^1 the corresponding instances of x. $f(x)$ unifies with the lhs of rule 2 $f(p(x'))$ (the variable of rule 2 is renamed into x' to avoid conflicts of variables), therefore $f(x)$ narrows into $f(x')$ and at the same time x is instantiated by $p(x')$. The synchronized productions $\{R_\epsilon^1 \Rightarrow R_\epsilon^2, X^1 \Rightarrow L_1^2\}$ encodes this fact. They mean that R_ϵ^1 and X^1 can be derived at the same time, and only at the same time, into respectively R_ϵ^2 and L_1^2. So we have $(R_\epsilon^1, X^1) \Rightarrow (R_\epsilon^2, L_1^2) \Rightarrow (R_\epsilon^2, p(X^2))$, where X^2 encodes the variable of rule 2 and R_ϵ^2 encodes its rhs. The derivation of L_1^2 into $p(X^2)$ is achieved thanks to a free production, and provides the term that instantiates x.

More generally for all $r^i|_u$ and l^j syntactically unifiable, we create the **pack of productions** (i.e. the set of synchronized productions) $\{NT(r^i, u) \Rightarrow NT(l^j, \epsilon), NT(r^i, u.v_1) \Rightarrow NT(l^j, v_1), \ldots NT(r^i, u.v_n) \Rightarrow NT(l^j, v_n)\}$ where $v_1, \ldots v_n$ are the variable occurrences of $r^i|u$.[2] For our example, r^1 unifies with l^1, l^2 and l^3 which give the synchronized productions $\{R_\epsilon^1 \Rightarrow R_\epsilon^1, X^1 \Rightarrow L_1^1\}$, $\{R_\epsilon^1 \Rightarrow R_\epsilon^2, X^1 \Rightarrow L_1^2\}$ and $\{R_\epsilon^1 \Rightarrow R_\epsilon^3, X^1 \Rightarrow L_1^3\}$. r^2 unifies with l^1, l^2 and l^3 too so we get $\{R_\epsilon^2 \Rightarrow R_\epsilon^1, X^2 \Rightarrow L_1^1\}$, $\{R_\epsilon^2 \Rightarrow R_\epsilon^2, X^2 \Rightarrow L_1^2\}$ and $\{R_\epsilon^2 \Rightarrow R_\epsilon^3, X^2 \Rightarrow L_1^3\}$. Finally $r^4|_1$ unifies with l^4 and l^5 so we get $\{R_1^4 \Rightarrow R_\epsilon^4, X^4 \Rightarrow L_1^4\}$ and $\{R_1^4 \Rightarrow R_\epsilon^5, X^4 \Rightarrow L_1^5\}$.

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 g such that $g(u) = l^j(\epsilon)$ (in other words $g(u)(x_1, \ldots, x_n)$ unifies with l^j), we create the synchronized productions $\{NT(g, u) \Rightarrow NT(l^j, \epsilon), ANT(g, u.1) \Rightarrow NT(l^j, 1), \ldots, ANT(g, u.n) \Rightarrow NT(l^j, n)\}$. The language derived from the non-terminal $NT(g, u)$ will express the terms issued by narrowing from $g(u)(x_1, \ldots, x_n)$ while the languages derived from the $ANT(g, u.i)$ will express the instances of the fictitious variables x_i. In example 5.1, $f(y_2)$ unifies with l^1, l^2 and l^3, this gives the synchronized productions $\{G_\epsilon^l \Rightarrow R_\epsilon^1, A_1^l \Rightarrow L_1^1\}$, $\{G_\epsilon^l \Rightarrow R_\epsilon^2, A_1^l \Rightarrow L_1^2\}$ and $\{G_\epsilon^l \Rightarrow R_\epsilon^3, A_1^l \Rightarrow L_1^3\}$. $g(x)$ unifies with l^4 and l^5 so we get $\{G_1^l \Rightarrow R_\epsilon^4, X^l \Rightarrow L_1^4\}$ and $\{G_1^l \Rightarrow R_\epsilon^5, X^l \Rightarrow L_1^5\}$[3].

The languages we want to 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.

[2] 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.

[3] The argument of the function symbol g is a variable, thus we do not need a non-terminal A_2^l for it.

5.1.3 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. For example 5.1, 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.

$$\begin{aligned}(G^l_\epsilon, A^l_1) &\Rightarrow (R^1_\epsilon, L^1_1) \Rightarrow (R^1_\epsilon, s(L^1_2)) \Rightarrow (R^1_\epsilon, s(s(X^1))) \Rightarrow (R^2_\epsilon, s(s(L^2_1)))\\ &\Rightarrow (R^2_\epsilon, s(s(p(X^2)))) \Rightarrow (R^3_\epsilon, s(s(p(L^3_1)))) \Rightarrow (0, s(s(p(L^3_1)))) \Rightarrow (0, s(s(p(0))))\end{aligned}$$

This encodes the narrowing derivation

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

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. For each non-variable occurrence u of the lhs l (resp rhs r) of the goal such that $u = \epsilon$ or u is just under a function, let $\{v_1, \dots, v_n\}$ (exhaustive list) be the occurrences appearing under u and such that for each i, the branch $[u, v_i[$ contains only one function f and v_i is just under f, or v_i is a variable occurrence and the branch $[u, v_i[$ contains only constructors, we create the grammar Gr^l_u defined by the axiom $(NT(l,u),\ ANT(l,\ v_1),\ \dots,\ ANT(l,\ v_n)))$ (resp G^r_u defined by the axiom $(NT(r,\ u),\ ANT(r,\ v_1),\ \dots,\ ANT(r,\ v_n))))$.

5.1.4 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 to 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 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:
$((G^l_\epsilon, 0), (X^l, 0)) \Rightarrow ((R^1_\epsilon, 1), (L^1_1, 1)) \Rightarrow (c((R^1_1, 1), (R^1_2, 1)), (L^1_1, 1))$
$\Rightarrow (c((R^1_1, 1), (R^1_2, 1)), c((X^1, 1), (Y^1, 1))) \Rightarrow (c((R^1_\epsilon, 2), (R^1_2, 1)), c((L^1_1, 2), (Y^1, 1)))$
$\Rightarrow (c(c((R^1_1, 2), (R^1_2, 2)), (R^1_2, 1)), c((L^1_1, 2), (Y^1, 1)))$
$\Rightarrow (c(c((R^1_1, 2), (R^1_2, 2)), (R^1_2, 1)), c(c((X^1, 2), (Y^1, 2)), (Y^1, 1)))$
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.

Theorem 5.2 *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.*

5.2 General definition of TTSG's

In the following, NT is a finite set of non-terminal symbols and recall that $\mathcal{C}$ 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 subsection 5.3). 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 5.3 *A* **production** *is a rule of the form $X \Rightarrow t$ where $X \in NT$ and $t \in T_{\mathcal{C} \cup NT}$. 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 5.4 *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 that may contain 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 $X_1, \ldots X_n$ occur at the same time in a computation 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.

5.3 Intersection of TTSG's over one component

This subsection describes the second step of our method. Let us consider again example 5.1. 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 5.1, 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 data-bases).

Definition 5.5 *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* $\{(t_1, \ldots, t_{n_1}, t_{n_1+1}, \ldots, t_{n_1+n_2-1}) |\ (t_1, \ldots, t_{n_1}) \in E_1$ *and* $(t_{n_1+1}, \ldots, t_{n_1+k_2-1}, t_{k_1}, t_{n_1+k_2}, \ldots, t_{n_1+n_2-1}) \in E_2\}$. $\diamond$

For example the one component 2, 1 intersection of the sets of pairs $E_1 = \{(0, s(0)), (s(0), 0)\}$ 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 5.6 *Emptiness of intersection of languages recognized by TTSG's is undecidable.*

It is proved by encoding the Post Correspondance Problem. In fact, 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 TTSG's that come 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**, which is the case for TTSG's coming from a unification problem. 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 5.7 *The first component of any TTSG produced from the unification problem (which computes the terms obtained by narrowing) has the external synchronization property.*

Example 5.8 The intersection over one component will be intuitively presented by the present example. By the way, the necessity of tuples of controle will be explained. This example does not come from a unification problem, but it is easier to understand. Remark that all components are 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 a new TTSG G_3 from the previous ones 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, let us 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 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 $((X,(0,\perp),(YZ,(0,0)),(T,(\perp,0))$. $\perp$ means that this field of the control is not used by the non-terminal. A possible derivation for G_3 is $((X,(0,\perp),(YZ,(0,0)),(T,(\perp,0)) \Rightarrow ((X',(1,\perp),(Y'Z,(1,0)),(T,(\perp,0))$. Now $\{Y'Z \Rightarrow Y'Z', T \Rightarrow T'\}_2$ is applicable and we get $((X',(1,\perp)),(Y'Z',(1,1)),(T',(\perp,1)))$ and then $(s(X,(1,\perp)), s(Y''Z,(1,1)), s(s(T,(\perp,1)))) \Rightarrow (s(X,(1,\perp)), s(Y''Z',(1,2)), s^2(T',(\perp,2)))$ $\Rightarrow^* (s(X,(1,\perp)), s^2(YZ,(1,2)), s^4(T,(\perp,2))) \Rightarrow (s(X',(2,\perp)), s^2(Y'Z,(2,2)), s^4(T,(\perp,2)))$ $\Rightarrow (s(X',(2,\perp)), s^2(Y'Z',(2,3)), s^4(T',(\perp,3))) \Rightarrow^* (s(0), s(s(0)), s(s(s(s(0)))))$

For lack of space, the general algorithm to compute intersection is not given here (see [10]). The point now is to prove that if one component (different from i_2) of G_2 is externally synchronized then the corresponding component of G_3 keeps this property. This is necessary to be able to compute incrementally several intersections.

Lemma 5.9 *Let G_1 be a TTSG whose i_1^{th} component is externally synchronized and G_2 another TTSG whose j_2^{th} component is too. Let G_3 the one component i_1, i_2 intersection ($i_2 \neq j_2$) of G_1 and G_2. The component that corresponds to j_2 (i.e. $n_1 + j_2$ if $j_2 < i_2$ and $n_1 + j_2 - 1$ otherwise) in G_3 is still externally synchronized.*

5.4 Decidability

Given a computation tp of a TTSG, a Class is an exhaustive set of non-terminal occurrences of tp that have identical control tuples for some field (of rank k). A TransClass is the transitive closure of Classes when k is running. The notion of TransClass is interesting because different TransClasses can be derived independently.

Lemma 5.10 *The size of the TransClasses of a TTSG G that comes from a unification problem is not greater than $card(NT)^{n+1}$, where NT is the set of non-terminals initially created from the unification problem and n is the number of intersections achieved to build G.*

Lemma 5.11 *Emptiness of languages recognized by TTSG's built from unification problems is decidable.*

The reasoning is similar to that used for a regular tree grammar. Thus we get :

Theorem 5.12 *The satisfiability of linear equations in theories given as confluent, constructor based, linear, without σ_{in} (See section 4), 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 5.13 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 work [7, 11, 2, 12, 9, 1] but satisfies ours. Therefore we are able to solve linear equations modulo this theory.

6 Conclusion

We have presented an original approach using Tree Tuple Synchronized Grammars (TTSG) to solve linear equations modulo theories given as rewrite systems satisfying some restrictions. This approach allows to represent infinite sets of solutions in a finite way, and provides a decision procedure for unifiability. Moreover we have shown that each restriction is needed to decide unifiability. In the future it would be nice to extend TTSG's to deal with non-linear goals, the decidability will then be lost but it is often interesting to have a finite representation of the set of solutions. Another way may consist in studying the place of TTSG's in the known hierarchy of tree grammars.

References

[1] J. Christian. Some Termination Criteria for Narrowing and E-Unification. In S. Springs, editor, *Proceedings 11th International Conference on Automated Deduction, Albany (NY, USA)*, volume 607 of *Lecture Notes in Artificial Intelligence*, 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 North-Holland, 1990.

[4] E. Domenjoud. Combination techniques for non-disjoint equational theories. In D. Kapur, editor, *proceedings of the 12th internationnal Conference on Automated Deduction*, LNCS, pages 267–281. Springer-Verlag, 1994.

[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, *Proceedings 5th International Conference on Automated Deduction, Les Arcs (France)*, volume 87 of *LNCS*, pages 318–334. Springer-Verlag, July 1980.

[8] J. Jaffar and M. J. Maher. Constraint Logic Programming: a survey. *Journal of Logic Programming*, 19-20:503–581, 1994.

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

[10] S. Limet and P. Réty. On decidability of unification modulo rewrite systems. Technical report, Laboratoire d'Informatique Fondamentale d'Orléans, 1996 to appear (contact authors to get it).

[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.