Conservative Type Extensions for XML Data

Jacques Chabin[1], Mirian Halfeld Ferrari[1],
Martin A. Musicante[2], and Pierre Réty[1]

[1] Université d'Orléans, LIFO, Orléans, France
[2] Universidade Federal do Rio Grande do Norte, DIMAp Natal, Brazil

Abstract. We introduce a method for building a *minimal* XML type (belonging to standard class of regular tree grammars) as an extension of other given types. Not only do we propose an easy-to-handle XML type evolution method, but we prove that this method computes the smallest extension of a given tree grammar, respecting pre-established constraints. We also adapt our technique to an interactive context, where an advised user is guided to build a new XML type from existing ones. A basic prototype of our tool is implemented.

1 Introduction

We deal with the problem of exchanging valid XML data in a multi-system environment. We assume that $I_1, \ldots, I_n$ are local systems that inter-operate with a global system I which should be capable of receiving information from any local system. Local systems $I_1, \ldots, I_n$ deal with sets of XML documents $X_1, \ldots, X_n$, respectively. Each set X_i conforms to schema constraints $\mathcal{D}_i$ and follows an ontology O_i. We want to associate I to a schema for which documents from any local system are valid, and, in this way, we consider that this new schema $\mathcal{D}$ is an evolution for all local systems.

Every real application goes through type evolution, and, thus, in general, our approach is useful whenever one wants not only to generate XML types from given ones but also to preserve the possibility of processing previous versions of software systems. In other words, we focus on a conservative type evolution, *i.e.*, in allowing some backward-compatibility on the types of data processed by new versions, in order to ensure that old clients can still be served. In a multi-system environment, this means to keep a service capable of processing information from any local source, without abolishing type verification.

In the XML world, it is well known that type (or schema) definitions and regular tree grammars are similar notions and that some schema definition languages can be represented by using specific classes of regular tree grammars. As mentioned in [Mani and Lee, 2002], the theory of regular tree grammars provides an excellent framework for understanding various aspects of XML type languages. They have been actively used in many applications such as: XML document processing (*e.g.* XQuery[3] and XDuce[4]) and XML document validation algorithms

[3] `http://www.w3.org/TR/xquery/`
[4] `http://xduce.sourceforge.net/`

(*e.g.* RELAX-NG[5]). They are also used for analyzing the expressive power of the different schema languages [Murata et al., 2005].

Regular tree grammars define sets of trees (in contrast to more traditional string grammars, which generate sets of strings). The rules of *general regular tree grammars* (or RTG) have the form $X \to a\,[R]$, where X in a non-terminal symbol, a is a terminal symbol, and R is a regular expression over non-terminal symbols. *Local tree grammars* (LTG) are regular tree grammars where rules for the same terminal symbol have the same non-terminal on their left-hand side. *Single-type tree grammars* (STTG) are regular tree grammars where distinct non-terminal symbols that appear in a same regular expression of the grammar, always generate distinct terminal symbols. Notice that the restriction for a grammar to be an LTG is stronger than the restriction for STTGs. This means that every LTG is also an STTG.

The interest of regular tree grammars in the context of XML progrssing is that each of the two mainstream languages for typing XML documents, *i.e.*, DTD[6] and XML Schema (XSD)[7], correspond, respectively to LTG and STTG [Murata et al., 2005].

Given an XML type and its corresponding tree grammar G, the set of XML documents described by G corresponds to the language (set of trees) $L(G)$ generated by the tree grammar. Then, given regular tree languages L_1, L_2, … L_n we propose an algorithm for generating a new type that corresponds to a tree language which contains the union of L_1, L_2, … L_n *but which should be an LTG or a STTG*. Notice that even if the grammars $G_1, \ldots, G_n$ that generate $L_1, \ldots, L_n$ are LTGs (resp. STTGs), in general their union $G = G_1 \cup \cdots \cup G_n$ is not an LTG (resp. not a STTG) [Murata et al., 2005]. This is because the union of the sets of rules from these grammars may not respect the conditions imposed by the definitions of LTGs and STTGs. This problem will be illustrated in the next section.

In this context, our proposal can be formally expressed as follows: We present a method for extending a given regular tree grammar G into a new grammar G' respecting the following property: the language generated by G' is the smallest set of unranked trees that contains the language generated by G and the grammar G' is a Local Tree Grammar (LTG) or a Single-Type Tree Grammar (STTG).

Thus, the contribution of this paper is twofold:

1. We introduce two algorithms to transform a given regular grammar G into a new grammar G' such that: (i) $L(G) \subseteq L(G')$; (ii) G' is an LTG or a STTG; and (iii) $L(G')$ is the smallest language that respects constraints (i) and (ii). We offer formal proofs of some interesting properties of our methods.
2. We propose an interactive tool capable of guiding the user in the generation of a new type.

[5] `http://relaxng.org/`
[6] `http://www.w3.org/TR/REC-xml/` .
[7] `http://www.w3.org/XML/Schema` .

Paper organization: Section 2 gives an overview of our contributions. Section 3 introduces notations, concepts and properties needed in the paper. In Section 4 we consider the extension of a regular tree grammar to a local tree grammar while in Section 5 we deal with its extension to a single-type tree grammar. These proposals are revisited and complete versions of the algorithms we have presented in [Chabin et al., 2010]. In Section 6, we modify the first algorithm to get an interactive tool for helping with the construction of XML types. In Section 7, we discuss time complexity and show some experiment results. Section 8 discusses some related work and Section 9 concludes the paper.

2 Overview

This section is an overview of our contributions. In our work, we consider the existence of an ontology associated to each grammar. An ontology alignment allows the implementation of translation functions that establish the correspondence among different words with the same meaning. In the following examples, we represent the tables (a kind of dictionary) over which the translation functions are implemented. All information in these tables is obtained from a given ontology alignment. A word without any entry in a table is associated to itself.

The first example illustrates the evolution of an XML type, expressed by a tree grammar, when the *resulting type should be a DTD (i.e., a LTG).*

Example 1. Let G_1 be a regular tree grammar, resulting from the union of other regular tree grammars. We suppose that the data administrator needs a type for which both organisations of research laboratory are valid. He has an additional constraint: the resulting grammar should be expressed via an LTG (whose translation to a DTD is direct).

G_1		Translation table
$R_1 \to lab[T_1^*]$	$R_2 \to lab[Emp^*]$	$researcher \leftrightarrow employee$
$T_1 \to team[Res^*]$	$Emp \to employee[IsIn]$	$team \leftrightarrow group$
$Res \to researcher[\epsilon]$	$IsIn \to group[\epsilon]$	

By translating the rules of G_1 according to the translation table we verify that the non-terminals *Res* and *Emp* generate the same terminal *researcher*, and consequently are in competition, which is forbidden for local tree grammars or DTDs (*i.e.*, G_1 is not an LTG). The same conclusion is obtained for non-terminals T_1 and $IsIn$ which generate terminal *team.* Our goal is to transform the new G_1 into an LTG, and this transformation will result in a grammar that generates a language L such that $L(G_1) \subseteq L$.

In this context, we propose an algorithm that extends G_1 into a new local tree grammar G_A. The solution proposed is very simple. Firstly, rules concerning non-terminals *Res* and *Emp* are combined, given the rule $S \to researcher[IsIn \mid \epsilon]$. Secondly, in regular expressions (of other rules), *Res* and *Emp* should be replaced by S. Thus, we have $T_1 \to team[S^*]$ and $R_2 \to lab[S^*]$. All competing non-terminals are treated in the same way, giving rise to grammar G_A with rules: $R \to lab[T^* \mid S^*]$; $S \to researcher[T \mid \epsilon]$ and $T \to team[S^* \mid \epsilon\ ]$. □

The result obtained in Example 1 respects the imposed constraints: the obtained grammar is the least LTG (in the sense of language inclusion) and it generates a language that contains the language $L(G_1)$. This algorithm is presented in Section 4.

Next, we consider the situation where the resulting type is supposed to be specified by a XSD (or a single-type tree grammar). The following example illustrates an evolution in this context.

Example 2. Let us now consider G_2, a regular tree grammar resulting from the union of other regular tree grammars. We focus only on the rules concerning publication types. The translation table summarizes the correspondence among terminals. An ontology alignment associates *article* and *paper*.

G_2	Translation table
$R_3 \rightarrow article[Title.TitleJournal.Year.Vol]$	$article \leftrightarrow paper$
$R_4 \rightarrow paper[Title.TitleConf.Year]$	
$R_5 \rightarrow publication[(R_3 \mid R_4)^*]$	

We propose an algorithm that extends G_2 to a single type grammar G_B, which can then be translated into an XSD. Notice that the above rules of G_2 violate STTG constraints, since rule R_5 contains a regular expression with competing non-terminals. Indeed, rules $R_3 \rightarrow article[Title.TitleJournal.Year.Vol]$ and $R_4 \rightarrow paper[Title.TitleConf.Year]$ have equivalent terminals, according to the translation table. Thus, non-terminals R_3 and R_4 are competing ones. This is not a problem for a STTG. However, both non-terminals appear in the regular expression of rule R_5 and this is forbidden for a STTG. In this case, our method replaces the above rules by the following ones:

G_B
$R_6 \rightarrow paper[Title.TitleJournal.Year.Vol \mid Title.TitleConf.Year]$
$R_7 \rightarrow publication[R_6^*]$

□

To get an LTG, competing non-terminals should be merged, which is simple. To get a STTG, the situation is more complicate: only competing non-terminals that appear in the same regular expression should be merged. Consequently, if R_1, R_2, R_3 are competing, but R_1, R_2 appear in the regular expression E whereas R_1, R_3 appear in E', then R_1 should be merged with R_2 (and not with R_3) within E, whereas R_1 should be merged with R_3 (and not with R_2) within E'. To do it in the general case, we introduce an equivalence relation over non-terminals, and consider equivalence classes as being the non-terminals of the new grammar. The algorithm is given in Section 5.

Now, suppose that the administrator wants to build a new type based on the types he knows, *i.e.*, by merging, in a flexible way, different types, *e.g.* G_3 and G_4. We propose a tool to guide him in this construction by considering one rule (of G_3 and G_4) at a time. The new type is built with the help of a dependency graph $D = (V, E)$. The set of nodes V contains the terminals of both grammars: the set of arcs E represents the notion of *dependency* among terminals in these grammars. The pair (a, b) of terminals is in E *iff* a production rule generating a contains a non-terminal which is associated to the terminal b. We just consider grammars where the dependency graph has no cycles. The following example illustrates this contribution.

Example 3. Let us consider the two grammars below. We concentrate in the rule concerning researchers.

G_3	G_4
$R_1 \rightarrow researcher[IdR.Name.Pub]$	$R_2 \rightarrow researcher[Name.Team.Ref]$

Figure 1 shows the dependency graph for these rules. We consider that non-terminals *IdR, Name, Pub, Name, Team, Ref* are the left-hand side of rules whose right-hand side contain terminals *idR, name, publications, team* and *references*, respectively. Arcs are colored according to the grammar they come from: red (filled arrow) to indicate they come from both grammars, blue (dashed arrow) only from G_3 and green (dotted arrow) only from G_4.

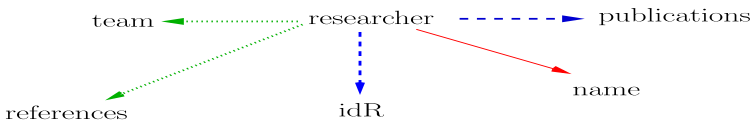


Fig. 1. Dependency graph for grammars G_3 and G_4 .

Our interactive tool proposes to follow this graph in a topological order: start with nodes with no output arcs, process them, delete them from the graph together with their input arcs, and so on. Processing a node here means writing its production rule. For each competing terminal, the user can propose a regular expression to define it. This regular expression is built only with non-terminals appearing as left-hand side of production rules already defined.

For instance, in our case, our interactive tool starts by proposing rules for each $A \in \{IdR, Name, Pub, Team, Ref\}$ (here, we consider that all these non-terminals are associated to rules of the format $A \rightarrow a[\epsilon]$). Then, the administrator can define the local type R, for researchers by using any of the above non-terminals. Suppose that the chosen rule is: $R \rightarrow researcher[(Name.Team.Pub)]$.

After this choice (and since in our example, no other type definition is expected), all rules defining useless types are discarded (*i.e.*, rules for *IdR* and *Ref* are discarded). Thus, we obtain a new grammar G_B, built by the administrator, guided by our application. More details of this tool are presented in Section 6.

Finally, if our administrator needs to accept documents of types G_3 or G_4 or G_B, he may again use the first algorithm to define:

$$R \rightarrow researcher[(IdR.Name.Pub) \mid (Name.Team.Ref) \mid (Name.Team.Pub)].$$

□

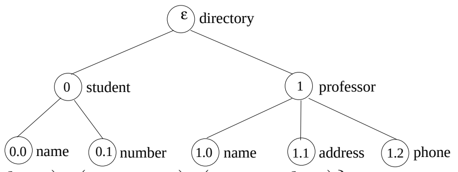
The previous examples illustrate that our contribution is twofold. On the one hand, to propose algorithms that automatically compute a minimal type extension of given types. On the other hand to apply these algorithms as guides to allow the interactive definition of new types.

3 Theoretical Background

It is a well known fact that type definitions for XML and regular tree grammars are similar notions and that some schema definition languages can be represented by using specific classes of regular tree grammars. Thus, DTD and XML Schema, correspond, respectively, to Local Tree Grammars and Single-Type Tree Grammars [Murata et al., 2005]. Given an XML type T and its corresponding tree grammar G, the set of XML documents described by the type T corresponds to the language (set of trees) generated by G.

An XML document is an unranked tree, defined in the usual way as a mapping t from a set of positions (nonempty and closed under prefixes) $Pos(t)$ to an alphabet Σ. For $v \in Pos(t)$, $t(v)$ is the label of t at the position v, and $t|_v$ denotes the subtree of t at position v. Positions are sequences of integers in $\mathbb{N}^*$ and the set $Pos(t)$ satisfies: $j \geq 0, u.j \in Pos(t), 0 \leq i \leq j \Rightarrow u.i \in Pos(t)$, where the "." denotes the concatenation of sequences of integers. As usual, ϵ denotes the empty sequence of integers, *i.e.* the root position. The following figure shows a tree whose alphabet is the set of element names appearing in an XML document. In this case we have $t(\epsilon) = directory$, $t(0) = student$ and so on.

Given a tree t we denote by $t|_p$ the subtree whose root is at position $p \in Pos(t)$,*i.e.*, $Pos(t|_p) = \{s \mid p.s \in Pos(t)\}$ and for each $s \in Pos(t|_p)$ we have $t|_p(s) = t(p.s)$.



For instance, in the figure $t|_0 = \{(\epsilon, student), (0, name), (1, number)\}$, or equivalently, $t|_0 = student(name, number)$.

Given a tree t such that the position $p \in Pos(t)$ and a tree t', we note $t[p \leftarrow t']$ as the tree that results of substituting the subtree of t at position p by t'.

Definition 1 (Sub-tree, forest). Let L be a set of trees. $ST(L)$ is the set of sub-trees of elements of L, i.e. $ST(L) = \{t \mid \exists u \in L, \exists p \in Pos(u), t = u|_p\}$. A *forest* is a (possibly empty) tuple of trees. For $a \in \Sigma$ and a forest $w = \langle t_1, \ldots, t_n \rangle$, $a(w)$ is the tree defined by $a(w) = a(t_1, \ldots, t_n)$. On the other hand, $w(\epsilon)$ is defined by $w(\epsilon) = \langle t_1(\epsilon), \ldots, t_n(\epsilon) \rangle$, *i.e.*, the tuple of the top symbols of w. □

Definition 2 (Regular Tree Grammar, derivation). A *regular tree grammar* (RTG) is a 4-tuple $G = (N, T, S, P)$, where: N is a finite set of *non-terminal symbols*; T is a finite set of *terminal symbols*; S is a set of *start symbols*, where $S \subseteq N$ and P is a finite set of *production rules* of the form $X \rightarrow a\,[R]$, where $X \in N$, $a \in T$, and R is a regular expression over N (We say that, for a production rule, X is the left-hand side, $a\,[R]$ is the right-hand side, and R is the content model.)

For an RTG $G = (N, T, S, P)$, we say that a tree t built on $N \cup T$ derives (in one step) into t' iff (i) there exists a position p of t such that $t|_p = A \in N$ and a production rule $A \rightarrow a\,[R]$ in P, and (ii) $t' = t[p \leftarrow a(w)]$ where $w \in L(R)$ ($L(R)$ is the set of words of non-terminals generated by R). We write $t \rightarrow_{[p, A \rightarrow a\,[R]]} t'$. More generally, a derivation (in several steps) is a (possibly empty) sequence of one-step derivations. We write $t \rightarrow_G^* t'$.

The *language* $L(G)$ generated by G is the set of trees containing only terminal symbols, defined by : $L(G) = \{t \mid \exists A \in S,\ A \rightarrow_G^* t\}$. □

Remark: As usual, in this paper, our algorithms start from grammars in reduced form and (as in [Mani and Lee, 2002]) in normal form. A *regular tree grammar* (RTG) is said to be in **reduced form** if (i) every non-terminal is reachable from a start symbol, and (ii) every non-terminal generates at least one tree containing only terminal symbols. A regular tree grammar (RTG) is said to be in **normal form** if distinct production rules have distinct left-hand-sides. □

To distinguish among sub-classes of regular tree grammars, we should understand the notion of competing non-terminals. Moreover, we define an equivalence relation on the non-terminals of grammar G, so that all competing non-terminals are in the same equivalence class. In our schema evolution algorithms (Sections 4 and 5), these equivalence classes form the non-terminals of the new grammar.

Definition 3 (Competing Non-terminals). Let $G = (N, T, S, P)$ be a regular tree grammar. Two non-terminals A and B are said to be *competing with each other* if $A \neq B$ and G contains production rules of the form $A \rightarrow a[E]$ and $B \rightarrow a[E']$ (i.e. A and B generate the same terminal symbol).
Define a **grouping relation over competing non-terminals** as follows. Let $\parallel$ be the relation on N defined by: for all $A, B \in N$, $A \parallel B$ iff $A = B$ or A and B are competing in P. For any $\chi \subseteq N$, let $\parallel_\chi$ be the restriction of $\parallel$ to the set χ ($\parallel_\chi$ is defined only for elements of χ). □

Lemma 1. *Since G is in normal form, $\parallel$ is an equivalence relation. Similarly, $\parallel_\chi$ is an equivalence relation for any $\chi \subseteq N$.* □

Definition 4 (Local Tree Grammar and Single-Type Tree Grammar). A *local tree grammar* (LTG) is a regular tree grammar that does not have competing non-terminals. A *local tree language* (LTL) is a language that can be generated by at least one LTG. A *single-type tree grammar* (STTG) is a regular tree grammar in normal form, where (i) for each production rule, non terminals in its regular expression do not compete with each other, and (ii) starts symbols do not compete with each other. A *single-type tree language* (STTL) is a language that can be generated by at least one STTG. □

In [Murata et al., 2005] the expressive power of these classes of languages is discussed. We recall that LTL $\subset$ STTL $\subset$ RTL (*RTL for regular tree language*). Moreover, the LTL and STTL are closed under intersection but not under union; while the RTL are closed under union, intersection and difference. Note that converting an LTG into normal form produces an LTG as well.

4 Transforming an RTG into an LTG

Given a regular tree grammar $G_0 = (N_0, T_0, S_0, P_0)$, we propose a method to compute a local tree grammar G that generates the least local tree language containing $L(G_0)$.

In [Chabin et al., 2010], we have introduced a very intuitive version of our algorithm: replace each pair of competing non-terminals by a new non-terminal, until there are no more competing non-terminals.

In this section, we prefer to use the well-known formalism of equivalence classes (Lemma 1), which makes the proofs simpler, and allows an uniform notation *w.r.t.* to the algorithm in the next section. Competing non-terminals are grouped together within an equivalence class, and the equivalence classes are the non-terminals of the new grammar G.

Algorithm 1 (RTG into LTG Transformation)
Notation:
(i) For any $A \in N_0$, $\hat{A}$ denotes the equivalence class of A w.r.t. relation $\|$, *i.e.*, $\hat{A}$ contains A and the non-terminals that are competing with A in P_0.
(ii) For any regular expression R, $\hat{R}$ is the regular expression obtained from R by replacing each non-terminal A by $\hat{A}$.
(iii) As usual, $N_0/_{\|}$ denotes the quotient set, i.e. the set of the equivalence classes.

Let $G_0 = (N_0, T_0, S_0, P_0)$ be a regular tree grammar. We define a new regular tree grammar $G = (N, T, S, P)$, obtained from G_0, as follows:

Let $G = (N_0/_{\|}, T_0, S, P)$ where:

- $S = \{\hat{A} \mid A \in S_0\}$,
- $P = \{ \{A_1, \ldots, A_n\} \to a\,[\hat{R}] \mid \{A_1, \ldots, A_n\} \in N_0/_{\|}$,
  and $A_1 \to a[R_1], \ldots, A_n \to a[R_n] \in P_0$, and $R = (R_1|\cdots|R_n)$. $\square$

The following example illustrates our algorithm.

Example 4. Let us consider merging the rules for two different DTDs for cooking recipes. Assuming that the vocabulary translations have already been done (on the basis of an alignment ontology), we build the grammar G_0 below. Each $A \in \{$*Name, Number, Unit, Quantity, Step, Item*$\}$ is associated to a production rule having the format $A \to a[\epsilon]$, meaning that label a is attached to data.

$$\begin{array}{rlrl}
Recipe_a \to & r[Ingreds.Recipe_a^*.Instrs_a] & Ingreds \to & is[OneIng_a^*] \\
OneIng_a \to & ing[Name.Unit.Quantity] & Instrs_a \to & ins[Step^*] \\[1ex]
Recipe_b \to & r[Required.OneIng_b^*.Instrs_b] & Required \to & req[Item^*] \\
OneIng_b \to & ing[Name.Quantity.Unit] & Instrs_b \to & ins[(Number.Step)^*]
\end{array}$$

Clearly, non-terminals $Recipe_a$ and $Recipe_b$, $OneIng_a$ and $OneIng_b$, $Instrs_a$ and $Instrs_b$ are competing. The equivalence classes for G_0 are $\{Recipe_a, Recipe_b\}$, $\{OneIng_a, OneIng_b\}$, $\{Instrs_a, Instrs_b\}$, $\{Ingreds\}$, $\{Required\}$, $\{Number\}$, $\{Name\}$, $\{Unit\}$, $\{Quantity\}$, $\{Step\}$, $\{Item\}$. Each equivalence class is now seen as a new non-terminal. Our algorithm combines rules of G_0 whose left-hand non-terminals (in N_0) are in the same equivalence class. The obtained result is the LTG G below. To shorten the notations, for each non-terminals like X, Y_a, Y_b we write X instead of $\{X\}$, and Y instead of $\{Y_a, Y_b\}$. The missing rules are of the form $A \to a[\epsilon]$.

$$\begin{aligned}
\mathit{Recipe} &\to r[(\mathit{Ingreds}.\mathit{Recipe}^*.\mathit{Instrs})|(\mathit{Required}.\mathit{OneIng}^*.\mathit{Instrs}])\\
\mathit{Ingreds} &\to \mathit{is}[\mathit{OneIng}^*]\\
\mathit{OneIng} &\to \mathit{ing}[(\mathit{Name}.\mathit{Unit}.\mathit{Quantity})|(\mathit{Name}.\mathit{Quantity}.\mathit{Unit})]\\
\mathit{Instrs} &\to \mathit{ins}[\mathit{Step}^*|(\mathit{Number}.\mathit{Step})^*]\\
\mathit{Required} &\to \mathit{req}[\mathit{Item}^*] \qquad \Box
\end{aligned}$$

One of the most important beauty of our algorithm is its simplicity. However, one fundamental contribution of this paper is the proof that, with this very simple method, we can compute the smallest extension of a given tree grammar, respecting the constraints imposed on an LTG. This result is stated in the following theorem.

Theorem 1. *The grammar returned by Algorithm 1 generates the least LTL that contains $L(G_0)$.* □

The intuition behind the proof of Theorem 1 is as follows. Let G be the grammar produced by our algorithm and let G' be any LTG such that $L(G_0) \subseteq L(G')$, we have to prove that $L(G_0) \subseteq L(G)$ (soundness), and that $L(G) \subseteq L(G')$ (minimality: $L(G)$ is the least LTL containing $L(G_0)$). Proving soundness is not very difficult. Minimality comes from the following steps: **(A)** As production rules of an LTG in normal form define a bijection between the sets of terminals and non-terminals, there is only one rule in G of the form $\hat{A}_1 \to a[R]$ producing subtrees with root a. By the construction of our algorithm this rule should correspond to rules $A_i \to a[R_i]$ in G_0 with $i \in \{1, \cdots, n\}$. All A_i are competing in G_0 and no other symbol in N_0 is competing with a A_i so $\hat{A}_1 = \cdots = \hat{A}_n = \{A_1, \cdots, A_n\}$. And we have $R = \hat{R}_1 \mid \cdots \mid \hat{R}_n$. **(B)** Consequently, we can prove that if $a(w)$ is a subtree of $t \in L(G)$, then there is at least one tree in $L(G_0)$ with $a(w')$ as a subtree, s.t. $w'(\epsilon) = w(\epsilon)$ (i.e. forests w' and w have the same tuple of top-symbols). **(C)** w is a forest composed by subtrees of $L(G)$, and by induction hypothesis applied to each component of w (each component is a strict subtree of $a(w)$), we know that w is also a forest composed by subtrees of $L(G')$. On the other hand, since $L(G_0) \subseteq L(G')$, $a(w')$ is a subtree of $L(G')$. **(D)** As G' is an LTG, and thanks to some properties of local languages, we can replace each subtree of $a(w')$, rooted by the elements of $w'(\epsilon)$, by the corresponding subtree of $a(w)$ and thus, $a(w)$ is a subtree of $L(G')$. **(E)** Finally, as this is valid for every subtree, we have that $L(G) \subseteq L(G')$.

Appendix A presents the proof of Theorem 1.

5 Transforming an RTG into a STTG

Given a regular tree grammar $G_0 = (N_0, T_0, S_0, P_0)$, the following algorithm computes a single-type tree grammar G that generates the least single-type tree language containing $L(G_0)$. It is based on grouping competing non-terminals into equivalence classes, in a way different from Algorithm 1. Here, we group competing non-terminals $A_1, \ldots, A_n$ together, only if they appear in the same regular expression R of G_0, and in this case the set $\{A_1, \ldots, A_n\}$ is a non-terminal of G. If $A_1, \ldots, A_n$ do not appear in the same regular expression, we have to consider subsets of $\{A_1, \ldots, A_n\}$. This is why Algorithm 2 (and proofs) is more complicated than Algorithm 1.

Algorithm 2 (RTG into STTG Transformation)
Notation:
(i) For any regular expression R, $N(R)$ denotes the set of non-terminals occurring in R.
(ii) For any $\chi \subseteq N_0$ and any $A \in \chi$, $\hat{A}^{\chi}$ denotes the equivalence class of A w.r.t. relation $\|_{\chi}$, i.e. $\hat{A}^{\chi}$ contains A and the non-terminals of χ that are competing with A in P_0.
(iii) $\sigma_{N(R)}$ is the substitution defined over $N(R)$ by $\forall A \in N(R)$, $\sigma_{N(R)}(A) = \hat{A}^{N(R)}$. By extension, $\sigma_{N(R)}(R)$ is the regular expression obtained from R by replacing each non-terminal A in R by $\sigma_{N(R)}(A)$.

Let $G_0 = (N_0, T_0, S_0, P_0)$ be a regular tree grammar. We define a new regular tree grammar $G = (N, T, S, P)$, obtained from G_0, according to the following steps:

1. Let $G = (\mathcal{P}(N_0), T_0, S, P)$ where:
   – $S = \{\hat{A}^{S_0} \mid A \in S_0\}$,
   – $P = \{\, \{A_1, \ldots, A_n\} \to a\,[\sigma_{N(R)}(R)] \mid$
   $A_1 \to a[R_1], \ldots, A_n \to a[R_n] \in P_0,\ R = (R_1|\cdots|R_n)\}$,
   where $\{A_1, \ldots, A_n\}$ indicates all non-empty sets containing competing non-terminals (not only the maximal ones).
2. Remove unreachable non-terminals and unreachable rules in G; return G. $\square$

The difference between STTG and LTG versions (Section 4) is in the use of non-maximal sets of competing non-terminals. In particular, Algorithm 2 considers (step 1) *each* set of competing non-terminals as a left-hand side (and not only maximal sets) to build the production rules of G. Thus, at step 1, G may create unreachable rules (from the start symbols), which are then removed at step 2. Algorithm 2 eases our proofs. An optimized version, where just the needed non-terminals are generated, is given in [Chabin et al., 2010].

The following example illustrates that for an STTG only competing non-terminals appearing in the same regular expression are combined to form a new non-terminal.

Example 5. Let G_0 be a non-STTG grammar having the following set P_0 of productions rules (*School* is the start symbol). It describes a French school with students enrolled to an international English section (*IntStudent*) and normal French students (*Student*). Different options are available for each student class.

$$
\begin{aligned}
\mathit{School} &\rightarrow \mathit{school}[\mathit{IntStudent} \mid \mathit{Student}] \\
\mathit{Student} &\rightarrow \mathit{student}[\mathit{Name}.\mathit{Option3}] \\
\mathit{IntStudent} &\rightarrow \mathit{intstudent}[\mathit{Name}.(\mathit{Option1} \mid \mathit{Option2})] \\
\mathit{Option1} &\rightarrow \mathit{option}[\mathit{EL}.\mathit{GL}] & \mathit{Option2} &\rightarrow \mathit{option}[\mathit{EL}.\mathit{SL}] \\
\mathit{Option3} &\rightarrow \mathit{option}[\mathit{EL}] & \mathit{Name} &\rightarrow \mathit{name}[\epsilon] \\
\mathit{EL} &\rightarrow \mathit{english}[\epsilon] & \mathit{GL} &\rightarrow \mathit{german}[\epsilon] \\
\mathit{SL} &\rightarrow \mathit{spanish}[\epsilon]
\end{aligned}
$$

The grammar G obtained by our approach has the rules below where non terminals are named by their equivalence class. Clearly, they can be denoted by shorter notations.

$$
\begin{aligned}
\{\mathit{School}\} &\rightarrow \mathit{school}[\{\mathit{IntStudent}\} \mid \{\mathit{Student}\}] \\
\{\mathit{IntStudent}\} &\rightarrow \mathit{intstudent}[\{\mathit{Name}\}.\{\mathit{Option1}, \mathit{Option2}\}] \\
\{\mathit{Student}\} &\rightarrow \mathit{student}[\{\mathit{Name}\}.\{\mathit{Option3}\}] \\
\{\mathit{Option1}, \mathit{Option2}\} &\rightarrow \mathit{option}[(\{\mathit{EL}\}.\{\mathit{GL}\}) \mid (\{\mathit{EL}\}.\{\mathit{SL}\})] \\
\{\mathit{Option3}\} &\rightarrow \mathit{option}[\{\mathit{EL}\}] \\
\{\mathit{Name}\} &\rightarrow \mathit{name}[\epsilon] \\
\{\mathit{EL}\} &\rightarrow \mathit{english}[\epsilon] \\
\{\mathit{GL}\} &\rightarrow \mathit{german}[\epsilon] \\
\{\mathit{SL}\} &\rightarrow \mathit{spanish}[\epsilon]
\end{aligned}
$$

Notice that although *Option1*, *Option2* and *Option3* are competing non-terminals; our approach does not produce new non-terminals corresponding to the combination of all of them. For instance, *Option1* and *Option2* are combined in order to generate the non-terminal $\{\mathit{Option}1, \mathit{Option}2\}$, but we do not need to produce a non-terminal {*Option1, Option3*} since *Option1* and *Option3* do not appear together in a regular expression. We also generate {*Option3*} as non-terminal because *Option3* is alone in the rule defining *Student*. □

The following example offers an interesting illustration of the extension of the original language.

Example 6. Consider a non-STTG grammar G_0 having the following set P_0 of productions rules (*Image* is the start symbol):

$$
\begin{aligned}
\mathit{Image} &\rightarrow \mathit{image}[\mathit{Frame1} \mid \mathit{Frame2} \mid \mathit{Background}.\mathit{Foreground}] \\
\mathit{Frame1} &\rightarrow \mathit{frame}[\mathit{Frame1}.\mathit{Frame1} \mid \epsilon] \\
\mathit{Frame2} &\rightarrow \mathit{frame}[\mathit{Frame2}.\mathit{Frame2}.\mathit{Frame2} \mid \epsilon] \\
\mathit{Background} &\rightarrow \mathit{back}[\mathit{Frame1}] \\
\mathit{Foreground} &\rightarrow \mathit{fore}[\mathit{Frame2}]
\end{aligned}
$$

Grammar G_0 defines different ways of decomposing an image: recursively into two or three frames or by describing the background and the foreground separately. Moreover, the background (resp. the foreground) is described by binary decompositions (resp. ternary decompositions). In other words, the language of G_0 contains the union of the trees: *image(bin(frame))*; *image(ter(frame))* and *image (back (bin (frame)), fore (ter (frame)))* where *bin* (resp. *ter*) denotes the set of all binary (resp. ternary) trees that contains only the symbol *frame*. The result is G, which contains the rules below (the start symbol is $\{Image\}$) :

$$\begin{aligned}
\{\mathit{Image}\} &\rightarrow \mathit{image}[\{\mathit{Frame1}, \mathit{Frame2}\} \mid \{\mathit{Background}\}.\{\mathit{Foreground}\}] \\
\{\mathit{Background}\} &\rightarrow \mathit{back}[\{\mathit{Frame1}\}] \\
\{\mathit{Foreground}\} &\rightarrow \mathit{fore}[\{\mathit{Frame2}\}] \\
\{\mathit{Frame1}, \mathit{Frame2}\} &\rightarrow \mathit{frame}[\epsilon \mid \{\mathit{Frame1}, \mathit{Frame2}\}.\{\mathit{Frame1}, \mathit{Frame2}\} \\
&\qquad \mid \{\mathit{Frame1}, \mathit{Frame2}\}.\{\mathit{Frame1}, \mathit{Frame2}\}.\{\mathit{Frame1}, \mathit{Frame2}\}] \\
\{\mathit{Frame1}\} &\rightarrow \mathit{frame}[\{\mathit{Frame1}\}.\{\mathit{Frame1}\} \mid \epsilon] \\
\{\mathit{Frame2}\} &\rightarrow \mathit{frame}[\{\mathit{Frame2}\}.\{\mathit{Frame2}\}.\{\mathit{Frame2}\} \mid \epsilon]
\end{aligned}$$

Grammar G is a STTG that generates the union of *image(tree(frame))* and *image (back (bin (frame)), fore (ter (frame)))* where *tree* denotes the set of all trees that contain only the symbol *frame* and such that each node has 0 or 2 or 3 children. Let $L_G(X)$ be the language obtained by deriving in G the non-terminal X. Actually, $L_G(\{Frame1, Frame2\})$ is the least STTL that contains $L_{G_0}(Frame1) \cup L_{G_0}(Frame2)$. □

An important part of our work consist in proving the following theorem. We have presented part of this proof in [Chabin et al., 2010], but the interested reader can find its complete version in Appendix B.

Theorem 2. *The grammar returned by Algorithm 2 generates the least STTL that contains* $L(G_0)$. □

From this result, we are able to ensure that our algorithm generates the least STTL responding to our goals. This is an important result when dealing with type we want to extend in a controlled way.

6 Interactive Generation of New Types

This section introduces an interactive tool which may help an advised user to build an XML type based on existing ones. This tool is useful when an administrator decides to propose a new type which should co-exist with other (similar) types for a certain time. For instance, the description of research laboratories in a university may vary. Our administrator wants to organize information in a more uniform way by proposing a new type, a schema evolution, based on the existing types (since old types represent all information available and needed). In this paper we outline the main ideas of our tool which can be used in different contexts where slightly different XML types exist and should be catalogued. Indeed, this kind of application needs the two parts of our proposal: to extend the union of types to a standard XML schema language and to interactively allow the construction of a new type. We illustrate the interactive function of our tool (outlined in Section 1) in a more complete example.

Example 7 (Interactive approach). We consider the rules from two LTG after translating terminals into a unified nomenclature, according to a translation table. Each grammar shows a different organization of research laboratories. In G_1 laboratories are organized with researchers and teams. Publications are sub-elements of researchers. Teams are composed by researchers, identified by their identification numbers. All $A \in \{Dom, IdR, First, Last, TitleP, TitleConf, Year, Vol\}$ are associated to production rules having the format $A \to a[\epsilon]$.

$Lab \to lab[Dom.R^*.Team^*]$ $\qquad$ $R \to researcher[IdR.Name.P]$
$P \to publications[CPaper^*.JPaper^*]$ $\qquad$ $Team \to team[IdR^*]$
$CPaper \to confPaper[TitleP.TitleConf.Year]$
$JPaper \to jourPaper[TitleP.TitleJ.Year.Vol]$
$Name \to name[First.Last]$

Grammar G_2 represents an organization where researchers and publications are stored in an independent way, but where references are supposed to link informations. A team is just a sub-element of a researcher. All $A \in \{ Dom, Name, Code, TitleP, TitleConf, Year, Vol \}$ are associated to production rules having the format $A \to a[\epsilon]$.
$Lab \to lab[Dom.R^*.P]$ $\qquad$ $R \to researcher[Name.Team.Ref]$
$Ref \to references[Code^*]$ $\qquad$ $P \to publications[CPaper^*.JPaper^*]$
$ConfPaper \to confPaper[Code.TitleP.TitleConf.Year]$
$JPaper \to jourPaper[Code.TitleP.TitleJ.Year.Vol]$

To guide the construction of a new grammar we use a dependency graph. Figure 2 shows the dependency graph for grammars G_1 and G_2.

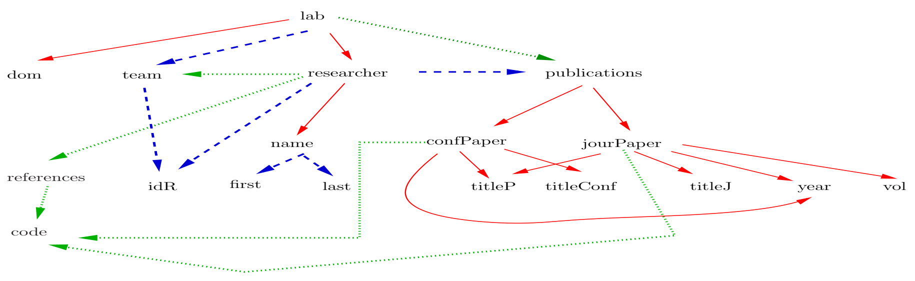


Fig. 2. Dependency graph for grammars G_1 and G_2 .

As mentioned in Example 3, to aid an advised user in the construction of a new XML type, we propose to follow this graph in a topological order and, for each competing terminal, to ask the user a regular expression to define it. This regular expression is built only by non-terminals already defined. From Figure 2 the user starts by considering terminals $a \in \{idR, name, team, titleP, titleConf, titleJ, year, code, vol\}$. Let G_3 be the grammar obtained by this interactive method. Clearly, all terminals a are associated to *data* and thus the corresponding grammar rules are of the format $A \to a[\epsilon]$. The following production rules correspond to the user choices.

$Lab \to lab[Dom.R^*]$ $\qquad R \to researcher[Name.Team.P]$
$P \to publications[CPaper^*.JPaper^*]$
$CPaper \to confPaper[TitleP.TitleConf.Year]$
$JPaper \to jourPaper[TitleP.TitleJ.Year.Vol]$ $\square$

We now formally present our interactive algorithm (which adapts Algorithm 1 to an interactive context). We recall that for a regular expression R (resp. a tree t), $NT(R)$ denotes the set of non-terminals occurring in R (resp. in t).

Definition 5 (Order relation over terminals). Let $G = (N, T, S, P)$ be an RTG. Let $a, b \in T$. We define the relation $\leadsto_G$ over terminals by $a \leadsto_G b$ iff there exist production rules $A \to a[R]$, $B \to b[R']$ in P such that $B \in NT(R)$. In other words, b may be a child of a. $\square$

Non-recursivity is defined as usual, i.e. over non-terminals: it means that a non-terminal A can never derive a tree that contains A again. Using terminals, we get a stronger property, called strong non-recursivity, which means that the dependency graph is acyclic.

Definition 6 (Recursivity).
A grammar G is *non-recursive* iff $\neg(\exists A \in N,\ A \to_G^+ t \wedge A \in NT(t))$, where t is a tree. A grammar G is *strongly non-recursive* iff $\neg(\exists a \in T,\ a \leadsto_G^+ a)$, where $\to_G^+$ and $\leadsto_G^+$ are the transitive closures of $\to_G$ and $\leadsto_G$, respectively. $\square$

Lemma 2. *If G is strongly non-recursive, then G is non-recursive.*

Proof: By contraposition. Suppose $\exists A_0 \in N,\ A_0 \to_G^+ t \wedge A_0 \in NT(t))$. Then: $\exists n \in \mathbb{N}\backslash\{0\},\ \forall i \in \{0, \dots, n-1\},\ \exists A_i \to a_i[R_i] \in P,\ A_{i+1} \in NT(R_i) \wedge A_n = A_0$. By definition of $\leadsto_G$ we have: $a_0 \leadsto_G a_1 \leadsto_G \cdots \leadsto_G a_{n-1} \leadsto_G a_0$. $\square$

Lemma 3. *If G is strongly non-recursive, then $\leadsto_G^+$ is a strict (partial) order.*

Proof: $\leadsto_G^+$ is transitive. On the other hand, for all $a, b \in T$, $a \leadsto_G^+ b \wedge b \leadsto_G^+ a$ implies $a \leadsto_G^+ a$, which is impossible since G is strongly non-recursive. $\square$

Algorithm 3 (Interactive Generation of an LTG)
Notation: Let $G_0 = (N_0, T_0, S_0, P_0)$ be a regular tree grammar[8] strongly non-recursive. For each terminal a, consider all the rules in P_0 that generates a, say $A_1 \to a[R_1], \dots, A_n \to a[R_n]$. Then we define $\hat{a} = \{A_1, \dots, A_n\}$. Note that $\hat{a} \in N_0/_{\parallel}$, i.e. $\hat{a}$ is an equivalence class.

We define a new regular tree grammar $G = (N, T, S, P)$, obtained from G_0, according to the steps:

1. Let $G = (N_0/_{\parallel}, T_0, S, P)$ where:

[8] Recall that G_0 is in reduced normal form and thus, for each $A \in N_0$ there exists exactly one rule in P_0 whose left-hand-side is A.

– $S = \{\hat{A} \mid A \in S_0\}$,
– $P = \{\, \hat{a} \to a\,[R] \mid a \in T_0$,
  and $\hat{a} = \{A_1, \ldots, A_n\} \in N_0/_{\|}$,
  and $A_1 \to a[R_1], \ldots, A_n \to a[R_n] \in P_0$,
  and (i) $R = (\hat{R_1}| \cdots |\hat{R_n})$ or (ii) R is defined by the user s.t.
  $\forall B \in NT(R),\ B = \hat{b} \wedge a \rightsquigarrow^{+}_{G_0} b$.

2. Remove all unreachable or unproductive non-terminals and rules in G, then return it. □

The aiding tool for XML type construction we propose is based on Algorithm 3. However, to make it more user friendly, each time a user wants to propose a new local type (the interactive step mentioned in item 1(ii)), some facilities are offered. The first one aims at releasing the user of thinking about grammar "technical problems" and distinction concerning terminal and non-terminals. Therefore, our tool allows the user to propose regular expressions built over XML labels (*i.e.*, the terminals of G). Indeed, this opportunity matches the use of DTDs. Grammar G resulting from Algorithm 3 is automatically obtained by replacing each terminal b (used by the user in the construction of the type) by non-terminal $\hat{b}$. Note that the limitation of the user choice in item 1(ii) (only b's s.t. $a \rightsquigarrow^{+}_{G_0} b$ are allowed) is necessary to prevent from introducing cycles in the dependency graph, i.e. to get a strongly non-recursive grammar. The second facility aims at guiding the user in a good definition order. Thus, at each step, our tool may guide the user to choose new (local) types according to the order established by a topological sort of the dependency graph: one may choose the type of a terminal a once the type of every $b \in T_0$ such that $a \rightsquigarrow^{+}_{G_0} b$ has already been treated (bottom-up approach).

We are currently discussing some other improvements to our tool. As a short term optimisation, we intend to allow a global design of an XML type before using Algorithm 3. By using a graphical interface, the user can, in fact, transform the dependency graph into a tree. In this way, he establishes a choice before entering in the details of each local type. For instance, in Example 7, Figure 2, terminal *team* has two parents, meaning that it can take part in the definition of *researcher* or *laboratory*. However, a user probably wants to choose one of these possibilities and not use *team* in both definitions (which is allowed by our algorithm), to avoid redundancy. By deleting the arc between *lab* and *team*, the user fixes, beforehand, his choice, avoiding useless computation. We currently consider the existence of an ontology alignment from which we can obtain a translation table for different terminals used in the grammars. A long term improvement concerns the methods to automatically generate this table. We can fin in [Gu et al., 2008] some initial clues to deal with this aspect.

Now we prove that grammars obtained by Algorithm 3 are strongly non-recursive LTGs.

Lemma 4. $\forall a, b \in T_0,\ a \rightsquigarrow_G b \implies a \rightsquigarrow^{+}_{G_0} b$. □

Proof: Suppose $a \rightsquigarrow_G b$. Then there exist rules $\hat{a} \to a[R],\ \hat{b} \to b[R'] \in P$ s.t. $\hat{b} \in NT(R)$. Therefore there exist $A_1 \to a[R_1], \ldots, A_n \to a[R_n] \in P_0$ and

$B_1 \to b[R'_1], \ldots, B_k \to b[R'_k] \in P_0$ s.t. $\hat{a} = \{A_1, \ldots, A_n\}$ and $\hat{b} = \{B_1, \ldots, B_k\}$. To build rule $\hat{a} \to a[R]$, there are two possibilities:
Case (i): $R = (\hat{R}_1 | \cdots | \hat{R}_n)$. Since $\hat{b} \in NT(R)$, there exists $j \in \{1, \ldots, n\}$ s.t. $\hat{b} \in NT(\hat{R}_j)$. Then $\exists p \in \{1, \ldots, k\}$, $B_p \in NT(R_j)$. Finally we have $A_j \to a[R_j] \in P_0$, $B_p \to b[R'_p] \in P_0$, and $B_p \in NT(R_j)$. Consequently $a \rightsquigarrow_{G_0} b$.
Case (ii) $\forall C \in NT(R)$, $C = \hat{c} \wedge a \rightsquigarrow^+_{G_0} c$. Since $\hat{b} \in NT(R)$, we have $a \rightsquigarrow^+_{G_0} b$.
□

Theorem 3. *The grammar returned by Algorithm 3 is a strongly non-recursive LTG in normal form.* □

Proof: In Algorithm 3, for each $a \in T_0$ we define only one rule in P that generates a: it is the rule $\hat{a} \to a[R]$. Therefore there are no competing non-terminal in G, then G is an LTG. On the other hand, suppose that G is not in normal form, i.e. $\exists\, b \in T_0$, $b \neq a \wedge \hat{b} \to b[R'] \in P \wedge \hat{b} = \hat{a} = \{C_1, \ldots, C_n\}$. Then for all $i \in \{1, \ldots, n\}$, there exist two rules $C_i \to a[R_i]$, $C_i \to b[R'_i] \in P_0$, which is impossible since G_0 is in normal form.
Suppose that G is not strongly non-recursive. Then $\exists a \in T_0$, $a \rightsquigarrow^+_G a$. From Lemma 4, we get $a \rightsquigarrow^+_{G_0} a$ which is impossible since G_0 is strongly non-recursive. □

7 Algorithm Analysis and Experiments

Algorithm 1 is polynomial. Recall that, for each original rule, the algorithm verifies whether there exist competing non-terminals by visiting the remaining rules and merges rules where competition is detected. The algorithm proceeds by traversing each regular expression of the rules obtained from this first step, replacing each non-terminal by the associated equivalent class. Thus, in the worst case, Algorithm 1 runs in time $O(N^2 + N.l)$, where N is the number of production rules and l is the maximal number of non-terminals in a regular expression.

In the following we consider an example which illustrates the worst case for Algorithm 2. Indeed, in the worst case, the number of non-terminals of the STTG returned by Algorithm 2 is exponential in the number of the non-terminals of the initial grammar G_0. However, in real cases, it is difficult to find such a situation.

Example 8. Let us consider a grammar where the production rules have the following form:

1	$S \to s[A_1 \| B_1 \| C_1]$	2	$A_1 \to a[A_2 \mid \epsilon]$
3	$A_2 \to a[A_1]$	4	$B_1 \to a[B_2 \mid \epsilon]$
5	$B_2 \to a[B_3]$	6	$B_3 \to a[B_1]$
7	$C_1 \to a[C_2 \mid \epsilon]$	8	$C_2 \to a[C_3]$
9	$C_3 \to a[C_4]$	10	$C_4 \to a[C_1]$

Clearly, this grammar is not a STTG, since in the first rule we have a regular expression with three competing non-terminals. By using Algorithm 2, the first rule is transformed into $\{S\} \to s[\{A_1, B_1, C_1\} \mid \{A_1, B_1, C_1\} \mid \{A_1, B_1, C_1\}]$. Trying to merge rules 2, 5 and 7 we find $\{A_1, B_1, C_1\} \to a[\sigma(A_2 | B_2 | C_2)]$ where the regular expression $A_2|B_2|C_2$ has also three competing non-terminals that should be *put together* to give a new non-terminal $\{A_2, B_2, C_2\}$. The reasoning goes on in the same way to obtain $\{A_1, B_3, C_3\}$, $\{A_2, B_1, C_4\}$ and so on. The number of non-terminals grows exponentially. □

A prototype tool implementing Algorithm 1 and Algorithm 2 may be downloaded from [Chabin et al.,]. It was developed using the ASF+SDF Meta-Environment [van den Brand et al., 2001] and demonstrates the feasibility of our approach. To study the scalability of our method, Algorithm 1 has been implemented in Java. In this context, we have also developed some tools for dealing with tree grammars. Thus, given a tree grammar G, we can test whether it is in reduced form or in normal form or whether the grammar is already an LTG.

Table 1 shows the results of some experiments obtained by the Java implementation of Algorithm 1. Our experiments were done on an Intel Dual Core P8700, 2.53GHz with 2GB of memory. To perform these tests we have developed an RTG generator[9]: from n_1 terminals, we generate n_2 rules (or non-terminals, since each non-terminal is associated to one rule). When $n_1 \leq n_2$ the generated RTG has n_1 non-terminals that are not competing and $n_2 - n_1$ competing non-terminals. When $n_1 = n_2$ the generated grammar is an LTG. The regular expression of each rule has the form $E_1 \mid \cdots \mid E_n$ where each E_i is a conjunction of k non-terminals randomly generated and randomly adorned by $*$ or ?. The values of n and k are also chosen at random, limited by given thresholds.

Example number	**Number of terminals**	**Number of non-terminals**	**Runtime for Algorithm 1 (ms) LTG transformation**
1	250	300	349
2	250	500	536
3	250	1000	2316
4	1000	1000	1956
5	1000	2000	8522
6	2000	2000	8093
7	1000	4000	36349
8	1000	8000	163236
9	1000	10000	265414

Table 1. Runtime for Algorithm 1 in milliseconds.

From Table 1 it is possible to see that time execution increases polynomially according to the number of non-terminals (roughly, when the number of non-terminals is multiplied by 2, time is multiplied by 4). We can also notice that when changing only the number of terminals, the impact of competition on execution time is not very important. For instance, lines 2 and 3 show grammars having the same set of terminals but a different number of non-terminals (or production rules). Clearly, grammar on line 3 needs much more time to be proceeded. However, lines 3 and 4 (or lines 5 and 6) show two grammars having the

[9] Available at `http://www.univ-orleans.fr/lifo/Members/chabin/logiciels.html`

same set of non-terminals but a different number of terminals. In our examples, this fact indicates that grammar of line 3 has more competing non-terminals than grammar on line 4. Notice however both cases are treated in approximately the same time.

We have just developed prototypes over which we can evaluate the initial interest of our proposal. Even if some software engineer effort is necessary to transform these prototypes into a software tool, the performed tests on our Java implementation show that we can expect a good performance of our method in the construction of tools for manipulating and integrating XML types.

8 Related Work

XML type evolution receives more and more attention nowadays, and questions such as incremental re-validation ([Guerrini et al., 2005]), document correction w.r.t type evolution ([Bouchou et al., 2006]) or the impact of the evolution on queries ([Genevès et al., 2009,Moro et al., 2007]) are some of the studied aspects. However, data administrators still need simple tools for aiding and guiding them in the evolution and construction of XML types, particularly when information integration is needed. This paper aims to respond to this demand.

Algorithms 1 and 2 allow the conservative evolution of schemas. Our work complements the proposals in [Bouchou et al., 2009,da Luz et al., 2007], since we consider not only DTD but also XSD, and adopts a global approach where all the tree grammar is taken into account as a whole. Our algorithms are inspired in some grammar inference methods (such as those dealing with ranked tree languages in [Besombes and Marion, 2003,Besombes and Marion, 2006]) that return a tree grammar or a tree automaton from a set of positive examples (see [Angluin, 1992,Sakakibara, 1997] for surveys). Our method deals with unranked trees, starts from a given RTG G_0 (representing a set of positive examples) and finds the least LTG or STTG that contains $L(G_0)$. As we consider an initial tree grammar we are not exactly inserted in the learning domain, but their methods inspire us and give us tools to solve our problem, namely, the evolution of a original schema (and not the extraction of a new schema).

In [Garofalakis et al., 2000,Bex et al., 2006,Bex et al., 2007] we find examples of work on XML schema inference . In [Bex et al., 2006] DTD inference consists in an inference of regular expressions from positive examples. As the seminal result from Gold [Gold, 1967] shows that the class of all regular expressions cannot be learnt from positive examples, [Bex et al., 2006] identifies classes of regular expressions that can be efficiently learnt. Their method is extended to deal with XMLSchema (XSD) in [Bex et al., 2007].

The approach in [Abiteboul et al., 2009] can be seen as the inverse of ours. Let us suppose a library consortium example. Their approach focus on defining the subtypes corresponding to each library supposing that a target global type of a distributed XML document is given. Our approach proposes to find the integration of different library subtypes by finding the least library type capable of verifying all library subtypes.

The usability of our method is twofold: as a theoretical tool, it can help answering the decision problem announced in [Martens et al., 2006]; as an applied tool, it can easily be adapted to the context of digital libraries, web services, etc. In [Martens et al., 2006], the authors are interested in analyzing the actual expressive power of XSD. With some non-trivial amount of work, part of their theorem proofs can be used to produce an algorithm similar to ours. Indeed, in [Gelade et al., 2010] (a work simultaneous to ours in [Chabin et al., 2010]), the authors decide to revisit their results in [Martens et al., 2006] to define approximations of the union (intersection and complement) of XSD schemas. Our methods are similar, but our proposal works directly over grammars, allowing the implementation of a user friendly tool easily extended to an interactive mode, while results in [Gelade et al., 2010] are based on the construction of type automata.

A large amount of work have been done on the subject of matching XML schemas or ontology alignment ([Shvaiko and Euzenat, 2005] as a survey) and we can find a certain number of automatic tools for generating schema matchings such as SAMBO [Lambrix et al., 2008] or COMA++ [Maßmann et al., 2011]. Generally, a schema matching gives a set of edges, or correspondences, between pairs of elements, that can be stored into translation tables (a kind of dictionary). An important perspective of our work concerns the generation of translation tables by using methods such as the one proposed in [Gu et al., 2008], since, until now these semantics aspects have been considered as a 'given information'.

An increasing demand on data exchange and on constraint validation have motivated us to work on the generation of a new set of constraints from different local sets of type or integrity restrictions. This new set of constraints should keep all non contradictory local restrictions. The type evolution proposed here is well adapted to our proposes and it seems possible to combine it with an XFD filter, as the one in [Amavi and Halfeld Ferrari, 2012], in order to obtain a (general) set of constraints allowing interoperability. This paper focus only on schema constraints and proposes an extension that guarantees the validity of any local document. Thus, as explained in the introduction, our approach is very interesting when local systems $I_1, \ldots, I_n$, inter-operate with a global system I which should receive information from any local source (or format) and also ensure type constraint validation.

9 Conclusion

XML data and types age or need to be adapted to evolving environments. Different type evolution methods propose to trigger document updates in order to assure document validity. Conservative type evolution is an easy-to-handle evolution method that guarantees validity after a type modification.

This paper proposes conservative evolution algorithms that compute a local or single-type grammar which extends minimally a given original regular grammar. The paper proves the correctness and the minimality of the generated grammars. An interactive approach for aiding in the construction of new schemas is also introduced. Our three algorithms represent the basis for the construction of a platform whose goal is to support administration needs in terms of maintenance, evolution and integration of XML types.

We are currently working on improving and extending our approach to solve other questions related to type compatibility and evolution. Except for the terminal translation table, our approach is inherently syntactic: only structural aspects of XML documents are considered and our new grammars are built by syntactic manipulation of the original production rules. However, schemas can be more expressive than DTD and XSD, associated to integrity constraints (as in [Bouchou et al., 2012]) or expressed by a semantically richer data model (as in [Wu et al., 2001]).

In [Amavi and Halfeld Ferrari, 2012] we find an algorithm that computes, from given local sets of XFD, the biggest set of XFD that does not violate any local document. This algorithm is a first step towards an extension of our approach which will take into account integrity constraints. Notice that we understand this extension by the implementation of different procedures, one for each kind of integrity constraints. In other words, by using the uniform formalism proposed in [Bouchou et al., 2012] for expressing integrity constraints on XML documents and following the ideas exposed in [Amavi and Halfeld Ferrari, 2012], we can build sets of integrity constraints (inclusion dependencies, keys, etc) adapted to our global schema. In this way, the evolution of richer schema would correspond to the parallel evolution of different sets of constraints.

We intend not only to extend our work in these directions but also to enrich our platform with tools (such as the one proposed in [Amavi et al., 2011]) for comparing or classifying types with respect to a 'type distance' capable of choosing the closest type for a given document (as discussed, for instance, in [Tekli et al., 2011,Bertino et al., 2008]). Interesting theoretical and practical problems are related to all these perspectives.

References

[Abiteboul et al., 2009] Abiteboul, S., Gottlob, G., and Manna, M. (2009). Distributed xml design. In *PODS '09: Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 247–258. ACM.

[Amavi et al., 2011] Amavi, J., Chabin, J., Halfeld Ferrari, M., and Réty, P. (2011). Minimal Tree Language Extensions: A Keystone of XML Type Compatibility and Evolution. In Springer-Verlag, editor, *Proceedings of the Int. Conf. on Implementation and Application of Automata, CIAA*, volume 6807 of *LNCS*, pages 30–41.

[Amavi and Halfeld Ferrari, 2012] Amavi, J. and Halfeld Ferrari, M. (2012). An axiom system for XML and an algorithm for filtering XFD (also a poster published in sac'2013). Technical Report RR-2012-03, LIFO/Université d'Orléans.

[Angluin, 1992] Angluin, D. (1992). Computational learning theory: survey and selected bibliography. In *STOC '92: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 351–369, New York, NY, USA. ACM.

[Bertino et al., 2008] Bertino, E., Giovanna Guerrini, G., and Mesiti, M. (2008). Measuring the structural similarity among XML documents and dtds. *J. Intell. Inf. Syst.*, 30:55–92.

[Besombes and Marion, 2003] Besombes, J. and Marion, J.-Y. (2003). Apprentissage des langages réguliers d'arbres et applications. *Traitement automatique de langues*, 44(1):121–153.

[Besombes and Marion, 2006] Besombes, J. and Marion, J.-Y. (2006). Learning tree languages from positive examples and membership queries. *Theoretical Computer Science*.

[Bex et al., 2006] Bex, G. J., Neven, F., Schwentick, T., and Tuyls, K. (2006). Inference of concise DTDs from XML data. In *VLDB*, pages 115–126.

[Bex et al., 2007] Bex, G. J., Neven, F., and Vansummeren, S. (2007). Inferring xml schema definitions from xml data. In *VLDB*, pages 998–1009.

[Bouchou et al., 2006] Bouchou, B., Cheriat, A., Halfeld Ferrari, M., and Savary, A. (2006). XML document correction: Incremental approach activated by schema validation. In *International Database Engeneering and Applications Symposium (IDEAS)*.

[Bouchou et al., 2009] Bouchou, B., Duarte, D., Halfeld Ferrari, M., and Musicante, M. A. (2009). Extending XML Types Using Updates. In Hung, D., editor, *Services and Business Computing Solutions with XML: Applications for Quality Management and Best Processes*, pages 1–21. IGI Global.

[Bouchou et al., 2012] Bouchou, B., Halfeld Ferrari Alves, M., and de Lima, M. A. V. (2012). A grammarware for the incremental validation of integrity constraints on xml documents under multiple updates. *T. Large-Scale Data- and Knowledge-Centered Systems*, 6:167–197.

[Chabin et al.,] Chabin, J., Halfeld Ferrari, M., Musicante, M. A., and Réty, P. A software to transform a RTG into a LTG or a STTG. `http://www.univ-orleans.fr/lifo/Members/rety/logiciels/RTGal - gorithms.html`.

[Chabin et al., 2010] Chabin, J., Halfeld Ferrari, M., Musicante, M. A., and Réty, P. (2010). Minimal Tree Language Extensions: A Keystone of XML Type Compatibility and Evolution. In Springer-Verlag, editor, *Proceedings of Int. Colloquium on Theoretical Aspects of Computing, ICTAC*, volume 6255 of *LNCS*, pages 60–75.

[da Luz et al., 2007] da Luz, R., Halfeld Ferrari, M., and Musicante, M. A. (2007). Regular expression transformations to extend regular languages (with application to a datalog XML schema validator). *Journal of Algorithms*, 62(3-4):148–167.

[Garofalakis et al., 2000] Garofalakis, M. N., Gionis, A., Rastogi, R., Seshadri, S., and Shim, K. (2000). Xtract: A system for extracting document type descriptors from xml documents. In *SIGMOD Conference*, pages 165–176.

[Gelade et al., 2010] Gelade, W., Idziaszek, T., Martens, W., and Neven, F. (2010). Simplifying xml schema: single-type approximations of regular tree languages. In *ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS*, pages 251–260.

[Genevès et al., 2009] Genevès, P., Layaïda, N., and Quint, V. (2009). Identifying query incompatibilities with evolving xml schemas. *SIGPLAN Not.*, 44:221–230.

[Gold, 1967] Gold, E. M. (1967). Language identification in the limit. *Information and Control*, 10(5):447–474.

[Gu et al., 2008] Gu, J., Xu, B., and Chen, X. (2008). An XML query rewriting mechanism with multiple ontologies integration based on complex semantic mapping. *Information Fusion*, 9(4):512–522.

[Guerrini et al., 2005] Guerrini, G., Mesiti, M., and Rossi, D. (2005). Impact of XML schema evolution on valid documents. In *WIDM'05: Proceedings of the 7th annual ACM international workshop on Web information and data management*, pages 39–44, New York, NY, USA. ACM Press.

[Lambrix et al., 2008] Lambrix, P., Tan, H., and Liu, Q. (2008). Sambo and sambodtf results for the ontology alignment evaluation initiative 2008. In *OM*.

[Mani and Lee, 2002] Mani, M. and Lee, D. (2002). XML to Relational Conversion using Theory of Regular Tree Grammars. In *In VLDB Workshop on EEXTT*, pages 81–103. Springer.

[Martens et al., 2006] Martens, W., Neven, F., Schwentick, T., and Bex, G. J. (2006). Expressiveness and complexity of XML schema. *ACM Trans. Database Syst.*, 31(3):770–813.

[Maßmann et al., 2011] Maßmann, S., Raunich, S., Aumüller, D., Arnold, P., and Rahm, E. (2011). Evolution of the coma match system. In *OM*.

[Moro et al., 2007] Moro, M. M., Malaika, S., and Lim, L. (2007). Preserving xml queries during schema evolution. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 1341–1342. ACM.

[Murata et al., 2005] Murata, M., Lee, D., Mani, M., and Kawaguchi, K. (2005). Taxonomy of XML schema languages using formal language theory. *ACM Trans. Inter. Tech.*, 5(4):660–704.

[Papakonstantinou and Vianu, 2000] Papakonstantinou, Y. and Vianu, V. (2000). DTD inference for views of XML data. In *PODS - Symposium on Principles of Database System*, pages 35–46. ACM Press.

[Sakakibara, 1997] Sakakibara, Y. (1997). Recent advances of grammatical inference. *Theor. Comput. Sci.*, 185(1):15–45.

[Shvaiko and Euzenat, 2005] Shvaiko, P. and Euzenat, J. (2005). A survey of schema-based matching approaches. pages 146–171.

[Tekli et al., 2011] Tekli, J., Chbeir, R., Traina, A. J. M., and Traina, C. (2011). XML document-grammar comparison: related problems and applications. *Central European Journal of Computer Science*, 1(1):117–136.

[van den Brand et al., 2001] van den Brand, M., Heering, J., de Jong, H., de Jonge, M., Kuipers, T., Klint, P., Moonen, L., Olivier, P., Scheerder, J., Vinju, J., Visser, E., and Visser, J. (2001). The ASF+SDF meta-environment: a component-based language development environment. *Electronic Notes in Theoretical Computer Science*, 44(2).

[Wu et al., 2001] Wu, X., Ling, T. W., Lee, M.-L., and Dobbie, G. (2001). Designing semistructured databases using ora-ss model. In *WISE (1)*, pages 171–.

A Appendix: Proof of Theorem 1

We start by proving that G (the grammar obtained by Algorithm 1) is an LTG. Then we show that Algorithm 1 proposes a grammar which generates a language containing $L(G_0)$.

Lemma 5. *G is an LTG.* $\square$

Proof : By contradiction. Suppose $\hat{A} = \{A_1, ..., A_n\}$ and $\hat{B} = \{B_1, ..., B_k\}$ are competing in G, we have $\hat{A} \to a[R]$ and $\hat{B} \to a[R']$ in P. By construction of the rules of G (Algorithm 1) we must have in P_0 the following rules: $A_1 \to a[R_1], \cdots, A_n \to a[R_n], B_1 \to a[R'_1], \cdots, B_k \to a[R'_k]$. We deduce that $A_1, \cdots, A_n, B_1, \cdots, B_k$ are competing in G_0. Thus $\hat{A} = \hat{B}$. This is impossible since by definition, competing non-terminals are different. As, by construction, there is one rule in P for each element of $N = N_0/_{\|}$, G is in normal form. $\square$

The following lemma shows that the algorithm preserves the trees generated by the grammar G_0.

Lemma 6. *If $X \to^*_{G_0} t_0$ then $\hat{X} \to^*_G t_0$.* $\square$

Proof : By induction on the length of $X \to^*_{G_0} t_0 = a(w)$. Let us consider the first step of the derivation $X \to_{G_0} a[R_X]$ and $\exists U \in L(R_X)$, $U \to^*_{G_0} w$. By construction of G, $\hat{X}$ is in N and $\hat{X} \to a[R]$ is in P with $R = \hat{R_1} \mid \cdots \mid \hat{R_n}$, R_X is one of R_i so $\hat{U} \in L(R)$. Then $\hat{X} \to_G a[R]$ and by induction hypothesis, $\hat{U} \to^*_G w$ therefore $\hat{X} \to^*_G a(w) = t_0$. $\square$

We can now begin to show the relationship between the original language, as described by the grammar G_0 and the language generated by the grammar G obtained by the Algorithm 1.

Lemma 7. ***(A)*** *$L(G_0) \subseteq L(G)$.* ***(B)*** *If $X \to^*_{G_0} t$ and $\hat{X} \to^*_G t'$ then $t(\epsilon) = t'(\epsilon)$ (i.e. t and t' have the same top symbol).* $\square$

Proof : Item **(A)** is an immediate consequence of Lemma 6. As G_0 is in normal form, we have only one rule in G_0 of the form $X \to a[R_X]$, *i.e.*, the terms generated by X in G_0 have a as top symbol. Non-terminal X is in $\hat{X}$. From Lemma 5, we know that $\hat{X} \to a[R]$ is the unique rule in G whose left-hand side is $\hat{X}$. $\square$

From Example 4, we can, for instance, derive $Instrs \to^*_G ins(step, step)$ and $Instrs \to^*_G ins(number, step)$. Different terms with the same top label. Now, the next lemma states that the resulting grammar G does not introduce any new terminal symbol at the root of the generated trees.

Lemma 8. *$\forall t \in L(G)$, $\exists t' \in L(G_0)$ such that $t'(\epsilon) = t(\epsilon)$.*

Proof : Let $t = a(w) \in L(G)$. Then there exists a rule $\{A_1, \cdots, A_n\} \to a[R]$ in P with $\{A_1, \cdots, A_n\} \in S$ (a start symbol in G). By definition of S, $\exists i$ such that $A_i \in S_0$ and $A_i \to^*_{G_0} a(w') = t'$ because G_0 is in the reduced form. So $t' \in L(G_0)$ and t and t' have the same root symbol. $\square$

Next, we show that for every subtree generated by G, its root appears in at least one subtree of the language generated by G_0 (recall that $w'(\epsilon) = w(\epsilon)$ means that forests w' and w have the same tuple of top-symbols):

Lemma 9. *If $t \in ST(L(G))$, such that $t = a(w)$, then, $\exists t' \in ST(L(G_0))$, $t' = a(w') \wedge w'(\epsilon) = w(\epsilon)$.*

Proof : Let $t \in ST(L(G))$ such that $t = a(w)$. There exists $\hat{A}_1 \to a[R] \in P$ such that $\hat{A}_1 \to a(U)$, $U \in L(R)$, $U \to^*_G w$. By construction, $R = \hat{R_1}| \dots |\hat{R_n}$, and $\forall i$, $\exists A_i \in N_0$, $A_i \to a[R_i] \in P_0 \wedge A_i \in \hat{A}_1$. Therefore there exists j such that $U \in L(\hat{R_j})$. Consider $A_j \to a[R_j] \in P_0$. There exists $U' \in L(R_j)$ such that $\hat{U'} = U$. Now, since G_0 is in reduced form, there exists a forest w' such that $U' \to^*_{G_0} w'$. Consequently $A_j \to_{G_0} a(U') \to^*_{G_0} a(w')$. Let $t' = a(w')$. Since G_0 is in reduced form, the rule $A_j \to a[R_j]$ is reachable in G_0, then $t' \in ST(L(G_0))$. From Lemma 7, since $\hat{U'} = U$, we have $w(\epsilon) = w'(\epsilon)$. $\square$

As an illustration of Lemma 9, we observe that from the grammars of Example 4, given the tree *r(is(ing(name,unit,qty)), r(req(item),ing(name,unit,qty), ins(step,step)), ins(step))* from $L(G)$, for its sub-tree t = *r(req(item),ing(name, unit,qty),ins(step,step))*= $r(w) \in ST(L(G))$, we have *t'* = *r(req(item),ing(name, qty,unit),ins(number,step))*= $r(w') \in ST(L(G_0))$, $t(\epsilon) = t'(\epsilon)$ and $w(\epsilon) = w'(\epsilon)$.

Now we need some properties of local tree languages. The following lemma states that the type of the subtrees of a tree node is determined by the label of its node (i.e. the type of each node is *locally* defined). Recall that $ST(L)$ is the set of sub-trees of elements of L.

Lemma 10 (See [Papakonstantinou and Vianu, 2000] (Lemma 2.10)).

Let L be a local tree language (LTL). Then, for each $t \in ST(L)$, each $t' \in L$ and each $p' \in Pos(t')$, we have that : $\quad t(\epsilon) = t'(p') \implies t'[p' \leftarrow t] \in L$. $\square$

We also need a weaker version of the previous lemma:

Corollary 1. *Let L be a local tree language (LTL). Then, for each $t, t' \in ST(L)$, and each $p' \in Pos(t')$, we have that :* $\quad t(\epsilon) = t'(p') \implies t'[p' \leftarrow t] \in ST(L)$. $\square$

In practical terms, Corollary 1 gives us a rule of thumb on how to "complete" a regular language in order to obtain a local tree language. For instance, let $L = \{f(a(b), c), f(a(c), b)\}$ be a regular language. According to Corollary 1, we know that L is not LTL and that the least local tree language L' containing L contains all trees where a has c as a child together with all trees where a has b as a child. In other words, $L' = \{f(a(b), c), f(a(c), b), f(a(c), c), f(a(b), b)\}$.

Now, we can prove that the algorithm just adds what is necessary to get an LTL (and not more), in other words, that $L(G)$ is the least local tree language that contains $L(G_0)$. This is done in two stages: first for subtrees, then for trees.

Lemma 11. *Let L' be an LTL such that $L(G_0) \subseteq L'$. Then $ST(L(G)) \subseteq ST(L')$.*

Proof : By structural induction on the trees in $ST(L(G))$. Let $t = a(w) \in ST(L(G))$. From Lemma 9, there exists $t' \in ST(L(G_0))$ such that $t' = a(w') \wedge w'(\epsilon) = w(\epsilon)$. Since $L(G_0) \subseteq L'$, we have $ST(L(G_0)) \subseteq ST(L')$, then $t' \in ST(L')$. If w is an empty forest (i.e. the empty tuple), w' is also empty, therefore $t = t' \in ST(L')$. Otherwise, let us write $w = (a_1(w_1), \ldots, a_n(w_n))$ and $w' = (a_1(w'_1), \ldots, a_n(w'_n))$ (since $w(\epsilon) = w'(\epsilon)$, w and w' have the same top symbols). Since $t = a(w) \in ST(L(G))$, for each $j \in \{1, \ldots, n\}$, $a_j(w_j) \in ST(L(G))$, then by induction hypothesis $a_j(w_j) \in ST(L')$. L' is an LTL, and for each j we have : $a_j(w_j) \in ST(L')$, $t' \in ST(L')$, $(a_j(w_j))(\epsilon) = a_j = t'(j)$. By applying Corollary 1 n times, we get $t'[1 \leftarrow a_1(w_1)] \ldots [n \leftarrow a_n(w_n)] = t \in ST(L')$. □

Theorem 4. *Let L' be an LTL such that $L' \supseteq L(G_0)$. Then $L(G) \subseteq L'$.*

Proof : Let $t \in L(G)$. Then $t \in ST(L(G))$. From Lemma 11, $t \in ST(L')$. On the other hand, from Lemma 8, there exists $t' \in L(G_0)$ such that $t'(\epsilon) = t(\epsilon)$. Then $t' \in L'$. From Lemma 10, $t'[\epsilon \leftarrow t] = t \in L'$. □

This result ensures that the grammar G of Example 4 generates the least LTL that contains $L(G_0)$.

B Appendix: Proof of Theorem 2

The proof somehow looks like the proof concerning the transformation of an RTG into an LTG (Section 4). However it is more complicate since in a STTL (and unlike what happens in an LTL), the confusion between $t|_p = a(w)$ and $t'|_{p'} = a(w')$ should be done only if position p in t has been generated by the same production rule as position p' in t', i.e. the symbols occurring in t and t' along the paths going from root to p (resp. p' in t') are the same. This is why we introduce notation $path(t, p)$ to denote these symbols (Definition 7).

Lemma 12. *Let $\chi \in \mathcal{P}(N_0)$ and $A, B \in \chi$. Then $\hat{A}^\chi$ and $\hat{B}^\chi$ are not competing in P.* □

Proof: By contradiction. Suppose $\hat{A}^\chi$ and $\hat{B}^\chi$ are competing in P. Then there exist $\hat{A}^\chi \to a[R_1] \in P$ and $\hat{B}^\chi \to a[R_2] \in P$. From the construction of P, there exist $C \in \hat{A}^\chi$ (then $C \parallel_\chi A$) and $C \to a[R'_1] \in P_0$, as well as $D \in \hat{B}^\chi$ (then $D \parallel_\chi B$) and $D \to a[R'_2] \in P_0$. Thus, $C \parallel_\chi D$ and by transitivity $A \parallel_\chi B$, then $\hat{A}^\chi = \hat{B}^\chi$, which is impossible since competing non-terminals are not equal. □

Example 9. Consider the grammar of Example 6.
Let $\chi = \{\mathit{Frame1}, \mathit{Frame2}, \mathit{Background}\}$. The equivalence classes induced by $\|_\chi$ are $\widehat{\mathit{Frame1}}^\chi = \widehat{\mathit{Frame2}}^\chi = \{\mathit{Frame1}, \mathit{Frame2}\}$; $\widehat{\mathit{Background}}^\chi = \{\mathit{Background}\}$; which are non-competing non-terminals in P. $\square$

Lemma 13. *$G = (N, T, S, P)$ is a STTG.* $\square$
Proof: (1) There is no regular expression in P containing competing non-terminals: If $\hat{A}^{S_0}, \hat{B}^{S_0}$ are in S, then $A, B \in S_0$. From Lemma 12, $\hat{A}^{S_0}$ and $\hat{B}^{S_0}$ are not competing in P. For any regular expression R, let $\hat{A}^{N(R)}, \hat{B}^{N(R)} \in N(\sigma_{N(R)}(R))$. Thus, $A, B \in N(R)$. From Lemma 12, $\hat{A}^{N(R)}$ and $\hat{B}^{N(R)}$ are not competing in P. (2) G is in normal form: As for each A_i there is at most one rule in P_0 whose left-hand-side is A_i (because G_0 is in normal form), there is at most one rule in P whose left-hand-side is $\{A_1, \dots, A_n\}$. $\square$

The next lemma establishes the basis for proving that the language generated by G contains the language generated by G_0. It considers the derivation process over G_0 at any step (supposing that this step is represented by a derivation tree t) and proves that, in this case, at the same derivation step over G, we can obtain a tree t' having all the following properties: (i) the set of positions is the same for both trees ($Pos(t) = Pos(t')$); (ii) positions associated to terminal are identical in both trees; (iii) if position p is associated to a non-terminal A in t then position $p \in Pos(t')$ is associated to the equivalence class $\hat{A}^\chi$ for some $\chi \in \mathcal{P}(N_0)$ such that $A \in \chi$.

Lemma 14. *Let $Y \in S_0$. If G_0 derives:*
$t_0 = Y \to \cdots \to t_{n-1} \to_{[p_n,\, A_n \to a_n[R_n]]} t_n$ *then G can derive:* $t'_0 = \hat{Y}^{S_0} \to \cdots \to t'_{n-1} \to_{[p_n,\, \hat{A_n}^{\chi_n} \to a_n[\sigma_{N(R_n|\cdots)}(R_n|\cdots)]]} t'_n$
s.t. $\forall i \in \{0, \dots, n\}, Pos(t'_i) = Pos(t_i) \wedge$
$\forall p \in Pos(t_i)$*:* $(t_i(p) \in T_0 \implies t'_i(p) = t_i(p)) \wedge$
$(t_i(p) = A \in N_0 \implies \exists \chi \in \mathcal{P}(N_0), A \in \chi \wedge t'_i(p) = \hat{A}^\chi)$ $\square$
Proof: See [Chabin et al., 2010].

The following corollary proves that the language of the new grammar G, proposed by Algorithm 2, contains the original language of G_0.

Corollary 2. *$L(G_0) \subseteq L(G)$.* $\square$

In the rest of this section we work on proving that $L(G)$ is the least STTL that contains $L(G_0)$. To prove this property, we first need to prove some properties over STTLs. We start by considering paths in a tree. We are interested by paths (sequence of labels) starting on the root and achieving a given position p in a tree t. For example, $path(a(b, c(d)), 1) = a.c$.

Definition 7 (Path in a tree t to a position p). *Let t be a tree and $p \in Pos(t)$, then $path(t, p)$ is recursively defined by:* (1) $path(t, \epsilon) = t(\epsilon)$ *and* (2) $path(t, p.i) = path(t, p).t(p.i)$ *where $i \in I\!N$.* $\square$

Given a STTG G, let us consider the derivation process of two trees t and t' belonging to $L(G)$. The following lemma proves that positions (p in t and p' in t') having identical paths are derived by using the same rules. A consequence of this lemma (when $t' = t$ and $p' = p$) is the well known result about the unicity in the way of deriving a given tree with a STTG [Mani and Lee, 2002].

Lemma 15. *Let G' be a STTG, let $t, t' \in L(G')$.*
Let $X \to^*_{[p_i, rule_{p_i}]} t$ be a derivation of t and $X' \to^*_{[p'_i, rule'_{p'_i}]} t'$ be a derivation of t' by G' (X, X' are start symbols). Then $\forall p \in Pos(t)$, $\forall p' \in Pos(t')$,
$(path(t,p) = path(t',p') \implies rule_p = rule'_{p'})$ □

Proof: Suppose $path(t,p) = path(t',p')$. Then we have $length(p) = length(p')$. The proof is by induction on $length(p)$.

• If $length(p) = 0$, then $p = p' = \epsilon$, and $t(\epsilon) = t'(\epsilon) = a$. Therefore $rule_\epsilon = (X \to a[R])$ and $rule'_\epsilon = (X' \to a[R'])$. If $X \neq X'$ then two start symbols are competing, which is impossible since G' is a STTG. If $X = X'$ and $R \neq R'$ then G' is not in normal form, which contradicts the fact that G' is a STTG.
Therefore $rule_\epsilon = rule'_\epsilon$, then $rule_p = rule'_{p'}$.

• Induction step. Suppose $p = q.k$ and $p' = q'.k'$ ($k, k' \in \mathbb{N}$), and $path(t,p) = path(t',p')$. Then $path(t,q) = path(t',q')$. By induction hypothesis, $rule_q = rule'_{q'} = (X \to a[R])$. There exits $w, w' \in L(R)$ s.t. $w(k) = A$, $w'(k') = A'$ and $rule_p = (A \to b[R_1])$, $rule'_{p'} = (A' \to b[R'_1])$ where $b = t(p) = t'(p')$.
If $A \neq A'$ then $A \in N(R)$ and $A' \in N(R)$ are competing, which is impossible since G' is a STTG. If $A = A'$ and $R_1 \neq R'_1$, then G' is not in normal form, which contradicts the fact that G' is a STTG. Consequently $rule_p = rule'_{p'}$. □

In a STTL, it is possible to permute sub-trees that have the same paths.

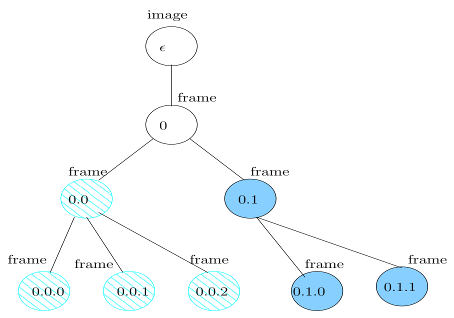
Lemma 16 (Also in [Martens et al., 2006]). *Let G' be a STTG.*
Then, $\forall t, t' \in L(G')$, $\forall p \in Pos(t)$, $\forall p' \in Pos(t')$, $(path(t,p) = path(t',p') \implies t'[p' \leftarrow t|_p] \in L(G'))$ □

Example 10. Let G be the grammar of Example 6. Consider a tree t as shown in Figure 3. The permutation of subtrees $t|_{0.0}$ and $t|_{0.1}$ gives us a new tree t'. Both t and t' are in $L(G)$. □

Definition 8 (Branch derivation). *Let G' be an RTG. A* branch-derivation *is a tuple of production rules*[10] *of G' : $\langle A^1 \to a^1[R^1], \ldots, A^n \to a^n[R^n]\rangle$ s.t. $\forall i \in \{2, \ldots, n\}, A^i \in N(R^{i-1})$.* □

Notice that if A^1 is a start symbol, the branch-derivation represents the derivation of a branch of a tree[11]. This branch contains the terminals $a^1, \ldots, a^n$ (the path to the node having a^n as label). Now, let us prove properties over the grammar G built by Algorithm 2.

[10] Indices are written as super-scripts for coherence with the notations in Lemma 17.
[11] This tree may contain non-terminals.



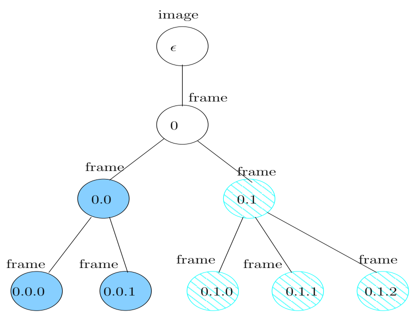


Fig. 3. Trees t and t' with permuted sub-trees.

Lemma 17. *Consider a branch-derivation in* G*:*
$\langle \{A^1_1, \ldots, A^1_{n_1}\} \to a^1 \sigma_{N(R^1_1|\cdots|R^1_{n_1})}[R^1_1|\cdots|R^1_{n_1}], \ldots,$
$\{A^k_1, \ldots, A^k_{n_k}\} \to a^k \sigma_{N(R^k_1|\cdots|R^k_{n_k})}[R^k_1|\cdots|R^k_{n_k}]\rangle$ *and let* $i_k \in \{1, \ldots, n_k\}$*. Then there exists a branch-derivation in* G_0*:* $(A^1_{i_1} \to a^1[R^1_{i_1}], \ldots, A^k_{i_k} \to a^k[R^k_{i_k}])$. □

Proof: By induction on k.

- $k = 1$. There is one step. From Definition 2, $A^k_{i_k} \to a^k[R^k_{i_k}] \in P_0$.

- Induction step. By induction hypothesis applied on the last $k-1$ steps, there exists a branch-derivation in G_0: $(A^2_{i_2} \to a^2[R^2_{i_2}], \ldots, A^k_{i_k} \to a^k[R^k_{i_k}])$. Moreover $\{A^2_1, \ldots, A^2_{n_2}\} \in N(\sigma_{N(R^1_1|\cdots|R^1_{n_1})}(R^1_1|\cdots|R^1_{n_1}))$. Then there exists $i_1 \in \{1, \ldots, n_1\}$ s.t. $A^2_{i_2} \in N(R^1_{i_1})$. And from Definition 2, $A^1_{i_1} \to a^1[R^1_{i_1}] \in P_0$. □

The following example illustrates Lemma 17 and its proof.

Example 11. Let G be the grammar of Example 6 and t the tree of Figure 3. The branch-derivation corresponding to the node 0.0.0 contains the first and the fourth rules of G presented in Example 6 (notice that the fourth rule appears three times). Figure 4 illustrates this branch-derivation on a derivation tree. For instance, the first rule in G is

$$\{Image\} \to image[\{Frame1, Frame2\} \mid \{Background\}.\{Foreground\}] \qquad \text{(R1)}$$

and G_0 has the production rule
$Image \to image[Frame1 \mid Frame2 \mid Background.Foreground]$. Then, the branch-derivation gives us the fourth rule in G, namely:

$$\begin{aligned}\{Frame1, Frame2\} \to\ & frame[\epsilon \\ & \mid \{Frame1, Frame2\}.\{Frame1, Frame2\} \\ & \mid \{Frame1, Frame2\}.\{Frame1, Frame2\}.\{Frame1, Frame2\}].\end{aligned}$$

Notice that the left-hand side $\{Frame1, Frame2\}$ is a non terminal in the right-hand side of (R1). Now, consider each non terminal of G_0 forming the non terminal $\{Frame1, Frame2\}$ in G. Clearly, $Frame1$ is on the right-hand side of the second rule in P_0 while

$Frame2$ is on the right-hand side of the third rule in P_0 (as shown in Example 6). We can observe the same situation for all the rules in the branch-derivation. Thus, as proved in Lemma 17, the branch-derivation in G_0 that corresponds to the one considered in this example is:

$\langle$ $Image \rightarrow image[Frame1 \mid Frame2 \mid Background.Foreground]$
$Frame2 \rightarrow frame[Frame2.Frame2.Frame2 \mid \epsilon]$
$Frame2 \rightarrow frame[Frame2.Frame2.Frame2 \mid \epsilon]$
$Frame2 \rightarrow frame[Frame2.Frame2.Frame2 \mid \epsilon]$ $\rangle$ $\Box$

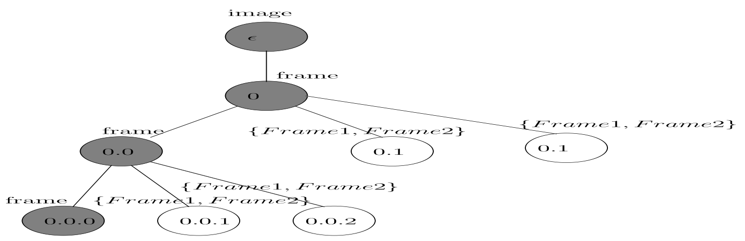


Fig. 4. Derivation tree in G. Grey nodes illustrate a branch-derivation.

The following lemma somehow expresses what the algorithm of Definition 2 does. Given a forest $w = (t_1, \ldots, t_n)$, recall that $w(\epsilon) = \langle t_1(\epsilon), \ldots, t_n(\epsilon)\rangle$, i.e. $w(\epsilon)$ is the tuple of the top symbols of w.

Lemma 18. $\forall t \in L(G), \forall p \in Pos(t)$,
$t|_p = a(w) \implies \exists t' \in L(G_0), \exists p' \in pos(t'), t'|_{p'} = a(w') \wedge w'(\epsilon) = w(\epsilon) \wedge path(t', p') = path(t, p)$. $\Box$

Proof: There exists a branch-derivation in G that derives the position p of t
$(\{A^1_1, \ldots, A^1_{n_1}\} \rightarrow a^1\sigma_{N(R^1_1|\cdots|R^1_{n_1})}[R^1_1|\cdots|R^1_{n_1}], \ldots,$
$\{A^k_1, \ldots, A^k_{n_k}\} \rightarrow a^k\sigma_{N(R^k_1|\cdots|R^k_{n_k})}[R^k_1|\cdots|R^k_{n_k}])$
and $u \in L(\sigma_{N(R^k_1|\cdots|R^k_{n_k})}(R^k_1|\cdots|R^k_{n_k}))$ s.t. $u \rightarrow^*_G w$.
Then there exists i_k s.t. $u \in L(\sigma_{N(R^k_1|\cdots|R^k_{n_k})}(R^k_{i_k}))$. Thus, there exists $v \in L(R^k_{i_k})$ s.t. $u = \sigma_{N(R^k_1|\cdots|R^k_{n_k})}(v)$. Note that $\forall Y \in N_0$, $\forall \chi \in \mathcal{P}(N_0)$, Y and $\hat{Y}^\chi$ generate the same top-symbol. So u and v generate the same top-symbols. Since G_0 is in reduced form, there exists w' s.t. $v \rightarrow^*_{G_0} w'$, and then $w'(\epsilon) = w(\epsilon)$.
From Lemma 17, there exists a branch-derivation in G_0: $(A^1_{i_1} \rightarrow a^1[R^1_{i_1}], \ldots, A^k_{i_k} \rightarrow a^k[R^k_{i_k}])$. Since G_0 is in reduced form, there exists $t' \in L_{G_0}(A^1_{i_1})$ (i.e. t' is a tree derived from $A^1_{i_1}$ by rules in P_0, and t' contains only terminals), and there exists $p' \in Pos(t')$ s.t. this branch-derivation derives in G_0 the position p' of t'. Since $v \in L(R^k_{i_k})$ and $v \rightarrow^*_{G_0} w'$, one can even choose t' s.t. $t'|_{p'} = a^k(w')$. Since $a^k = a$,

we have $t'|_{p'} = a(w')$. On the other hand, $path(t', p') = a^1 \dots a^k = path(t, p)$. Finally, since $t \in L(G)$, $\{A^1_1, \dots, A^1_{n_1}\} \in S$. Since $A^1_{i_1} \in \{A^1_1, \dots, A^1_{n_1}\}$, from Definition 2 we have $A^1_{i_1} \in S_0$. Therefore $t' \in L(G_0)$. $\square$

Example 12. Let G be the grammar of Example 6 and t the tree of Figure 3. Let $p = 0$. Using the notations of Lemma 18, $t|_0 = frame(w)$ where $w = \langle frame(frame, frame, frame),\ frame(frame, frame)\rangle$. We have $t \notin L(G_0)$. Let $t' = image(frame(frame(frame, frame),\ frame)) \in L(G_0)$ and (with $p' = p = 0$) $t'|_{p'} = frame(w')$ where $w' = \langle frame(frame, frame), frame\rangle$. Thus $w'(\epsilon) = w(\epsilon)$. Note that others $t' \in L(G_0)$ suit as well. $\square$

We end this section by proving that the grammar obtained by our algorithm generates the least STTL which contains $L(G_0)$.

Lemma 19. *Let L' be a STTL s.t. $L(G_0) \subseteq L'$. Let $t \in L(G)$. Then, $\forall p \in Pos(t), \exists t' \in L', \exists p' \in pos(t'), (t'|_{p'} = t|_p \wedge path(t', p') = path(t, p))$.* $\square$

Proof: We define the relation $\sqsupset$ over $Pos(t)$ by $p \sqsupset q \iff \exists i \in \mathbb{N}, p.i = q$. Since $Pos(t)$ is finite, $\sqsupset$ is noetherian. The proof is by noetherian induction on $\sqsupset$. Let $p \in pos(t)$. Let us write $t|_p = a(w)$.
From Lemma 18, we know that: $\exists t' \in L(G_0), \exists p' \in pos(t'), t'|_{p'} = a(w') \wedge w'(\epsilon) = w(\epsilon) \wedge path(t', p') = path(t, p)$. Thus, $t|_p = a(a_1(w_1), \dots, a_n(w_n))$ and $t'|_{p'} = a(a_1(w'_1), \dots, a_n(w'_n))$.
Now let $p \sqsupset p.1$. By induction hypothesis: $\exists t'_1 \in L', \exists p'_1 \in pos(t'_1), t'_1|_{p'_1} = t|_{p.1} = a_1(w_1) \wedge path(t'_1, p'_1) = path(t, p.1)$. Notice that $t'_1 \in L'$, $t' \in L(G_0) \subseteq L'$, and L' is a STTL. Moreover $path(t'_1, p'_1) = path(t, p.1) = path(t, p).a_1 = path(t', p').a_1 = path(t', p'.1)$.
As $path(t'_1, p'_1) = path(t', p'.1)$, from Lemma 16 applied on t'_1 and t', we get $t'[p'.1 \leftarrow t'_1|_{p'_1}] \in L'$. However $(t'[p'.1 \leftarrow t'_1|_{p'_1}])|_{p'} = a(a_1(w_1), a_2(w'_2), \dots, a_n(w'_n))$ and $path(t'[p'.1 \leftarrow t'_1|_{p'_1}], p') = path(t', p') = path(t, p)$. By applying the same reasoning for positions $p.2, \dots, p.n$, we get a tree $t'' \in L'$ such that $t''|_{p'} = t|_p$ and $path(t'', p') = path(t, p)$. $\square$

Corollary 3 (When $p = \epsilon$, and then $p' = \epsilon$). *Let L' be a STTL such that $L' \supseteq L(G_0)$. Then $L(G) \subseteq L'$.* $\square$