A New Result about the Decidability of the Existential One-Step Rewriting Theory*

Sébastien Limet and Pierre Réty

LIFO, Université d'Orléans (France), {limet|rety}@lifo.univ-orleans.fr

Abstract. We give a decision procedure for the whole existential fragment of one-step rewriting first-order theory, in the case where rewrite systems are linear, non left-left-overlapping (i.e. without critical pairs), and non ϵ-left-right-overlapping (i.e. no left-hand-side overlaps on top with the right-hand-side of the same rewrite rule[2]). The procedure is defined by means of tree-tuple synchronized grammars.

1 Introduction

Given a signature Σ, the theory of one-step rewriting for a finite rewrite system is the first order theory over the universe of ground Σ-terms that uses the only predicate symbol $\rightarrow$, where $x \rightarrow y$ means x rewrites into y by one step.

It has been shown undecidable in [11]. Sharper undecidability results have been obtained for some subclasses of rewrite systems, about the $\exists^*\forall^*$-fragment [10, 8] and the $\exists^*\forall^*\exists^*$-fragment [12].

It has been shown decidable for the positive existential fragment [9], in the case of unary signatures [3], in the case of linear rewrite systems whose left and right members do not share any variables [2][3], and for the whole existential fragment in the case of shallow rewrite systems without collapsing rules [1].

In comparison with [1], our result also excludes collapsing rules because of the stronger restriction: no ϵ-left-right-overlaps. On the other hand, it includes non shallow rewrite rules. In comparison with [2], linearity of rewrite rules is also assumed, but it includes the rewrite rules whose members share variables. To get this possibility using a technique based on tree automata or tree grammars, we must ensure that the two occurrences of each shared variable are replaced by identical terms. This cannot be performed by ground tree transducers. Automata with equality and disequality constraints can, but they cannot perform the closure of (one-step) rewriting by context application.

On the other hand tree-tuple synchronized grammars (*TTSG* for short) can correctly perform both variable instantiations and context applications, thanks to synchronizations. TTSG's have been introduced by the authors to solve some equational unification [4] and disunification [6] problems.

---

* A full version of this paper is available in [5].

[2] Formally, for each rewrite rule $l -> r$, the two terms l and r' are not unifiable, where r' is a renaming of r such that $Var(l) \cap Var(r') = \emptyset$.

[3] Even the theory of several-step rewriting is decidable.

Given an existential formula in the prenex form, our decision procedure consists in the following steps:

First: because of the restrictions, every predicate of the form $x \rightarrow x$ has no solutions (see Lemma 3 in Section 3.3): they are replaced by the predicate without solutions $\perp$. Next, the formula is transformed into a disjunction of conjunctions of items of the form $x \rightarrow y$ or $\neg(x \rightarrow y)$[4]. The solutions of $x \rightarrow y$ and of $\neg(x \rightarrow y)$ are tree-pair languages, which can both be generated by TTSG's (see Section 3).

Second: the solutions of a conjunctive factor are obtained by making natural joins (called intersection in this paper). Let $x \rightarrow y \wedge C$ (or $\neg(x \rightarrow y) \wedge C$) be a conjunctive factor and assume that the solutions of C have been computed already, as a tree-tuple language. The size of tuples is the number of distinct variables appearing in C. Three kinds of intersections are needed: over n components, $n \in \{0, 1, 2\}$, according to the number of variables shared by $x \rightarrow y$ and C. Of course, if $n = 0$ this is the cartesian product.

Third: the solutions of the entire formula (without quantifiers) are obtained by making unions of languages of tree-tuples of different sizes. By making cartesian products with the language of all ground terms, we can lengthen tuples so that they all have the same length[5].

Fourth: the validity of the existentially quantified formula is tested thanks to an emptiness test performed on the solutions of the formula without quantifiers.

2 Overview of TTSG's

This section briefly recalls notions and existing results about TTSG's. For a more formal presentation and proofs see [4]. We use the classic notions of terms and term rewriting systems. Let us just precise a few notations. For any occurrences u, v, $u < v$ means that u is a strict prefix of v, i.e. $v = u.w$ for some $w \neq \epsilon$. For any term t, $O(t)$ is the set of occurrences of t and $t(u)$ is the symbol of t that appears at occurrence u. Moreover we extend the notion of occurrences to tuples in a natural way i.e. $(t_1, \ldots, t_n)|_{i.u} = t_i|_u$ if $i \in [1, n]$. The arity of any symbol f is denoted by $ar(f)$. $\Sigma_{>i}$ contains the symbols of Σ whose arity is greater than i. *lhs* means left-hand-side.
A *TTSG* is a tree-tuple grammar whose terminals are the symbols of Σ, and that contains:

- *free productions*, i.e. productions like those of a regular tree language: $X \Rightarrow f(X_1, \ldots, X_n)$.
- *synchronized productions*: $\{X_1 \Rightarrow Y_1, \ldots, X_n \Rightarrow Y_n\}$, which means that X_1 is replaced by Y_1,...,X_n by Y_n at the same time (and only at the same

[4] The symbols of Σ are not allowed in formulas.

[5] Lengthening tuples can be necessary even if the lengths are equal. Consider for example two tree-pair languages. If the one provides the valuations of the variables x, y while the other provides those of x, z, we have to extend both into triple languages providing the valuations of x, y, z, before making their union.

time). They are always empty, i.e. they do not generate any terminals. A set of productions synchronized together, like $\{X_1 \Rightarrow Y_1, \ldots, X_n \Rightarrow Y_n\}$, is called *a pack of synchronized productions*.

A TTSG is denoted by (Σ, NT, P, I) where NT is the set of non-terminals, P is the set of productions, and I is the axiom. Languages generated by TTSG's are called *synchronized languages*. Note that regular tree-tuple languages are a particular case of synchronized languages.

Example 1. Let $\Sigma = \{f, g, a, b\}$ where f, g are monadic symbols and a, b are constants. Consider the TTSG whose productions are:

$$\{X \Rightarrow F,\ X' \Rightarrow G\},\ \{X \Rightarrow A,\ X' \Rightarrow B\}$$
$$F \Rightarrow f(X),\ G \Rightarrow g(X'),\ A \Rightarrow a,\ B \Rightarrow b$$

Starting from the axiom (X, X'), we can derive

$$(X, X') \Rightarrow (F, G) \Rightarrow^* (f(X), g(X')) \Rightarrow (f(F), g(G)) \Rightarrow^* (f(f(X)), g(g(X'))$$
$$\Rightarrow^* (f^n(X), g^n(X')) \Rightarrow (f^n(A), g^n(B)) \Rightarrow^* (f^n(a), g^n(b))$$

Thus we get the language of pairs $\{(f^n(a), g^n(b)) \mid n \in I\!N\}$.

However when dealing with non monadic signatures, there might be confusions when some non-terminals occur several times in a tree. For example if X occurred twice and X' also did in a tree-tuple derived from the axiom, there would be two possible combinations of synchronizations. To avoid this ambiguity, a *control integer* is associated to each non-terminal occurrence. To make understanding easier, the way it works is explained in the sequel by means of Example 3.

A TTSG is *without internal synchronizations* if for each tree-tuple tt derived from the axiom and for each pack of productions PP that can be applied to tt, PP does not apply to several occurrences of the same component of tt.

The TTSG defined in the above example is without internal synchronizations, whereas the below one is with.

Example 2. Consider the 1-tuple language defined by the TTSG whose productions are: $\{X \Rightarrow F,\ Y \Rightarrow A\},\ \{X \Rightarrow A,\ Y \Rightarrow A\},\ F \Rightarrow f(X, Y),\ A \Rightarrow a$

Starting from the axiom F we have the derivation: $F \Rightarrow f(X, Y)$

Now, the first (and also the second) pack of synchronized productions applies to the single component $f(X, Y)$, and derives X and Y at the same time. Since X and Y belong to the same component there are internal synchronizations.

The *cartesian product* of two TTSG's is a TTSG, obtained by making the union of their productions and the concatenation of their axioms.

The *union* of two TTSG's that generate tuples of the same length is still a TTSG. Let $G = (\Sigma, NT, P, (A_1, \ldots, A_n))$ and $G' = (\Sigma', NT', P', (A'_1, \ldots, A'_n))$ s.t. $NT \cap NT' = \emptyset$. Then $G \cup G' = (\Sigma \cup \Sigma', NT \cup NT' \cup \{B_1 \ldots, B_n\}, P \cup P' \cup \{\{B_1 \Rightarrow A_1, \ldots, B_n \Rightarrow A_n\},\ \{B_1 \Rightarrow A'_1, \ldots, B_n \Rightarrow A'_n\}\}, (B_1, \ldots, B_n))$.

The *intersection* over p components consists in making the cartesian product, except that p fixed components in each grammar are merged (by making intersections) together. It is the same as the natural join in relational data-bases. Performing the intersection of two TTSG's is much more difficult than making

their union. It has been shown in [4] that the intersection over one component of two TTSG's is still a TTSG if at least one of the two components to be merged has no internal synchronizations. In this case, the resulting component has none either. The resulting TTSG needs a more precise control, carried out thanks to integer pairs[6] instead of single integers. Fortunately, the TTSG's defined in the sequel have no internal synchronizations.

The *emptiness* of TTSG's is decidable [4], assuming that no non-terminal appears more than once with the same control value. Some TTSG's defined in the sequel do not satisfy this property, but they will if modifying them slightly, as explained in [5].

3 Languages for positive and negative predicates

3.1 Basic languages

Let us define a few basic regular or synchronized languages we need in the following. The sets of non-terminals used by the below grammars are pairwise disjoint in order to avoid confusions when considering the union of their productions.

1. The language of ground terms.
   This regular language is generated by the grammar GrG whose axiom is G (G for ground) and whose productions are $G \Rightarrow f(G, \ldots, G)$ for each $f \in \Sigma$.
2. The language of pairs of arbitrary ground terms.
   This regular language is generated by the grammar $GrTT'$ whose axiom is (T, T') (T for term) and whose productions are:
   $$\left.\begin{array}{l} T \Rightarrow f(T, \ldots, T) \\ T' \Rightarrow f(T', \ldots, T') \end{array}\right\} \text{ for each } f \in \Sigma$$
3. The language of pairs of identical ground terms.
   To get two identical terms, synchronizations are obviously necessary. This synchronized language is generated by the grammar $GrTT$ whose axiom is (T_1, T'_1), whose synchronized productions are:
   $$\{T_i \Rightarrow T_f,\ T'_i \Rightarrow T'_f\} \text{ for each } i \in [1, MaxArity] \text{ and each } f \in \Sigma$$
   and whose free productions are:
   $$T_f \Rightarrow f(T_1, \ldots, T_n),\ \ T'_f \Rightarrow f(T'_1, \ldots, T'_n) \quad \text{for each } f \in \Sigma$$
   where $MaxArity$ is the maximal arity of symbols of Σ, and $T_1, T_2, \ldots, T_f, \ldots$ are non-terminals (respectively $T'_1, T'_2, \ldots, T'_f, \ldots$).

   Example 3. Let $\Sigma = \{a, b, f\}$ where a, b are constants and f is a binary symbol. The productions are:
   $$\{T_1 \Rightarrow T_a,\ T'_1 \Rightarrow T'_a\},\ \ \{T_1 \Rightarrow T_b,\ T'_1 \Rightarrow T'_b\},\ \ \{T_1 \Rightarrow T_f,\ T'_1 \Rightarrow T'_f\},$$
   $$\{T_2 \Rightarrow T_a,\ T'_2 \Rightarrow T'_a\},\ \ \{T_2 \Rightarrow T_b,\ T'_2 \Rightarrow T'_b\},\ \ \{T_2 \Rightarrow T_f,\ T'_2 \Rightarrow T'_f\},$$
   $$T_a \Rightarrow a,\ T'_a \Rightarrow a,\ T_b \Rightarrow b,\ T'_b \Rightarrow b,\ T_f \Rightarrow f(T_1, T_2),\ T'_f \Rightarrow f(T'_1, T'_2)$$

[6] And integer tuples when making several intersections incrementally.

From the axiom (T_1, T'_1) we can derive (the control is written just above the non-terminals):

$$(\overset{0}{T_1},\ \overset{0}{T'_1}) \Rightarrow (\overset{1}{T_f},\ \overset{1}{T'_f}) \Rightarrow^* (f(\overset{1}{T_1}, \overset{1}{T_2}),\ f(\overset{1}{T'_1}, \overset{1}{T'_2}))$$
$$\Rightarrow (f(\overset{2}{T_f}, \overset{1}{T_2}),\ f(\overset{2}{T'_f}, \overset{1}{T'_2})) \Rightarrow^* (f(f(\overset{2}{T_1}, \overset{2}{T_2}), \overset{1}{T_2}),\ f(f(\overset{2}{T'_1}, \overset{2}{T'_2}), \overset{1}{T'_2}))$$
$$\Rightarrow (f(f(\overset{3}{T_a}, \overset{2}{T_2}), \overset{1}{T_2}),\ f(f(\overset{3}{T'_a}, \overset{2}{T'_2}), \overset{1}{T'_2})) \Rightarrow^* (f(f(a, \overset{2}{T_2}), \overset{1}{T_2}),\ f(f(a, \overset{2}{T'_2}), \overset{1}{T'_2}))$$

Free productions leave control unchanged, whereas synchronized productions increase it into a value not yet used. At this stage, T_2 appears twice in the first component, while T'_2 appears twice in the second one. Of course the leftmost T_2 must not be derived together with the rightmost T'_2 into identical terms. This is the role of control integers: only non-terminals that have the same control integer can be derived (synchronized) together. So:

$$\Rightarrow (f(f(a, \overset{4}{T_b}), \overset{1}{T_2}),\ f(f(a, \overset{4}{T'_b}), \overset{1}{T'_2})) \Rightarrow^* (f(f(a, b), \overset{1}{T_2}),\ f(f(a, b), \overset{1}{T'_2}))$$
$$\Rightarrow (f(f(a, b), \overset{5}{T_a}),\ f(f(a, b), \overset{5}{T'_a})) \Rightarrow^* (f(f(a, b), a),\ f(f(a, b), a))$$

4. The language of contexts.
   A *context* is a ground term that contains one hole denoted by $\perp$ ($\perp \notin \Sigma$). This regular language is generated by the grammar GrC whose axiom is C (C for context) and whose productions are those of GrT plus:

   $$C \Rightarrow \perp$$
   $$C \Rightarrow f(T, \ldots, T, C, T, \ldots, T) \left\{ \begin{array}{l} \text{for each } f \in \Sigma_{>0} \\ \text{and each position of } C \text{ as an argument of } f \end{array} \right.$$

5. The language of pairs of identical contexts.
   The hole of the second component is denoted by $\perp'$ instead of $\perp$. This synchronized language is generated by the grammar $GrCC$ whose axiom is (C_1, C'_1), whose productions are those of $GrTT$ plus the synchronized productions:

   $$\{C_1 \Rightarrow C_\perp,\ C'_1 \Rightarrow C'_\perp\}$$
   $$\{C_1 \Rightarrow C_{f,i},\ C'_1 \Rightarrow C'_{f,i}\} \text{ for each } f \in \Sigma_{>0} \text{ and each } i \in \{1, \ldots, ar(f)\}$$

   and the free productions:

   $$C_\perp \Rightarrow \perp,\ C'_\perp \Rightarrow \perp'$$
   $$\left. \begin{array}{l} C_{f,i} \Rightarrow f(T_1, \ldots, T_{i-1}, C_1, T_{i+1}, \ldots, T_n) \\ C'_{f,i} \Rightarrow f(T'_1, \ldots, T'_{i-1}, C'_1, T'_{i+1}, \ldots, T'_n) \end{array} \right| \begin{array}{l} \text{for each } f \in \Sigma_{>0} \\ \text{and each } i \in \{1, \ldots, ar(f)\} \end{array}$$

   where $C_1, C_\perp, C_{f,i}, \ldots$ are non-terminals (respectively $C'_1, C'_\perp, C'_{f,i}, \ldots$).

6. The language of pairs of different ground terms.
   The idea consists in generating a branch that contains at least one clash (only the first one is forced), and anything elsewhere. This synchronized language is generated by the grammar $GrT\overline{T}$ whose axiom is (D, D') (D for different), whose productions are those of $GrTT'$ plus the synchronized productions (G_f is for generating every term whose root symbol is f):

   $$\{D \Rightarrow D_{f,i},\ D' \Rightarrow D'_{f,i}\} \text{ for each } f \in \Sigma_{>0} \text{ and each } i \in \{1, \ldots, ar(f)\}$$
   $$\{D \Rightarrow G_f,\ D' \Rightarrow G'_g\} \text{ for each } f, g \in \Sigma \text{ s.t. } f \neq g$$

and the free productions:

$$\left.\begin{array}{r} D_{f,i} \Rightarrow f(T,\ldots,T,D,T,\ldots,T) \\ D'_{f,i} \Rightarrow f(T',\ldots,T',D',T',\ldots,T') \end{array}\right| \begin{array}{l}\text{for each } f \in \Sigma_{>0},\ i \in \{1,\ldots,ar(f)\},\\ \text{s.t. } D\ (D') \text{ is the } i^{th} \text{ argument of } f\end{array}$$

$$\left.\begin{array}{r} G_f \Rightarrow f(T,\ldots,T) \\ G'_f \Rightarrow f(T',\ldots,T') \end{array}\right\} \text{ for each } f \in \Sigma$$

where $D_{f,i},\ldots,G_f,\ldots$ are non-terminals (respectively $D'_{f,i},\ldots,G'_f,\ldots$).

We can also deal with formulas that contain the equality predicate since the solutions of $x \doteq y$ are generated by $GrTT$, and those of $\neg(x \doteq y)$ by $GrT\overline{T}$.

3.2 The language $\mathcal{L}$ of solutions of $x \rightarrow y$

Now, the grammar Gr that generates $\mathcal{L}$ can be easily defined. Roughly speaking, we have to generate all $(C[\sigma l], C[\sigma r])$.

1. The language of rewrite rules.
   This is the set of pairs of terms (l, r) where $l \rightarrow r$ is a rewrite rule whose variables are replaced in the following way: each variable x of l (resp. x of r) is replaced by the new symbol X (resp. X'). Note that $X, X' \notin \Sigma$. This language is finite, then regular. Let $GrLR$ be a grammar that generates this language, and let (L, R) be its axiom.
2. The language $\mathcal{L}$.
   $\mathcal{L}$ is generated by the grammar Gr whose axiom is (C_1, C'_1), whose productions are those of $GrCC$ (for generating two identical contexts), plus those of $GrLR$ (for generating left and corresponding right-hand-sides), plus those of GrG and $GrTT$ (for generating ground instances), plus the synchronized productions (for making links between the four used grammars):

$$\begin{array}{r} \{\bot \Rightarrow L,\ \bot' \Rightarrow R\} \\ \{X \Rightarrow T_1,\ X' \Rightarrow T'_1\} \text{ for each } x \in Var(l) \cap Var(r) \\ \{X \Rightarrow G\} \text{ for each } x \in Var(l) \setminus Var(r) \\ \{X' \Rightarrow G\} \text{ for each } x \in Var(r) \setminus Var(l) \end{array} \Bigg\} \text{ for each rule } l \rightarrow r$$

   Note that $\bot, \bot', X, X', \ldots$ are now considered as non-terminals.

3.3 The language $\neg\mathcal{L}$ of solutions of $\neg(x \rightarrow y)$

We guess that the languages generated by TTSG's are not closed by complement. Fortunately, we need only to complement the language $\mathcal{L}$, which is a particular case. Instead of computing the complement of Gr, we directly represent $\neg\mathcal{L}$ by a TTSG (denoted $\overline{Gr}$), by studying the solutions of $\neg(x \rightarrow y)$ carefully.

Definition 1. Let t_1, t_2 be two different ground terms. Let $u_{c_1},\ldots,u_{c_n}$ (exhaustive list) be the outer clash occurrences between t_1 and t_2, i.e.

$$\forall i,\ u_{c_i} \in O(t_1) \cap O(t_2) \wedge t_1(u_{c_i}) \neq t_2(u_{c_i}) \wedge \forall u < u_{c_i},\ t_1(u) = t_2(u)$$

A *father* of clashes is any occurrence u_{fa} located outer than or equal to all outer clash occurrences (i.e. $\forall i,\ u_{fa} \leq u_{c_i}$). Note that all fathers are located on the same branch, going from occurrence ϵ to the *innermost father* u_{if}.

The notions of father and innermost father are essential with respect to one-step rewriting. Indeed, if $t_1 \rightarrow_{[u,l\rightarrow r,\sigma]} t_2$, all clashes between t_1 and t_2 must disappear when replacing σl by σr. Then necessarily u is a father occurrence (otherwise at least one clash located outer than u or in another branch would not disappear when rewriting t_1). In other words, if u is not a father occurrence, then $t_1 \not\rightarrow_{[u]} t_2$.

Assume now that some left-hand-side l overlaps in t_1 at some father occurrence u, i.e. $t_1 = t_1[u \leftarrow \sigma l]$ and $u \leq u_{if}$.

Lemma 2. *Within $t_1 = t_1[u \leftarrow \sigma l]$, if l does not cover the innermost father occurrence u_{if}, i.e. $u_{if} = u.v.w$ where v is a variable occurrence of l, then $t_1 \not\rightarrow_{[u,l\rightarrow r]} t_2$.*

Lemma 3. *No term can rewrite by one step into itself.*

Consequently, $t_1 \not\rightarrow t_2$ iff one of the three following cases holds.

1. Some left-hand-sides $l_1, \ldots, l_k$ (exhaustive list) overlap in t_1 at father occurrences, and cover the innermost father occurrence u_{if}, but the terms reached by one-step rewriting thanks to these redexes are all different from t_2.
2. No left-hand-side overlaps in t_1 at father occurrences.
3. t_1 and t_2 are identical.

Let $\mathcal{L}_{case-1}$, $\mathcal{L}_{case-2}$, $\mathcal{L}_{case-3}$ be the languages of pairs of ground terms (t_1, t_2) defined by respectively case 1, case 2, case 3. Thus $\neg\mathcal{L} = \mathcal{L}_{case-1} \cup \mathcal{L}_{case-2} \cup \mathcal{L}_{case-3}$.

The language $\mathcal{L}_{case-3}$ is generated by the grammar $GrTT$. For $\mathcal{L}_{case-1}$ (resp. $\mathcal{L}_{case-2}$), we split the problem into two simpler ones. We define:

- the language $\mathcal{L}_0$ of pairs of different ground terms (t_1, t_2), where the innermost father occurrence u_{if} is marked by replacing in t_1 the symbol (say g) appearing at u_{if} by a new symbol g_{if}. This amounts to deal with the extended set of symbols $\Sigma' = \Sigma \cup \Sigma_{if}$ that contains an additional symbol g_{if} for each symbol $g \in \Sigma$.
- and the language $\mathcal{L}_1$ (resp. $\mathcal{L}_2$) of pairs of ground terms (t_1, t_2), such that t_1 contains one symbol of Σ_{if} at an arbitrary position u_{if} and such that the condition of case 1 (resp. case 2) is satisfied, u_{if} being considered as the innermost father.

Note that $\mathcal{L}_1$ (resp. $\mathcal{L}_2$) does not ensure that u_{if} is the actual innermost father position with respect to the existing clashes between t_1 and t_2. This is the role of $\mathcal{L}_0$. Therefore $\mathcal{L}_{case-1} = \mathcal{L}_0 \cap \mathcal{L}_1$ (resp. $\mathcal{L}_{case-2} = \mathcal{L}_0 \cap \mathcal{L}_2$) after replacing each symbol of Σ_{if} by the corresponding symbol of Σ. Note that this symbol replacement does not change anything with respect to emptiness.

Generating $\mathcal{L}_0$: $\mathcal{L}_0$ contains all pairs of the form (C is any context and '...' means any ground term(s)):

1. $(C[f_{if}(\ldots)], C[g(\ldots)])$ where $f \neq g$. This is the particular case where there is only one outer clash. And then the innermost father and the outer clash coincide.
2. $(C[g_{if}(\ldots, s_1, \ldots, s'_1, \ldots)], (C[g(\ldots, s_2, \ldots, s'_2, \ldots)]$ where $s_1 \neq s_2$, $s'_1 \neq s'_2$.

The grammar Gr_0 that generates $\mathcal{L}_0$ can be easily defined thanks to the previously defined grammars. Its axiom is (C_1, C'_1) and its productions are those of $GrCC$ (for generating two identical contexts), plus those of $GrTT'$ (for generating pairs of arbitrary ground terms), plus those of $GrT\overline{T}$ (for generating pairs of different ground terms), plus the productions for case 1 (O as Outer clash, NO as Non Outer clash) which are:

$$\{\bot \Rightarrow O_f,\ \bot' \Rightarrow O'_g\} \text{ for each } f, g \in \Sigma \text{ s.t. } f \neq g$$
$$\left.\begin{array}{l} O_f \Rightarrow f_{if}(T, \ldots, T) \\ O'_f \Rightarrow f(T', \ldots, T') \end{array}\right\} \text{for each } f \in \Sigma$$

and the productions for case 2 ($\mathbf{T}$ means $T, \ldots, T$):

$$\{\bot \Rightarrow NO_{g,i,j},\ \bot' \Rightarrow NO'_{g,i,j}\} \left\{\begin{array}{l} \text{for each } g \in \Sigma_{\geq 2} \\ \text{and each } i, j \in \{1, \ldots, ar(g)\} \text{ s.t. } i < j \end{array}\right.$$

$$\left.\begin{array}{l} NO_{g,i,j} \Rightarrow g_{if}(\mathbf{T}, D_1, \mathbf{T}, D_2, \mathbf{T}) \\ NO'_{g,i,j} \Rightarrow g(\mathbf{T}', D'_1, \mathbf{T}', D'_2, \mathbf{T}') \end{array}\right| \begin{array}{l} \text{for each } g \in \Sigma_{\geq 2},\ i, j \in \{1, \ldots, ar(g)\} \text{ s.t. } i < j \\ \text{where } D_1\ (D'_1) \text{ is the } i^{th} \text{ argument} \\ D_2\ (D'_2) \text{ is the } j^{th} \text{ argument} \end{array}$$

$$\{D_1 \Rightarrow D,\ D'_1 \Rightarrow D'\}$$
$$\{D_2 \Rightarrow D,\ D'_2 \Rightarrow D'\}$$

where $O_f, \ldots, NO_{g,i,j}, \ldots$ are non-terminals (respectively $O'_g, \ldots, NO'_{g,i,j}, \ldots$).

Generating $\mathcal{L}_1$: Assume some left-hand-side l overlaps in t_1 at some father occurrence u, s.t. l covers u_{if} within t_1. We have $u \leq u_{if}$ and $t_1 = t_1[u \leftarrow \sigma l]$. Let t' be the term obtained by rewriting t_1 at occurrence u: $t' = t_1[u \leftarrow \sigma r]$.

Let us first show that the following assumption is impossible: Suppose there is another (or the same) left-hand-side l' that overlaps in t_1 at some position $u' \leq u_{if}$, s.t. l' also covers u_{if}. Since l and l' are both linear, and moreover they share the same occurrence u_{if} of t_1, they overlap necessarily, which is in contradiction with the non left-left-overlapping restriction.

Therefore, to ensure that $t_1 \not\to t_2$, checking that t_2 is different from t' is enough. In other words, $\mathcal{L}_1$ contains all pairs (t_1, t_2) of the form $(C[\sigma l_{if}], \overline{C[\sigma r]})$ where C is any context, σ is any substitution, $l \to r$ is any rewrite rule, l_{if} is obtained from l by replacing one arbitrary symbol (say f) by the corresponding symbol of Σ_{if} (i.e. f_{if}), and $\overline{C[\sigma r]}$ denotes any term different from $C[\sigma r]$.

However, the above explanation does not take into account the possibility of having several t' for fixed t_1, u, $l \to r$; in other words several $C[\sigma r]$ for one $C[\sigma l_{if}]$. This case happens when r contains some variables that do not appear in l, since these variables can be instantiated by any ground term. So, forcing a clash with respect to some t' within the instances of such variables does not ensure that we get a term different from all potential t'. To process this case correctly, we just have to force the clash elsewhere.

Besides, the clashes between t_1 and t_2 occur only below u_{if}[7]. As u_{if} belongs to the part brought by l_{if}, there is no clash in the part brought by C. So computing $\overline{C[\sigma r]}$ as $C[\overline{\sigma}r] \cup C[\overline{\theta r}]$ where $\overline{\sigma}r$ denotes an instance of r different from σr and $\overline{\theta r}$ denotes any term non instance of r, is enough. Or again, $\mathcal{L}_1$ can be computed as the union of the languages $\mathcal{L}'_1$ and $\mathcal{L}''_1$ defined below.

- $\mathcal{L}'_1 = (C[\sigma l_{if}], C[\overline{\sigma}r])$.
  The TTSG that generates $\mathcal{L}'_1$ looks like Gr (language $\mathcal{L}$ in Section 3.2). The main difference is that an arbitrary variable of $Var(l) \cap Var(r)$ (don't know choice) is instantiated in l and r by two different terms, thanks to $GrT\overline{T}$, while other variable occurrences are replaced by any ground term. Thus there is at least one clash between the substitution applied to l_{if} and the one applied to r.
- $\mathcal{L}''_1 = (C[\sigma l_{if}], C[\overline{\theta r}])$.
  The TTSG that generates $\mathcal{L}''_1$ looks like Gr as well, except that to generate $\overline{\theta r}$, we generate an arbitrary branch of r that reaches to a clash with a non variable occurrence of r, and any ground terms on the other branches and below the clash.

Generating $\mathcal{L}_2$: $\mathcal{L}_2$ contains all pairs (t_1, t_2) such that t_1 is any ground term that does not belong to the projection on the first component of $\mathcal{L}_1$, and t_2 is any ground term. The projection on the first component of $\mathcal{L}_1$ contains the terms of the form $C[\sigma l_{if}]$, which is obviously a regular language. Its complement is also a regular language. $\mathcal{L}_2$ is obtained by computing the cartesian product between the complement and the language of all ground terms.

The complement is computed with respect to $\Sigma \cup \Sigma_{if}$. So it owns, among others, terms that contain no or several symbols of Σ_{if}. These terms are wrong because they do not make sense[8]. But it does not matter since these wrong terms will be deleted when making intersection with $\mathcal{L}_0$, all terms of which just contain one symbol of Σ_{if}.

4 Intersection over several components

The intersection over several components is needed because of conjunctions in formulas and also for computing $\mathcal{L}_{case-1}$ and $\mathcal{L}_{case-2}$. Lengthening the tuples by making cartesian products with the language of all ground terms[9] (to get tuples of the same size), and nesting tree-tuples below a new symbol (to get single trees), allows to transform the intersection over several components into a classical intersection between tree languages. Thus in the following, we deal with tree synchronized grammars (*TSG* for short) instead of TTSG's. However these TSG's contain internal synchronizations even if the initial TTSG's do not. As

[7] Actually, this is not ensured by $\mathcal{L}_1$, but this will hold when computing $\mathcal{L}_0 \cap \mathcal{L}_1$.
[8] There is only one innermost father.
[9] In the same way as done when making unions. See footnote 5, or [7] for more details.

guessed in [4], we think that in general, the resulting language cannot be generated by a TSG. Fortunately, the TTSG's to be processed satisfy the following particular property : the difference of depth between any two synchronized points is bounded. Of course it still holds in the corresponding TSG's. However this difference is not necessarily equal to 0, which causes a major difficulty, as explained in Section 4.1. This difficulty is solved in Section 4.2 by first transforming the grammars to make the depth difference equal to 0.

4.1 Intersection of TSG's

Recall that in the most general case, control numbers are actually integer-tuples. So in the following, the synchronized grammars are denoted by 5-tuples instead of 4-tuples because a new component (the first one) has been added to mention the size of these integer-tuples. Packs of synchronized productions are denoted by $\{\ldots\}_k$ where k is an integer (called *the level of the pack*) meaning that only the k^{th} component of the control tuple is incremented when applying the pack (see [4] for more details). $Rec(G)$ denotes the language recognized (generated) by the grammar G.

The TSG intersection is mainly based on the regular tree grammar intersection. The difference is that we have to take care of control numbers. In order to make the intersection of G_1 and G_2, consider G_3 as defined below.

Definition 4. Let $G_1 = (S_1, \mathcal{C}_1, NT_1, P_1, I_1)$ and $G_2 = (S_2, \mathcal{C}_2, NT_2, P_2, I_2)$ be two TSG's.
Let $G_3 = (S_1 + S_2, \mathcal{C}_1 \cup \mathcal{C}_2, NT_1 \times NT_2, Fr \cup Sn_1 \cup Sn_2, I_1 I_2)$ where

- $Fr = \{X_1X_2 \Rightarrow c(Y_{1,1}Y_{2,1}, \ldots, Y_{1,n}Y_{2,n})$ such that
  $X_1 \Rightarrow c(Y_{1,1}, \ldots, Y_{1,n}) \in P_1$ and $X_2 \Rightarrow c(Y_{2,1}, \ldots, Y_{2,n}) \in P_2\}$
- $Sn_1 = \{\{X_1Y_1 \Rightarrow X_1'Y_1, \ldots, X_nY_n \Rightarrow X_n'Y_n\}_k$ such that
  $\{X_1 \Rightarrow X_1', \ldots, X_n \Rightarrow X_n'\}_k \in P_1$ and $\forall i \in [1,n]\, Y_i \in NT_2\}$
- $Sn_2 = \{X_1Y_1 \Rightarrow X_1Y_1', \ldots, X_nY_n \Rightarrow X_nY_n'\}_{S_1+k}$ such that
  $\{\{Y_1 \Rightarrow Y_1', \ldots, Y_n \Rightarrow Y_n'\}_k \in P_2$ and $\forall i \in [1,n]\, X_i \in NT_1\}$

The control tuples of G_3 are obtained by concatenating those of G_1 with those of G_2. This allows to avoid confusions between the control coming from G_1 and that coming from G_2 when deriving G_3.

It is easy to verify that $Rec(G_3) \subseteq Rec(G_1) \cap Rec(G2)$ because of the way G_3 is built (The proof is done with a classical induction on the length of the derivations of G_3). But unfortunately $Rec(G_1) \cap Rec(G_2) \not\subseteq Rec(G_3)$ as illustrated by the following example.

Example 4. Let $\mathcal{C} = \{c, s, 0\}$, $NT_1 = \{X_i | i \in [1,6]\}$, $NT_2 = \{Y_i | i \in [1,6]\}$,
$P_1 = \{X_1 \Rightarrow c(X_2, X_3), X_2 \Rightarrow s(X_4), X_5 \Rightarrow s(X_6), X_6 \Rightarrow 0, \{X_3 \Rightarrow X_5, X_4 \Rightarrow X_6\}_1\}$,
$P_2 = \{Y_1 \Rightarrow c(Y_3, Y_2), Y_2 \Rightarrow s(Y_4), Y_5 \Rightarrow s(Y_6), Y_6 \Rightarrow 0, \{Y_3 \Rightarrow Y_5, Y_4 \Rightarrow Y_6\}_1\}$
Let $G_1 = (1, \mathcal{C}, NT_1, P_1, X_1)$ and $G_2 = (1, \mathcal{C}, NT_2, P_2, Y_1)$
$G_3 = (2, \mathcal{C}, NT_1 \times NT_2, Fr \cup Sn_1 \cup Sn_2, X_1Y_1)$ where
$Fr = \{X_1Y_1 \Rightarrow c(X_2Y_3, X_3Y_2), X_2Y_2 \Rightarrow s(X_4Y_4), X_5Y_5 \Rightarrow s(X_6Y_6),$

$X_2Y_5 \Rightarrow s(X_4Y_6), X_5Y_2 \Rightarrow s(X_6Y_4), X_6Y_6 \Rightarrow 0\}$
$Sn_1 = \{\{X_3Y_i \Rightarrow X_5Y_i, X_4Y_j \Rightarrow X_6Y_j\}_1$ such that $i,j \in [1,6]\}$ and
$Sn_2 = \{\{X_iY_3 \Rightarrow X_iY_5, X_jY_4 \Rightarrow X_jY_6\}_2\}$ such that $i,j \in [1,6]\}$

Obviously $Rec(G_1) = Rec(G_2) = \{c(s(0), s(0))\}$. Unfortunately $Rec(G_3) = \emptyset$. Indeed $X_1Y_1 \overset{0,0}{\Rightarrow} c(X_2Y_3, X_3Y_2)$ and no more productions can be applied because X_2Y_3 requires a non terminal of the form X_iY_4 to perform a synchronization and X_3Y_2 requires a non terminal of the form X_4Y_j. So each branch of the term is waiting for the other, which is a kind of deadlock. This happens because G_1 and G_2 have leaning synchronizations, i.e. in their derivation appear some synchronization points that are at different depths. So a sufficient condition to get $Rec(G_3) \subseteq Rec(G_1) \cap Rec(G_2)$ is that G_1 and G_2 have no leaning synchronizations. This condition clearly still holds for G_3.

In fact, a leaning synchronization is a set of non-terminals such that their reduction may cause depth differences between further synchronization points.

In the following definition, $|u|$ is the depth of the occurrence u, $\Rightarrow_i$ is a step that applies a pack of productions of level i, and $\Rightarrow^*_{\neq i}$ tt denotes a derivation that does not apply any packs of level i s.t. tt is either irreducible or reducible only by packs of level i.

Definition 5. Let $G = (S, \mathcal{C}, NT, P, I)$ be a TSG.
A *synchronization of level* i is a set of non-terminals which is either $\{I\}$ or $\{X_1, \ldots, X_n\}$ such that $\exists\{X_1 \Rightarrow X'_1, \ldots, X_n \Rightarrow X'_n\}_i \in P$.
For $\{X_1, \ldots, X_n\}$ a synchronization of level i, $Next(\{X_1, \ldots, X_n\})$ denotes the set defined by $\{\{t_1, \ldots, t_n\}$ s.t. $(\overset{ct_1}{X_1}, \ldots, \overset{ct_n}{X_n}) \Rightarrow_i (\overset{ct'_1}{X'_1}, \ldots, \overset{ct'_n}{X'_n}) \Rightarrow^*_{\neq i} (t_1, \ldots, t_n)\}$.
For the synchronization $\{I\}$ and for each level i, $Next(\{I\}) = \{\{t\} | I \Rightarrow^*_{\neq i} t\}$.
A synchronization $\{X_1, \ldots, X_n\}$ of level i is said *leaning* if $\exists tt \in Next(\{X_1, \ldots, X_n\})$, $\exists u, u' \in O(tt)$ such that both $tt(u), tt(u')$ are non-terminals and $|u| \neq |u'|$.
A synchronization $\{X_1, \ldots, X_n\}$ of level i is said *finite* if $Next(\{X_1, \ldots, X_n\})$ is a finite set.
A TSG without leaning synchronizations is called a *balanced TSG*.

Lemma 6. *Let G be a balanced TSG, in any derivation of G, if $\{X_1 \Rightarrow Y_1, \ldots, X_n \Rightarrow Y_n\}_l$ applies at occ. $u_1 \ldots u_n$ then $|u_1| = \ldots = |u_n|$.*

This property prevents from deadlocks when making intersection because all synchronized non terminals are always at the same depth. Unfortunately, the TSG's built from one-step rewriting formulas are not necessarily balanced because variables may appear at different depths in the rewrite rules. The only non-balanced languages are $\mathcal{L}$ and $\mathcal{L}'_1$, due to the leaning synchronization $\{\perp, \perp'\}$ and the pack $\{\perp \Rightarrow L, \perp' \Rightarrow R\}$. This leaning synchronization cannot be applied infinitely many times (actually only once), which prevents from increasing the depth difference between synchronization points in an unbounded way. So the object of the next subsection is to give an algorithm to transform such grammars (called quasi-balanced) into balanced ones.

4.2 From quasi-balanced to balanced TSG's

Definition 7. Let G be a TSG and $\{X_1,...,X_n\}$ a synchronization of level i.
$\{X_1,\ldots,X_n\}$ is said *recursive* if there exists a tuple tt s.t. $(\overset{ct_1}{X_1},\ldots,\overset{ct_n}{X_n}) \Rightarrow^+ tt$ where $\forall j \in [1,n]$ $ct_j|_i = ct_1|_i$ (the other components of the controls being any) and there exist $u_1,\ldots,u_n \in O(tt)$ such that $\forall j$, $tt(u_j) = \overset{ct''_j}{X_j}$ and $ct''_j|_i = ct''_1|_i$.
Otherwise $\{X_1,\ldots,X_n\}$ is said *non-recursive*.
A TSG G is said *quasi-balanced* if any leaning synchronization is finite and non-recursive.

Now, we have to prove that the class of languages recognized by quasi-balanced TSG's is the same as the class of languages recognized by balanced TSG's. For that we give an algorithm that transforms any quasi-balanced TSG into a balanced one.

The main idea of this algorithm is to re-organize synchronizations so that the synchronized points appear at the same depth. To do that, the algorithm "cuts" the slopes at the depth of the higher synchronized non-terminal. This cutting operation needs the algorithm to look in advance at the tree-parts that will be generated later. For that it creates new 13¡

non-terminals whose names encode the tree-parts to be produced. For sake of simplicity we present the algorithm for TSG's whose control numbers are single integers (which is enough for the one-step rewriting problem).

First, let us define the new non-terminals and the way to decode their names.

Definition 8. Let $G = (1,\mathcal{C},NT,PP,I)$ be a TSG. $AheadNT(G) = \{\overset{i}{X^t} \mid X \in NT, i \in \mathbb{N}, t \in T(\mathcal{C}, NT \times \mathbb{N}) \cup \{\epsilon\}\}$ where ϵ is a new symbol.
Let $\overset{i}{X^t} \in AheadNT(G)$, $term(\overset{i}{X^t}) = \overset{i}{X}$ when $t = \epsilon$ and t otherwise.

Roughly speaking, X is the actual non-terminal, t is the term to be generated from X (if $t = \epsilon$ there is no restriction on the term generated from X). So the function $term$ allows to look ahead in the derivation. Note that $AheadNT(G)$ is infinite, but thanks to the non-recursivity of leaning synchronizations only a finite part will be used by the algorithm. In the following $term(\{A_1,\ldots,A_n\})$ where $\forall i$, $A_i \in AheadNT(G)$, will denote $\{term(A_1),\ldots,term(A_n)\}$. Now let us define some notations needed in the algorithm.

Definition 9. Let $G = (1,\mathcal{C},NT,P,I)$ be a TSG and s be a finite subset of $AheadNT(G)$. s is said

- *free* if $\forall \overset{i}{X^t} \in s$ there exists a free production $X \Rightarrow c(X_1,\ldots,X_n) \in P$ and either $t = \epsilon$ or $t = c(t_1,\ldots,t_n)$. Moreover we denote,

$NF(s) = \cup_{\overset{i}{X^t} \in s} \{\overset{i}{X_1^{t_1}},\ldots,\overset{i}{X_n^{t_n}}\}$ and $NFP(s) = \cup_{\overset{i}{X^t} \in s} \{\overset{i}{X^t} \Rightarrow c(\overset{i}{X_1^{t_1}},\ldots,\overset{i}{X_n^{t_n}})\}$.

– *minimally synchronized* if $\cup_{\overset{i}{X^t} \in s} term(\overset{i}{X^t})$ contains only synchronized non-terminals and all these non-terminals can be derived (i.e. none of them are waiting for another non-terminal of its synchronization) and $\forall \overset{i}{X^t} \in s$, $s \setminus \{\overset{i}{X^t}\}$ is not minimally synchronized. In that case, $NS(s) = \{\{\overset{i_1}{X_1^{t_1}}, \ldots, \overset{i_n}{X_n^{t_n}}\}$ such that $term(s) \Rightarrow^* term(\{\overset{i_1}{X_1^{t_1}}, \ldots, \overset{i_n}{X_n^{t_n}}\})$ where one and only one synchronized production has been applied to each non-terminal of $term(s)$. $NSP(s) = \{\overset{j_1}{Y_1^{s_1}} \Rightarrow \overset{i_1}{X_1^{t_1}}, \ldots, \overset{j_n}{Y_n^{s_n}} \Rightarrow \overset{i_n}{X_n^{t_n}}\}$ such that $s = \{\overset{j_1}{Y_1^{s_1}}, \ldots, \overset{j_n}{Y_n^{s_n}}\}$, $\{\overset{i_1}{X_1^{t_1}}, \ldots, \overset{i_n}{X_n^{t_n}}\} \in NS(s)$ and $term(s) \Rightarrow^* term(\{\overset{i_1}{X_1^{t_1}}, \ldots, \overset{i_n}{X_n^{t_n}}\})$.
– *completely synchronized* if $\forall \overset{i}{X^t} \in s$, $term(\overset{i}{X^t})$ contains only synchronized non-terminals. In that case, $Pa(s) = \{\{\overset{i_1}{X_1^{t_1}}, \ldots, \overset{i_n}{X_n^{t_n}}\}$ such that this set is minimally synchronized $\}$.

Now the algorithm is given thanks to these 4 inference rules.

$$\frac{D \quad TD \cup \{s\} \quad R}{D \cup \{s\} \; TD \cup \{s'\} \setminus D \; R \cup \{p\}} \textbf{(1)} \qquad \frac{D \quad TD \cup \{s\} \quad R}{D \cup \{s\} \; TD \cup Pa(s) \setminus D \; R} \textbf{(2)}$$

$$\frac{D \quad TD \cup \{s\} \quad R}{D \cup \{s\} \; TD \cup NF(s) \setminus D \; R \cup NFP(s)} \textbf{(3)}$$

$$\frac{D \quad TD \cup \{s\} \quad R}{D \cup \{s\} \; TD \cup NS(s) \setminus D \; R \cup NSP(s)} \textbf{(4)}$$

(1) Look Ahead if s is not free and $\overset{i}{Y^t} \in s$ such that $term(t) \Rightarrow term(t')$ with the free production $X \Rightarrow c(X_1, \ldots, X_n)$.
p denotes the production $\overset{i}{Y^t} \Rightarrow \overset{i}{Y^{t'}}$ and s' the set $(s \setminus \{\overset{i}{Y^t}\}) \cup \{\overset{i}{Y^{t'}}\}$
(2) Partition if s is completely synchronized but not minimally synchronized.
(3) Free if s is free.
(4) Sync if s is minimally synchronized.

Comments: D stands for Done, TD for To Do and R for Result. To transform a quasi-balanced TSG G into a balanced one, one has to initialize the sets D and R to $\emptyset$, and TD to $\{\overset{0}{I}\}$ (i.e. the axiom of G). Then the inference rules are applied in a don't care way as long as possible. At the end, R contains the productions of the new TSG (the non-terminals being deduced from the productions). For each step, s is a subset of AheadNT representing non-terminals appearing at the same depth in a derivation of G, associated with their ahead terms when leaning synchronizations are detected. The role of each inference rule is the following:

Look Ahead is applied when s is neither free nor totally synchronized, which means that at the same depth in a synchronization of G there are some non-

terminals waiting for synchronization and some others that can be derived with a free production. This case happens while visiting leaning synchronizations. So **Look Ahead** will apply the free productions on the Ahead terms of the non-terminals instead of the actual non-terminal to find the next synchronization points. It generates empty free productions (i.e. production that do not produce any terminal symbols). These production can be easily eliminated afterwards but they simplify all the proofs.

Partition is applied when s is completely synchronized, which means that $term(s)$ contains only synchronized non terminals, so the ahead terms are big enough to know what will happen when applying a synchronized pack of productions on non-terminals appearing at different depths. So partition divides s in subsets which are minimally synchronized, i.e. the non-terminals of each subset are actually bound by synchronization constraints.

Free is applied when s is free i.e. when all the non-terminals of s can be derived without synchronization constraints. So the free productions are applied on the Ahead non-terminals at the same time in order to keep the new non-terminals at the same depth. The free productions are stored in the result.

Sync is applied when s is minimally synchronized (i.e. all the non-terminals of $term(s)$ are actually to be synchronized) and the ahead terms are big enough to know what happens next, all the possibilities are explored and all the corresponding new synchronized productions are created. Note that thanks to **Free** we know that the ahead non-terminals of s always appear at the same depth in a derivation of the new TSG.

The application of the inference rules terminates because the depth between two synchronization points is bounded, so the depth of the Ahead terms is bounded, so the number of AheadNT's used in the algorithm is bounded, so the number of different sets of AheadNT's is bounded.

So from a quasi-balanced TSG $G = (1, \mathcal{C}, NT, P, I)$ we get $G' = (1, \mathcal{C}, NT, R, \overset{0}{I}{}^{\epsilon})$. G' is balanced from its construction. The last step is to prove that $Rec(G) = Rec(G')$. For that we make a correspondence between derivations of G' and derivations of G using the terms of G' where AheadNT's have been expanded with the function $term$.

5 Further work : weakening the restrictions

A referee has pointed out that our method can be easily extended to deal with rewrite systems that have critical pairs between different rules. If the rewrite system is $R = \{r_1, \ldots, r_n\}$, just replace each $x \to y$ in the formula by $x \to_{[r_1]} y \vee \ldots \vee x \to_{[r_n]} y$, bring the formula into a disjunctive form, and adjust the method (which is easy) to take into account the new predicate symbols.

Besides, we conjecture that the method could be extended to deal with rewrite systems that have non linear right-hand-sides, and collapsing rules[10].

[10] I.e. non ϵ-left-right-overlapping is still assumed, except for collapsing rules.

The non-linearity of rhs's creates internal synchronizations in languages. However the difference of depth between synchronization points is bounded. The intersection problem in this case has been solved already in Section 4, and the proof of lemma 2 seems to still hold. On the other hand, it seems nearly impossible to weaken the linearity of left-hand-sides because $\mathcal{L}_2$ is obtained by complementing a regular language, which would not be regular any more. Of course, a reduction automaton could be used, but it would not be deterministic, which prevents from computing the complement.

Within $\mathcal{L}_1$, the verification that the rewrite step is impossible is ensured by the failure of one rewrite rule applied at one occurrence. This is enough thanks to the restrictions. But we think that the number of failures to be ensured becomes unbounded if there are ϵ-left-right-overlapping rules that are non-collapsing.

So the question still arises: is the existential fragment of one-step rewriting theory without any restrictions decidable?

References

1. A.C. Caron, F. Seynhaeve, S. Tison, and M. Tommasi. Solving One-step Rewriting Formulae. In *12th Workshop on Unification, Roma (Italy)*. U. of Roma, 1998.
2. H. Comon, M. Dauchet, R. Gilleron, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications (TATA)*. http://l3ux02.univ-lille3.fr/tata.
3. F. Jacquemard. *Automates d'Arbres et Réécriture de Termes*. Thèse de Doctorat d'Université, Université de Paris-sud, 1996. In french.
4. S. Limet and P. Réty. E-Unification by Means of Tree Tuple Synchronized Grammars. *DMTCS (http://dmtcs.loria.fr/)*, 1:69–98, 1997.
5. S. Limet and P. Rety. A new result about the decidability of the existential one-step rewriting theory. Research Report RR-LIFO-98-12, LIFO, 1998.
6. S. Limet and P. Réty. Solving Disequations modulo some Class of Rewrite Systems. In *Proc. of RTA'98, Tsukuba (Japon)*, vol 1379 of *LNCS*, pages 121–135.1998.
7. S. Limet and F. Saubion. A general framework for r-unification. In C. Palamidessi, H. Glaser, and K. Meinke, editors, *proc of PLILP-ALP'98*, volume 1490 of *LNCS*, pages 266–281. Springer Verlag, 1998.
8. J. Marcinkowski. Undecidability of the First-order Theory of One-step Right Ground Rewriting. In *Proceedings 8th Conference RTA, Sitges (Spain)*, volume 1232 of *LNCS*, pages 241–253. Springer-Verlag, 1997.
9. J. Niehren, M. Pinkal, and P. Ruhrberg. On Equality up-to Constraints over Finite Trees ,Context Unification and One-step Rewriting. In W. Mc Cune, editor, *Proc. of CADE'97, Townsville (Australia)*, volume 1249 of *LNCS*, pages 34–48, 1997.
10. F. Seynhaeve, M. Tommasi, and R. Treinen. Grid Structures and Undecidable Constraint Theories. In *Proceedings of 6th Colloquium on Trees in Algebra and Programming*, volume 1214 of *LNCS*, pages 357–368. Springer-Verlag, 1997.
11. R. Treinen. The First-order Theory of One-step Rewriting is Undecidable. In H. Ganzinger, editor, *Proceedings 7th Conference RTA, New Brunswick (USA)*, volume 1103 of *LNCS*. Springer-Verlag, 1996.
12. S. Vorobyov. The First-order Theory of One-step Rewriting in Linear Noetherian Systems is Undecidable. In *Proceedings 8th Conference RTA, Sitges (Spain)*, volume 1232 of *LNCS*, pages 241–253. Springer-Verlag, 1997.

This article was processed using the LaTeX macro package with LLNCS style