UNIVERSITE D'ORLEANS
Faculté des Sciences
LIFO

Laboratoire d'Informatique Fondamentale d'Orléans
4, rue Léonard de Vinci, BP 6759
F-45067 Orléans Cedex 2
FRANCE

Rapport de Recherche

www : http://www.univ-orleans.fr/SCIENCES/LIFO/

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

Sébastien Limet and Pierre Réty
Université d'Orléans, LIFO

Rapport N^o **98-12**

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

April 26, 1999

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). 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 [13]. Sharper undecidability results have been obtained for some subclasses of rewrite systems, about the $\exists^*\forall^*$-fragment [12, 10] and the $\exists^*\forall^*\exists^*$-fragment [14].

It has been shown decidable for the positive existential fragment [11], in the case of unary signatures [4], in the case of linear rewrite systems whose left and right members do not share any variables [2][1], 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 ensure 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.

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

TTSG's have been introduced by the authors to solve some equational unification [5, 6] and disunification [7] problems, and have also been used for the validation of logic programs [8].

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 3.4 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 $\neg(x \rightarrow y)$[2]. 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 intersections. 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[3].

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

TTSG's, cartesian product, union of languages of tuples of the same size, intersection over one component, emptiness test, are recalled in Section 2, whereas the intersection over several components is a new result, presented in Section 4. The possibility of weakening the restrictions is discussed in Section 5.

2 Overview of TTSG's

This section briefly recalls notions and existing results about TTSG's. For a more formal presentation and proofs see [6]. We use the basic notions of terms and term rewriting systems as defined in [3]. 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, $t(u)$ 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 time). They are always

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

[3] 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.

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 2.1 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 combination of synchronizations. To avoid this ambiguity, a *control integer* is associated to each non-terminal occurrence. To make understanding easier, the way it works will be explained in the sequel by means of an example (see Example 3.1).

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 derive occurrences belonging to the same component of tt.

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

Example 2.2 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 [6] 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[4] instead of single integers. Fortunately, the TTSG's defined in the sequel have no internal synchronizations.

The *emptiness* of TTSG's is decidable [6], 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 the technical remark just before section 3.2.

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 as 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 as 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 the 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 3.1 Let $\Sigma = \{a,b,f\}$ where a,b are constants and f is a binary symbol. The productions are:
   $$\begin{array}{c} \{T_1 \Rightarrow T_a,\ T'_1 \Rightarrow T'_a\},\ \ \{T_1 \Rightarrow T_b,\ T'_1 \Rightarrow T'_b\},\ \ \{T_1 \Rightarrow T_f,\ T'_1 \Rightarrow T'_f\}, \\ \{T_2 \Rightarrow T_a,\ T'_2 \Rightarrow T'_a\},\ \ \{T_2 \Rightarrow T_b,\ T'_2 \Rightarrow T'_b\},\ \ \{T_2 \Rightarrow T_f,\ T'_2 \Rightarrow T'_f\}, \\ T_a \Rightarrow a,\ T'_a \Rightarrow a,\ T_b \Rightarrow b,\ T'_b \Rightarrow b,\ T_f \Rightarrow f(T_1,T_2),\ T'_f \Rightarrow f(T'_1,T'_2) \end{array}$$

[4] 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):

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

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:

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

Remark : Here, the use of control is rather trivial: only non-terminals appearing at identical occurrences in each component are synchronized. In the sequel, the control will also be used to synchronize non identical occurrences.

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 as context) and whose productions are those of GrT plus:

   $$C \Rightarrow \perp$$

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

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 as 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 $D'_{f,i}, \ldots, G'_f, \ldots$).

Note that 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 $GrT\overline{T}$.

Technical remark: Every grammar defined above is a TTSG. However some of them do not satisfy the following condition: within each computation derived by the grammar, no non-terminal must occur several times in the same *class* (i.e. with the same control integer). This is due to those free productions of the form $X \Rightarrow f(X_1, \ldots, X_n)$ when several non-terminals among $X_1, \ldots, X_n$ are identical: indeed, when applying a free production the same control integer is bound to all arguments. Unfortunately, the decidability of emptiness of TTSG's is ensured only if this condition is satisfied, which was the case for TTSG's coming from unification problems (see Lemma 8.1 on page 93 in [6]).

Fortunately, thanks to the use of primes for the non-terminals appearing in the second component, every grammar defined above does not contain any packs of productions of the form $\{X \Rightarrow Y, X \Rightarrow Z\}$. It is then possible to satisfy the condition by transforming grammars in the following way. Let us explain for the grammar GrG of ground terms, it is similar for others. Consider the production $G \Rightarrow f(G, \ldots, G)$. We can replace it by $G \Rightarrow f(N_1, \ldots, N_p)$ where $N_1, \ldots, N_p$ are new pairwise different non-terminals, and we add the following synchronized productions $\{N_1 \Rightarrow G\}, \ldots, \{N_p \Rightarrow G\}$. Note that $\{N_1 \Rightarrow G\}$ is a pack of synchronized productions on its own: this means that this empty production is synchronized with nothing, whereas its status is synchronized. The point of interest is that the control integer is modified when applying a pack of synchronized productions.

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

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 $l \to r$ 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):

$$\{\perp \Rightarrow L,\ \perp' \Rightarrow R\}$$
$$\left.\begin{array}{r} \{X \Rightarrow T_1,\ X' \Rightarrow T_1'\} \text{ for each } x \in Var(l) \cap Var(r) \\ \{X \Rightarrow G\} \text{ for each } x \in Var(l) \setminus Var(r) \\ \{X' \Rightarrow G\} \text{ for each } x \in Var(r) \setminus Var(l) \end{array}\right\} \text{ for each rule } l \rightarrow r$$

Note that $\perp, \perp', X, X', \ldots$ 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 represent $\neg\mathcal{L}$ by a TTSG (denoted $\overline{Gr}$) in a straightforward way, by studying the solutions of $\neg(x \rightarrow y)$ carefully.

Definition 3.2 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 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 3.3 *Within $t_1 = t_1[u \leftarrow \sigma l]$, if l does not cover the innermost father occurrence u_{if}, i.e. u_{if} is located in the part brought by σ, then $t_1 \not\rightarrow_{[u,l\rightarrow r]} t_2$.*

Proof: Let t' be the term obtained by rewriting t_1 at occurrence u: $t' = t_1[u \leftarrow \sigma r]$. Due to the linearity of l and r, as well as the non ϵ-left-right-overlapping restriction, there is necessarily at least one clash between l and r, i.e. there exists $u_c \in O(l) \cap O(r)$ s.t. $l(u_c) \neq r(u_c)$. Thus $t'(u.u_c) \neq t_1(u.u_c)$. On the other hand, within $t_1 = t_1[u \leftarrow \sigma l]$, the occurrence $u.u_c$ is in the part brought by l, whereas u_{if} is in the part brought by σ. As a consequence $u.u_c$ is not below (nor equal to) u_{if}. Moreover recall that clashes between t_1 and t_2 can appear only at occurrences located below (or equal to) u_{if}. Therefore $t_2(u.u_c) = t_1(u.u_c) \neq t'(u.u_c)$. In other words $t_1 \not\rightarrow_{[u,l\rightarrow r]} t_2$. $\Box$

Lemma 3.4 *No term can rewrite by one step into itself.*

Proof: Indeed, the case $t \rightarrow_{[l\rightarrow r,\sigma]} t$ is impossible: due to the non ϵ-left-right-overlapping restriction, necessarily $\sigma l \neq \sigma r$. $\Box$

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 $GrTT$. With respect to $\mathcal{L}_{case-1}$ (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.

3.3.1 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 $s_1 \neq s_2$ and $s'_1 \neq s'_2$.

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 $Gr\overline{TT}$ (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 ($\vec{T}$ means $T, \ldots, T$):

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

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

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

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

3.3.2 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 $t_1 \not\rightarrow t_2$, 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, $l \rightarrow r$ 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, $l \rightarrow r$; 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}[5]. 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.

[5]Rigorously speaking, this is not ensured by $\mathcal{L}_1$, but this will hold when computing $\mathcal{L}_0 \cap \mathcal{L}_1$.

- $\mathcal{L}_1'' = (C[\sigma l_{if}],\, C[\overline{\theta r}])$.
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.

A more precise definition of the TTSG's that generate $\mathcal{L}_1'$ and $\mathcal{L}_1''$ is given in the appendix (Section A).

3.3.3 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[6]. 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[7] (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 guessed in [6], 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. Proofs of this part are in the Appendix.

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*

[6] There is only one innermost father.

[7] In the same way as done when making unions. See footnote 3, or [9] for more details.

level of the pack) meaning that only the k^{th} component of the control tuple is incremented when applying the pack (see [6] 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 4.1 *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* $X_2 \Rightarrow c(Y_{2,1}, \ldots, Y_{2,n}) \in P_2\}$
- $Sn_1 = \{\{X_1Y_1 \Rightarrow X'_1Y_1, \ldots, X_nY_n \Rightarrow X'_nY_n\}_k$ *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 4.2 *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* $\overset{0,0}{X_1Y_1} \Rightarrow c(\overset{0,0}{X_2Y_3}, \overset{0,0}{X_3Y_2})$ *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 dead-lock. 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 $X_1, \ldots, X_n$ such that their reduction may cause depth differences between the 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 4.3 *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 of all tuples $\{\{t_1, \ldots, t_n\}$ s.t. $(\overset{ct_1}{X_1}, \ldots, \overset{ct_n}{X_n}) \Rightarrow_i (\overset{ct'_1}{X'_1}, \ldots, \overset{ct'_n}{X'_n}) \Rightarrow^*_{\neq i} (t_1, \ldots, t_n)$ [8]*.*
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 4.4 *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 occurrences $u_1 \ldots u_n$ then $|u_1| = \ldots = |u_n|$.*

This property prevents from deadlocks when making intersection. 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 recursively 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

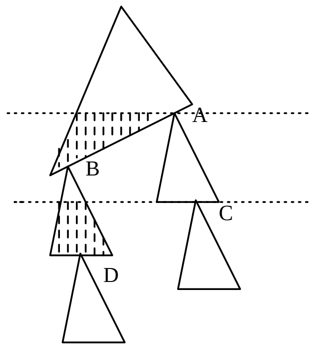
Definition 4.5 *Let G be a TSG and $\{X_1, \ldots, X_n\}$ a synchronization of level i of G.*
$\{X_1, \ldots, X_n\}$ is said recursive *if $(\overset{ct_1}{X_1}, \ldots, \overset{ct_n}{X_n}) \Rightarrow^+ tt$ where $\forall j \in [1, n]$ $ct_j|_i = ct_1|_i$ (the other components of the controls being 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 every 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 (see the figure).

[8] For the synchronization $\{I\}$ and for each level i, $Next(\{I\}) = \{\{t\} | I \Rightarrow^*_{\neq i} t\}$.



To do that, the algorithm "cuts" the slopes at the depth of the higher synchronized non-terminal (A and C on the figure). This cutting operation needs the algorithm to look in advance at the tree-parts that will be generated later (hachured parts in the figure). For that it creates new 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 of all, let us define the new non-terminals and the way to decode their names.

Definition 4.6 *Let $G = (1, \mathcal{C}, NT, PP, I)$ be a TSG.*
$AheadNT(G) = \{\overset{i}{X}{}^t \mid X \in NT,\ i \in I\!N\ ,\ t \in T(\mathcal{C}, NT \times I\!N) \cup \{\epsilon\}\ \}$ *where ϵ is a new symbol coding "no term".*
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 constraint). 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 4.7 *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 if $t \neq \epsilon$ $t = c(t_1, \ldots, t_n)$. Moreover we denote,*
  $$NFr(s) = \bigcup_{\overset{i}{X}{}^t \in s} \{\overset{i}{X}{}_1^{t_1}, \ldots, \overset{i}{X}{}_n^{t_n}\} \text{ and } NFrP(s) = \bigcup_{\overset{i}{X}{}^t \in s} \{\overset{i}{X}{}^t \Rightarrow c(\overset{i}{X}{}_1^{t_1}, \ldots, \overset{i}{X}{}_n^{t_n})\}.$$
- minimally synchronized *if $\bigcup_{\overset{i}{X}{}^t \in s} term(\overset{i}{X}{}^t)$ 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, $NSn(s) = \{\{\overset{i_1}{X}{}_1^{t_1}, \ldots, \overset{i_n}{X}{}_n^{t_n}\}$ 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 $term(s)$. $NSnP(s) = \{\overset{j_1}{Y}{}_1^{s_1} \Rightarrow \overset{i_1}{X}{}_1^{t_1}, \ldots, \overset{j_n}{Y}{}_n^{s_n} \Rightarrow \overset{i_n}{X}{}_n^{t_n}\}$ *such that* $s = \{\overset{j_1}{Y}{}_1^{s_1}, \ldots, \overset{j_n}{Y}{}_n^{s_n}\}$, $\{\overset{i_1}{X}{}_1^{t_1}, \ldots, \overset{i_n}{X}{}_n^{t_n}\} \in NSn(s)$ *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.

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

$$\frac{D \quad\quad TD \cup \{s\} \quad\quad R}{D \cup \{s\} \quad TD \cup NFr(s) \setminus D \quad R \cup NFrPr(s)}\textbf{Free(3)}$$

$$\frac{D \quad\quad TD \cup \{s\} \quad\quad R}{D \cup \{s\} \quad TD \cup NSn(s) \setminus D \quad R \cup NSnPr(s)}\textbf{Sync(4)}$$

(1) if s is not free and $p = \overset{i}{Y}{}^t \Rightarrow \overset{i}{Y}{}^{t'}$ and $\overset{i}{Y}{}^t \in s$ such that $term(t) \Rightarrow term(t')$ with the free production $X \Rightarrow c(X_1, \ldots, X_n)$.
(2) if s is completely synchronized but not minimally synchronized.
(3) if s is free.
(4) 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[9].

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 3.3 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?

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

References

[1] A.C. Caron, F. Seynhaeve, S. Tison, and M. Tommasi. Solving One-step Rewriting Formulae. In *Proceedings 12th International Workshop on Unification, Rome (Italy)*. University of Roma, 1998.

[2] H. Comon, M. Dauchet, R. Gilleron, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications (TATA)*. http://www.lifl.fr/˜tommasi/tata.

[3] N. Dershowitz and J.-P. Jouannaud. Rewrite Systems. In J. Van Leuven, editor, *Handbook of Theoretical Computer Science*, chapter 6, pages 243–320. Elsevier Science Publishers, 1990.

[4] F. Jacquemard. *Automates d'Arbres et Réécriture de Termes*. Thèse de Doctorat d'Université, Université de Paris-sud, 1996. In french.

[5] S. Limet and P. Réty. E-Unification by Means of Tree Tuple Synchronized Grammars. In *Proceedings of 6th Colloquium on Trees in Algebra and Programming*, volume 1214 of *LNCS*, pages 429–440. Springer-Verlag, 1997.

[6] S. Limet and P. Réty. E-Unification by Means of Tree Tuple Synchronized Grammars. *Discrete Mathematics and Theoritical Computer Science (http://dmtcs.loria.fr/)*, 1:69–98, 1997.

[7] S. Limet and P. Réty. Solving Disequations modulo some Class of Rewrite Systems. In *Proceedings of 9th Conference on Rewriting Techniques and Applications, Tsukuba (Japon)*, volume 1379 of *LNCS*, pages 121–135. Springer-Verlag, 1998.

[8] S. Limet and F. Saubion. On partial validation of logic programs. In M. Johnson, editor, *proc of the 6th Conf. on Algebraic Methodology and Software Technology, Sydney (Australia)*, volume 1349 of *LNCS*, pages 365–379. Springer Verlag, 1997.

[9] S. Limet and F. Saubion. A general framework for r-unification. In C. Palamidessi, H. Glaser, and K. Meinke, editors, *proc of the Conf. on Principle of Declarative Programming (PLILP-ALP'98)*, volume 1490 of *LNCS*, pages 266–281. Springer Verlag, 1998.

[10] 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.

[11] J. Niehren, M. Pinkal, and P. Ruhrberg. On Equality up-to Constraints over Finite Trees ,Context Unification and One-step Rewriting. In D. Kapur, editor, *Proceedings 14th International Conference on Automated Deduction, Townsville (Australia)*, volume 1249 of *LNCS*, pages 34–48. Springer-Verlag, 1997.

[12] 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.

[13] 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.

[14] 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.

Appendix

A TTSG's for $\mathcal{L}'_1$ and $\mathcal{L}''_1$

The TTSG's that generate $\mathcal{L}'_1$ and $\mathcal{L}''_1$ are obtained by modifying Gr as explained below. About $\mathcal{L}'_1$:

- The language of rewrite rules is modified: this is the set of all pairs (l_{if}, r) where $l_{if} \to r$ is any rewrite rule whose lhs has been modified by replacing one arbitrary symbol by the corresponding symbol of Σ_{if}, and whose variables are replaced as done for $\mathcal{L}$, but one of $Var(l) \cap Var(r)$, arbitrarily chosen, which is replaced by D in l (D' in r), to cause a clash. Note that this language is still finite.
- $\mathcal{L}'_1$ contains in addition the productions of $GrT\overline{T}$, and each production of the form $\{X \Rightarrow T_1,\ X' \Rightarrow T'_1\}$ is replaced by $\{X \Rightarrow G,\ X' \Rightarrow G\}$.

About $\mathcal{L}''_1$:

- The language of rewrite rules is modified: this is the set of all pairs (l_{if}, r') where $l_{if} \to r$ is any rewrite rule whose lhs has been modified by replacing one arbitrary symbol by the corresponding symbol of Σ_{if}, u_c is any non variable occurrence of r (the clash occurrence) and r' is a term recursively defined (from bottom to top) by:
  - $r'|_{u_c} = f(G, \ldots, G)$ for an arbitrary $f \in \Sigma$ s.t. $f \neq r(u_c)$.
  - $\forall v < u_c$ (i.e. $u_c = v.i.w$), $r'|_v = r(u_c)(G, \ldots, G, r'|_{v.i}, G, \ldots, G)$ where $r'|_{v.i}$ is the i-th argument of $r(u_c)$.

  Note that this language is still finite.
- $\mathcal{L}''_1$ is obtained from $\mathcal{L}$ by replacing each production of the form $\{X \Rightarrow T_1,\ X' \Rightarrow T'_1\}$ by $\{X \Rightarrow G\}$.

B Intersection over several components

B.1 TSG intersection

Definition B.1 *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. G_3 is the TSG*
$G_3 = (S_1 + S_2, \mathcal{C}_1 \cup \mathcal{C}_2,\ NT_1 \times NT_2, Fr \cup Sn_1 \cup Sn_2, I_1 I_2)$ where

- *$Fr = \{X_1 X_2 \Rightarrow c(Y_{1,1}Y_{2,1}, \ldots, Y_{1,n}Y_{2,n})$ such that $X_1 \Rightarrow c(Y_{1,1}, \ldots, Y_{1,n}) \in P_1$ and $X_2 \Rightarrow c(Y_{2,1}, \ldots, Y_{2,n}) \in P_2\}$*
- *$Sn_1 = \{\{X_1 Y_1 \Rightarrow X'_1 Y_1, \ldots, X_n Y_n \Rightarrow X'_n Y_n\}_k$ such that $\{X_1 \Rightarrow X'_1, \ldots, X_n \Rightarrow X'_n\}_k \in P_1$ and $\forall i \in [1, n]\, Y_i \in NT_2\}$*

- $Sn_2 = \{X_1Y_1 \Rightarrow X_1Y_1', \dots, X_nY_n \Rightarrow X_nY_n'\}_{S_1+k}$ *such that* $\{\{Y_1 \Rightarrow Y_1', \dots, Y_n \Rightarrow Y_n'\}_k \in P_2$ *and* $\forall i \in [1,n]\; X_i \in NT_1\}$

Lemma B.2 *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.* $Rec(G_3) \subseteq Rec(G_1) \cap Rec(G_2)$.

In the following $(i_1, \dots, i_n) * (j_1, \dots, j_m)$ denotes $(i_1, \dots, i_n, j_1, \dots, j_m)$.

Proof: It is proved by showing on induction on the number of steps that for every derivation $\overset{ct_1*ct_2}{I_1I_2} \Rightarrow^* t$ there exist a derivation $\overset{ct_1}{I_1}\Rightarrow^* t_1$ of G_1 and another $\overset{ct_2}{I_2}\Rightarrow^* t_2$ of G_2 such that $\forall u \in O(t)$ either $t(u) \in \mathcal{C}_1 \cup \mathcal{C}_2$ and $t(u) = t_1(u) = t_2(u)$ or $t|_u = \overset{ct_1'*ct_2'}{X_1X_2}$ and $t_1|_u = \overset{ct_1'}{X_1}$ and $t_2|_u = \overset{ct_2'}{X_2}$.

For the a 0 step derivation its obvious. If we suppose the property true for all derivation with at most n steps, let us prove it is true for a $n+1$ derivation. Let $\overset{ct_1*ct_2}{I_1I_2} \Rightarrow^* t \Rightarrow s$ such a derivation. From induction hypothesis, we know that
there exist a derivation $\overset{ct_1}{I_1}\Rightarrow^* t_1$ of G_1 and another $\overset{ct_2}{I_2}\Rightarrow^* t_2$ of G_2 such that $\forall u \in O(t)$ either $t(u) \in \mathcal{C}_1 \cup \mathcal{C}_2$ and $t(u) = t_1(u) = t_2(u)$ or $t|_u = \overset{ct_1'*ct_2'}{X_1X_2}$ and $t_1|_u = \overset{ct_1'}{X_1}$ and $t_2|_u = \overset{ct_2'}{X_2}$.

Now if $t \Rightarrow s$ by the free production $Y_1Y_2 \Rightarrow c(Y_{11}'Y_{21}', \dots, Y_{1n}'Y_{2n}')$. From the definition of G_3 we know that $Y_1 \Rightarrow c(Y_{11}', \dots, Y_{1n}')$ is a free production of G_1 and $Y_2 \Rightarrow c(Y_{21}', \dots, Y_{2n}')$ is a free one of G_2. Since $t(u) = \overset{ct_1'*ct_2'}{Y_1Y_2}$, from the induction hypothesis we know that $t_1(u) = \overset{ct_1'}{Y_1}$ and $t_2(u) = \overset{ct_2'}{Y_2}$, so the two free productions apply to respectively to t_1 and t_2 at occurrence u. Moreover $s = t[u \leftarrow c(\overset{ct_1'*ct_2'}{Y_{11}'Y_{21}'}, \dots, \overset{ct_1'*ct_2'}{Y_{1n}'Y_{2n}'})]$ and $t_1 \Rightarrow t_1[u \leftarrow c(\overset{ct_1'}{Y_{11}'}, \dots, \overset{ct_1'}{Y_{1n}'})]$ and $t_2 \Rightarrow t_2[u \leftarrow c(\overset{ct_2'}{Y_{21}'}, \dots, \overset{ct_2'}{Y_{2n}'})]$.

If $t \Rightarrow s$ by the synchronized productions $\{Y_{11}Y_{21} \Rightarrow Y_{11}'Y_{21} \dots Y_{1n}Y_{2n} \Rightarrow Y_{1n}'Y_{2n}\}_l$ with $l \leq S_1$, applied on occurrences $u_1 \dots u_n$. From the definition of G_3, we know that $\{Y_{11} \Rightarrow Y_{11}' \dots Y_{1n} \Rightarrow Y_{1n}'\}_l$ is a synchronized pack of G_1. Since $t|_{u_i} = \overset{ct_{1i}'*ct_{2i}'}{Y_{1i}Y_{2i}}$ for $i \in [1,n]$, from the induction hypothesis we can deduce that $t_1|_{u_i} = \overset{ct_{1i}'}{Y_{1i}}$ and $t_2|_{u_i} = \overset{ct_{2i}'}{Y_{2i}}$. So $\{Y_{11} \Rightarrow Y_{11}' \dots Y_{1n} \Rightarrow Y_{1n}'\}_l$ applies on occurrences $u_1, \dots, u_n$ on t_1 giving the term s_1 equals to t_1 except for the occurrences u_i $i \in [1,n]$ which are $\overset{ct_{1i}''}{Y_{1i}'}$ (note that $s|_{u_i} = \overset{ct_{1i}''*ct_{2i}'}{Y_{1i}'Y_{2i}}$).

If $t \Rightarrow s$ be the synchronized productions $\{Y_{11}Y_{21} \Rightarrow Y_{11}Y_{21}' \dots Y_{1n}Y_{2n} \Rightarrow Y_{1n}Y_{2n}'\}_l$ with $l > S_1$, then we get the symmetric case of the former one.

Therefore, if $t \in Rec(G_3)$ then $t \in Rec(G_1)$ and $t \in Rec(G_2)$. □

Lemma B.3 *Let* G *be a balanced TSG, in any derivation* $\overset{ct}{I}\Rightarrow^* t$ *of* G, *if* $\{X_1 \Rightarrow Y_1, \dots, X_n \Rightarrow Y_n\}_l$ *applies at occurrences* $u_1 \dots u_n$ *then* $|u_1| = \dots = |u_n|$.

Proof: It can be easily proved by induction on the number of steps of the derivation. The main evidence coming the definition of balanced synchronizations (i.e. non leaning synchronization). □

Lemma B.4 *Let G be a balanced TSG, any derivation $\overset{ct}{I} \Rightarrow^* t_n$ where $t_n \in Rec(G)$ can be performed in the following way: $t_0 = \overset{ct}{I} \Rightarrow^* t_1 \Rightarrow^* \ldots \Rightarrow^* t_n$ where $\forall i \in 0, n$ the non-terminals of t_i are all at depth $i+1$.*

Proof: It is proved by induction on i. For $i = 0$, it obvious taking any derivation from I to t_n. Now let us suppose the for $i < m$ we have $t_0 = \overset{ct}{I} \Rightarrow^* t_1 \Rightarrow^* t_i \Rightarrow^* t_n$ and let us prove that $t_i \Rightarrow^* t_{i+1} \Rightarrow^* t_n$.

If t_i contains some synchronized non-terminals then we can apply on these non-terminals the synchronized pack of productions applied in the derivation $t_i \Rightarrow^* t_n$. This is possible because of lemma B.3. We obtain a term which non-terminals are at the same depth as in t_i because synchronized productions are empty. Now we can apply a free production on each non-terminal (each of these productions giving exactly one terminal) so we obtain $t_i \Rightarrow^* t_{i+1}$ where all non-terminals of t_{i+1} are 1 deeper than those of t_i. Since in $t_i \Rightarrow^* t_{i+1}$ we applied only productions of $t_i \Rightarrow^* t_n$ and since productions can be applied in a don't care way, $t_{i+1} \Rightarrow^* t_n$. $\Box$

Lemma B.5 *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 balanced TSG's and G_3 computed from definition B.1.*
$Rec(G_1) \cap Rec(G_2) \subseteq Rec(G_3)$.

Proof: Let $t \in Rec(G_1) \cap Rec(G_2)$, we have $\overset{ct_1}{I_1} \Rightarrow^* t$ and $\overset{ct_2}{I_2} \Rightarrow^* t$. Since G_1 and G_2 are balanced, from lemma B.4, we can use derivations of the form $t_i \Rightarrow^* t_{i+1}$ for G_1 and $s_i \Rightarrow^* s_{i+1}$ for G_2 where the non-terminals of t_i and s_i are at depth $i+1$ and those of t_{i+1} and s_{i+1} are at depth $i+2$. Now we can prove by induction on i that if s_i and t_i are equal on the occurrences labelled with terminals and non-terminals label the others, then there exists a derivation $\overset{ct_1 * ct_2}{I_1 I_2} = r_0 \Rightarrow r_1 \Rightarrow^* r_k$ of G_3 such that for each i r_i is equal to t_i and s_i on occurrences labelled by terminals and for occurrences u such that $t_i|_u = \overset{ct_1}{X_1}$ and $s_i|_u = \overset{ct_2}{X_2}$, we have $r_i|_u = \overset{ct_1 * ct_2}{X_1 X_2}$. For $i = 0$ it is obvious with the axioms of each grammar. Now for the induction step, let t_i, s_i and r_i the terms obtain for each grammar.

If a synchronized pack of productions applies on occurrences $u_1, \ldots, u_n$ of t_i giving t_i', since $r_i|_u = \overset{ct_1 * ct_2}{X_1 X_2}$ when $t_i|_u = \overset{ct_1}{X_1}$ and $s_i|_u = \overset{ct_2}{X_2}$; and from the definition of $Sync_1$ in definition B.1, we know that there exists a synchronized pack of productions of G_3 that applies on occurrences $u_1, \ldots, u_n$ the resulting term keeps the property on the non-terminal occurrences.

If a synchronized pack of productions applies on occurrences $u_1, \ldots, u_n$ of s_i with a similar reasoning as above we get the same result.

So we can suppose that we apply first on t_i and s_i the synchronized productions giving terms t_i'' and s_i'' we get the term r_i'' in the derivation of G_3. These terms respect the expected property

Now only free productions applies on non-terminals of t_i'' and s_i''. If we apply exactly one free production on each non-terminal t_i'' and s_i'', we obtain t_{i+1} and s_{i+1} which are equal on the terminal occurrences, then this means that $\forall u$ such that $t_i''|_u = \overset{ct_1}{X_1}$ and $s_i''|_u = \overset{ct_2}{X_2}$, there exist $X_1 \Rightarrow c(X_{11}, \ldots, X_{1n})$ in G_1 and $X_2 \Rightarrow c(X_{21}, \ldots, X_{2n})$ in G_2 so $X_1 X_2 \Rightarrow$

$c(X_{11}X_{21},\ldots,X_{1n}X_{2n})$ is in G_3 and this production applies on occurrence u of r''_i which is $\overset{ct_1*ct_2}{X_1X_2}$ so we obtain r_{i+1}. $\square$

B.2 From quasi-balanced TSG to balanced TSG

Here are two lemmas on quasi balanced TSG's which look like lemmas B.3 and B.4. These lemmas state that in any derivation of a quasi-balanced TSG the difference of depth of two synchronized non-terminals is bounded.

Lemma B.6 *Let G be a quasi-balanced TSG, in any derivation $\overset{ct}{I}\Rightarrow^* t$ of G, if $\{X_1 \Rightarrow Y_1,\ldots,X_n \Rightarrow Y_n\}_l$ applies at occurrences $u_1\ldots u_n$ then $|u_i| - |u_j|$ $i,j \in [1,n]$ is bounded.*

Proof: From the definition of leaning synchronizations we know that the application of this kind of may cause a difference of depth between two synchronized non-terminals. But since these leaning productions are not recursive and finite, they apply at most one time in a same branch in a derivation (i.e. if X belongs to a leaning synchronization at occurrence u, the no non-terminals of the same synchronization can appear at occurrence $u.v$ later in the derivation). So the maximal difference of depth of two synchronized non-terminals is bounded to the sum of the maximal difference of depth of the leaning synchronizations of G. $\square$

Lemma B.7 *Let G be a balanced TSG, any derivation $\overset{ct}{I}\Rightarrow^* t_n$ where $t_n \in Rec(G)$ can be performed in the following way: $t_0 = \overset{ct}{I}\Rightarrow^* t_1 \Rightarrow^* \ldots \Rightarrow^* t_n$ where $\forall i \in 0,n$ the non-terminals of t_i the higher non-terminal of each class of t_i is at depth $i+1$.*

The proof is similar to the proof of lemma B.4.

Now we have to prove that the applications of the inference rules always terminates in one hand, and that the new TSG defines the same language than the quasi-balanced one.

First of all, let us define an equivalence between AheadNT's. This equivalence states that the values of the control numbers are less important than the way they divide non-terminal occurrences into classes.

Definition B.8 *Let ρ be a bijection from $I\!N$ to $I\!N$ and $\overset{i}{X}{}^t$ a Ahead non-terminal. ρ is extended to Ahead non-terminals in the following way: $\rho(\overset{i}{X}{}^t) = \overset{\rho(i)}{X}{}^{\rho(t)}$ where $\rho(t)$ is the term t whose non-terminal occurrences have been replaced by $\overset{\rho(i)}{Y}$. ρ is called a* control renaming.

Two sets s_1 s_2 of AheadNT's are equivalent if there exists a control renaming ρ such that $\rho(s_1) = s_2$.

Lemma B.9 *Let G be a quasi-balanced TSG. The application of the inference rules on G always terminates.*

Proof: In fact we have to suppose that in the application of the inference rules, comparison between sets of AheadNT's are made modulo a control renaming.

First of all one can verify that for any set $\{\overset{i_1}{X_1^{t_1}}, \ldots, \overset{i_n}{X_n^{t_n}}\}$ of AheadNT's treated by the algorithm the set of control numbers $\{i_1, \ldots, i_n\}$ (keeping only one occurrence of each integer) is exactly the set of control numbers used in $term(\overset{i_1}{X_1^{t_1}}, \ldots, \overset{i_n}{X_n^{t_n}})$. It is true for the axiom. If it is true for a set s then if we apply

- **Free** and **Look Ahead** The new set s' introduced in TD still verifies the property because no control number changes.
- **Partition** From the definition of Pa, we know that the resulting set s' does not separate non-terminals (of the ahead-terms) with the same control number, so the property holds for each s'.
- **Sync** The elements of $Sn(s)$ verify the property because all the non-terminals are derived be a synchronized productions (so each control number i has been replaced by a unique control number j).

So from lemma B.6 we know that the difference of depth of two non-terminals with the same control number is bounded in a derivation of G. Since the AheadNT's are constructed from derivations of G the depth of the Ahead terms is bounded. So the number of possible AheadNT's is bounded and since the number of subsets of a finite set is finite the number of sets that can appear during the application of the inference rules is finite.
□

Lemma B.10 *Let $G = (1, \mathcal{C}, NT, P, I)$ be a quasi-balanced TSG and let $G' = (1, \mathcal{C}, NT', R, \overset{0}{I^{\epsilon}}$) where R is the set of productions resulting from the application of the algorithm, NT' is the set of non-terminals appearing in R. $Rec(G) = Rec(G')$.*

Proof:

First we prove that if $\overset{0}{I} \Rightarrow_P^* t_1 \Rightarrow_P^* \ldots \Rightarrow_P^* t_k$ corresponding to lemma B.7 then there exists a derivation $\overset{0}{\overset{0}{I^{\epsilon}}} \Rightarrow_R^* t'$ such that $term(t') = t_k$ and all non-terminals of t' are at depth $k+1$.

The proof is done on induction on k. When $k = 0$, it is obviously true with the axioms of each TSG. Now, what happens in $t_k \Rightarrow^* t_{k+1}$ Let us decompose this derivation as follows $t_k \Rightarrow^* s_1 \Rightarrow^* s_2 \Rightarrow^* t_{k+1}$, where in the derivation from t_k to s_1 all possible free productions are applied on non-terminal belonging to the same class than the synchronized non-terminals appearing at depth $k+1$. From s_1 to s_2 only synchronized productions are applied and from s_2 to t_{k+1} only one free production is applied on each non-terminal appearing at depth $k+1$.

The derivation from $t_k \Rightarrow^* s_1$ correspond exactly to the application of **Look Ahead** so $t' \Rightarrow^* t''$ with the corresponding empty free production generated by **Look Ahead**. All non-terminals of t'' are still at depth $k+1$ because only empty productions have been applied and of course $term(t'') = s_1$.

The derivation from s_1 to s_2 corresponds to one application of **Partition** and some applications of **Sync**. Indeed, thanks to the derivation $t_k \Rightarrow^* s_1$ all the synchronized non-terminals appearing at depth $k+1$ in s_1 (i.e. coming from t_k) can be derived. So for each of these non-terminal X we can define $\{u, u_1, \ldots, u_n\}$ such that $|u| = |u_1| = \ldots = |u_n|$, u is the occurrence of X in t and $\{u, u_1, \ldots, u_n\}$ is the smallest set such that $\{t|_u, t|_{u_1}, \ldots, t|_{u_n}\}$ contains only synchronized non-terminals and all these non-terminals can be derived. $\{t|_u, t|_{u_1}, \ldots, t|_{u_n}\}$ are the ahead terms of non-terminals labelling occurrences $u, u_1, \ldots, u_n$ of t'', so this set of AheadNT is minimally synchronized, so **Sync** have generated all possible synchronized productions. Therefore, for each of synchronized non-terminal of t and each synchronized production of G there exists a corresponding one in G'. We obtain t'''. from the definition of **Sync** $term(t''') = s_2$ and all the non terminals of t''' are still at depth $k+1$ because synchronized productions are empty.

Now the derivation $s_2 \Rightarrow^* t_{k+1}$ is performed in G' by applying the corresponding free production generated by **Free**. This gives the term t'''' such that $term(t'''') = t_{k+1}$. All non-terminals of t'''' are at depth $k+2$ because exactly one free production has been applied on each non-terminals of t''' and this production generate one terminal symbol.

Second, we prove that if $\overset{0}{\overset{0}{I}}{}^{\epsilon} \Rightarrow_R^* t'$ then there exists a derivation $\overset{0}{I} \Rightarrow_P^* t$ such that $term(t') = t$.

This property is true for the axioms of G and G'. Now if t' is derived from the axiom of G', we suppose that t is derived from the axiom of G and $term(t') = t$.

If a free production $\overset{i}{X}{}^{s} \Rightarrow \overset{i}{X}{}^{s'}$ generated by **Look Ahead** is applied on occurrence u of t' giving t'', then this means that $term(t')|_u = s$ can be derived by a free production of G giving s', so $t = term(t') \Rightarrow_P t[u \leftarrow s'] = term(t'')$

If a free production $\overset{i}{X}{}^{s} \Rightarrow c(\overset{i}{X}{}_1^{s_1}, \ldots \overset{i}{X}{}_1^{s_n})$ generated by **Free** is applied on occurrence u of t' giving t'' then by a similar reasoning as the former case, we can deduce that t can be derived by the corresponding free production of G.

If a synchronized pack of productions $\{\overset{i_1}{X}{}_1^{s_1} \Rightarrow \overset{i_1}{Y}{}_1^{s_1}, \ldots, \overset{i_n}{X}{}_n^{s_n} \Rightarrow \overset{i_n}{Y}{}_n^{s_n}\}$ on occurrences $u_1, \ldots, u_n$ of t' giving t'', from the definition of **Sync** and always with the same reasoning we can deduce that the corresponding synchronized productions can be applied on t.

So we have proved that each derivation (that corresponds to lemma B.7) of G is equivalent to a derivation of G' and each derivation G' corresponds to a derivation of G, therefore we can deduce that $Rec(G) = Rec(G')$.

□

Publications récentes du LIFO
LIFO's recent publications

97-7 E. MELIN, B. RAFFIN, X. REBEUF, B. VIROT
A Unifying Approach of Data and Task Parallelism Based on Structural Clock Communication Paradigm (extended version)

97-8 S. ANANTHARAMAN, Editor
Proceedings of UNIF'97, International Workshop on Unification

97-9 F. BENTAYEB, D. LAURENT
Inversion de l'algrèbre relationnelle et mises à jour

97-10 I. PARFENOFF
An efficient parallel algorithm for maximum matching in P_4-tidy graphs

97-11 F. ROUSSEL, I. RUSU
An O(m2+mn) algorithm to recognize Meyniel graphs

97-12 M. BAHMA, G. HAINS
Prevision des performances de la θ-jointure et de la multi-jointure pour des machines SN

97-13 F. LOULERGUE, G. HAINS
Design of an event-oriented functionnal parallel language : the finite case

97-14 S. LIMET, F. SAUBION
A proof method for logic programming

97-15 E. MELIN, B. RAFFIN, X. REBEUF, B. VIROT
An Automatic Distribution of Sequential Code Fitting Irregular Data Accesses

Pour récupérer les rapports de recherche du LIFO / To get LIFO's research reports :

- Par WWW : http://www.univ-orleans.fr/SCIENCES/LIFO/Recherche/Publis/RR/index.html
- Par FTP : ftp://ftp-lifo.univ-orleans.fr/pub/RR
- Par courrier papier adressé à / By surface mail adressed to :
  Arnaud Lallouet, LIFO, Université d'Orléans
  4, rue Léonard de Vinci, BP 6759
  F-45067 Orléans cedex 2, FRANCE
  en mentionnant votre adresse.
- Par courrier électronique à / By e-mail to : Arnaud.Lallouet@lifo.univ-orleans.fr