Discrete Mathematics and Theoretical Computer Science **1**, 1997, 69–98

E-unification by means of tree tuple synchronized grammars

Sébastien Limet and Pierre Réty

LIFO - Université d'Orléans, B.P. 6759, 45067 Orléans cedex 2, France
E-Mail: {`limet, rety`} `@lifo.univ-orleans.fr`

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

Keywords: E-unification, narrowing, tree languages, decidability

1 Introduction

First order E-unification [1] is a tool that plays an important role in automated deduction, in particular in functional logic programming and for solving symbolic constraints (see Baader and Siekmann[2] for an extensive survey of the area). It consists of finding instances to variables that make two terms equal modulo to an equational theory given by a set of equalities, i.e. it amounts to solving an equation (called a 'goal'). General E-unification is undecidable and may have infinitely many solutions. This is why E-unification procedures, like narrowing, often loop, enumerating an infinite set of unifiers or computing unproductive branches [3, 4, 5, 6, 7, 8, 9, 10]. From the programming point of view, it is very important to avoid infinite loops when possible.

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

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

A solution being defined by the instances of the variables of the goal, i.e. by a tuple of terms, and terms being trees, the set of solutions can be viewed as a tree tuple language. To describe this language, we introduce an *ad hoc* kind of grammar, the Tree Tuple Synchronized Grammars (TTSG). Their particularity is the notion of synchronization, i.e. the fact that some productions must be applied at the same time. For this reason TTSGs can define languages like $\{d(a^i(0), b^i(0), c^i(0))\}$. The class of languages defined by TTSGs is larger than we need, and does not have nice properties. Fortunately, we do not build many TTSGs from a unification problem, and the recognized languages have particular properties:

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

As Example 8.8 shows, symmetric binary trees appear to be the solution to some unification problem that satisfy our restrictions. Therefore, introducing a new kind of grammar was necessary because, as far as we know, there is no standard grammar or tree automaton technique that can express the symmetric binary trees, and whose emptiness is decidable.

Some authors have already used tree languages to represent infinite sets of solutions. For example inGilleron *et al.* [11], they are used to solve set constraints, but without synchronization. The notion of synchronization has already appeared in string grammars, for example, as in parallel communicating grammar systems [12] and in tree grammars [13]. However the TTSGs are not identical to the coupled context-free grammars of Guan *et al.* [13] because we need a finer control of synchronizations which is achieved thanks to a tuple of integers. The following example explains the principle of our procedure.

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

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

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

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

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

Soundness and completeness of the method come from

- the correspondence between narrowing and the derivations of TTSGs built by Step 1 (Theorem 6.5), as well as soundness and completeness of narrowing of the confluent constructor-based rewrite systems (Theorem 2.1), and
- soundness and completeness of the intersection algorithm (Theorem 7.9).

Decidability of the emptiness of languages recognized by TTSGs built from unification problems (Theorem 8.6), comes from the following facts. The number of leaves of the computations (the tree tuples derived from the axiom) is of course not bounded. However, when considering a TTSG that comes from a unification problem, and thanks to the control, the leaves of any computation can be divided into subsets, called *transclasses* (Definition 7.3), whose size is bounded. Then by pumping techniques, emptiness can be tested.

This paper is organized as follows. Sect. 2 recalls some basic notions of rewriting techniques. Sect. 3 describes related work on the decidability of unifiability. Our restrictions are given in Sect. 4, where some undecidability results are also given. The TTSG is defined in Sect. 5, and the two steps of our algorithm are respectively given in Sects. 6 and 7. Then Sect. 8 sets our decidability result. Sect. 9 concludes the paper, and discusses future extensions to this work.

2 Preliminaries

We recall some basic notions that concern rewriting techniques. For more details see Dershowitz and Jouannaud [14].

Let Σ be a finite set of symbols and V be an infinite set of variables, $T_{\Sigma \cup V}$ is the first-order *term* algebra over Σ and V. Σ is partitioned in two parts: the set $\mathcal{F}$ of *function symbols*, and the set $\mathcal{C}$ of *constructors*. The terms of $T_{\mathcal{C} \cup V}$ are called *data terms*. A term is said to be *linear* if it does not contain several times the same variable.

Let t be a term, $\boldsymbol{O(t)}$ is the set of *occurrence*s of t, $\boldsymbol{t|_u}$ is the subterm of t at occurrence u, and $\boldsymbol{t(u)}$ is the symbol that labels the occurrence u of t. $\boldsymbol{t[u \leftarrow s]}$ is the term obtained by replacing in t the subterm at occurrence u by s. A *substitution* σ is a mapping from V into $T_{\Sigma \cup V}$ whose domain $D(\sigma) = \{x \in V \mid \sigma x \neq x\}$ is assumed to be finite. It is trivially extended to a mapping from $T_{\Sigma \cup V}$ to $T_{\Sigma \cup V}$. A *data substitution* σ is a substitution such that, for each variable x, σx is a data term. The operator **.** denotes the composition of substitutions. $\boldsymbol{\sigma|_{Var(t)}}$ is the restriction of σ to the set of variables of t.

In the following x, y, z denote variables, s, t, l, r denote terms, f, g, h function symbols, c is a constructor symbol, u, v, w are occurrences, and σ, θ are substitutions.

We generalize the occurrences (as well as the above notation) to tuples in the following way: let $p = (p_1, \ldots, p_n)$ a tuple, $\forall i \in [1, n]$ $p|_i = p_i$, and when the p_i's are terms, $p|_{i.u} = p_i|_u$. Moreover, we define the *concatenation* of two tuples by $(t_1, \ldots, t_n) * (t'_1, \ldots, t'_{n'}) = (t_1, \ldots, t_n, t'_1, \ldots, t'_{n'})$ and the *component elimination* by $(t_1, \ldots, t_i, \ldots, t_n)\backslash_i = (t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n)$

A Term Rewrite System (TRS) is a finite set of oriented equations called rewrite rules or *rules*; *lhs* means left-hand side and *rhs* means right-hand side. For a TRS R, the *rewrite relation* is denoted by $\boldsymbol{\rightarrow_R}$ and is defined by $t \rightarrow_R s$ if there exists a rule $l \rightarrow r$ in R, a non-variable occurrence u in t, and a substitution σ, such that $t|_u = \sigma l$ and $s = t[u \leftarrow \sigma r]$. The reflexive-transitive closure of $\rightarrow_R$ is denoted by $\boldsymbol{\rightarrow^*_R}$, and the symmetric closure of $\rightarrow^*_R$ is denoted by $\boldsymbol{=_R}$.

A TRS is said to be *confluent* if $t \rightarrow^*_R t_1$ and $t \rightarrow^*_R t_2$ implies $t_1 \rightarrow^*_R t_3$ and $t_2 \rightarrow^*_R t_3$ for some t_3. If the lhs (resp. rhs) of every rule is linear, the TRS is said to be *left-* (resp. *right-*)*linear*. If it is both left and right-linear the TRS is said to be *linear*. A TRS is *constructor based* if every rule is of the form $f(t_1, \ldots, t_n) \rightarrow r$, where the t_i's are data terms.

The (data)-substitution σ is a *(data)-$\boldsymbol{R}$-unifier* of t and t' if $\sigma t =_R \sigma t'$. The set S of substitutions is a *complete set of (data)-R-unifiers* of t and t' if it contains only (data)-R-unifiers of t and t', and for each (data)-R-unifier σ of t and t' there exist $\theta \in S$ and a substitution ρ such that $\sigma =_R \rho.\theta$. The theory defined by the TRS R is *finitary* if every pair of terms has at least a finite complete set of R-unifiers. When $R = \emptyset$ the most general unifier is denoted $\boldsymbol{mgu}$.

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

Theorem 2.1 *TRS R being confluent and constructor based, the set of data substitutions σ such that*

- *there exists a narrowing derivation*

$$t_0 \stackrel{?}{=} t'_0 \rightsquigarrow_{[\sigma_1]} t_1 \stackrel{?}{=} t'_1 \rightsquigarrow \ldots \rightsquigarrow_{[\sigma_n]} t_n \stackrel{?}{=} t'_n$$

  such that t_n and t'_n are unifiable by the mgu θ,

- *and $\sigma = \theta.\sigma_n \ldots \sigma_1|_{Var(t_0) \cup Var(t'_0)}$*

is a complete set of data-R-unifiers of t_0 and t'_0.

This is a consequence of the lifting lemma [3]. Note that the termination of R is not required here because we look only for data solutions, which are in normal form. So we do not need to normalize them before applying the lifting lemma.

3 Related Decidability Results

In term rewriting, some authors have already established decidability results for unifiability, assuming some restrictions on the TRS. The first result assumed the rewrite system to be ground [15]. Hullot [3] extended it to rewrite systems whose rhs's are either variables or ground terms (Mitra [16] allows rhs's to be data terms). Actually, these results are very restrictive because they forbid recursivity.

Christian [17] defines a new criterion: every rewrite rule's lhs is flat ($f(s_1, \ldots, s_n)$ is flat if $\forall i \in [1, n]$, s_i is either a variable or a ground data term), and the rewrite rules are oriented by a well founded ordering. Comon *et al.* [18] show that decidability also holds for shallow rewrite systems (the sides of rewrite rules have variables occurring at a depth of at most one). Nieuwenhuis [19] extends the shallow theories to standard theories that allow non-shallow variables.

The restriction of Kapur and Narendran [20], extended by Mitra [16] imposes that for every rule, every subterm of the rhs having a function symbol on top is a strict subterm of the lhs. For all these restrictions the theory is finitary, i.e. there always exists a finite complete set of unifiers. Most decidability proofs are thus based on the fact that there exists a complete narrowing strategy whose search space is always finite.

As regards non-finitary theories, a decidability result is established by Mitra [16, 21] for constructor-based rewrite systems, assuming that, for every function symbol f, there is at most one rewrite rule among the rules defining f that does not have a data term as the rhs. Moreover, this rhs must contain only one function symbol, and the subterm rooted by this function is flat in the sense of Christian [17]. Thanks to the notion of iterated substitution, Mitra is able to represent finitely the infinite set of unifiers and decide upon unifiability.

Kaji *et al*. [22] give a procedure that, when it terminates, decides upon unifiability by means of tree automata. They assume linearity for the goal, right linearity and (nearly) left linearity for the TRS. Unfortunately, their procedure does not represent the set of solutions, and does not terminate for an example like $\{s(x)+y \rightarrow s(x+y),\ 0+x \rightarrow x\}$ because of the superposition of $s(x)$ with $s(x+y)$. Note that our algorithm works (and terminates) for this example when solving linear equations, since our restrictions are satisfied (see Sect. 4).

Faßbender and Maneth [23] give a decision procedure for unifiability without representing the set of solutions. However, they need very strong restrictions. Only one function can be defined, and every constructor and every function is monadic (i.e. admits at most one argument).

4 Additional Restrictions and their Need

The four additional restrictions are:

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

These restrictions together define a class of rewrite systems incomparable with those of related work, and they allow non-finitary theories. For example, the rewrite system $\{f(s(x)) \rightarrow f(x),\ f(0) \rightarrow 0\}$. The (even minimal) complete set of solutions, and also the narrowing search space, may be infinite.

To show that the above restrictions are necessary all together to get the decidability of unifiability, we prove that if one of them is removed, then there exists a rewrite system satisfying the other three, and that encodes a well-known undecidable problem, the Post correspondence problem.

Definition 4.1 Post Correspondence Problem (Post 1946)
Let A and C be finite disjoint alphabets and ϕ and ϕ' be two morphisms from A^* to C^*. The Post correspondence problem for ϕ and ϕ' consists of finding a non-empty sequence $\alpha \in A^+$ such that $\phi(\alpha) = \phi'(\alpha)$.

Theorem 4.2 *There exists no uniform algorithm for solving the Post correspondence problem. The problem remains undecidable when ϕ and ϕ' are injective.*

We use the following code: given $A = \{a_1, \ldots, a_n\}$ and $C = \{c_1, \ldots, c_m\}$, to each $a_i \in A$ (resp. $c_i \in C$) we associate the unary symbol a_i (resp. c_i). The constant $\perp$ represents the end of words. Thus, the word abc is represented by the term $a(b(c(\perp)))$. If ω denotes the word abc, $\omega(\perp)$ denotes the term $a(b(c(\perp)))$. In the proofs $\perp$ will sometimes be omitted to simplify the expressions. We need to represent prefixes, i.e. beginning of words whose end is unknown. To do this we use non-ground terms, for example, the term $a(b(c(x)))$ denotes the prefix abc.

Lemma 4.3 *Linearity of rewrite rules is necessary to decide unifiability.*

Proof. Let $R_1 =$

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

where ω_i and ω'_i respectively denote $\phi(a_i)$ and $\phi'(a_i)$.

R_1 respects all the restrictions but linearity. Obviously, f encodes ϕ and f' encodes ϕ'. Then for any words $\alpha, \beta \in A^*$, the value of $h(\alpha)$ (resp. $h'(\beta)$) is $c(\alpha, \phi(\alpha))$ (resp. $c(\beta, \phi'(\beta))$). Therefore, $h(\alpha) = h'(\beta)$ implies $c(\alpha, \phi(\alpha)) = c(\beta, \phi'(\beta))$, and so $\alpha = \beta$ and $\phi(\alpha) = \phi'(\alpha)$ because c is a constructor. Thus, looking for α and β different from $\perp$ such that $h(\alpha) =_{R_1} h'(\beta)$ amounts to solving the Post correspondence problem.

If we can decide unifiability under restrictions 2–4, we can do it for any linear goal, then for all the goals $c_i = h(a_i(x)) \stackrel{?}{=} h'(a_i(y))$ $(i = 1, \ldots, n)$. This amounts to deciding the unifiability of $h(\alpha) \stackrel{?}{=} h'(\beta)$ forbidding $\alpha = \perp$ and $\beta = \perp$. This is therefore impossible. □

Lemma 4.4 *Forbidding σ_{in} is necessary to decide unifiability.*

Proof. The following rewrite system comes from Domenjoud [24]. Let $R_2 =$

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

where ω_i and ω'_i respectively denote $\phi(a_i)$ and $\phi'(a_i)$.

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

$$\begin{array}{lll} f(a_1(a_2(\perp)), \perp) & \to & \omega_1(f(a_2(\perp), a_1(\perp))) \\ & \to & \omega_1(\omega_2(f(\perp, a_2(a_1(\perp))))) \\ & \to & \omega_1(\omega_2(h(a_2(a_1(\perp))))) \end{array}$$

For the same reasons, $f'(\beta, \perp) \to^* \phi'(\beta)(h(\overline{\beta}))$.

Therefore, $f(\alpha, \perp) = f'(\beta, \perp)$ implies $\phi(\alpha)(h(\overline{\alpha})) = \phi'(\beta)(h(\overline{\beta}))$, and then $\phi(\alpha) = \phi'(\beta)$ and $\alpha = \beta$. Thus, $\phi(\alpha) = \phi'(\alpha)$, i.e. solving $f(\alpha, \perp) \stackrel{?}{=} f'(\beta, \perp)$ amounts to solve the Post correspondence problem, if $\alpha = \perp$ and $\beta = \perp$ are forbidden.

We conclude this proof in a similar way as the previous one, by considering the linear goals

$$c_i = f(a_i(x), \perp) \stackrel{?}{=} f'(a_i(y), \perp),\ i = 1, \ldots, n$$

□

Lemma 4.5 *Forbidding nested functions is necessary to decide unifiability.*

Proof. Let $R_3 =$

$$\begin{array}{llll} \{ & f(a_i(x), h(y)) & \rightarrow & \omega_i(f(x, g_i(y))) \mid i = 1, \ldots, n \\ & f'(a_i(x), h(y)) & \rightarrow & \omega'_i(f'(x, g_i(y))) \mid i = 1, \ldots, n \\ & f(\perp, h(y)) & \rightarrow & h(y) \qquad f'(\perp, h(y)) \rightarrow h(y) \\ & g_i(y) & \rightarrow & h(a_i(y)) \mid i = 1, \ldots, n \qquad \} \end{array}$$

where ω_i and ω'_i denote respectively $\phi(a_i)$ and $\phi'(a_i)$.

This rewrite system looks like the previous one, except that the second argument appearing under the constructor h, which avoids σ_{in}'s but introduces nested functions. Since the nested function $g_i(y)$ rewrites into $h(a_i(y))$, we get the same situation as in the previous proof if we consider the goals

$$c_i = f(a_i(x), h(\perp)) \stackrel{?}{=} f'(a_i(y), h(\perp)), \; i = 1, \ldots, n$$

□

Lemma 4.6 *Linearity of the goal is necessary to decide unifiability.*

Proof. Let $R_4 =$

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

where ω_i and ω'_i respectively denote $\phi(a_i)$ and $\phi'(a_i)$.

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

5 Formal Definitions of TTSGs

Only formal definitions are given here. For motivations and examples see Sects. 6 and 7.

NT is a finite set of non-terminals and the terminals are the constructors of the signature. Upper case letters denote elements of NT.

Definition 5.1 A *production* is a rule of the form $X \Rightarrow t$, where $X \in NT$ and $t \in T_{\mathcal{C} \cup NT}$. A *pack of productions* is a set of productions coupled with a non-negative integer (called *level*) and denoted $\{X_1 \Rightarrow t_1, \ldots, X_n \Rightarrow t_n\}_k$.

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

Definition 5.2 A *tuple of control* is a tuple of non-negative integers.

Definition 5.3 A *TTSG* is defined by a 5-tuple $(Sz, \mathcal{C}, NT, PP, TI)$, where

- Sz is a positive integer that defines the size of tuples of control,

- $\mathcal{C}$ is the set of constructors (*terminals* in the terminology of grammars),
- NT is the finite set of non-terminals,
- PP is a finite set of packs of productions of a level less than or equal to Sz,
- TI is the axiom of the TTSG. It is a tuple $((I_1, ct_1), \dots, (I_n, ct_n))$, where every I_i is a non-terminal and every ct_i is a Sz-tuple of control that may contain 0's and $\perp$'s.

$\perp$ means that this component of the control is not used. In fact, Sz is the number of intersections $+1$ used to build the grammar. This is why in Sect. 6, no intersections between TTSGs having yet been computed, the tuples of control are 1-tuples, i.e. non-negative integers.

A computation of the grammar is a tuple of trees derived from the axiom by applying productions. In a computation, a tuple of control is associated with each non-terminal occurrence. The control is for simulating variable renaming within narrowing. At this moment, single integers may suffice, but we need tuples of integers in order to get stability of TTSGs by intersection.

Intuitively, a free production $X \Rightarrow c(Y_1, \dots, Y_n)$ can be applied as soon as X appears in a computation of the grammar, and then $Y_1, \dots, Y_n$ preserves the same control as X. On the other hand, a pack of productions $\{X_1 \Rightarrow Y_1, \dots, X_n \Rightarrow Y_n\}_k$ can be applied iff $X_1, \dots X_n$ occur in the same computation and the kth components of their controls are identical (and are not $\perp$). The X_i's are then replaced by the Y_i's, and the kth component of control is set to a new fresh value.

Definition 5.4 The set of *computations* of a TTSG $Gr = (Sz, \mathcal{C}, NT, PP, TI)$, denoted $\boldsymbol{Comp(Gr)}$, is the smallest set defined by:

- TI is in $Comp(Gr)$,
- if tp is in $Comp(Gr)$ and $tp|_u = (X, ct)$, and the free production $X \Rightarrow c(Y_1, \dots, Y_n)$ is in PP, then $tp[u \leftarrow c((Y_1, ct), \dots, (Y_n, ct))]$ is in $Comp(Gr)$,
- if tp is in $Comp(Gr)$ and there exist n pairwise different occurrences $u_1, \dots, u_n$ of tp, such that $\forall i \in [1, n]\ tp|_{u_i} = (X_i, ct_i)$ and $ct_i|_k = a$ ($a \in \mathbb{N}$), and the pack of productions $\{X_1 \Rightarrow Y_1, \dots, X_n \Rightarrow Y_n\}_k \in PP$, then $tp[u_1 \leftarrow (Y_1, ct_1[k \leftarrow b])] \dots [u_n \leftarrow (Y_n, ct_n[k \leftarrow b])]$ (where b is a new integer) is in $Comp(Gr)$.

The symbol $\Rightarrow$ also denotes the above two deduction steps; a *derivation* of Gr is a sequence of computations $TI \Rightarrow tp_1 \Rightarrow \dots \Rightarrow tp_n$.

The *i*th *component* of a TTSG denotes any tree appearing as the *i*th component of a computation.

Definition 5.5 The language *recognized* by a TTSG Gr, denoted $\boldsymbol{Rec(Gr)}$, is the set of tuples of ground data terms $Comp(Gr) \cap T_{\mathcal{C}}^n$.

6 Step 1: Transformation of a TRS into TTSGs

This is the first step of our method. In the rest of the paper, the TRS is assumed to be confluent and constructor-based, and satisfies restrictions (1)–(4). The aim is to convert the TRS and the goal into several TTSGs that simulate narrowing.

TTSGs contain only regular productions (free productions) and synchronization constraints (synchronized productions). This is because, thanks to the restrictions, we have to simulate only particular narrowing steps:

- We start from a linear term, since the goal is linear. Consider the narrowing step $t \rightsquigarrow_{[u,l\rightarrow r,\sigma]} t' = (\sigma t)[u \leftarrow \sigma r]$, where t is linear. Now $t' = \sigma t[u \leftarrow r]$ because there is no σ_{in}, and then $t' = t[u \leftarrow r]$ because t is linear. Thus, the resulting term t' is simple (σ does not apply), and can be easily expressed by a grammar. This property is still preserved when applying a narrowing step on t', since t' is linear thanks to the linearity of r.
- Nested functions on rhs's may create σ_{in}, as shown in the proof of Lemma 4.5.
- Consider now the narrowing derivation
$$t_0 \rightsquigarrow_{[\sigma_1]} t_1 \rightsquigarrow \ldots \rightsquigarrow_{[\sigma_n]} t_n$$
The narrowing substitutions are composed of subterms of lhs's. If lhs's are not linear, then it may be that the term $\sigma_n \ldots \sigma_1 x$ contains 2^n times the same variable y. In other words, the number of variables that must be replaced next by the same term is not bounded. This cannot be easily expressed by a grammar.

Without the restrictions, we would get context-sensitive productions whose emptiness would be undecidable.

Consider again Example 1.1. For this example, three TTSGs will be constructed: one for $g(x)$, one for $f(y_2)$ and one for 0. Intuitively, the TRS and the goal are considered as a kind of tree automaton (in fact several automata), where states are the occurrences of the terms and transitions are subterm relationships and unification relationships. The idea is to extract tree grammars from the automata. Recall that the terminals of the grammars are the constructors.

6.1 Non-Terminals

To each occurrence of each term of the TRS and the goal we associate a non-terminal. Next the productions will be deduced from subterm relations and syntactic unifications. For each rewrite rule i (see Figure 1):

- to each non-variable occurrence u of the lhs (resp. rhs) is associated the non-terminal L^i_u (resp. R^i_u), except when $u = \epsilon$, where we even associate R^i_u to the lhs,
- to the occurrences of variables $x, y, \ldots$ are associated the non-terminals $X^i, Y^i, \ldots$. There is an exception when x is on a rhs and is the leaf of a branch that contains only constructors. In this case, we associate X'^i. This avoids confusion between the occurrences of x in the lhs and rhs.

In the same way, for the goal:

- the non-terminal G^l_u (resp G^r_u) is associated with each non-variable occurrence u of the lhs (resp. rhs) of the goal,
- the non-terminals $X^l, Y^l, \ldots$ (resp $X^r, Y^r, \ldots$) are associated with the occurrences of the variables $x, y, \ldots$ of the goal.

$NT(t, u)$ denotes the non-terminal associated with the occurrence u of t.

An additional non-terminal A^l_u (resp. A^r_u) is associated with the non-variable arguments of the function of the goal (here, occurrence 1 of $f(g(x))$ to encode the variable y_2). t being a side of the goal, $ANT(t, u)$ denotes the additional non-terminal associated with the occurrence u of t.

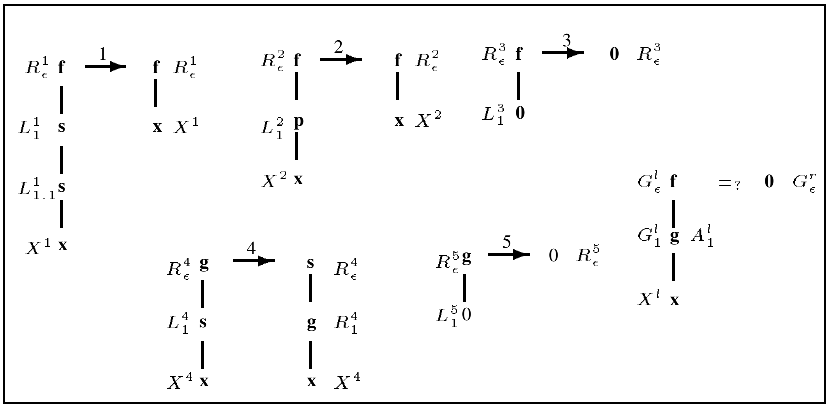

Fig. 1:

6.2 Productions

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

6.2.1 Explanations

The way in which the productions are deduced is motivated by narrowing techniques. From the correspondence between rewriting and narrowing (lifting lemma [3]), the languages $\mathcal{L}_1$, $\mathcal{L}_2$ of Example 1.1 are the ground instances of the data solutions computed by narrowing. This is why we look for narrowing possibilities. For instance, the subterm $g(x)$ of the rhs of Rule (4) in Example 1.1 unifies with the lhs of the same rule. Therefore, the narrowing step $g(x) \rightsquigarrow_{[\epsilon, r_4, x \mapsto s(x')]} s(g(x'))$ is possible. This step achieves two operations: it maps the variable x to $s(x')$; and it sets the result of the narrowing step to $s(g(x'))$.

From the TTSG's point of view, this narrowing step is simulated as follows. The subterm $g(x)$ is represented by the non-terminal R^4_1 (see Figure 1) and the variable x by X^4. Therefore, the pair (R^4_1, X^4) encodes $(g(x), x)$. The fact that $g(x)$ unifies with $g(s(x'))$ (the renamed version of the lhs of Rule 4) is encoded by the empty production $R^4_1 \Rightarrow R^4_\epsilon$. The fact that the previous unification instantiates x is encoded by the empty production $X^4 \Rightarrow L^4_1$. To force these two operations to be achieved at the same time, the two productions are synchronized in the pack of productions $\{R^4_1 \Rightarrow R^4_\epsilon,\ X^4 \Rightarrow L^4_1\}$. Thus, when it is applied on (R^4_1, X^4), we get (R^4_ϵ, L^4_1), which means that the unification is about to be done, and therefore so is the narrowing step, but the new constructors produced by the unification and the narrowing step have not yet appeared.

This is the aim of the free productions deduced from subterm relationships. In our example, we have just narrowed $g(x)$ on top with Rule (4), and we get $s(g(x'))$. So the narrowing step generates a term with the constructor s on top whose argument is the function call $g(x')$. This is encoded by the free production $R^4_\epsilon \Rightarrow s(R^4_1)$. In the same way, x is instantiated by $s(x')$ which is encoded by the free production $L^4_1 \Rightarrow s(X^4)$. The narrowing step is achieved entirely by the derivation

$$(R^4_\epsilon, L^4_1) \Rightarrow (s(R^4_1), L^4_1) \Rightarrow (s(R^4_1), s(X^4))$$

One can easily see that a second application of Rule (4) on $s(g(x'))$ can be simulated by again applying the pack of productions and then the two free productions.

6.2.2 Making Productions

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

$L^1_1 \Rightarrow s(L^1_{1.1}),\ L^1_{1.1} \Rightarrow s(X^1),\ L^2_1 \Rightarrow p(X^2),\ L^3_1 \Rightarrow 0,\ R^3_\epsilon \Rightarrow 0,$
$L^4_1 \Rightarrow s(X^4),\ R^4_\epsilon \Rightarrow s(R^4_1),\ L^5_1 \Rightarrow 0,\ R^5_\epsilon \Rightarrow 0,\ G^r_\epsilon \Rightarrow 0$

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

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

where $v_1,\dots v_n$ are the variable occurrences of $r^i|_u$. Note that if $\theta = mgu(r^i|_u, l^j)$, from the σ_{in} restriction we know that $\theta r^i|_u = l^j$, therefore $v_1,\dots,v_n$ are also occurrences of l^j.

For our example, r^1 unifies with l^1, l^2 and l^3, which gives the synchronized productions

$$\{R^1_\epsilon \Rightarrow R^1_\epsilon, X^1 \Rightarrow L^1_1\},\ \{R^1_\epsilon \Rightarrow R^2_\epsilon, X^1 \Rightarrow L^2_1\},\ \{R^1_\epsilon \Rightarrow R^3_\epsilon, X^1 \Rightarrow L^3_1\}$$

r^2 also unifies with l^1, l^2 and l^3, so we get

$$\{R^2_\epsilon \Rightarrow R^1_\epsilon, X^2 \Rightarrow L^1_1\},\ \{R^2_\epsilon \Rightarrow R^2_\epsilon, X^2 \Rightarrow L^2_1\},\ \{R^2_\epsilon \Rightarrow R^3_\epsilon, X^2 \Rightarrow L^3_1\}$$

Finally, $r^4|_1$ unifies with l^4 and l^5, so we get

$$\{R^4_1 \Rightarrow R^4_\epsilon, X^4 \Rightarrow L^4_1\},\ \{R^4_1 \Rightarrow R^5_\epsilon, X^4 \Rightarrow L^5_1\}$$

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

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

The language derived from $NT(t,u)$ expresses the terms issued by narrowing from $t(u)(x_1,\dots,x_n)$, while the languages derived from $ANT(t,u.i)$ express the instances of the fictitious variables x_i. In Example 1.1, $f(y_2)$ unifies with l^1, l^2 and l^3; this gives the synchronized productions

$$\{G^l_\epsilon \Rightarrow R^1_\epsilon, A^l_1 \Rightarrow L^1_1\},\ \{G^l_\epsilon \Rightarrow R^2_\epsilon, A^l_1 \Rightarrow L^2_1\},\ \{G^l_\epsilon \Rightarrow R^3_\epsilon, A^l_1 \Rightarrow L^3_1\}$$

$g(x)$ unifies with l^4 and l^5, so we get

$$\{G^l_1 \Rightarrow R^4_\epsilon, X^l \Rightarrow L^4_1\},\ \{G^l_1 \Rightarrow R^5_\epsilon, X^l \Rightarrow L^5_1\}$$

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

6.3 Productions to Express Ground Instances

The languages we want to express are the ground data instances of the solutions provided by narrowing. The productions described so far express the solutions provided by narrowing. Let us see on an example of the problem. We consider the rewrite rule

$$f(x) \xrightarrow{1} 0$$

and the term $f(y)$. The productions associated with this problem are

$$R^1_\epsilon \Rightarrow 0, \{G^l_\epsilon \Rightarrow R^1_\epsilon, Y \Rightarrow X^1\}$$

From the pair (G^l_ϵ, Y) which represents $(f(y), y)$, we get the derivation:

$$(G^l_\epsilon, Y) \Rightarrow (R^1_\epsilon, X^1) \Rightarrow (0, X^1)$$

No more production is applicable, therefore from a grammar point of view, this language is empty because of the non-terminal X^1. The meaning of this derivation is that $f(y)$ narrows to 0 for any value of y. Since we are only interested in ground data solutions, the non-terminal X^1 has to be derived into any ground data term. For this, the grammar that generates any term of $T_{\mathcal{C}}$ is constructed.

A new non-terminal ANY is introduced, and for all constructors $c \in \mathcal{C}$, the free production $ANY \Rightarrow c(ANY, \ldots, ANY)$ is created. The variables concerned with the problem are any variables x^j appearing on the lhs of the rewrite rule but not on the rhs, or variables appearing in the goal in a branch that contains only constructors. For all these variables the production $X^j \Rightarrow ANY$ is created. For technical reasons, this production is considered as synchronized, because it does not generate any constructors.

In the previous example, the only production created for ANY is $ANY \Rightarrow 0$, because 0 is the only one. $X^1 \Rightarrow ANY$ is created for X^1. Thus, the derivation terminates as follows:

$$(0, X^1) \Rightarrow (0, ANY) \Rightarrow (0, 0)$$

In fact, the problem we just solved may be more complicated if a rewrite rule has the form

$$f(x) \xrightarrow{2} s(x)$$

because in this case, we have to simulate that the x occurring on the lhs and the x occurring on the rhs must be instantiated by the same ground data term.

This can easily be simulated thanks to some synchronized productions, as follows (introduction of the control is necessary to understand why these productions work well):

- $\forall c \in \mathcal{C}$ the free productions $ANY^c_{Res} \Rightarrow c(ANY^1_{Res}, \ldots, ANY^n_{Res})$ and $ANY^c_{Var} \Rightarrow c(ANY^1_{Var}, \ldots, ANY^n_{Var})$ are created.
- For all integers i between 1 and the maximal arity of the constructors and $\forall c \in \mathcal{C}$, the synchronized productions $\{ANY^i_{Res} \Rightarrow ANY^c_{Res}, ANY^i_{Var} \Rightarrow ANY^c_{Var}\}$ are created.

The variables concerned with this new case are those appearing in a branch on a rhs of the rewrite rule which contains only constructors. For each of these variables x^j and for each constructor $c \in \mathcal{C}$, the synchronized productions $\{X'^j \Rightarrow ANY^c_{Res}, X^j \Rightarrow ANY^c_{Var}\}$ are created.

Now consider the term $f(y)$. If the only constructors are 0, s, the productions associated with this problem are

$$\begin{array}{c}\{G^l_\epsilon \Rightarrow R^2_\epsilon,\, Y \Rightarrow X^2\},\, R^2_\epsilon \Rightarrow s(X'^2)\\ \{X'^2 \Rightarrow ANY^0_{Res}, X^2 \Rightarrow ANY^0_{Var}\},\, \{X'^2 \Rightarrow ANY^s_{Res}, X^2 \Rightarrow ANY^s_{Var}\}\\ \{ANY^0_{Res} \Rightarrow 0, ANY^0_{Var} \Rightarrow 0\},\, \{ANY^s_{Res} \Rightarrow s(ANY^1_{Res}), ANY^s_{Var} \Rightarrow s(ANY^1_{Var})\}\\ \{ANY^1_{Res} \Rightarrow ANY^0_{Res}, ANY^1_{Var} \Rightarrow ANY^0_{Var}\},\, \{ANY^1_{Res} \Rightarrow ANY^s_{Res}, ANY^1_{Var} \Rightarrow ANY^s_{Var}\}\end{array}$$

From the pair (G^l_ϵ, Y), which represents $(f(y), y)$, a possible derivation is:

$$\begin{array}{c}(G^l_\epsilon, Y) \Rightarrow (R^2_\epsilon, X^2) \Rightarrow (s(X'^2), X^2) \Rightarrow (s(ANY^s_{Res}), ANY^s_{Var})\\ \Rightarrow (s(s(ANY^1_{Res})), s(ANY^1_{Var})) \Rightarrow (s(s(ANY^0_{Res})), s(ANY^0_{Var})) \Rightarrow (s(s(0)), s(0))\end{array}$$

6.4 Grammars

Many productions have been deduced from the TRS and the goal. Let us now define the grammars that are constructed with them. All the grammars considered have the same terminals (the constructors), the same non-terminals, and the same productions, as defined before. Just the axioms (tuples of non-terminals) are different. Note that the grammars could be optimized by removing non-reachable non-terminals and non-usable productions. For Example 1.1 we get the grammars:

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

Here is an example of derivation for Gr^l_ϵ:

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

This encodes the narrowing derivation

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

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

In the general case, the definition of the grammars (i.e. of theirs axioms) is a little technical because of the constructors that may appear in the goal. Below, $[u, v_i[$ denotes the set of occurrences appearing along the branch going from u to v_i (v_i is not included). For each non-variable occurrence u of the lhs l (resp. rhs r) of the goal such that $u = \epsilon$ or u is just under a function, let $\{v_1, \ldots, v_n\}$ (exhaustive list) be the occurrences appearing under u, such that for each i

- the branch $[u, v_i[$ contains only one function f and v_i is just under f,
- or v_i is a variable occurrence and the branch $[u, v_i[$ contains only constructors.

We create the grammar Gr^l_u defined by the axiom $(NT(l, u), ANT(l, v_1), \ldots, ANT(l, v_n)))$ (resp. G^r_u defined by the axiom $(NT(r, u), ANT(r, v_1), \ldots, ANT(r, v_n))))$.

6.5 Control

Synchronized grammars, as defined previously, are close to regular tree grammars (and very close to the coupled grammars of Guan *et al.* [13]), and are easy to use, but unfortunately they cannot encode our complete problem because they do not take into account variable renamings. Indeed, consider the rewrite system $\{f(c(x,y)) \xrightarrow{1} c(f(x),f(y))\}$ and the goal $f(x) = t$, where t is an arbitrary term. The Tree Grammar Gr^l_ϵ contains the productions $L^1_1 \Rightarrow c(X^1,Y^1), R^1_\epsilon \Rightarrow c(R^1_1,R^1_2), \{R^1_1 \Rightarrow R^1_\epsilon, X^1 \Rightarrow L^1_1\}, \{R^1_2 \Rightarrow R^1_\epsilon, Y^1 \Rightarrow L^1_1\}, \{G^l_\epsilon \Rightarrow R^1_\epsilon, X^l \Rightarrow L^1_1\}$ and the axiom is (G^l_ϵ, X^l). A possible derivation of Gr^l_ϵ is:

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

This encodes the narrowing derivation:

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

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

The solution to this problem consists of using an integer number, called *control*, to encode variable renamings. In a grammar computation, each non-terminal is coupled with an integer of control, which is set to a new fresh value when a synchronized production is applied on it. When a free production is applied, the control number is preserved. Moreover, a pack of productions will be applied only on non-terminals that have the same control number. For example, the previous derivation is transformed into:

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

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

6.6 Correspondence Between TTSGs and Narrowing

Let us now prove that there is a one-to-one correspondence between TTSG derivations and narrowing derivations. Let us first define some notation and definitions needed to express and prove the results of this section.

Given a TTSG Gr, a computation tp is *in simplified form* if tp cannot be derived by a free production. If it is not, the simplified form of tp, denoted by $\boldsymbol{tp\downarrow}$, is a computation in simplified form derived from tp by free productions. It is unique because, from the construction of TTSG, there is at most one free production $x \Rightarrow t$ for a given non-terminal X.

Each tuple of control ct is here a 1-tuple, i.e. an integer. $\boldsymbol{Id_{ct}(r^j)}$ denotes the term r^j whose variable x^j is renamed into x^j_{ct}.

The next definition gives the way to get terms of narrowing derivations (i.e. result of narrowing and instantiation of variables) and computations of TTSGs.

Definition 6.1 Let tp be a computation in simplified form, and suppose that the goal is $g \stackrel{?}{=} g'$. The term (that contains no non-terminals) *associated to* tp is the tuple issued from tp by replacing every non-terminal of the form

- (X^i, ct) by x^i_{ct},
- (R^i_v, ct) by $Id_{ct}(r^i|_v)$ (note that $r^i(v)$ is a function),
- $(G^l_u, 0)$ by $g(u)(A^l_{u.1}, \ldots, A^l_{u.p})$, where $A^l_{u.1}, \ldots, A^l_{u.p}$ are new variables numbered by occurrences (note that $g(u)$ is a function),
- in the same way, $(G^r_u, 0)$ by $g'(u)(A^r_{u.1}, \ldots, A^r_{u.p})$, where $A^r_{u.1}, \ldots, A^r_{u.p}$ are new variables numbered by occurrences,
- $(A^l_{v_i}, 0)$ by $A^l_{v_i}$ and $(A^r_{v_i}, 0)$ by $A^r_{v_i}$.

It is denoted by $\boldsymbol{terms(tp)}$.

In Example 1.1, $terms(s((R^4_1, 1)), s((X^4, 1))) = (s(g(x^4_1)), s(x^4_1))$.

Lemma 6.2 *(Technical) Let $(c, c_1, \ldots, c_n)$ be a computation in simplified form. If $c|_{w.u} = (R^j_u, ct)$ then $\forall x^j \in Var(r^j|_u)$, (X^j, ct) appears in $(c_1, \ldots, c_n)$.*

Proof. Along the derivation that has created $(c, c_1, \ldots, c_n)$ a computation like $((R^j_\epsilon, ct), \ldots)$ has appeared, i.e. a pack like $P = \{R^i_v \Rightarrow R^j_\epsilon,\ X^i \Rightarrow L^i_w,\ \ldots\}$ has been used while the control number ct has been created. As there is no σ_{in}, every non-terminal associated with the variables of l^j appears in $(L^j_w\downarrow, k)$. In the derivation, the $L^j_w\downarrow$'s appeared before (or at the same time as) $(c, c_1, \ldots, c_n)$, because $(c, c_1, \ldots, c_n)$ is in simplified form. If $x^j \in Var(r^j|_u)$, then $x^j \in Var(l^j)$, so the associated non-terminal (X^j, ct) appeared before (or at the same time as) $(c, c_1, \ldots, c_n)$. It has not disappeared because x^j occurs in r^j under a function ($r^j(u)$ is a function), and then only a synchronized pack containing $R^j_u \Rightarrow \ldots$ could remove it; this is impossible because (R^j_u, k) would also have been removed. □

Lemma 6.3 *(Correspondence in one step) Let $(c, c_1, \ldots, c_n)$ be a computation in simplified form, and let $(s, s_1, \ldots, s_n) = terms(c, c_1, \ldots, c_n)$. If*

$$(c, c_1, \ldots, c_n) \Rightarrow_{[P]} . \Rightarrow^*_{[free-prods.]} (d, d_1, \ldots, d_n)$$

where P is a synchronized pack that does not contain ANY and $(d, d_1, \ldots, d_n)$ is in simplified form, then

$$s \leadsto_{[\sigma]} t$$

and $(t, \sigma s_1, \ldots, \sigma s_n) = terms(d, d_1, \ldots, d_n)$.
Conversely, if

$$s \rightsquigarrow_{[\sigma]} t$$

then

$$(c, c_1, \ldots, c_n) \Rightarrow_{[P]} . \Rightarrow^*_{[free-prods.]} (d, d_1, \ldots, d_n)$$

where P is a synchronized pack that does not contain ANY, $(d, d_1, \ldots, d_n)$ is in simplified form, and $(t, \sigma s_1, \ldots, \sigma s_n) = terms(d, d_1, \ldots, d_n)$.

Proof. 1. First sense.
1.1. For a computation different from the axiom.
Since $(c, c_1, \ldots, c_n) \Rightarrow_{[P=\{R^j_u \Rightarrow R^i_\epsilon,\ X^j \Rightarrow L^i_v,\ \ldots\}]} \ldots$ the pack P is in the grammar. Then $r^j|_u$ and l^i are unifiable. Thus, if $w.u$ is the occurrence of R^j_u in c, $s|_{w.u} = terms(c|_{w.u}) = terms((R^j_u, ct)) = Id_{ct}(r^j|_u)$ which is unifiable with l^i. Then $s \rightsquigarrow_{[w.u, l^i \rightarrow r^i, \sigma]} t = s[w.u \leftarrow Id_{ct'}(r^i)]$. On the other hand, from the grammar building, for each variable x^j_k in $Id_{ct}(r^j|_u)$ at occurrence v we have $\sigma x^j_k = Id_{ct'}(l^i|_v) = terms((L^i_v \downarrow, ct'))$. As $\forall b,\ d_b = c_b[(X^j, ct) \leftarrow (L^i_v \downarrow, ct')]$ we get $terms(d_b) = terms(c_b)[p \leftarrow terms((L^i_v \downarrow, ct')) = terms(c_b)[x^j_k \leftarrow \sigma(x^j_k)] = \sigma(terms(c_b)) = \sigma s_b$. On the other hand, $d = c[w.u \leftarrow (R^i_\epsilon, ct')\downarrow]$. Then $terms(d) = terms(c)[w.u \leftarrow terms((R^i_\epsilon, ct')\downarrow)] = s[w.u \leftarrow Id_{ct'}(r^i)] = t$.
1.2. For the axiom (in simplified form).
The reasoning is made on the lhs of the goal, but it is exactly the same for the rhs. Since $(c, c_1, \ldots, c_n) \Rightarrow_{[P=\{G^l_u \Rightarrow R^i_\epsilon,\ A_{u.k} \Rightarrow L^i_k,\ \ldots\}]} \ldots$ the pack P is in the grammar, and so $g(u) = l^i(\epsilon)$. Then

$$s|_u = g(u)(A^l_1, \ldots, A^l_n) \rightsquigarrow_{[\epsilon, l^i \rightarrow r_i, \sigma]} t|_u = Id_{ct'}(r_i)$$

with $\forall j,\ \sigma(A^l_j) = Id_{ct'}(l^i|_j)$. As $\forall j,\ d_j = L^i_j\!\downarrow, terms(d_j) = Id_{ct'}(l^i|_j) = \sigma(a_j) = \sigma(terms(c_j))$. On the other hand, $terms(d|_u) = terms((R^i_\epsilon, ct')\!\downarrow) = Id_{ct'}(r^i) = t|_u$.
2. Conversely.
2.1. For a computation different from the axiom.
Assume $s \rightsquigarrow_{[w.u, l^i \rightarrow r^i, \sigma]} t$. Thus, $s|_{w.u}$ and l^i are unifiable, so $s(w.u)$ is a function. As $s = terms(c)$, the function occurrences of s are non-terminal occurrences of c, of the form R^j_u. Thus, $s|_{w.u} = terms(c|_{w.u}) = terms((R^j_u, ct)) = Id_{ct}(r^j|_u)$. Therefore, $r^j|_u$ and l^i are unifiable, then the pack $P = \{R^j_u \Rightarrow R^i_\epsilon,\ X^j \Rightarrow L^i_v,\ \ldots\}$ is in the grammar. From the previous technical lemma, every X^j appears in $(c_1, \ldots, c_n)$ with ct as control number. Then $(c, c_1, \ldots, c_n) \Rightarrow_{[P]} . \Rightarrow^*_{[free-prods.]} (d, d_1, \ldots, d_n)$ and $(d, d_1, \ldots, d_n)$ is simplified form. We show as in (1.1) that $(t, \sigma s_1, \ldots, \sigma s_n) = terms(d, d_1, \ldots, d_n)$.
2.2. For the axiom (in simplified form).
$s|_u = g(u)(A^l_1, \ldots, A^l_n) \rightsquigarrow_{[\epsilon, l^i \rightarrow r^i, \sigma]} t|_u = Id_{ct'}(r^i)$, so $g(u) = l^i_\epsilon$, then the grammar contains the pack $P = \{G^l_u \Rightarrow R^i_\epsilon,\ A_{u.k} \Rightarrow L^i_k,\ \ldots\}$. So as $(c|_u, c_1, \ldots, c_n) = (G^l_u, A_{u.1}, \ldots, A_{u.k})$, we have

$$(c, c_1, \ldots, c_n) \Rightarrow_{[P]} . \Rightarrow^*_{[free-prods.]} (d, d_1, \ldots, d_n)$$

We show as in 1.2 that $(t, \sigma s_1, \ldots, \sigma s_n) = terms(d, d_1, \ldots, d_n)$. □

Lemma 6.4 *(Correspondence in several steps) Let $TI = (G_u\!\downarrow, A_{v_1}, \ldots, A_{v_n})$ be the simplified form of the axiom, and let $(s, a_1, \ldots, a_n)$ be its associated tuple of terms, where $a_1, \ldots, a_n$ are variables. If*

$TI \Rightarrow^* (d, d_1, \dots, d_n)$ by productions that do not use ANY and $(d, d_1, \dots, d_n)$ is in simplified form, then

$$s \rightsquigarrow^*_{[\sigma]} t \text{ and } (t, \sigma a_1, \dots, \sigma a_n) = terms(d, d_1, \dots, d_n)$$

Conversely, if $s \rightsquigarrow^*_{[\sigma]} t$, then $TI \Rightarrow^* (d, d_1, \dots, d_n)$ by productions that do not contain ANY, $(d, d_1, \dots, d_n)$ is in simplified form, and $(t, \sigma a_1, \dots, \sigma a_n) = terms(d, d_1, \dots, d_n)$.

Proof. From the correspondence in one step and by induction over the length of derivations. □

Theorem 6.5 *The tree tuple language recognized by the grammar gives exactly the ground data instances of the data terms computed by narrowing, thanks to the first component of tuples, as well as the corresponding instances of variables thanks to other components. Formally, let TI be the simplified form of the axiom, and let $(s, a_1, \dots, a_n)$ be its associated tuple of terms ($a_1, \dots, a_n$ are variables). If $(d, d_1, \dots, d_n)$ is a tuple recognized by the grammar, i.e. $TI \Rightarrow^* (d, d_1, \dots, d_n)$ and $d, d_1, \dots, d_n$ do not contain any non-terminal, and so $d, d_1, \dots, d_n$ are ground data-terms, then $s \rightsquigarrow^*_{[\sigma]} t$ and there exists a substitution θ such that $d = \theta t,\ d_1 = \theta\sigma a_1, \dots, d_n = \theta\sigma a_n$.*

Conversely, if $s \rightsquigarrow^*_{[\sigma]} t$ where t is a data term, and there exists a data substitution θ such that $\theta t,\ \theta\sigma a_1, \dots, \theta\sigma a_n$ are ground, then there is a tuple $(d, d_1, \dots, d_n)$ recognized by the grammar such that $d = \theta t,\ d_1 = \theta\sigma a_1, \dots, d_n = \theta\sigma a_n$.

Proof. 1. First sense.
From the grammar building, if X^j appears as the lhs of some production that does not contain ANY, then x^j appears in r^j under a function. Therefore, the set of X^j's as the lhs of productions that do not contain ANY and the set of X^j's as the lhs of productions that contain ANY are disjoint. Moreover, if X^j is derived into ANY, it can only be derived into trees that contain non-terminals like ANY. Therefore, if the given derivation contains the step $X^j \Rightarrow ANY$, the steps issued from X^j can be done independently to the others: the order of steps in the derivation can be changed so that the productions dealing with ANY are done last.

$$TI \Rightarrow^*_{[prods-without-ANY]} (e, e_1, \dots, e_n) \Rightarrow^*_{[prods-with-ANY]} (d, d_1, \dots, d_n)$$

where $(e, e_1, \dots, e_n)$ is in simplified form. From the correspondence in several steps $s \rightsquigarrow^*_{[\sigma]} t$ and $(t, \sigma a_1, \dots, \sigma a_n) = terms(e, e_1, \dots, e_n)$. Applying productions containing ANY until having a tuple without non-terminals amounts to replacing all X^j by ground data terms. Within $(t, \sigma a_1, \dots, \sigma a_n)$, this amounts to instantiating each variable x^j by a ground data term: let θ be this substitution. So we get $\theta t = d, \theta\sigma a_1 = d_1, \dots, \theta\sigma a_n = d_n$.
2. Conversely.
From the correspondence in several steps

$$TI \Rightarrow^*_{[prods-without-ANY]} (e, e_1, \dots, e_n)$$

, where $(e, e_1, \dots, e_n)$ is in simplified form and $t, \sigma a_1, \dots, \sigma a_n) = terms(e, e_1, \dots, e_n)$. For each variable x^j of $(t, \sigma a_1, \dots, \sigma a_n)$, some productions containing ANY can be applied to derive X^j into θx^j. Thus $(e, e_1, \dots, e_n) \Rightarrow^* (d, d_1, \dots, d_n)$. Since t is a data term, $(e, e_1, \dots, e_n)$ contains only non-terminals like X^j, then $(d, d_1, \dots, d_n)$ contains only constructors, i.e. is a recognized tuple, and we have $d = \theta t,\ d_1 = \theta\sigma a_1, \dots, d_n = \theta\sigma a_n$. □

7 Step 2: Intersection of TTSGs over One Component

This section describes the second step of our method. Let us consider again Example 1.1. Recall that we have decomposed the problem into three parts $g(x) \stackrel{?}{=} y_1$, $f(y_2) \stackrel{?}{=} y_3$ and $0 \stackrel{?}{=} y_4$. In Sect. 6.4, three TTSGs have been deduced from the problem to solve each of the three parts. The point now is to reconstruct the initial problem thanks to the one-component-intersection of sets of tuples. This operation corresponds to the join operation in relational algebra (relational databases).

Definition 7.1 Let E_1 be a set of n_1-tuples and E_2 be a set of n_2-tuples. The *one component* k_1, k_2 *intersection* of E_1 and E_2 is the set of n_1+n_2-1-tuples defined by $\{tp_1 * (tp_2\backslash_{k_2}) \mid tp_1 \in E_1 \text{ and } tp_2 \in E_2 \text{ and } tp_1|_{k_1} = tp_2|_{k_2}\}$.

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

To get the solutions of the initial goal, we have to compute incrementally the one component k_1, k_2 intersection for each pair of grammars Gr_1, Gr_2 such that the k_1th component of the axiom of G_1 is G^l_u and the k_2th component of the axiom of G_2 is A^l_u with the same u (resp. G^r_u and A^r_u). For our example, this means that the possible instantiations of y_2 are restricted to the possible results for $g(x)$. At the end, we also have to compute the intersection for components G^l_ϵ and G^r_ϵ.

When considering any TTSGs, we have the following result:

Lemma 7.2 *Emptiness of intersection over one component of languages recognized by TTSGs is undecidable.*

Proof. The post correspondence problem can be encoded by the intersection over one component of two languages recognized by TTSGs. Let b be a binary constructor and consider the 1-tuple tree languages $L = \{b(\alpha, \phi(\alpha)) | \, \alpha \in A^+\}$ and $L' = \{b(\alpha, \phi'(\alpha)) | \, \alpha \in A^+\}$. Let us write $\phi(a_i) = c_{i,1} \dots c_{i,p_i}$ for each $a_i \in A$. Then L is recognized by the TTSG Gr defined by the productions $I \Rightarrow b(X, Y)$ and for each $a_i \in A : \{X \Rightarrow X_i, Y \Rightarrow Y_i\}$, $X_i \Rightarrow a_i Z$, $Y_i \Rightarrow c_{i,1} Y_{i,2}, Y_{i,2} \Rightarrow c_{i,2} Y_{i,3}, \dots, Y_{i,p_i} \Rightarrow c_{i,p_i} T$, $\{Z \Rightarrow X, T \Rightarrow Y\}$, $\{Z \Rightarrow \bot, T \Rightarrow \bot\}$, and the axiom I. The TTSG Gr' that recognized L' is defined in the same way. Now checking emptiness of $L \cap L'$ amounts to checking the existence of $\alpha \in A^+$ such that $\phi(\alpha) = \phi'(\alpha)$, i.e. solving the post correspondence problem. □

Moreover, the intersection of languages recognized by TTSGs is not always a language recognized by a TTSG. Fortunately, we do not consider any TTSGs, only TTSGs that come from a unification problem, and in this case the problem is decidable thanks to some particular properties. Sect. 7.1 gives preliminaries that define these properties, while Sect. 7.2 presents the intuitive idea and the detail of the intersection algorithm.

7.1 Preliminaries

Emptiness of intersection becomes decidable if the component k_1 (or k_2) has the property of external synchronization, which is the case for TTSGs coming from a unification problem. Roughly speaking, this means that at most one production can be applied on this component when using a pack of synchronized productions. So, an externally synchronized component of a TTSG behaves as a regular tree language in the sense that any branch of this component can be generated independently from the others.

Recall that now the control is not one integer, but a tuple of integers.

Definition 7.3 Let Gr be a TTSG and $tp \in Comp(Gr)$:

- $\boldsymbol{Class(tp, u, level)}$ is the set of occurrences of tp that may be directly synchronized with u on a given level and is defined if $tp|_u = (X, ct_u)$ by $\{v|\, tp|_v = (Y, ct_v)$ and $ct_v|_{level} = ct_u|_{level}$ and $ct_v|_{level} \neq \perp\}$.
- $\boldsymbol{TransClass(tp, u)}$ is the set of occurrences of tp that may be indirectly synchronized with u. It is the union of the following inductively defined sets:
  - $E_0 = \bigcup_{j=1}^{Sz} Class(tp, u, j)$
  - $E_{i+1} = \bigcup_{j=1}^{Sz} \{v|\, \exists w \in E_i$ such that $v \in Class(tp, w, j)\}$.

For example let $Sz = 2$ and consider the pair $tp = (c((X, (0, 0)),\, (Y, (1, 0))), c((Z, (1, 1)),\, (T, (2, 2)))$. The occurrence of X is 1.1.

- $Class(tp, 1.1, 2) = \{1.1, 1.2\}$, i.e. the occurrences of X and Y.
- $TransClass(tp, 1.1) = \{1.1, 1.2, 2.1\}$.

$Class$ and $TransClass$ contain non-terminal occurrences, i.e. occurrences of leaves. $TransClass$ is roughly the transitive closure of $Class$ – then two different TransClasses are disjoint.

The following lemma means that two different TransClasses can be derived independently.

Lemma 7.4 *Two different TransClasses of the same computation tp cannot be merged when tp is derived further. Formally, let $TC = \{u_1, \ldots, u_n\}$ and $TC' = \{u'_1, \ldots, u'_{n'}\}$ two different TransClasses of $tp \in Comp(Gr)$, and let tp' be a computation derived form tp by the grammar (i.e. $tp \Rightarrow^* tp'$). Then any two non-terminal occurrences v, v' of tp', of the form $v = u_i.w$, $v' = u'_j.w'$, do not belong to the same TransClass of tp'.*

Proof. Obvious as packs of synchronized productions may divide Classes but never join them. □

Definition 7.5 Let $G = (Sz, \mathcal{C}, NT, PP, TI)$ be a TTSG. The ith component of G is said to be *externally synchronized* if $\forall tp \in Comp(G)$ and for all packs $\{X_1 \Rightarrow Y_1, \ldots, X_n \Rightarrow Y_n\}_k$ of PP that can be applied on tp (at occurrences $\{u_1, \ldots, u_n\}$) we have

1. there is at most one $v \in \{u_1, \ldots, u_n\}$ such that $v = i.w$,
2. and if such v exists then the tuple tp' obtained by applying $\{X_1 \Rightarrow Y_1, \ldots, X_n \Rightarrow Y_n\}_k$ on tp satisfies $TransClass(tp', v) \cap \{v' \in O(tp)|v' = i.w'\} = v$.

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

Lemma 7.6 *The first component of every TTSG produced from the unification problem has the external synchronization property.*

Proof. By construction, the first component contains only non-terminals like $R^?_?$ or like X^j, where x^j is a variable occurring only under constructors on some rhs. So X^j does not appear as the lhs of any productions. Moreover, in every pack of productions there is only one production whose lhs is of the form $R^?_?$.

On the other hand, the control tuples of such TTSGs are 1-tuples. So the Classes and TransClasses are the same, therefore (2) is satisfied. □

7.2 Algorithm

Before showing how the intersection is computed, let us explain how it works and why tuples of control are needed, instead of single integers.

Example 7.7 *This example does not come from a unification problem, but it is easier to understand, and every component is nevertheless externally synchronized. Let*

$$\begin{aligned} G_1 = \quad & (1,\ \{s, 0\},\ \{X, X', Y, Y', Y''\}, \\ & \{X' \Rightarrow 0, Y' \Rightarrow 0, X' \Rightarrow s(X), Y' \Rightarrow s(Y''), Y'' \Rightarrow s(Y), \{X \Rightarrow X', Y \Rightarrow Y'\}_1\}, \\ & ((X, 0), (Y, 0))) \end{aligned}$$

and

$$\begin{aligned} G_2 = \quad & ((1,\ \{s, 0\},\ \{Z, Z', T, T', T''\}, \\ & \{Z' \Rightarrow 0, T' \Rightarrow 0, Z' = s(Z), T' \Rightarrow s(T''), T'' \Rightarrow s(T), \{Z \Rightarrow Z', T \Rightarrow T'\}_1\}, \\ & ((Z, 0), (T, 0))) \end{aligned}$$

G_1 and G_2 generate the same language i.e. the pairs $(s^n(0), s^{2n}(0))$. The 2,1 intersection of G_1 and G_2 is then the language of triples

$$\mathcal{L}_3 = \{(s^n(0), s^{2n}(0), s^{4n}(0))\}$$

The question is how to build from G_1 and G_2 a new TTSG G_3 that generates $\mathcal{L}_3$? The idea is that the first component of $\mathcal{L}_3$ will be generated by the productions of G_1, the last component of $\mathcal{L}_3$ will be generated by the productions of G_2, therefore G_3 contains the non-terminals and the productions of both G_1 and G_2. The second component of $\mathcal{L}_3$ is the intersection of the second component of G_1 with the first component of G_2. The productions that generate it are built using the same idea as for the intersection of regular languages, i.e. by computing the Cartesian product of the grammars.

More precisely, let us note first that only the non-terminals Y, Y', Y'' (resp. Z, Z') may appear in the second (resp. the first) component of G_1 (resp. G_2). Thus, for the intersection, the set of non-terminals is the Cartesian product $\{YZ, Y'Z, Y''Z, YZ', Y'Z', Y''Z'\}$, and the free productions are $\{Y'Z' \Rightarrow s(Y''Z), Y'Z' \Rightarrow 0, Y''Z' \Rightarrow s(YZ)\}$. The packs of productions are constructed such that if a synchronization was possible in the initial grammars, it is still possible in the intersection: for each pack of productions of G_1 (resp G_2) that deals with Y or Y' or Y'' (resp. Z or Z'), we create a new pack in G_3. We get $\{\{X \Rightarrow X', YZ \Rightarrow Y'Z\}_1, \{X \Rightarrow X', YZ' \Rightarrow Y'Z'\}_1\}$ from the packs of G_1 and $\{\{YZ \Rightarrow YZ', T \Rightarrow T'\}_1, \{Y'Z \Rightarrow Y'Z', T \Rightarrow T'\}_1, \{Y''Z \Rightarrow Y''Z', T \Rightarrow T'\}_1\}$ from the packs of G_2. The axiom of G_3 is $((X, 0), (YZ, 0), (T, 0))$.

Unfortunately, G_3 is not the desired grammar. Indeed, using the pack $\{X \Rightarrow X', YZ \Rightarrow Y'Z\}_1$, the axiom is derived into $((X', 1), (Y'Z, 1), (T, 0))$. Now the pack $\{Y'Z \Rightarrow Y'Z', T \Rightarrow T'\}_1$ cannot be applied because the control numbers of $Y'Z$ and T are not equal, and no other production can derive $Y'Z$. The axiom can also be derived using the pack $\{YZ \Rightarrow YZ', T \Rightarrow T'\}_1$), but we get the same conclusion. Thus the language recognized by G_3 is empty. This problem is solved by considering pairs of integers as control in G_3, the first (resp. second) level being incremented when applying a pack that comes from G_1 (resp. G_2). So the packs of productions coming from G_2 must have 2 as rank, and are then

$$\{\{YZ \Rightarrow YZ', T \Rightarrow T'\}_2, \{Y'Z \Rightarrow Y'Z', T \Rightarrow T'\}_2, \{Y''Z \Rightarrow Y''Z', T \Rightarrow T'\}_2\}$$

The axiom is now $((X,(0,\perp),(YZ,(0,0)),(T,(\perp,0))$. $\perp$ *means that this level of control is not used by the non-terminal. A possible derivation for* G_3 *is*

$$((X,(0,\perp),(YZ,(0,0)),(T,(\perp,0)) \Rightarrow ((X',(1,\perp),(Y'Z,(1,0)),(T,(\perp,0))$$

Now $\{Y'Z \Rightarrow Y'Z', T \Rightarrow T'\}_2$ *is applicable, and we get*

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

The following definition is needed for Theorem 7.9. It defines the set of non-terminals that may be derived from a given non-terminal.

Definition 7.8 Let PP be a set of packs of productions. $\boldsymbol{Reach(X, PP)}$ is inductively defined by

- X is in $Reach(X, PP)$,
- if Y is in $Reach(X, PP)$ and $Y \Rightarrow Z$ is a production that belongs to a pack of PP, then Z is in $Reach(X, PP)$,
- if Y is in $Reach(X, PP)$ and there is a free production in PP such that $Y \Rightarrow c(Y_1, \ldots, Y_n)$, then $\{Y_1, \ldots, Y_n\} \subseteq Reach(X, PP)$.

Theorem 7.9 *(Algorithm for intersection) Let* $G_1 = (Sz_1, \mathcal{C}_1, NT_1, PP_1, TI_1)$ *and* $G_2 = (Sz_2, \mathcal{C}_2, NT_2, PP_2, TI_2)$ *be two TTSGs such that the* i_1*th component of* G_1 *is externally-synchronized. The language recognized by the TTSG* $G_3 = (Sz_3, \mathcal{C}_3, NT_3, PP_3, TI_3)$ *as defined below is the one component* i_1, i_2 *intersection of* G_1 *and* G_2. (I_{i_1}, ct_{i_1}) *(resp.* (I_{i_2}, ct_{i_2})*) denotes the* i_1*th (resp.* i_2*th) component of the axiom* TI_1 *(resp.* TI_2*) of* G_1 *(resp.* G_2*).*

- $Sz_3 = Sz_1 + Sz_2$
- $\mathcal{C}_3 = \mathcal{C}_1 \cup \mathcal{C}_2$
- $NT_3 = NT_1 \cup NT_2 \cup Reach(I_{i_1}, PP_1) \times Reach(I_{i_2}, PP_2)$
- $PP_3 = PP_1 \cup PP_2 \cup Cons \cup Sync_1 \cup Sync_2$ *where*
  - $Cons = \{X_1X_2 \Rightarrow c(Y_{1,1}Y_{2,1}, \ldots, Y_{1,n}Y_{2,n})$ *such that* $X_1 \in Reach(I_{i_1}, PP_1)$ *and* $X_1 \Rightarrow c(Y_{1,1}, \ldots, Y_{1,n}) \in PP_1$ *and* $X_2 \in Reach(I_{i_2}, PP_2)$ *and* $X_2 \Rightarrow c(Y_{2,1}, \ldots, Y_{2,n}) \in PP_2\}$
  - $Sync_1 = \{\{X_1X_2 \Rightarrow Y_1X_2, Z_1 \Rightarrow Z'_1, \ldots, Z_n \Rightarrow Z'_n\}_k$ *such that* $X_1 \in Reach(I_{i_1}, PP_1)$ *and* $\{X_1 \Rightarrow Y_1, Z_1 \Rightarrow Z'_1, \ldots, Z_n \Rightarrow Z'_n\}_k \in PP_1$ *and* $X_2 \in Reach(I_{i_2}, PP_2)\}$

- $Sync_2 = \{\{X_{1,1}X_{2,1} \Rightarrow X_{1,1}X'_{2,1}, \ldots, X_{1,n}X_{2,n} \Rightarrow X_{1,n}X'_{2,n}, X_{2,n+1} \Rightarrow X'_{2,n+1}, \ldots, X_{2,m} \Rightarrow X'_{2,m}\}_{Sz_1+k}$ *such that*
  $\{X_{1,1}, \ldots, X_{1,n}\}$ *is a multiset whose elements are in* $Reach(I_{i_1}, PP_1)$ *and*
  $\{X_{2,1}, \ldots, X_{2,n}\} \subseteq Reach(I_{i_2})$ *and* $\{X_{2,1} \Rightarrow X'_{2,1}, \ldots, X_{2,m} \Rightarrow X'_{2,m}\}_k \in PP_2$

- $TI_3 = (TI'_1[i_1 \leftarrow (I_{i_1}I_{i_2}, ct_{i_1} * ct_{i_2})]) * (TI'_2 \backslash_{i_2})$ *where* TI'_1 *(resp.* TI'_2*) comes from* TI_1 *(resp.* TI_2*) by replacing each control tuple* ct_j *by* $ct_j * (\perp_1, \ldots, \perp_{Sz_2})$ *(resp.* $(\perp_1, \ldots, \perp_{Sz_1} * ct_j)$*).*

The definition of $Sync_2$ is more complicated than that of $Sync_1$ because the i_2th component of G_2 is not assumed to be externally synchronized. So several productions (in fact n) of the pack $\{X_{2,1} \Rightarrow X'_{2,1}, \ldots, X_{2,m} \Rightarrow X'_{2,m}\}_k$ of PP_2 may be applied together on the i_2th component of G_2.

The following definition is needed in the proof of Theorem 7.9.

Definition 7.10 Let tp and tp' be two computations of (eventually different) TTSGs. Let $t = tp|_i$ and $t' = tp'|_{i'}$. We write $t \approx_l t'$ (resp. $t \approx_r t'$) if

- $\forall u \in O(t)$, $t(u)$ is a constructor symbol, which implies $t(u) = t'(u)$,
- and $\forall u \in O(t)$, $t(u) = (X, ct)$ implies $t'(u) = (X', ct')$ and $ct' = ct * (\perp, \ldots, \perp)$ (resp. $ct' = (\perp, \ldots, \perp) * ct$).

Proof. 1. First let us prove that for $tp_1 \in Comp(G_1)$ and $tp_2 \in Comp(G_2)$ satisfying
(1) $\forall u \in O(tp_1|_{i_1})$, $tp_1|_{i_1.u}$ is a constructor, which implies $tp_1|_{i_1}(u) = tp_2|_{i_2}(u)$
(2) and $\forall u \in O(tp_1|_{i_1})$, $tp_1|_{i_1.u} = (X_1, ct_1)$, which implies $tp_2|_{i_2.u} = (X_2, ct_2)$,
then there exists $tp_3 \in Comp(G_3)$ such that

- (A) $\forall j \in [1, i_1 - 1] \cup [i_1 + 1, n_1]$ $tp_3|_j \approx_l tp_1|_j$ and
- (B) $\forall j \in [n_1 + 1, n_1 + i_2 - 1]$ $tp_3|_j \approx_r tp_2|_{j-n_1}$ and
- (C) $\forall j \in [n_1 + i_2, n_1 + n_2]$ $tp_3|_j \approx_r tp_2|_{j+1-n_1}$ and
- (D) $\forall u \in O(tp_1|_{i_1})$ s.t. $tp_1|_{i_1.u}$ is a constructor, $tp_1|_{i_1}(u) = tp_3|_{i_1}(u)$ and $\forall u \in O(tp_1|_{i_1})$ such that $tp_1|_{i_1.u} = (X_1, ct_1)$ and $tp_2|_{i_2.u} = (X_2, ct_2)$ then $tp_3|_{i_1.u} = (X_1X_2, ct_1 * ct_2)$

It is proved by induction on the total number of productions used to generate tp_1 and tp_2. The basic case is obvious because the axiom of G_3 is constructed to respect these properties.

For the induction step, let $tp_1 \in Comp(G_1)$ and $tp_2 \in Comp(G_2)$ such that $tp_1|_{i_1}$ and $tp_2|_{i_2}$ satisfy (1) and (2). By the induction hypothesis, we suppose that there exists $tp_3 \in Comp(G_3)$ such that tp_3 verifies (A) (B) (C) and (D):

- If the next production applied on tp_1 does not affect any occurrence of $tp_1|_{i_1}$, we obtain tp'_1 and tp'_1 and tp_2 verifies (1) and (2). $tp_3 \Rightarrow tp'_3$ with the same production at the same occurrence because $PP_1 \subseteq PP_3$. Moreover, tp'_3 verifies (A) (B) (C) and (D).
- If the next production applied on tp_2 does not affect any occurrence of $tp_2|_{i_2}$, we obtain the same result.

- If the production applied on tp_1 is a pack of productions

$$\{X_1 \Rightarrow X'_1, \ldots, X_n \Rightarrow X'_n\}_k$$

that affects the occurrences $u_1, \ldots u_n$ of tp_1, where one of them, say v, is $i_1.v'$, $\forall i \in [1, n]$, we have $tp_1|_{u_i} = (X_i, ct_i)$, let tp'_1 the computation obtained by the application of the pack. tp'_1 and tp_2 verifies (1) and (2), because the synchronized productions do not produce any new constructor. Moreover, $\forall i \in [1, n]$ we have $tp_3|_{u_i} = (X_i, ct_i * (\perp, \ldots, \perp))$, except v, for which we have $tp_3|_v = (X_j Y, ct_j * ct')$. By construction of G_3, the pack $\{X_1 \Rightarrow X'_1, \ldots, X_j Y \Rightarrow X'_j Y, \ldots, X_n \Rightarrow X'_n\}_k$ is in PP_3, therefore it can be applied on tp_3, and we obtain tp'_3 that verifies (A) (B) (C) and (D).

- If the production applied on tp_2 is a pack of productions with similar reasoning as above, we get the same result.

- If the production applied on tp_2 is a free production that affects one occurrence of $tp_2|_{i_2}$: Let $X_2 \Rightarrow c(Y_{2,1}, \ldots, Y_{2,n})$ be the production, v the occurrence in $tp_2|_{i_2}$, and tp'_2 the computation obtained. If there exists $tp''_1 \in Comp(G_1)$ such that $tp_1 \Rightarrow^* tp''_1$ and $tp''_1|_{i_1}(v) = tp'_2|_{i_2}(v) = c$, then there exists tp'_1 in $Comp(G_1)$ such that $tp_1 \Rightarrow^* tp'_1$ and $tp'_1|_{i_1} = tp_1|_{i_1}[v \leftarrow c((Y_{1,1}, ct_1), \ldots, (Y_{1,n}, ct_1))]$ and a free production in PP_1 $X_1 \Rightarrow c(Y_{1,1}, \ldots, Y_{1,n})$. Indeed, the i_1th component of G_1 is externally synchronized, therefore each branch of this component can be generated independently from the others. Moreover, every free production generates one and only one constructor, so tp'_1 and $X_1 \Rightarrow c(Y_{1,1}, \ldots, Y_{1,n})$ do exist. Now, tp'_1 and tp'_2 verifies (1) and (2). Notice that all the productions but $X_1 \Rightarrow c(Y_{1,1}, \ldots, Y_{1,n})$ in $tp_1 \Rightarrow^* tp'_1$ do not affect any occurrence of $tp_1|_{i_1}$ or are synchronized productions, therefore from the second and the fourth point of this proof we can deduce that $tp_3 \Rightarrow^* tp''_3$ and $tp''_3|_{i_1}(v) = (X_1 X_2, ct_1 * ct_2)$. From the construction of G_3, we know that $X1X2 \Rightarrow c(Y_{1,1}Y_{2,1}, \ldots, Y_{1,n}Y_{2,n})$ is in PP_3, therefore this production can be applied on tp''_3 at occurrence v, and we obtain tp'_3 that verifies (A) (B) (C) and (D).

To conclude this part of the proof we can see that if $tp_1 \in Rec(G_1)$ and $tp_2 \in Rec(G_2)$, then there exists $tp_3 \in Rec(G_3)$ that verifies (A) (B) (C) and (D). This means that tp_3 is the intersection over one component of tp_1 and tp_2.
2. Conversely, we have to prove that if $tp_3 \in Rec(G_3)$, then there exists $tp_1 \in Rec(G_1)$ and $tp_2 \in Rec(G_2)$, such that $tp_3 = tp_1 * (tp_2 \backslash_{i_2})$. To prove that, we prove that if $tp_3 \in Comp(G_3)$ then there exist $tp_1 \in Comp(G_1)$ and $tp_2 \in Comp(G_2)$ such that tp_3 verifies (A), (B), (C) and (D) wrt tp_1 and tp_2.

It is proved by induction on the number of productions applied to obtain tp_3. The case $n = 0$ is obvious from the construction of G_3. For the induction step we will suppose that the lemma is true for a given tp_3, and we prove that if a new production is applied on tp_3 obtaining tp'_3, we can find $tp'_1 \in Comp(G_1)$ and $tp'_2 \in Comp(G_2)$ such that tp'_3 verifies (A), (B), (C) and (D) wrt tp'_1 and tp'_2.

- If the next production applied on tp_3 does not affect any occurrence of $tp_3|_{i_1}$, then it is a production either of PP_1 or of PP_2, therefore the same production can be applied at the same occurrence (modulo a shift for tp_2) on either tp_1 or tp_2. Therefore, we prove the induction step.

- If the next production applied on tp_3 is a free one, say

$$X1X2 \Rightarrow c(Y_{1,1}Y_{2,1}, \ldots, Y_{1,n}Y_{2,n})$$

applied at an occurrence v of $tp_3|_{i_1}$, from the construction of G_3 we have $X1 \Rightarrow c(Y_{1,1}, \ldots, Y_{1,n}) \in PP_1$ and $X2 \Rightarrow c(Y_{2,1}, \ldots, Y_{2,n}) \in PP_2$, therefore we can apply the first one at the occurrence v of $tp_1|_{i_1}$ and the other at the occurrence v of $tp_2|_{i_2}$, and the resulting computations allow us to verify the induction step.

- If the next production applied on tp_3 is a pack of productions that affects at least one occurrence of $tp_3|_{i_1}$, then from the construction of PP_3, the corresponding pack of productions exists in PP_1 or in PP_2 (depending on whether the pack belongs to $Sync_1$ or $Sync_2$). Therefore, the corresponding pack of productions can be applied on either tp_1 or tp_2, and we get the tuples that prove the induction step.

□

The point now is to prove that if one component (different from i_2) of G_2 is externally synchronized, then the corresponding component of G_3 keeps this property. This is necessary to be able to compute incrementally several intersections using the previous algorithm.

Lemma 7.11 *Let G_1 be a TTSG whose i_1th component is externally synchronized and G_2 another TTSG whose j_2th component is also externally synchronized. Let G_3 be a one component i_1, i_2 intersection ($i_2 \neq j_2$) of G_1 and G_2. The component that corresponds to j_2 (i.e. $n_1 + j_2$ if $j_2 < i_2$ and $n_1 + j_2 - 1$, otherwise) in G_3 is still externally synchronized.*

Proof. Point (1) of Definition 7.5 is obvious. For point (2), when building G_3, the only possible merging between two TransClasses of G_2 might come from the intersection component, due to an internal synchronization on the i_1th component of G_1. However, it is assumed to be externally synchronized. □

To end Example 1.1, let us see the grammars obtained by intersection. We have had three grammars:

- Gr_1^l, whose axiom is (G_1^l, X^l).
- Gr_ϵ^l, whose axiom is (G_ϵ^l, A_1^l).
- Gr_ϵ^r, whose axiom is (G_ϵ^r).

For the sake of simplicity, only new productions are given. '_' represents any non-terminal allowed by Theorem 7.9.

We have $Reach(G_1^l) = \{G_1^l, R_\epsilon^4, R_1^4, R_\epsilon^5\}$, $Reach(A_1^l) = \{A_1^l, L_1^1, L_{1.1}^1, X^1, L_1^2, X^2, L_1^3\}$, and $Reach(G_\epsilon^r) = \{G_\epsilon^r\}$. So the intersection of Gr_1^l with Gr_ϵ^l gives the grammar Gr^{left} with the following new productions:

$$
\begin{array}{ll}
\{ & R_\epsilon^4 L_1^1 \Rightarrow s(R_1^4 L_{1.1}^1), R_\epsilon^4 L_{1.1}^1 \Rightarrow s(R_1^4 X^1), R_\epsilon^5 L_1^3 \Rightarrow 0, \\
& \{R_1^4\_ \Rightarrow R_\epsilon^4\_, X^4 \Rightarrow L_1^4\}_1, \{R_1^4\_ \Rightarrow R_\epsilon^5\_, X^4 \Rightarrow L_1^5\}_1, \\
& \{G_1^l\_ \Rightarrow R_\epsilon^4\_, X^l \Rightarrow L_1^4\}_1, \{G_1^l\_ \Rightarrow R_\epsilon^5\_, X^l \Rightarrow L_1^5\}_1, \\
& \{R_\epsilon^1 \Rightarrow R_\epsilon^1, \_X^1 \Rightarrow \_L_1^1\}_2, \{R_\epsilon^1 \Rightarrow R_\epsilon^2, \_X^1 \Rightarrow \_L_1^2\}_2, \\
& \{R_\epsilon^1 \Rightarrow R_\epsilon^3, \_X^1 \Rightarrow \_L_1^3\}_2, \{R_\epsilon^2 \Rightarrow R_\epsilon^1, \_X^2 \Rightarrow \_L_1^1\}_2, \\
& \{R_\epsilon^2 \Rightarrow R_\epsilon^2, \_X^2 \Rightarrow \_L_1^2\}_2, \{R_\epsilon^2 \Rightarrow R_\epsilon^3, \_X^2 \Rightarrow \_L_1^3\}_2, \\
& \{G_\epsilon^l \Rightarrow R_\epsilon^1, \_A_1^l \Rightarrow \_L_1^1\}_2, \{G_\epsilon^l \Rightarrow R_\epsilon^2, \_A_1^l \Rightarrow \_L_1^2\}_2, \\
& \{G_\epsilon^l \Rightarrow R_\epsilon^3, \_A_1^l \Rightarrow \_L_1^3\}_2\}
\end{array}
$$

and the axiom is $((G_1^l A_1^l, (0, 0)), (X^l, (0, \perp)), (G_\epsilon^l, (\perp, 0)))$.

We have $Reach(G^l_\epsilon) = \{G^l_\epsilon, R^1_\epsilon, R^2_\epsilon, R^3_\epsilon\}$. So the intersection of Gr^{left} with Gr^r_ϵ gives the final grammar with the following new productions:

$$
\begin{array}{ll}
\{ & R^3_\epsilon G^r_\epsilon \Rightarrow 0, \\
& \{R^1_{\epsilon}\_ \Rightarrow R^1_\epsilon\_, \_X^1 \Rightarrow \_L^1_1\}_2, \{R^1_{\epsilon}\_ \Rightarrow R^2_\epsilon\_, \_X^1 \Rightarrow \_L^2_1\}_2, \\
& \{R^1_{\epsilon}\_ \Rightarrow R^3_\epsilon\_, \_X^1 \Rightarrow \_L^3_1\}_2, \{R^2_{\epsilon}\_ \Rightarrow R^1_\epsilon\_, \_X^2 \Rightarrow \_L^1_1\}_2, \\
& \{R^2_{\epsilon}\_ \Rightarrow R^2_\epsilon\_, \_X^2 \Rightarrow \_L^2_1\}_2, \{R^2_{\epsilon}\_ \Rightarrow R^3_\epsilon\_, \_X^2 \Rightarrow \_L^3_1\}_2, \\
& \{G^l_{\epsilon}\_ \Rightarrow R^1_\epsilon\_, \_A^l_1 \Rightarrow \_L^1_1\}_2, \{G^l_{\epsilon}\_ \Rightarrow R^2_\epsilon\_, \_A^l_1 \Rightarrow \_L^2_1\}_2, \\
& \{G^l_{\epsilon}\_ \Rightarrow R^3_\epsilon\_, \_A^l_1 \Rightarrow \_L^3_1\}_2\}\}
\end{array}
$$

and the axiom is $((G^l_1 A^l_1, (0, 0, \perp)), (X^l, (0, \perp, \perp)), (G^l_\epsilon G^r_\epsilon, (\perp, 0, 0)))$.

A possible derivation of this TTSG is

$$
\begin{array}{l}
((G^l_1 A^l_1, (0, 0, \perp)), (X^l, (0, \perp, \perp)), (G^l_\epsilon G^r_\epsilon, (\perp, 0, 0))) \\
\quad \Rightarrow ((R^4_\epsilon A^l_1, (1, 0, \perp)), (L^4_1, (1, \perp, \perp)), (G^l_\epsilon G^r_\epsilon, (\perp, 0, 0))) \\
\quad \Rightarrow ((R^4_\epsilon L^1_1, (1, 1, \perp)), (L^4_1, (1, \perp, \perp)), (R^1_\epsilon G^r_\epsilon, (\perp, 1, 0))) \\
\quad \Rightarrow ((s(R^4_1 L^1_{1.1}, (1, 1, \perp))), (L^4_1, (1, \perp, \perp)), (R^1_\epsilon G^r_\epsilon, (\perp, 1, 0))) \\
\quad \Rightarrow ((s(R^4_1 L^1_{1.1}, (1, 1, \perp))), (s(X^4, (1, \perp, \perp))), (R^1_\epsilon G^r_\epsilon, (\perp, 1, 0))) \\
\quad \Rightarrow ((s(R^4_\epsilon L^1_{1.1}, (2, 1, \perp))), (s(L^4_1, (2, \perp, \perp))), (R^1_\epsilon G^r_\epsilon, (\perp, 1, 0))) \\
\quad \Rightarrow ((s(s(R^4_1 X^1, (2, 1, \perp)))), (s(s(X^4, (2, \perp, \perp)))), (R^1_\epsilon G^r_\epsilon, (\perp, 1, 0))) \\
\quad \Rightarrow ((s(s(R^5_\epsilon X^1, (3, 1, \perp)))), (s(s(L^5_1, (2, \perp, \perp)))), (R^1_\epsilon G^r_\epsilon, (\perp, 1, 0))) \\
\quad \Rightarrow ((s(s(R^5_\epsilon L^1_3, (3, 2, \perp)))), (s(s(L^5_1, (2, \perp, \perp)))), (R^3_\epsilon G^r_\epsilon, (\perp, 2, 0))) \\
\quad \Rightarrow (s(s(0)), s(s(0)), 0)
\end{array}
$$

This derivation illustrates how the synchronizations enforce two applications of r_4 for one application of r_1.

8 Decidability

This section shows how unifiability can be decided thanks to TTSGs. First, we give a bound on the size of the TransClasses that is used to limit the length of the derivations. At the end of this section, two examples of problems we can solve with our technique are given.

In the following lemmas, a Class (resp. TransClass) of a given TTSG G denotes any Class (resp. TransClass) in any computation of G.

Lemma 8.1 *Let G be a TTSG built from a unification problem by Step 1 (no intersection has been achieved). The non-terminals appearing at two different occurrences of the same Class of G are different.*

Proof. This is true for the axiom, since the goal is linear. Actually, linearity of each side of the goal, considered independently of each other, suffices.

When applying a pack of synchronized productions $\{\ldots \Rightarrow R^j_\epsilon, \ldots \Rightarrow L^j_{v_1}, \ldots \Rightarrow L^j_{v_n}\}$, the newly generated non-terminals belong to the same Class, but they are different to each other. After that, they are reduced into a simplified form (this notion was defined at the beginning of Sect. 6.6) by free productions. However, the non-terminals appearing in the simplified form of $R^j_e psilon$ come from the rhs of rewrite rule j, and are then of the form $R^j_?$, X'^j, Y'^j,... . The same X'^j cannot appear twice because of the linearity of the rhs's. On the other hand, $L^j_{v_1}, \ldots, L^j_{v_n}$ are replaced by independent subterms of the rhs l^j that contain non-terminals like $L^j_?$, X^j, Y^j,... . So the same X^j cannot appears twice because of the linearity of the lhs's. $\square$

Lemma 8.2 *The size of the TransClasses of a TTSG G that comes from a unification problem is not greater than $card(NT)^{n+1}$, where NT is the set of non-terminals associated with the unification problem, and n is the number of intersections achieved to build G.*

Proof. This is proved by induction on n.

If no intersection has been done, i.e. $n = 0$, we can see that the TransClasses are in fact the Classes. From the previous lemma, the size of any Class cannot be greater than the number of non-terminals of G, i.e. $card(NT)$.

For the induction step we consider the TTSG G_1 (resp. G_2), made thanks to n_1 intersections (resp. n_2), whose TransClass Size is not greater than $card(NT)^{n_1+1}$ (resp. $card(NT)^{n_2+1}$), and we also assume that the i_1th component of G_1 is externally synchronized. Let G_3 be the i_1, i_2 intersection over one component of G_1 and G_2.

When building G_3, the only possible merging between two TransClasses of G_2 may come from the intersection component, due to an internal synchronization on the i_1th component of G_1. This is impossible, since it is assumed to be externally synchronized. Therefore, if c_3 is a TransClass of G_3, then it comes from the merger of some TransClass c_2 of G_2 with some TransClasses $c_1^1, \ldots, c_1^k$ of G_1, where k is the number of occurrences of c_2 that belong to the i_2th component of G_2. From the induction hypothesis, we get

$$\begin{aligned} Size(c_3) &= Size(c_1^1) + \ldots + Size(c_1^k) + Size(c_2) - k \\ &\leq k.card(NT)^{n_1+1} + card(NT)^{n_2+1} - k \\ &= k.(card(NT)^{n_1+1} - 1) + card(NT)^{n_2+1} \end{aligned}$$

This expression is maximal when k is, i.e. $k = card(NT)^{n_2+1}$. This case appears when the size of c_2 is maximal and all the occurrences of c_2 belong to the i_2th component. Therefore,

$$Size(c_3) \leq card(NT)^{n_2+1} \times card(NT)^{n_1+1} = card(NT)^{n_1+n_2+2}$$

Since the number of intersections for building G_3 is $n = n_1 + n_2 + 1$, we get $Size(c_3) \leq card(NT)^{n+1}$.
□

We introduce now the notion of an NT-TransClass. Intuitively, an NT-TransClass is composed by the non-terminals appearing at the occurrences of a given TransClass. But this is not sufficient, because the subtrees derived by applying productions from an NT-TransClass also depend upon the Classes appearing within the TransClass, and we want the notion of an NT-TransClass to define in a unique way the language derived from it. To make formalisation easier, we give a more general definition, in the sense that a NT-TransClass is associated with each TransClass, but the reverse is false. This means that some NT-TransClasses (as defined below) cannot really appear in a computation.

Definition 8.3 An *NT-TransClass* is composed by

- a multiset of non-terminals, and
- for each $i \in [1, Sz]$, a partition of the multiset.

The size of an NT-TransClass is defined as being the size of the multiset.

For each level i, the partition simulates the way in which the Classes are organized. Note that the values of control tuples do not matter; only the sets they define are important.

Lemma 8.4 *Under the same assumptions as the previous lemma, there are finitely many NT-TransClasses whose size is not greater than* $card(NT)^{n+1}$.

Proof. For any p, there are finitely many NT-TransClasses of size p, because

- there are finitely many multisets of size p whose elements come from the finite set NT, and
- for each multiset, for each $i \in [1, Sz]$ there are finitely many partitions.

□

Lemma 8.5 *The emptiness of languages recognized by TTSGs built from unification problems is decidable.*

Proof. The reasoning is similar to that used for a regular tree grammar: if there is a derivation of the grammar giving a term whose leaves are all terminals (i.e. the recognized language is not empty), then there is a derivation that does not contain several times of the same non-terminal in the same branch, and that gives a term whose leaves are all terminals. So to test emptiness, only derivations that do not apply more than n (n being the number of non-terminals of the grammar) at comparable occurrences have to be generated.

For a TTSG, we use the same reasoning on NT-TransClasses instead of single non-terminals. From Lemma 8.2, the size of TransClasses is bounded by $card(NT)^{n+1}$. So is the size of NT-TransClasses. Then from the previous lemma, there are finitely many NT-TransClasses. Moreover, given two different TransClasses of a given computation, their associated NT-TransClasses are next derived independently. Then, as for regular tree languages, only derivations that do not contain several times the same NT-TransClass in the same sub-derivation must be generated to test emptiness. So, only derivations whose depth is less than the number of NT-TransClasses are needed. □

Thus we get:

Theorem 8.6 *The satisfiability of linear equations in theories given as confluent, constructor-based, linear, without* σ_{in} *(see Sect. 4), without nested functions in rhs's, rewrite systems is decidable. Moreover, the set of solutions can be expressed by a tree tuple synchronized grammar.*

Example 8.7 *Let*

$$\begin{array}{ll} R = \{ & 0 + x \to x,\ x + 0 \to x, \\ & s(x) + s(y) \to s(s(x+y)),\ s(x) + p(y) \to x + y, \\ & p(x) + s(y) \to x + y,\ p(x) + p(y) \to p(p(x+y)) \quad \} \end{array}$$

define the addition in positive and negative integers. This rewrite system does not satisfy the restrictions given in previous work [3, 16, 18, 19, 20, 17, 22, 23], but does satisfy ours. Therefore, we are able to solve linear equations modulo this theory.

Example 8.8 *Let*

$$\left\{ \begin{array}{llll} r_1 : & f(c(x,x'), c(y,y')) & \to & c(f(x,y'), f(x',y)) \\ r_2 : & f(0,0) & \to & 0 \end{array} \right.$$

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

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

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

9 Conclusion

We have presented an original approach using Tree Tuple Synchronized Grammars (TTSGs) to solve linear equations modulo theories given as rewrite systems satisfying some restrictions. This approach allows us to represent infinite sets of solutions in a finite way, and provides a decision procedure for unifiability. Moreover, we have shown that each restriction is needed to decide unifiability.

In future, it would be nice to use TTSGs to deal with equational disunification [25, 26], i.e. finding substitutions that are not the solution of a given equation. This may be achieved if it is possible to compute the set minus between two languages recognized by TTSGs.

TTSG productions look like regular tree language productions because non-terminals are of arity 0. However, they enforce synchronization constraints as well as constraints due to control. Since control tuples contain non-bounded integers, a tree automata that can recognize the language defined by a TTSG need an infinite memory. The question is: What is the place of TTSGs in the known hierarchies of tree grammars?

TTSGs can define infinite sets of (tuples of) ground terms. In automated deduction, several authors have studied recurrent schematizations of infinite sets of terms, like ω-terms, I-terms, R-terms and primal grammars (see elsewhere [27, 28, 29] for an overview). A language like

$$\underbrace{f^n(a) * \ldots * f^n(a)}_{p},\ n,p \in I\!N$$

can be defined both by a primal grammar and by a TTSG that does not come from a unification problem. On the other hand, the symmetric trees cannot be defined by a primal grammar, whereas they are defined by a TTSG coming from a unification problem, as shown in Example 8.8. It would be interesting to study further the possible links between recurrent schematizations and TTSGs.

Acknowledgements

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

References

[1] Plotkin, G. (1972). Building-in equational theories. *Machine Intelligence* **7**, 73–90.

[2] Baader, F. and Siekmann, J. (1993). Unification theory. In: D. M. Gabbay, C. J. Hogger and J. A. Robinson (eds.), *Handbook of Logic in Artificial Intelligence and Logic Programming*. Oxford University Press.

[3] Hullot, J.-M. (1980). Canonical forms and unification. In: W. Bibel and R. Kowalski (eds.), *Proceedings 5th International Conference on Automated Deduction*, Les Arcs, France. (Volume 87 of *LNCS*, pp. 318–334. Springer-Verlag.)

[4] Nutt, W., Réty, P. and Smolka, G. (1989). Basic narrowing revisited. *J. Symbolic Computation* **7**(3 & 4), 295–318.

[5] You, J.-H. (1989). Enumerating outer narrowing derivations for constructor-based term rewriting systems. *J. Symbolic Computation* **7**(3 & 4), 319–342.

[6] Chabin, J. and Réty, P. (1991). Narrowing directed by a graph of terms. In: R. V. Book (ed.), *Proceedings 4th Conference on Rewriting Techniques and Applications*, Como, Italy. (Volume 488 of *LNCS*, pp. 112–123. Springer-Verlag.)

[7] Moreno-Navarro, J. J. and Rodriguez-Artalejo, M. (1992). Logic programming with functions and predicates: The language BABEL. *J. Logic Programming* **12**(3), 191–223.

[8] Bockmayr, A., Krischer, S. and Werner, A. (1995). Narrowing strategies for arbitrary canonical systems. *Fundamenta Informaticae* **24**(1, 2), 125–155.

[9] Antoy, A., Echahed, R. and Hanus, M. (1994). A needed narrowing strategy. *Proceedings 21st ACM Symposium on Principle of Programming Languages*, Portland, OR, pp. 268–279.

[10] Burghardt, J. (1995) Regular substitution sets: a means of controlling E-unification. In: J. Hsiang (ed.), *Proceedings 6th Conference on Rewriting Techniques and Applications*, Kaiserslautern, Germany. (Volume 914 of *LNCS*, pp. 382–396. Springer-Verlag.)

[11] Gilleron, R., Tison, S. and Tommasi, M. (1994). Some new decidability results on positive and negative set constraints. *Proc. First International Conference on Constraints in Computational Logics*. Volume 845 of *LNCS*, pp 336–351. Springer-Verlag.

[12] Dumitrescu, S., Paun, G. and Salomaa, A. (1996). Pattern Languages versus Parallel Communicating Grammar Systems. *Technical Report 42*, Turku Centre for Computer Science, Finland.

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

[14] Dershowitz, N. and Jouannaud, J.-P. (1990). Rewrite systems. In: J. Van Leuven (ed.), *Handbook of Theoretical Computer Science*. Elsevier.

[15] Kozen, D. (1981). Positive first order logic is np-complete. *IBM J. Res. &Develop.* **25**(4), 327–332.

[16] Mitra, S. (1994). Semantic Unification for Convergent Rewrite Systems. *Phd thesis*, University Illinois at Urbana-Champaign.

[17] Christian, J. (1992). Some termination criteria for narrowing and E-unification. *Proceedings 11th International Conference on Automated Deduction*, Albany, NY. (Volume 607 of *Lecture Notes in Artificial Intelligence*, pp. 582–588. Springer-Verlag.)

[18] Comon, H., Haberstrau, M. and Jouannaud, J.-P. (1994). Syntacticness, cycle-syntacticness and shallow theories. *Infor. & Computation* **111**(1), 154–191.

[19] Nieuwenhuis, R. (1996). Basic paramodulation and decidable theories. *Procedings of the 11th Annual IEEE Symposium on Logic in Computer Science*.

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

[21] Mitra, S. (1994). Semantic Unification for Convergent System. *Technical Report CS-R-94-1855*, University of Illinois at Urbana-Champaign.

[22] Kaji, Y., Fujiwara, T. and Kasami, T. (1994). Solving a unification problem under constrained substitutions using tree automata. *Proceedings of the 14th Conference on FST & TCS*, Madras, India. (Volume 880 of *LNCS*, pp. 276–287. Springer-Verlag.)

[23] Faßbender, H. and Maneth, S. (1996). A strict border for the decidability of E-unification for recursive functions. *Proceedings of the Intern. Conf. on Algebraic and Logic Programming*. Volume 1139 of *LNCS*, pp. 194–208. Springer-Verlag.

[24] Domenjoud, E. (1994). Combination techniques for non-disjoint equational theories. In: D. Kapur (ed.), *Proceedings 12th International Conference on Automated Deduction*, Albany, NY. *LNCS*, pp. 267–281. Springer-Verlag.

[25] Fernández, M. (1992). Narrowing based procedures for equational disunification. *Applicable Algebra in Engineering Communication and Computing* **3**, 1–26.

[26] Comon, H. (1993). Complete axiomatizations of some quotient term algebras. *Theoretical Computer Science* **118**(2).

[27] Hermann, M. (1996). Overview of Existing Recurrent Schematizations. *Proc. of the CADE-13 Workshop on Term Schematization and their Applications*.

[28] Amaniss, A. (1996). Classification, extension and applications of some schematization methods of infinite sets of terms. *Proc. of the CADE-13 Workshop on Term Schematization and their Applications*.

[29] Amaniss, A. (1996). *Méthodes de schématisation pour la démonstration automatique*. *Thèse de Doctorat d'Université*, Université de Nancy I.