RAIRO-Theor. Inf. Appl. 43 (2009) 379–402
DOI: 10.1051/ita/2009006

Available online at:
www.rairo-ita.org

NESTED SIBLING TREE AUTOMATA *

Françoise Gire[1] and Jean-Marc Talbot[2]

Abstract. In the XML standard, data are represented as unranked labeled ordered trees. Regular unranked tree automata provide a useful formalism for the validation of schemas enforcing regular structural constraints on XML documents. However some concrete application contexts need the expression of more general constraints than the regular ones. In this paper we propose a new framework in which context-free style structural constraints can be expressed and validated. This framework is characterized by: (i) the introduction of a new notion of trees, the so-called *typed unranked labeled trees* (*tulab trees* for short) in which each node receives one of three possible types (*up*, *down* or *fix*), and (ii) the definition of a new notion of tree automata, the so-called *nested sibling tulab tree automata*, able to enforce context-free style structural constraints on *tulab* tree languages. During their structural control process, such automata are using visibly pushdown languages of words [R. Alur and P. Madhusudan, Visibly pushdown languages, 36th ACM symposium on Theory of Computing, Chicago, USA (2004) 202–211] on their alphabet of states. We show that the resulting class *NSTL* of *tulab* tree languages recognized by nested sibling *tulab* tree automata is robust, *i.e.* closed under Boolean operations and with decision procedures for the classical membership, emptiness and inclusion problems. We then give three characterizations of *NSTL*: a logical characterization by defining an adequate logic in which *NSTL* happens to coincide with the models of monadic second order sentences; the two other characterizations are using adequate encodings and map together languages of *NSTL* with some regular sets of 3-ary trees or with particular sets of binary trees.

Mathematics Subject Classification. 68Q45, 68P15.

Keywords and phrases. Automata, logic, unranked trees, XML schemas.

* *Work supported by the ANR project ANR-08-DEFIS-04*

[1] Université de Paris I, Centre de Recherche en Informatique Paris Cedex 13, France. gire@univ-paris1.fr

[2] Université de Provence, LIF Marseille France. jean-marc.talbot@lif.univ-mrs.fr

Article published by EDP Sciences

Introduction

The XML standard data exchange format for the Web [6] gave rise to a new interest for tree automata theory. Tree automata provide indeed a useful formalism for validating regular structural constraints against XML documents. They can be used as a toolbox to solve classical decision problems for schemas like the membership problem or the containment and equivalence problems. This explains the actual full expansion of research on tree automata in the XML research area. XML documents are unranked trees, that is, the number of children of a node labeled with a given symbol is not *a priori* bounded [1]. Therefore classical tree automata dealing with ranked labeled trees (in which each node has a fixed number of children) have been extended to deal with the representation of XML data. Research on the corresponding class of automata, the so called unranked tree automata, has been initiated in [7]. Most of known results for ranked labeled tree automata, carries over the class of unranked labeled tree automata [5,12,14]. Indeed the class of regular unranked tree languages, that is, the class of unranked tree languages recognizable by a bottom-up tree automaton, is a robust class: it is closed under boolean operations; membership, emptiness, containment and equivalence problems are decidable for bottom-up tree automata, and solvable in PTIME for the deterministic ones [15].

At the same time several attempts have been proposed to express on XML documents more complex structural constraints than the regular ones in order to fulfill concrete requirements. For example one may want to express a constraint like 'as many nodes with label a than nodes with label b must occur as children of a node with label c'. Such a constraint cannot be captured by an unranked tree automaton and is a particular case of the so-called counting constraints studied in [11,16] where specific automata are defined to deal with them. The resulting class of tree automata is a robust class in the above mentioned sense. However, when counting and regular constraints are mixed together, it has been shown that the resulting class of valid documents lacks some closure properties like the closure under complement.

In this paper, we propose a new framework in which such context-free style structural constraints can be expressed and validated. The idea is to define tree automata in which the structure control process at the tree nodes is performed using a class C of languages more expressive than the class of rational languages used for regular unranked tree automata. However to obtain a robust class of recognized tree languages the robustness of C is essential. The class of visibly pushdown languages recently introduced in [4] has been shown to be a robust subclass of the class of context-free languages. This class is also used in [13] to model the streaming process of XML documents. Visibly pushdown languages are languages of words over pushdown alphabets whose letters receive one of three available types $\{up, down, fix\}$. Roughly speaking these languages are recognized by pushdown automata whose stack can perform three kinds of moves (push a symbol, pop a symbol or do nothing) according to the type of the currently read letter. In this paper we show how the class of visibly pushdown languages can be

fruitfully used to achieve our goal of defining a framework in which context-free style structural constraints can be expressed and validated. This framework is characterized by two main features:

(i) it introduces and deals with a new notion of trees, the so-called *typed unranked labeled trees* (*tulab* trees): these trees are defined on typed tree domains in which each node receives one of three possible types;
(ii) it allows to express context-free style structural constraints through specific tree automata, the so-called *nested sibling* tulab *tree automata*. Such automata recognize a class of *tulab* tree languages that is a robust class, *i.e.* closed under Boolean operations and with decision procedures for the classical membership, emptiness and inclusion problems. They can be therefore used to express complex schemas for XML data and provide a toolbox to solve their associated decision problems.

Related work. In addition to [11,16] already mentioned above, our work can be related with others in the literature. Let us first notice that the kind of context-free constraints our *nested sibling tree automata* are able to express are structural horizontal constraints. Therefore in general these automata are expressively incomparable with classical pushdown tree automata on trees that enforce pushdown constraints along tree paths. Secondly our approach introducing type information on the tree domain can also be related to those of [3] and [2]: in these works the authors study languages of nested trees in which the nesting structure is moved from the alphabet to the underlying input shape as in our work. These languages are a class of graphs that naturally abstract branching behaviors of structured programs so that the problem of branching-time model checking can be phrased as a membership question for such languages. In [3] a class of automata running on such nested trees is defined and has nice closure properties (under union intersection and complement) as well as a decidable model-checking problem although the emptiness problem is undecidable. However these automata are still expressively incomparable to our *nested sibling tree automata* because they also express pushdown constraints along tree paths instead of across horizontal sibling paths. Furthermore they deal with unordered directed acyclic graphs instead of unranked ordered labeled trees.

In [8], an extension of pushdown-stack automata has been proposed, namely *tree automata with one memory*. This extension is twofold: the automata proceed on a tree instead of a word and the (word) stack has been replaced by a memory which is a tree. Although emptiness can be decided for these automata, they are neither closed under intersection, nor under union. A strict subclass of tree automata with one memory is tailored in [10] by adopting a visible-style discipline to manipulate the memory; under this restriction, this class of visibly tree automata with memory is (effectively) closed under Boolean operations. This class can not be directly compared with *nested sibling tree automata* as it deals with ranked trees. However, we will see how the two notions of automata are connected when visibly tree automata with memory run on firstChild-nextSibling encodings of *tulab* trees.

Organization of the paper. The paper is organized as follows: Section 2 is devoted to the introduction of typed unranked labeled trees while the definition of nested sibling *tulab* tree automata is given in Section 3 and illustrated by examples in Section 4. In Section 5 we show that the resulting class *NSTL* of *tulab* tree languages recognized by such automata is closed under Boolean operations and has decision procedures for the classical membership, emptiness and inclusion problems. In Section 6 three characterizations of *NSTL* are given. The first one is a logical characterization in the terms of the Thatcher and Wright theorem [17]: languages of *NSTL* happen to coincide with the models of monadic second order (MSO) sentences written in an adequate logic. The second one shows that, through an encoding of *tulab* trees into 3-ary trees, languages of *NSTL* and regular sets of 3-ary encoding trees are mapped together. The third one shows that, through a standard firstChild-nextSibling encoding of *tulab* trees into binary trees, there is a correspondence between languages of *NSTL* and languages of binary trees accepted by a particular subclass of visibly tree automata with memory. We conclude in Section 7. For sake of place some technical parts of proofs are given in the appendix.

1. Typed unranked labeled trees

Typed tree domain. A *typed tree domain* is a finite subset D of $\mathbb{N}^*$ (where $\mathbb{N}^*$ is the set of finite words over the alphabet $\mathbb{N}$ of positive integers) such that:

- D is a tree domain: the empty word ε belongs to D and if $wi \in D$ with $i \in \mathbb{N}$ and $w \in \mathbb{N}^*$ then both w and wj belong to D for all j, $1 \leq j < i$;
- each element of D has a type *up*, *down* or *fix*, denoted respectively by u, d and f.

As usual, ε of the tree domain D denotes the root of D while wi denotes the ith child of the node w.

Typed unranked labeled trees. A *typed unranked labeled tree* (*tulab tree* for short) T over an alphabet Σ, is a pair $T = (D, \lambda)$ where D is a typed tree domain and λ is a labeling mapping from D to Σ. We denote by $TTree(\Sigma)$, the set of *tulab* trees over Σ. So a *tulab* tree can be viewed as a standard unranked labeled tree with additional type information stored within the domain nodes.

Example 1.1. Let n be a positive integer and let $T_n = (D_n, \lambda_n)$ be the following *tulab* tree where $D_n = \{\varepsilon\} \cup \{i \mid 1 \leq i \leq 2n+1\}$, the type of nodes ε and $n+1$ is f, the type of each node i is u for $1 \leq i \leq n$, and d for $n+2 \leq i \leq 2n+1$, $\lambda_n(\varepsilon) = \lambda_n(n+1) = a$, $\lambda_n(i) = b$ for all nodes $i \neq n+1$ with $1 \leq i \leq 2n+1$. A representation of T_n with $n = 3$ is given below in Figure 1 where three colors are used to represent the three possible node types: black for type u, white for type f and gray for type d.

Let us remark that any *tulab* tree T over Σ can easily be translated into a standard labeled tree T' over another alphabet $\Sigma' = \{x^c, c \in \{u, d, f\} \text{ and } x \in \Sigma\}$ where the upper-script c of a label x^c explicitly gives the type of the corresponding

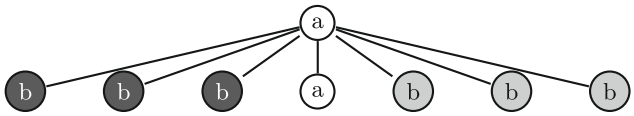

FIGURE 1. The *tulab* tree T_3.

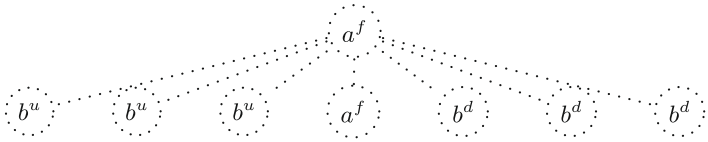

FIGURE 2. The labeled tree T_3'.

underlying node in T (see Fig. 2). Note that the main difference between T and T' is that, in T, the type information is directly stored within the domain nodes and therefore a same letter x can label nodes with different types (like b in Ex. 1.1) whereas in T', nodes of the domain are untyped and the type information is stored within the label symbol.

A language of *tulab* trees over Σ is a subset of $TTree(\Sigma)$. In this paper we define and study a class of *tulab* tree automata (the *nested sibling tree automata*) that recognize a class of *tulab* tree languages (the *nested sibling tulab tree languages*) satisfying structural constraints much more general than the regular ones expressed, for example, by DTDs in the XML context.

2. Nested sibling tree automata

In this section we introduce a class of automata running on *tulab* trees. Roughly speaking, such automata will control a particular structural constraint at a given node of the input *tulab* tree using a visibly pushdown language of words over the alphabet of states. The class of visibly pushdown languages has been introduced in [4] and is a robust subclass of context-free languages. Its use in our framework is motivated, firstly by its expressivity allowing to express complex structural constraints on *tulab* trees, and secondly by its closure properties assuring to get, as we will see later, a robust class of recognized *tulab* tree languages. We first recall the main features of the class of visibly pushdown languages of words.

Visibly pushdown (word) languages. Visibly pushdown languages are languages of words over pushdown alphabets. A *pushdown alphabet* $\widetilde{X}$ is defined as a classical alphabet X together with a partition of X giving a type to each letter; formally, a pushdown alphabet is a pair $\widetilde{X} = (X, (X_{up}, X_{down}, X_{fix}))$ where X is a finite alphabet, $(X_{up}, X_{down}, X_{fix})$ is a partition of X, X_{up} being a set of *up* letters, X_{down} a set of *down* letters and X_{fix} a set of *fix* letters, respectively. For each letter x in X, we say that x is of type τ in $\widetilde{X}$ when $x \in X_\tau$. This type information can be used to restrict the behavior of a classical pushdown automaton running

on a word of X^*: intuitively, the automaton will push onto the stack only when reading an *up* letter, pop the stack for a *down* letter and do not use the stack for a *fix* letter. Such restricted pushdown automata are called visibly pushdown automata (*vpa*) over $\widetilde{X}$ and have been studied in [4]. We refer the reader to [4] for a complete description of the class *VPA* of visibly pushdown automata. The class of languages over the alphabet X that are recognized by some visibly pushdown automaton over $\widetilde{X}$ is the class of *visibly pushdown languages* over $\widetilde{X}$ and is denoted by $VPL(\widetilde{X})$. For example the language $\{a^n c b^n \mid n \in N\}$ belongs to $VPL(\widetilde{X})$ with $\widetilde{X} = (\{a, b, c\}, (\{a\}, \{b\}, \{c\}))$. More generally *VPL* denotes the class of visibly pushdown languages of words over any pushdown alphabet. In [4] it is proved that *VPL* is a robust subclass of context-free languages of words. We recall below the main properties of *VPL* that we will use later. The term *renaming mapping* between two pushdown alphabets $\widetilde{X}$ and $\widetilde{U}$ refers to a mapping from X to U that is type-preserving *i.e.* that maps each letter of X to a letter of U with the same type in $\widetilde{X}$ and $\widetilde{U}$ respectively. A renaming mapping can be extended in the usual way to words over X^*

- $VPL(\widetilde{X})$ is closed under union, intersection and complement;
- *VPL* is closed under direct and inverse renaming mappings;
- the emptiness problem for *VPA* can be decided in polynomial time;
- the inclusion problem for *VPA* is *EXPTIME*-complete.

As explained above we will use visibly pushdown languages to enforce particular structural constraints on *tulab* trees. Let us recall that *tulab* trees are labeled over a classical alphabet but have typed nodes. The control processes of complex structural constraints will be defined through so-called *nested sibling tree automata* whose sets of state symbols are pushdown alphabets and whose runs will use the type information stored within the nodes of tulab trees.

Nested sibling tree automata (*nsta*). A *nested sibling tree automaton* (*nsta* for short) $\mathcal{A}$ over an alphabet Σ is a quadruple $\mathcal{A} = (\Sigma, \widetilde{Q}, \delta, F)$ where

- $\widetilde{Q} = (Q, (Q_{up}, Q_{down}, Q_{fix}))$ is a pushdown alphabet and Q is the set of state symbols;
- $\delta \subseteq (\Sigma \times Q \times Q^*)$ is the transition relation;
- for each (σ, q) in $\Sigma \times Q$, the set $L_A(\sigma, q) = \{w \in Q^* \mid (\sigma, q, w) \in \delta\}$ is a visibly pushdown language over $\widetilde{Q}$;
- $F \subseteq Q$ is the set of final states.

Let us remark that the above definition involves a possibly infinite transition relation. To be effectively given, this transition relation can be given, for instance, through the list of the *vpa*'s recognizing the languages $L_A(\sigma, q)$ for all (σ, q) in $\Sigma \times Q$.

So, a nested sibling tree automaton can be viewed as an unranked tree automaton (or hedge automaton, see [7]) with some specific features: (i) it uses a pushdown alphabet $\widetilde{Q}$ and (ii) while regular unranked tree automata require the $L_A(\sigma, q)$ languages to be regular languages over Q, nested sibling tree automata require the $L_A(\sigma, q)$ languages to be visibly pushdown languages over $\widetilde{Q}$. Running

on a *tulab* tree T, a *nsta* $\mathcal{A}$ will be able to take into account the information type of T's nodes thanks to its pushdown alphabet $\widetilde{Q}$. We describe next precisely this feature of *nsta*'s runs.

Nsta's run on *tulab* trees. A *run* r of a *nsta* $\mathcal{A}$ on a *tulab* tree $T = (D, \lambda)$ is a mapping, $r : D \to Q$, from the typed domain D to Q, such that: (i) r is type preserving *i.e.* r maps each node w of D to a state q of Q with the same type in $\widetilde{Q}$ as w in D and (ii) $\forall w \in D$, if $(w1, w2, \ldots, wk)$ is the (possibly empty) sequence of w's children nodes and $\lambda(w) = \sigma$ then the (possibly empty) word $r(w1)r(w2)\ldots r(wk)$ of Q^* must belong to the visibly pushdown language $L_A(\sigma, r(w))$ of $VPL(\widetilde{Q})$. A run r is a *successful run* of $\mathcal{A}$ on T if $r(\varepsilon) \in F$. A *tulab* tree T is recognized by $\mathcal{A}$ if there exists a successful run of $\mathcal{A}$ on T. We denote by $L(\mathcal{A})$ the language of *tulab* trees recognized by $\mathcal{A}$. We say that a language L of *tulab* trees is recognized by a *nsta* $\mathcal{A}$ if $L = L(\mathcal{A})$. It is then called a *nested sibling tulab tree* language. We denote by *NSTL* the class of nested sibling *tulab* tree languages.

Deterministic and complete nested sibling automata. A *nsta* $\mathcal{A}$ is called *deterministic* (respectively *complete*) if for each (σ, w) of $\Sigma \times Q^*$ and for each type τ of $\{up, down, fix\}$, there is *at most* (respectively *at least*) one state q of Q_τ such that $w \in L_A(\sigma, q)$. Notice that this definition of determinism for *nsta*s slightly differs from the one for regular unranked tree automata, as it takes into account how *nsta*s run on *tulab* trees: a local run description at a node n is given by the word w built with the sequence of n's children states, the label σ of n, and the type τ of n in D. Therefore two states q and q' can satisfy $w \in L_A(\sigma, q)$ and $w \in L_A(\sigma, q')$ if they have different types in $\widetilde{Q}$ because they cannot be assigned to the same node n of the typed domain D. Clearly, if $\mathcal{A}$ is deterministic (respectively complete, complete and deterministic), there exists *at most* (respectively *at least*, *exactly*) one run of $\mathcal{A}$ on any *tulab* tree T.

We denote by $NSTA(\Sigma)$ (respectively $DNSTA(\Sigma)$, $CDNSTA(\Sigma)$) the family of *nsta*s (respectively deterministic, complete and deterministic *nsta*s) over Σ. Similarly we denote by $NSTL(\Sigma)$ (respectively $DNSTL(\Sigma)$, $CDNSTL(\Sigma)$) the class of languages of *tulab* trees that are recognized by some *nsta* of $NSTA(\Sigma)$ (respectively $DNSTA(\Sigma)$, $CDNSTA(\Sigma)$).

3. Examples

In this section we give some examples of nested sibling *tulab* tree languages illustrating how this class of languages can capture complex structural constraints.

Example 3.1. Let us define $L_1 = \{T_n = (D_n, \mu_n) | n \in \mathbb{N}\}$, where D_n is as in Example 1.1 and μ_n is defined by: $\mu_n(\varepsilon) = \mu_n(n+1) = a$, $\mu_n(i) = b$ for each node i with $1 \leq i < n$ and $\mu_n(i) = c$ for each node i with $n + 2 \leq i \leq 2n + 1$. It is easy to show that L_1 is a nested sibling *tulab* tree language because it is trivial to define a nested sibling tree automaton $\mathcal{A}$ recognizing it by using the pushdown alphabet $\widetilde{Q} = (\{q_0, q_1, q_2\}, (Q_{up} = \{q_1\}, Q_{down} = \{q_2\}, Q_{fix} = \{q_0\}))$ as alphabet of states and the fact that the language $\{q_1^n q_0 q_2^n \mid n \in \mathbb{N}\}$ is a visibly pushdown

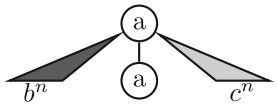

FIGURE 3. A *tulab* tree of L_1.

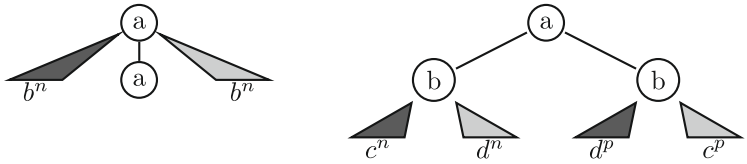

FIGURE 4. Left panel: the *tulab* tree T_n. Right panel: the *tulab* tree $U_{n,p}$.

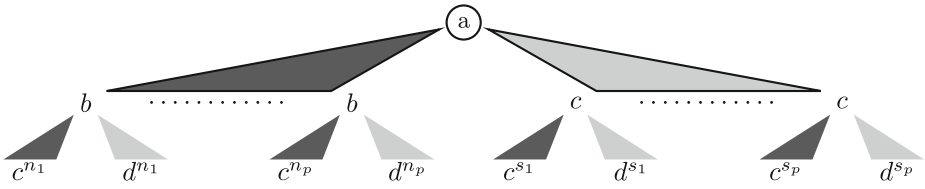

FIGURE 5.

language over $\widetilde{Q}$. L_1 is a simple language of *tulab* trees requiring context-free style structural constraints. Figure 3 shows a representation of a *tulab* tree of L_1.

Example 3.2. Let us define the languages of *tulab* trees: $L_2 = \{T_n \mid n \in \mathbb{N}\}$ and $L_3 = \{U_{n,p} \mid n, p \in \mathbb{N}\}$, where T_n and $U_{n,p}$ are the *tulab* trees represented in Figure 4. As in Example 3.1, it is easy to show that L_2 and L_3 are nested sibling *tulab* tree languages requiring context-free structural constraints. It is interesting to notice here that a same label can appear at nodes having different types: in T_n, as already pointed out in Example 1.1, the label b appears at sibling nodes with different types *up* and *down*, and in $U_{n,p}$ the same observation can be made about the label c (or d) that can appear at nodes with different types without sibling relationship but with fathers labeled with the same symbol b.

Example 3.3. The language $L_4 = \{T_{p,\overrightarrow{n},\overrightarrow{s}} \,|\, p \in \mathbb{N}, \overrightarrow{n}, \overrightarrow{s} \in \mathbb{N}^p\}$ where $T_{p,\overrightarrow{n},\overrightarrow{s}}$ is represented in Figure 5 with $\overrightarrow{n} = \{n_1, \ldots, n_p\}$ and $\overrightarrow{s} = \{s_1, \ldots, s_p\}$, is a nested sibling *tulab* tree language requiring complex context-free constraints at different levels in the *tulab* tree.

Remark 3.4. Let us notice that, as pointed in Example 1.1, *tulab* trees can be translated into standard unranked labeled trees labeled with symbols from a pushdown alphabet. Therefore instead of introducing this new notion of *tulab*

trees one might think to define runs of nested sibling tree automata on standard unranked trees labeled with a pushdown alphabet of symbols. Such runs would then associate with some node ν a state having the same type as ν's label. However it is easy to see that such a definition would lead to a class of recognized tree languages that would lack some closure properties like the closure under mappings. Let us for example consider the translated version $T_n^{'}$ of T_n of Example 3.1. It is not difficult to see that the language $\{T_n^{'} \,|\, n \in \mathbb{N}\}$ would then be recognized by a nested sibling tree automaton. However if we consider the mapping μ that maps the letters b^u and c^d on the same letter b^u and keeps the letter a^f unchanged, it is clear that the language $\{\mu(T_n^{'}) \,|\, n \in N\}$ would no longer be recognized by a nested sibling tree automaton. The closure under mappings would be only obtained under renaming (*i.e.* type-preserving) mappings. Because the type information in a *tulab* tree is stored within the nodes instead of within the labels, this drawback disappears for the class *NSTL*. The next section shows that the definitions given in Section 2 ensure *NSTL* to be a robust class of *tulab* tree languages.

4. Robustness results

4.1. Closure properties of nested sibling tree languages

In this section we review the closure properties of the class *NSTL* under Boolean operations as well as under direct and inverse mappings. The proof of those properties is based on a simple adaptation of constructions for unranked tree automata, using additionally the closure properties of *VPL*.

Proposition 4.1 (Closure under union and intersection)**.** *NSTL*(Σ) *is closed under union and intersection.*

Proof. Let L_1 and L_2 be two *tulab* trees languages of *NSTL*(Σ): $L_1 = L(A_1)$ with $A_1 = (\Sigma, \widetilde{Q^1}, \delta^1, F^1)$, $\widetilde{Q^1} = (Q^1, (Q^1_{up}, Q^1_{down}, Q^1_{fix}))$, and $L_2 = L(A_2)$ with $A_2 = (\Sigma, \widetilde{Q^2}, \delta^2, F^2)$, $\widetilde{Q^2} = (Q^2, (Q^2_{up}, Q^2_{down}, Q^2_{fix}))$.

Closure under union: it is straightforward to check that the automaton obtained by the union of, respectively, sets of states, transition relations and sets of final states, accepts precisely the union of L_1 and L_2 (assuming without loss of generality that Q^1 and Q^2 are disjoint).

Closure under intersection: we define the *nsta* automaton $\mathcal{A} = (\Sigma, \widetilde{Q}, \delta, F)$ by $\widetilde{Q} = (Q, (Q^1_{up} \times Q^2_{up}, Q^1_{down} \times Q^2_{down}, Q^1_{fix} \times Q^2_{fix}))$ where $Q = \bigcup_{\tau \in \{up, down, fix\}} Q^1_\tau \times Q^2_\tau$, $\delta = \{(\sigma, (q,t), (q_1,t_1) \dots (q_n,t_n)) \,|\, (\sigma, q, q_1 \dots q_n) \in \delta^1 \text{ and } (\sigma, t, t_1 \cdots t_n) \in \delta^2\}$ and $F = (F^1 \times F^2) \cap Q$.

For all $(\sigma, (q,t))$ in $\Sigma \times Q$, $L_A(\sigma, (q,t))$ coincides with $\pi_1^{-1}(L_{A_1}(\sigma, q)) \cap \pi_2^{-1}(L_{A_2}(\sigma, t))$, where π_1 and π_2 are the first and the second projection from Q to Q_1 and Q_2 respectively. But π_1 and π_2 are renaming mappings from $\widetilde{Q}$ to $\widetilde{Q_1}$ and $\widetilde{Q_2}$. Therefore using the closure properties of *VPL* we deduce that $L_A(\sigma, q)$ belongs to *VPL*$(\widetilde{Q})$. Now clearly: $\mathrm{L}(\mathcal{A}) = L_1 \cap L_2$. □

Proposition 4.2 (Closure under direct and inverse mapping)**.** *Let* Σ, Ω *be two alphabets,* f *be a mapping from* Σ *to* Ω, L *and* H *be languages of* $NSTL(\Sigma)$ *and* $NSTL(\Omega)$ *respectively. Then the* tulab *trees languages* $f(L) = \{f(T) \mid T \in L\}$ *and* $f^{-1}(H) = \{T \in TTree(\Sigma) \mid f(T) \in H\}$ *belong to* $NSTL(\Omega)$ *and* $NSTL(\Sigma)$ *respectively.*

Proof. Closure under direct mapping: we assume $L = L(\mathcal{A})$ with $\mathcal{A} = (\Sigma, \widetilde{Q}, \delta, F)$. We define $\mathcal{B} = (\Omega, \widetilde{Q}, \eta, F)$ with $(\omega, q, w) \in \eta$ if there is σ in Σ such that $\omega = f(\sigma)$ and $(\sigma, q, w) \in \delta$. For each (ω, q) of $\Omega \times Q$, we have $L_B(\omega, q) = \bigcup_{\sigma \in f^{-1}(\omega)} L_A(\sigma, q)$. Using the closure of $VPL(\widetilde{Q})$ under union, we deduce that $L_B(\omega, q)$ belongs to $VPL(\widetilde{Q})$ and therefore that $\mathcal{B}$ is a *nsta* over Ω. Now clearly $L(\mathcal{B}) = f(L(\mathcal{A}))$ as any successful run of $\mathcal{B}$ on a *tulab* tree T directly simulates a successful run of $\mathcal{A}$ on some tree S of $L(\mathcal{A})$ such that $f(S) = T$.

Closure under inverse mapping: we assume $H = L(\mathcal{A})$ with $\mathcal{A} = (\Omega, \widetilde{Q}, \delta, F)$. We define $\mathcal{B} = (\Sigma, \widetilde{Q}, \eta, F)$ with $(\sigma, q, w) \in \eta$ if $(f(\sigma), q, w) \in \delta$. For each (σ, q) of $\Sigma \times Q$, we have $L_B(\sigma, q) = L_A(f(\sigma), q)$. So $L_B(\sigma, q)$ belongs to $VPL(\widetilde{Q})$ and therefore $\mathcal{B}$ is a *nsta* over Σ. Because any successful run of $\mathcal{B}$ on a *tulab* tree T directly simulates a successful run of $\mathcal{A}$ on $f(T)$ we deduce that $L(\mathcal{B}) = f^{-1}(L(\mathcal{A}))$. □

To investigate the closure under complementation of $NSTL(\Sigma)$ we first focus on the equivalence between complete deterministic and non deterministic *nsta*s, the closure under complementation of $NSTL(\Sigma)$ being closely related to this equivalence. Actually for each complete and deterministic *nsta* $\mathcal{A}$ and each *tulab* tree T there is exactly one run of $\mathcal{A}$ over T and $TTree(\Sigma) \backslash L(\mathcal{A})$ is recognized by the *nsta* deduced from $\mathcal{A}$ by changing the set of final states F into $Q \backslash F$. It turns out that complete deterministic and non deterministic nested sibling tree automata can be proved to be equivalent.

Proposition 4.3 (Equivalence between *nsta*s and complete deterministic *nsta*s)**.** *For each* nsta $\mathcal{A}$ *of* $NSTA(\Sigma)$ *there is a complete deterministic* nsta $\mathcal{B}$ *such that* $L(\mathcal{A}) = L(\mathcal{B})$. *Therefore,* $NSTL(\Sigma) = CDNSTL(\Sigma)$.

Proof. Let $\mathcal{A} = (\Sigma, \widetilde{Q}, \delta, F)$ be a *nsta* over Σ. We build the following complete deterministic *nsta* $\mathcal{B}$ by simulating as usual in a deterministic way all possible runs of $\mathcal{A}$ on a given *tulab* tree. Such a construction is used in [7] in the framework of regular unranked tree automata. In the later we use the notation $P(E)$ to denote the set of the subsets of a set E. So $\mathcal{B}$ is given by $(\Sigma, \widetilde{R}, \Delta, \Phi)$ where:

(1) the pushdown alphabet $\widetilde{R} = (R, (R_{up}, R_{down}, R_{fix}))$ with $R_{up} = P(Q_{up})$, $R_{down} = P(Q_{down})$, $R_{fix} = P(Q_{fix})$ and $R = R_{up} \cup R_{down} \cup R_{fix}$;

(2) Δ is the *set of transitions*: if S is in R_τ (where τ is *up*, *down* or *fix*), $S_1, \ldots, S_n$ are in R and σ is in Σ, $(\sigma, S, S_1 \ldots S_n) \in \Delta$ if and only if $S = \{q \in Q_\tau \mid \exists q_1 \in S_1, \ldots, \exists q_n \in S_n$ *such that* $(\sigma, q, q_1 \ldots q_n) \in \delta\}$;
(3) Φ is the *set of final states*: $\Phi = \{S \in R \mid S \cap F \neq \emptyset\}$.

$\mathcal{B}$ is complete and deterministic, because given a type τ and $(\sigma, S_1 S_2 \ldots S_n)$ in $\Sigma \times R^*$, there is exactly one S of R_τ such that $(S, \sigma, S_1 S_2 \ldots S_n) \in \Delta$, that is $S = \{q \in Q_\tau \mid \exists q_1 \in S_1, \exists q_2 \in S_2, \ldots, \exists q_n \in S_n$ such that $(\sigma, q, q_1 q_2 \ldots q_n) \in \delta\}$.

Moreover, as in the standard case, there is a successful run of $\mathcal{B}$ on a *tulab* tree T if and only if there is a successful run of $\mathcal{A}$ on T. So $L(\mathcal{A}) = L(\mathcal{B})$. It remains to prove that $\mathcal{B}$ is a *nsta*: so given $\sigma \in \Sigma$ and $S \in R_\tau$, we show below that $L_B(\sigma, S)$ belongs to $VPL(\widetilde{R})$.

$L_B(\sigma, S) = \{S_1 S_2 \ldots S_n \in R^* \mid (\sigma, S, S_1 S_2 \ldots S_n) \in \Delta\}$. Therefore, $S_1 \ldots S_n$ belongs to $L_B(\sigma, S)$ if and only if the two following conditions (i) and (ii) hold:

(i) for each $q \in S$, there are $q_1 \in S_1, \ldots, q_n \in S_n$ such that $(\sigma, q, q_1 \ldots q_n) \in \delta$;
(ii) if there are $q_1 \in S_1, \ldots, q_n \in S_n$ such that $(\sigma, q, q_1 \ldots q_n) \in \delta$ for some state q in Q_T, then q is in S.

We denote by $L(i)$ (resp. by $L(ii)$) the language of words $S_1 \ldots S_n$ of R^* satisfying (i) (resp. (ii)), we define the pushdown alphabet $\widetilde{U} = (U, (U_{up}, U_{down}, U_{fix}))$ with $U_\tau = \{(q, T) \in Q_\tau \times R_\tau \mid q \in T\}$ for any type τ of $\{up, down, fix\}$, and we denote by π_1 (resp. π_2) the first (resp. second) projection from U to Q (resp. to R). Let us remark that π_1 and π_2 are renaming mappings.

If we define $L'_{\sigma,q} = \{(q_1, S_1)(q_2, S_2) \ldots (q_n, S_n) \in U^* \mid (\sigma, q, q_1 q_2 \ldots q_n) \in \delta\}$, we can write: $L(i) = \bigcap_{q \in S} \pi_2(L'_{\sigma,q})$ and $L'_{\sigma,q} = \pi_1^{-1}(L_A(\sigma, q))$.

So finally, $L(i) = \bigcap_{q \in S} \pi_2(\pi_1^{-1}(L_A(\sigma, q)))$. Because each $L_A(\sigma, q)$ belongs to $VPL(\widetilde{Q})$ we deduce from the closure properties of the family of visibly pushdown languages that $L(i)$ belongs to $VPL(\widetilde{R})$.

Now a word $S_1 \ldots S_n$ satisfies (ii) if and only if there is no $q \in Q_\tau \backslash S$ such that $S_1 \cdots S_n \in \pi_2(\pi_1^{-1}(L_A(\sigma, q)))$. Therefore, $L(ii) = R^* \backslash \bigcup_{q \in Q_\tau \backslash S} \pi_2(\pi_1^{-1}(L_A(\sigma, q)))$. Using again the closure properties of the family of visibly pushdown languages we deduce that $L(ii)$ and therefore $L_B(\sigma, S)$ are languages of $VPL(\widetilde{R})$. Finally $\mathcal{B}$ is a complete and deterministic *nsta* such that $L(\mathcal{A}) = L(\mathcal{B})$. □

We immediately deduce Corollary 4.4

Corollary 4.4 (Closure under complementation)**.** *NSTL*(Σ) *is closed under complementation.*

4.2. Decision problems for nested sibling tree automata

We focus now on decision problems for nested sibling tree automata. The next propositions are devoted to the membership and emptiness problems respectively. Their proofs generalize decision algorithms that are well-known for regular tree automata.

Each nested sibling tree automaton $(\Sigma, \widetilde{Q}, \delta, F)$ considered in this section is supposed to be effectively given: in particular the set δ of transitions is effectively

given for example through the list of the *vpa*'s recognizing the languages $L_A(\sigma, q)$ for $(\sigma, q) \in (\Sigma \times Q)$.

Proposition 4.5 (Emptiness problem)**.** *Let $\mathcal{A} = (\Sigma, \widetilde{Q}, \delta, F)$ be a nested sibling tree automaton over Σ. Deciding whether $L(\mathcal{A})$ is empty is in PTIME.*

Proof. The proof is standard and amounts to saturate a subset Acc of $\widetilde{Q}$ until a fix-point is reached. Starting from $Acc = \varnothing$, we consider each transition rule, that is, each *vpa* $\mathcal{V}_{(\sigma,q)}$ for any pair (σ, q), adding q to Acc if $L(\mathcal{V}_{(\sigma,q)}) \cap Acc^*$ is non-empty. Eventually, $L(\mathcal{A})$ is empty iff $Acc \cap F = \varnothing$.

Using the fact that intersection of two *vpa*'s can be done in polynomial time and that emptiness of *vpa*'s is in PTIME [4], the complexity bound follows. □

Proposition 4.6 (Membership problem)**.** *Let $\mathcal{A} = (\Sigma, \widetilde{Q}, \delta, F)$ be a nested sibling tree automaton over Σ and t be a* tulab *tree of $TTree(\Sigma)$. Deciding whether t belongs to $L(\mathcal{A})$ can be done in linear time for the non-uniform problem and in PTIME for the uniform problem.*

Proof. For the non-uniform complexity (meaning that the automata $\mathcal{A}$ is not part of the input), we may assume that $\mathcal{A}$ is deterministic and for each of its transitions each *vpa* $\mathcal{V}_{(\sigma,q)}$ (for any pair (σ, q)) is a deterministic *vpa*. Therefore, one can decorate in a bottom-up way the tree with states in linear time using the fact that only one rule can be applied at a position in the tree and that picking this rule amounts to test the membership of a word to the language defined by some deterministic *vpa* which can be done in linear time in the size of the word.

For the uniform complexity, we rely on the generic algorithm proposed in [9] for hedge automata. This algorithm computes (possibly in a bottom way) a set of states for each position in the tree. A node is labeled by a set containing q if its children are labeled by sets $S_1 \dots S_k$ containing respectively $q_1, \dots, q_k$ and the word $q_1 \dots q_k$ belongs to $L_A(\sigma, q)$. Testing whether a word built from the sets $S_1 \dots S_k$ belongs to $L_A(\sigma, q)$ is therefore the crucial point of the membership test. Obviously, this can be solved by building a trivial word automaton for the language $S_1 \dots S_k$ with k states and at most $k \cdot |\widetilde{Q}|$ transitions. It suffices then to test the emptiness of the intersection of this automaton and the *vpa* defining the transitions which can be done in PTIME. □

From Propositions 4.1 and 4.5, and from Corollary 4.4 we obtain:

Corollary 4.7 (Inclusion problem)**.** *Let $\mathcal{A}$ and $\mathcal{B}$ be two nested sibling tree automata over Σ. Deciding whether $L(\mathcal{A})$ is included in $L(\mathcal{B})$ is in EXPTIME.*

Proof. The inclusion test "$\mathrm{L}(\mathcal{A}) \subseteq^? \mathrm{L}(\mathcal{B})$" is equivalent to test the emptiness of $L(\mathcal{A}) \cap (TTree(\Sigma) \setminus L(\mathcal{B}))$. So the complexity of the inclusion problem directly follows from the constructions proving Propositions 4.1 and 4.5 and Corollary 4.4. □

5. Different characterizations of *NSTL*

5.1. Logical characterization of *NSTL*

In this section, we give a logical characterisation of nested sibling *tulab* tree languages. We propose an extension of monadic second order logic (MSO) for unranked trees using the matching relation from [4] and call this logic MSO_μ.

The logic MSO is the extension of the first-order logic FO with quantification over unary relations, *i.e.* over sets. Additionally, we consider the matching relation μ which holds between two elements of the tree domain if they are the matching pair of *up* and *down* positions.

Let ζ be the signature $\{\mathsf{lab}_a \,|\, a \in \Sigma\} \cup \{\mathsf{child}, \mathsf{sibling}, \mu\}$ where the lab_a's are unary predicates. and $\mathsf{child}, \mathsf{sibling}, \mu$ are binary ones. We associate with a *tulab* tree $T = (D, \lambda)$ a finite ζ-structure $\mathcal{S}^T$ whose domain is D and for the relations: for all d, d' in D, (i) $\mathsf{lab}_a(d)$ holds in $\mathcal{S}^T$ if $\lambda(d) = a$, (ii) $\mathsf{child}(d, d')$ holds in $\mathcal{S}^T$ if $d' = d.i$ for some i in $\mathbb{N}$, (iii) $\mathsf{sibling}(d, d')$ holds in $\mathcal{S}^T$ if there exists d'' in D and i in $\mathbb{N}$ such that $d = d''.i$, $d' = d''.(i+1)$ and, (iv) $\mu(d, d')$ holds in $\mathcal{S}^T$ if

- there exists d'' in D and i, j in $\mathbb{N}$ such that $i < j$, $d = d'' \cdot i$ and $d' = d'' \cdot j$;
- d is of type *up* and d' of type *down*;
- $|\{d'' \cdot k \mid i < k < j\}|_{up} = |\{d'' \cdot k \mid i < k < j\}|_{down}$;
- for all $i < l < j$, $|\{d'' \cdot k \mid i \leq k \leq l\}|_{down} < |\{d'' \cdot k \mid i \leq k \leq l\}|_{up}$

where $|S|_\tau$ is the number of positions of type τ in the set S.

We assume a countable set of first-order variables ranging over by $x, y, z, \ldots$ and a countable set of second-order variables ranging over by $X, Y, Z, \ldots$. MSO_μ formulas are given by the following syntax:

$$\psi ::= \mathsf{lab}_a(x) \,|\, \mathsf{child}(x, y) \,|\, \mathsf{sibling}(x, y) \,|\, x \in X \,|\, \psi \vee \psi \,|\, \neg\psi \,|\, \exists x \cdot \psi \,|\, \exists X \cdot \psi \mid \mu(x, y).$$

Let $\mathcal{S}$ be a ζ-structure with domain D. Let ρ be a valuation mapping first-order variables to elements from D and second-order variables to subsets of D. We write $\mathcal{S} \models_\rho \psi$ when the structure $\mathcal{S}$ is a model of the formula ψ under the valuation ρ; this is defined in the usual Tarskian manner and we have in particular, (i) $\mathcal{S} \models_\rho \mathsf{lab}_a(x)$ if $\mathsf{lab}_a(\rho(x))$ holds in $\mathcal{S}$, (ii) $\mathcal{S} \models_\rho \mathsf{child}(x, y)$ if $\mathsf{child}(\rho(x), \rho(y))$ holds in $\mathcal{S}$, (iii) $\mathcal{S} \models_\rho \mathsf{sibling}(x, y)$ if $\mathsf{sibling}(\rho(x), \rho(y))$ holds in $\mathcal{S}$ and, (iv) $\mathcal{S} \models_\rho \mu(x, y)$ if $\mu(\rho(x), \rho(y))$ holds in $\mathcal{S}$.

An MSO_μ-sentence is an MSO_μ formula without free variables. A set of *tulab* trees L is MSO_μ-*definable* if and only if there exists an MSO_μ sentence ψ such that $L = \{T \,|\, T \models \psi\}$.

It is well-known that a language of unranked trees is accepted by some unranked tree automaton iff it is MSO-definable. We prove a similar result for nested sibling *tulab* tree languages.

Proposition 5.1. *A set of* tulab *trees is recognized by a nested sibling tree automaton iff it is MSO_μ-definable.*

Proof. To show that any language of *NSTL* is definable by some MSO_μ sentence, we simply prove that for any *nsta* A, there exists an MSO_μ sentence ϕ_A which states the existence of some accepting run on *tulab* trees.

We consider the visibly pushdown alphabet $\widetilde{X} = (X, (X_{up}, X_{fix}, X_{down}))$ and a visibly pushdown word automaton $M = (\widetilde{X}, P, P_{in}, \Gamma, \delta, F)$ where P is a finite set of states, $P_{in}, F \subseteq P = \{p_1, \ldots, p_n\}$ are respectively the set of initial and final states, $\Gamma = \{\gamma_1, \ldots, \gamma_m\} \cup \{\bot\}$ is a finite stack alphabet ($\bot$ is the symbol used at the bottom of the stack) and δ contains three kinds of transitions:

- **[push]** (q, a, q', γ) with q, q' in Q, a in X_{up}, γ in $\Gamma \backslash \{\bot\}$;
- **[pop]** (q, a, γ, q') with q, q' in Q, a in X_{down}, γ in $\Gamma \backslash \{\bot\}$;
- **[internal]** (q, a, q') with q, q' in Q, a in X_{fix}.

We assume $X_{up} = \{u_1, \ldots, u_{m_u}\}$, $X_{fix} = \{f_1, \ldots, f_{m_f}\}$, $X_{down} = \{d_1, \ldots, d_{m_d}\}$.

Let us denote $\leq$ the reflexive-transitive closure of $\mathsf{sibling}$. If S is a set of positions totally ordered by the relation $\leq$ and labeled with elements from X, and U_i for $1 \leq i \leq m_u$ (respectively F_i for $1 \leq i \leq m_f$, D_i for $1 \leq i \leq m_d$) is the set of S's positions labeled with u_i (respectively f_i, d_i), there exists a MSO_μ formula $\phi_M(S, U_1, \ldots, U_{m_u}, F_1, \ldots, F_{m_f}, D_1, \ldots, D_{m_d})$ that holds iff the word S is accepted by the visibly pushdown word automaton M over the pushdown alphabet $(X, (X_{up}, X_{fix}, X_{down}))$. Details of the construction of $\phi_M(S, U_1, \ldots, U_{m_u}, F_1, \ldots, F_{m_f}, D_1, \ldots, D_{m_d})$ are given in the appendix.

Now, let us define a MSO_μ formula stating the existence of an accepting run for a nested sibling tree automaton $\mathcal{A}$. The construction is in fact similar to the one for formula ϕ_M. Let $\mathcal{A} = (\Sigma, \widetilde{Q}, \delta, F)$ and $Q = Q_{up} \cup Q_{down} \cup Q_{fix}$ where $Q_{up} = \{u_1, \ldots, u_{m_u}\}$, $Q_{fix} = \{f_1, \ldots, f_{m_f}\}$ and $Q_{down} = \{d_1, \ldots, d_{m_d}\}$, δ is the transition relation and F the set of final states.

$$\exists U_1 \ldots U_{m_u} \exists F_1 \ldots F_{m_f} \exists D_1 \ldots D_{m_d}$$

$$\begin{array}{l}
\forall x \ (\bigvee_{1 \leq i \leq m_u} x \in U_i \ \vee \ \bigvee_{1 \leq i \leq m_f} x \in F_i \ \vee \ \bigvee_{1 \leq i \leq m_d} x \in D_i) \wedge \\
(\bigwedge_{1 \leq i,j \leq n,\, i \neq j} x \in U_i \Rightarrow \neg(x \in U_j) \wedge \\
(\bigwedge_{1 \leq i,j \leq n,\, i \neq j} x \in F_i \Rightarrow \neg(x \in F_j) \wedge \\
(\bigwedge_{1 \leq i,j \leq n,\, i \neq j} x \in D_i \Rightarrow \neg(x \in D_j) \wedge \\
(\bigvee_{1 \leq i \leq m_u} x \in U_i \Rightarrow \neg(\bigvee_{1 \leq i \leq m_f} x \in F_i \ \vee \ \bigvee_{1 \leq i \leq m_d} x \in D_i)) \wedge \\
(\bigvee_{1 \leq i \leq m_f} x \in F_i \Rightarrow \neg(\bigvee_{1 \leq i \leq m_u} x \in U_i \ \vee \ \bigvee_{1 \leq i \leq m_d} x \in D_i)) \wedge \\
(\bigvee_{1 \leq i \leq m_d} x \in D_i \Rightarrow \neg(\bigvee_{1 \leq i \leq m_u} x \in U_i \ \vee \ \bigvee_{1 \leq i \leq m_f} x \in F_i)) \wedge
\end{array}$$

$$\forall x, y \, \mu(x, y) \Rightarrow \bigvee_{1 \leq i \leq m_u} x \in U_i \ \wedge \ \bigvee_{1 \leq i \leq m_d} y \in D_i \wedge$$

$$\begin{array}{l}
\forall x \ (\exists Z \ (\forall y \, \mathsf{child}(x, y) \Leftrightarrow y \in Z) \wedge \bigvee_{(a,q,L) \in \delta} \mathsf{lab}_a(x) \\
\qquad \wedge \ x \in X_q \wedge \phi_{M_L}(Z, U_1, \ldots, U_{m_u}, F_1, \ldots, F_{m_f}, D_1, \ldots, D_{m_d})) \wedge
\end{array}$$

$$\forall x (\forall y \neg \mathsf{child}(y, x)) \Rightarrow \bigvee_{q \in F} x \in X_q.$$

The first part of the formulas says that each position in the tree is uniquely labeled with a symbol from $\widetilde{Q}$. The second part states that this labelling is compatible

with the relation μ. The next part checks that the labelling corresponds locally to a transition rule where M_L is an automaton such that $L = L(M_L)$. The final part states that the run is an accepting one.

Now let us prove that the set of *tulab* trees that are models of some MSO_μ sentence is the language accepted by a nested sibling tree automaton.

The proof goes by induction over the structure of the formula: the models of a formula ϕ with free variables $x_1 \dots x_n X_1 \dots X_m$ are *tulab* trees over the alphabet $\Sigma \times \{0,1\}^{n+m}$. For a tree t, the idea is to represent a valuation $x_i \mapsto n$ (n being a node in t) within the tree by setting to 1 the ith component of the label only at the node n; this idea generalizes to sets of positions by setting to 1 the component corresponding to the variable X at each node belonging to the set associated with X by the valuation.

The induction uses closure by union, complementation, renaming to compute automata for disjunction, negation, existential quantifications of formulas. Thus, it suffices to build automata for atomic formulas. The construction for $x \in X$, $\mathsf{lab}_a(x)$, $\mathsf{child}(x,y)$, $\mathsf{sibling}(x,y)$ are similar to the one for (regular) automata for unranked trees. The construction for $\mu(x,y)$ is given in the appendix. □

5.2. Nested sibling tree languages and regular sets of 3-ary trees

In this section we describe a mapping from $TTree(\Sigma)$, the set of *tulab* trees, into a particular class of 3-ary labeled trees, called 3-stacktrees. This encoding will proceed in the same way that *VPL*s are encoded into sets of binary trees in [4] and maps together languages of *NSTL* with regular sets of 3-stacktrees. The 3-ary encoding trees are labeled over the typed alphabet $\tilde{\Sigma} = \{x^c \mid c \in \{u,d,f\} \text{ and } x \in \Sigma\} \cup \{\#\}$ in which a letter x^c is of type c and $\#$ of type f. A 3-ary tree over $\tilde{\Sigma}$ is a structure (Δ, θ) where Δ is a 3-ary tree domain (*i.e.* Δ contains the empty word, for $i = 0,1,2$, $wi \in \Delta$ implies $w \in \Delta$ and $\forall j < i, wj \in \Delta$) and θ is a labeling mapping from Δ to $\tilde{\Sigma}$. The set of all 3-ary trees over $\tilde{\Sigma}$ is denoted by $3Tree(\tilde{\Sigma})$.

Actually the encoding Enc maps a possibly empty sequence $(t_1, t_2, \dots, t_n)$ of *tulab* trees into a 3-ary tree over $\tilde{\Sigma}$. The idea is rather simple. Let us use the notation $t_i = [x_i, c_i](\sigma_i)$ to denote that t_i's root has the label x_i and the type c_i and that σ_i is the sequence of subtrees rooted at the children of t_i's root. We denote ϵ the empty sequence.

For the sequence of *tulab* trees σ, the mapping $\mathsf{Enc}(\sigma)$ is defined recursively as follows:

- $\mathsf{Enc}(\epsilon) = \#$;
- $\mathsf{Enc}([x_1, c_1](\sigma_1), t_2, \dots, t_n)$ is the 3-ary tree whose root is labeled with $x_1^{c_1}$, its 2-child is a tree isomorphic to $\mathsf{Enc}(\sigma_1)$ and depending on c_1:
  - $c_1 = u$ and depending on σ:
    - there exists an i such that $[x_i, c_i]$ is the matching position of $[x_1, c_1]$ in the word $[x_1, c_1] \dots [x_n, c_n]$, then its 0-child is isomorphic to $\mathsf{Enc}(t_2, \dots, t_{i-1})$ and its 1-child is isomorphic to $\mathsf{Enc}(t_i, \dots, t_n)$;

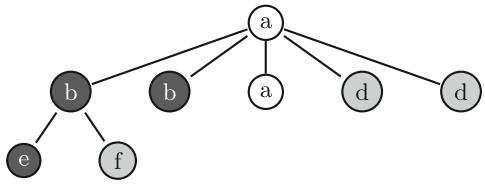

FIGURE 6. The *tulab* tree T_4.

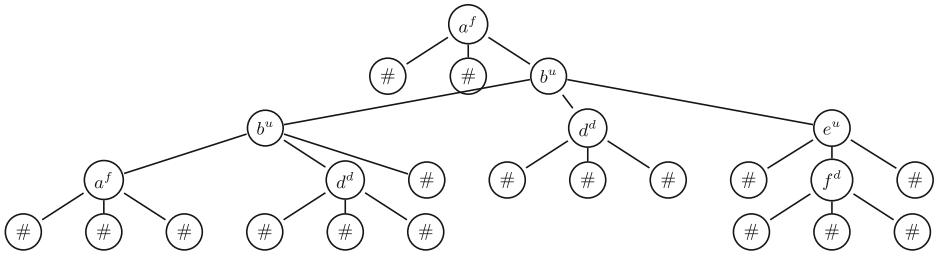

FIGURE 7. The T_4's encoding 3-stacktree.

$*$ otherwise, its 0-child is isomorphic to $\mathsf{Enc}(t_2, \ldots, t_n)$ and its 1-child is #.

– $c_1 \in \{f, d\}$: its 0-child is # and its 1-child is isomorphic to $\mathsf{Enc}(t_2, \ldots, t_n)$.

A tree of $\mathsf{Enc}(TTree(\Sigma))$ is called a 3-stacktree. We denote by $3STree(\tilde{\Sigma})$ the set of 3-stacktrees over $\tilde{\Sigma}$. Let us remark that $3STree(\tilde{\Sigma})$ is a strict subset of $3Tree(\tilde{\Sigma})$ because Enc is not a bijection. The tree encoding Enc is illustrated by Figures 6 and 7.

In [4] a mapping is provided between $\widetilde{\Sigma}^*$ and the set $2STree(\tilde{\Sigma})$ of particular binary trees over $\tilde{\Sigma}$ called 2-stacktrees. Through this mapping, languages of *VPL* are mapped onto regular languages of 2-stacktrees. We prove a similar result for nested sibling *tulab* tree languages.

Proposition 5.2. *Let G be a* tulab *tree language. Then G is a language of NSTL iff $\mathsf{Enc}(G)$ is a regular language.*

Proof. Let G be in *NSTL*, we denote $L = \mathsf{Enc}(G)$. Then $G = L(\mathcal{A})$ for a *nsta* automaton $\mathcal{A} = (\Sigma, \widetilde{Q}, \delta, F)$ with $\widetilde{Q} = (Q, (Q_{up}, Q_{down}, Q_{fix}))$. For each run r of the automaton $\mathcal{A}$ over a tree t of G we build the *tulab* tree t^r over the typed alphabet $\widetilde{\Sigma \times Q} = \{(\sigma^c, q) \,|\, \sigma \in \Sigma, q \in Q_c\}$ deduced from t by replacing each label σ of a node w with type c by the label $(\sigma^c, r(w))$ with type c. Let us denote G^{dec} the set of all such decorated trees t^r built from trees of G and successful runs of $\mathcal{A}$ over these trees. We have the following equalities: $L = \mathsf{Enc}(G)$ and $\mathsf{Enc}(G) = \pi_1(\mathsf{Enc}(G^{dec}))$, π_1 denoting the first projection from $\widetilde{\Sigma \times Q}$ to $\widetilde{\Sigma}$. We prove that L is regular by proving that $\mathsf{Enc}(G^{dec})$ is regular.

G^{dec} is in *NSTL* because $G^{dec} = L(\mathcal{B})$ where $\mathcal{B} = (\widetilde{\Sigma \times Q}, \widetilde{\Sigma \times Q}, \Delta, \widetilde{\Sigma \times F})$ with Δ given by: $\forall(\sigma^c, q) \in \widetilde{\Sigma \times Q}$, $L_{\mathcal{B}}((\sigma^c, q), (\sigma^c, q)) = \pi_2^{-1}(L_A(\sigma, q))$ (π_2 denoting the second projection from $\widetilde{\Sigma \times Q}$ to $\widetilde{Q}$). From $\mathcal{B}$, we construct a finite tree automaton $\mathcal{C} = (\widetilde{\Sigma \times Q}, U, \gamma, V)$ recognizing $\mathsf{Enc}(G^{dec})$. Let us denote by η the mapping from $\widetilde{\Sigma \times Q}^*$ to $2STree(\widetilde{\Sigma \times Q})$ that differs from the mapping of [4] only by the image of ϵ: $\eta(\epsilon) = \#$ instead of the empty tree as in [4]. For each (σ, q) of $\widetilde{\Sigma \times Q}$, $L_{\mathcal{B}}((\sigma, q), (\sigma, q))$ is a language of *VPL* and therefore by [4], $\eta(L_{\mathcal{B}}((\sigma, q), (\sigma, q)))$ is a regular language of 2-stacktrees recognized by some finite tree automaton $\mathcal{A}_{(\sigma,q)} = (\widetilde{\Sigma \times Q}, Q_{(\sigma,q)}, \delta_{(\sigma,q)}, F_{(\sigma,q)})$.

Then $\mathcal{C}$ is as follows:

- the set of states is $U = \bigcup_{(\sigma,q) \in \widetilde{\Sigma \times Q}} Q_{(\sigma,q)}$;
- the transitions γ: $((\sigma, q), t, t_1 t_2 t_3) \in \gamma$ *if and only if* $t_3 \in F_{(\sigma,q)}$ and $((\sigma, q), t, t_1 t_2) \in \delta_{(\sigma',q')}$ if $t \in Q_{(\sigma',q')}$;
- the set of final states: $V = \bigcup_{(\sigma,q) \in \widetilde{\Sigma \times Q}} F_{(\sigma,q)}$.

One can verify that $\mathsf{Enc}(G^{dec}) = L(\mathcal{C})$. Intuitively, during a computation of $\mathcal{C}$ on $\mathsf{Enc}(\tau)$ where $\tau \in G^{dec}$, the states t, t_1 and t_2 of a transition $((\sigma, q), t, t_1 t_2 t_3)$ of γ are used to simulate the verification in τ of the position of a node ν labeled with (σ, q) with respect to its brothers, while state t_3 is used to verify the correctness of the sequence of ν's children.

Conversely let L be a regular language of *3-stacktrees* recognized by some finite tree automaton $\mathcal{A} = (\widetilde{\Sigma}, Q, \delta, F)$ where $\widetilde{\Sigma}$ is a typed alphabet containing one 0-ary symbol $\#$ with type f. For some subset G of $TTree(\Sigma)$, $L = \mathsf{Enc}(G)$. Let L^{dec} be the set of all decorated trees t^r built from trees t of L and successful runs r of $\mathcal{A}$ on t by replacing each label a of a node w by the pair $(a, r(w))$. L^{dec} is a language of *3-stacktrees* over the typed alphabet $\widetilde{\Sigma} \times Q$ (where each (σ, q) has the type of σ in $\widetilde{\Sigma}$). L^{dec} is regular as well because $L^{dec} = L(\mathcal{B})$ where $\mathcal{B} = (\widetilde{\Sigma} \times Q, \widetilde{\Sigma} \times Q, \Delta, \widetilde{\Sigma} \times F)$ with $\Delta = \{((\sigma, t), (\sigma, t), (\sigma_1, t_1)(\sigma_2, t_2)(\sigma_3, t_3)) \mid (\sigma, t, t_1 t_2 t_3) \in \delta\}$.

We construct from $\mathcal{B}$ a *nsta* $\mathcal{C}$ recognizing G. If P_{3to2} is the mapping from 3-ary trees to 2-ary trees transforming each 3-ary tree into a 2-ary one by forgetting each 2-child, then for each state (σ, q), $P_{3to2}(L(\mathcal{B}, (\sigma, q)))$ is a regular language of 2-stacktrees over $\widetilde{\Sigma} \times Q$ where $L(\mathcal{B}, (\sigma, q))$ is the set of 3-stacktrees whose root is associated with (σ, q) by at least a run of $\mathcal{B}$. Therefore by [4], $\eta^{-1}(P_{3to2}(L(\mathcal{B}, (\sigma, q))))$ is a language of *VPL* that we denote $VPL((\sigma, q))$. Then we set $\mathcal{C} = (\Sigma, \widetilde{\Sigma} \times Q, \gamma, \widetilde{\Sigma} \times F)$ with γ defined by: $\forall(a, q) \in \Sigma \times Q$ and $\forall c \in \{u, d, f\}$, $L_{\mathcal{C}}(a, (a^c, q)) = VPL((a^c, q))$ and $L_{\mathcal{C}}(a, (\sigma, q)) = \emptyset$ if $\sigma \neq a^c$ for each $c \in \{u, d, f\}$.

One verifies that $G = L(\mathcal{C})$. In fact, because the definition of $L_{\mathcal{C}}(a, (a^c, q))$ as $VPL((a^c, q))$ and the inductive definition of Enc there is a one to one correspondence through Enc between decorated trees τ^ρ representing successful runs ρ of $\mathcal{C}$ on *tulab* trees τ and 3-ary trees accepted by $\mathcal{B}$. Therefore there is a one to one correspondence between successful runs of $\mathcal{C}$ on a *tulab* tree τ and successful runs of $\mathcal{A}$ on $t = \mathsf{Enc}(\tau)$. Hence τ belongs to $L(\mathcal{C})$ iff $\mathsf{Enc}(\tau)$ belongs to L. □

Let us remark that the robustness results of *NSTL* given in Section 4.1 can be directly deduced from Proposition 5.2.

- *NSTL*'s closures under intersection and complementation follow from Proposition 5.2, from the equalities $\mathsf{Enc}^{-1}(L_1) \cap \mathsf{Enc}^{-1}(L_2) = \mathsf{Enc}^{-1}(L_1 \cap L_2)$, $\overline{\mathsf{Enc}^{-1}(L)} = \mathsf{Enc}^{-1}(\overline{L}) = \mathsf{Enc}^{-1}(\overline{L} \cap 3STree(\tilde{\Sigma}))$ and from the fact that $L_1 \cap L_2$ and $\overline{L} \cap 3STree(\tilde{\Sigma})$ are regular sets of 3-stacktrees if so are L_1, L_2 and L.
- Equivalence between *nsta*s and complete deterministic *nsta*s can also be deduced from proposition 5.2 and the remark that the *nsta* $\mathcal{C}$ built from the automaton $\mathcal{A}$ recognizing a regular language of 3-stacktrees L is complete and deterministic if so is $\mathcal{A}$.

However the use of the encoding Enc does not lead to the direct automata constructions given in the previous sections, but to indirect algorithms including encoding and decoding phases.

5.3. *NSTL* and visibly tree automata with memory

In this section we give a relationship between *nsta*s and a subclass of *visibly tree automata with memory* (*vtam* for short) introduced in [10].

*vtam*s defined in [10] are running on binary trees labeled over a typed alphabet. We show, using the standard firstChild-nextSibling encoding from *tulab* trees into binary labeled trees, that encodings of *NSTL* languages coincide with languages of binary trees that are recognized by particular *vtam*s that we call *visibly tree automata with right stack* (*vtars* for short).

Roughly speaking, a *vtars* is a *vtam* that (i) only uses (word) stack memories and (ii) works only with right son memories during its runs. For sake of place, we omit the general definition of a *vtam* and directly give the formal definition of a *vtars* that is self containing. Given a ranked typed alphabet $\widetilde{\Sigma}$ whose symbols have arity 2 except a particular symbol $\#$ with arity 0, a *vtars* $\mathcal{A}$ on $\widetilde{\Sigma}$ is a tuple (Γ, Q, Q_f, Δ) where Γ is a stack alphabet with a special bottom-of-stack symbol $\perp$, Q is a finite set of state symbols, disjoint from $\widetilde{\Sigma} \cup \Gamma$, $Q_f \subseteq Q$ is the subset of final states and Δ is a set of rewrite rules of one of the following forms:

(i) $f(q_1(w_1), q_2(w_2)) \longrightarrow q(\gamma w_2)$, if f's type is *up*;
(ii) $f(q_1(w_1), q_2(\gamma w_2)) \longrightarrow q(w_2)$ or $f(q_1(w_1), q_2(\perp)) \longrightarrow q(\perp)$, if f's type is *down*;
(iii) $f(q_1(w_1), q_2(w_2)) \longrightarrow q(w_2)$, if f's type is *fix*;
(iv) $\# \longrightarrow q(\perp)$,

where $f \in \widetilde{\Sigma}$, q_1, q_2 and $q \in Q$, w_1 and $w_2 \in (\Gamma \setminus \{\perp\})^*\{\perp\}$ and $\gamma \in \Gamma \setminus \{\perp\}$.

Identifying a binary tree t labeled over $\widetilde{\Sigma}$ with its corresponding term over $\widetilde{\Sigma}$, we say that t is accepted by $\mathcal{A}$ in state $q \in Q$ and with memory $m \in (\Gamma \setminus \{\perp\})^*\{\perp\}$ if and only if $t \longrightarrow^* q(m)$ and we define $L(\mathcal{A}, q) = \{t \mid t \longrightarrow^* q(m), m \in (\Gamma \setminus \{\perp\})^*\{\perp\}\}$. The language of binary trees recognized by $\mathcal{A}$ is the union of languages of $\mathcal{A}$ recognized in its final states $L(\mathcal{A}) = \bigcup_{q \in Q_f} L(\mathcal{A}, q)$.

We now define the standard firstChild-nextSibling encoding *Enc* from possibly empty sequences of *tulab* trees over Σ into binary trees labeled over the typed alphabet $\tilde{\Sigma} = \{x^c \,|\, c \in \{u, d, f\} \text{ and } x \in \Sigma\} \cup \{\#\}$ in which a letter x^c is of arity 2 and type c and $\#$ of arity 0 and type f. Using the notation $t = [x, c](\sigma)$ introduced in Section 5.2 to denote a *tulab* tree t, *Enc* is defined as follows:

- $Enc(\epsilon) = \#$;
- $Enc([x_1, c_1](\sigma_1), t_2, \ldots, t_n) = x_1^{c_1}(Enc(\sigma_1), Enc(t_2, \ldots, t_n))$.

The following result gives the relationship between *NSTL* languages and languages recognized by a *vtars*.

Proposition 5.3. *Let L be a language of* tulab *trees. Then L is a nested sibling* tulab *tree language iff* $\mathrm{Enc}(L) = L(\mathcal{A})$ *for some* vtars $\mathcal{A}$.

Proof. Let $L = L(\mathcal{A})$ be a nested sibling *tulab* tree language where $\mathcal{A}$ is a *nsta* $\mathcal{A} = (\Sigma, \widetilde{Q}, \delta, F)$ with $\widetilde{Q} = (Q, (Q_{up}, Q_{down}, Q_{fix}))$. The construction of a *vtars* $\mathcal{B}$ recognizing $Enc(L)$ is guided by the following intuition: given a *tulab* tree t, maximal right branches in $Enc(t)$ (*i.e.* paths with maximal length only using right child relationship) are in bijection with child node sequences σ in t. Therefore $\mathcal{B}$'s runs on $Enc(t)$ will simulate on such branches, using only right stack memories, runs of a *vpa* over strings of Q^* verifying that the sequence of states associated with σ's nodes during $\mathcal{A}$'s runs on t is correct.

Formally, for each $(x, q) \in \Sigma \times \widetilde{Q}$, let $\mathcal{B}_{(x,q)} = (\Gamma_{(x,q)}, T_{(x,q)}, T^{in}_{(x,q)}, T^{f}_{(x,q)}, \Delta_{(x,q)})$ be a *vpa* automaton over strings of Q^* recognizing the visibly pushdown language $\widetilde{L_{\mathcal{A}}(x, q)} = \{\widetilde{w} \,|\, w \in L_{\mathcal{A}}(x, q)\}$ where $\widetilde{w}$ denotes the mirror image of w. $T_{(x,q)}$ (resp. $T^{in}_{(x,q)}$, $T^{f}_{(x,q)}$) is the set of (resp. initial, final) states while $\Gamma_{(x,q)}$ is the stack alphabet containing a special bottom-of-stack symbol $\perp$.

We define $T = \bigcup_{(x,q)\in\Sigma\times\widetilde{Q}} T_{(x,q)}$, $\Gamma = \bigcup_{(x,q)\in\Sigma\times\widetilde{Q}} \Gamma_{(x,q)}$ and consider a new stack symbol U and two new state symbols $q_\perp$ and $t_\perp$.

Then $\mathcal{B} = (\Gamma \cup \{U\}, (Q \cup \{q_\perp\}) \times (T \cup \{t_\perp\}), F \times \{t_\perp\}, \Delta)$ where rewrite rules of Δ have one of the following forms:

- form (a) deals with leaves of $Enc(t)$: $\# \longrightarrow (q_\perp, t)(\perp)$ where t is in $\{t_\perp\} \cup \bigcup_{(y,s)\in\Sigma\times\widetilde{Q}} T^{in}_{(y,s)}$;
- form (b) deals with internal nodes of $Enc(t)$:
  $x^c((q_1, t_1)(w_1), (q_2, t_2)(w_2)) \longrightarrow (q, t)(w)$ with (i) $q \in Q_c$, (ii) $t_2, t \in T_{(y,s)}$, $w_2, w \in \Gamma^*_{(y,s)}$ and $(q, t_2, w_2) \longrightarrow (t, w) \in \Delta_{(y,s)}$ for some $(y, s) \in \Sigma \times \widetilde{Q}$ and (iii) $t_1 \in T^{f}_{(x,q)}$.
  Because $\mathcal{B}_{(y,s)}$ is a *vpa*, moves of the stack described by rules of $\Delta_{(y,s)}$ such as $(q, t_2, w_2) \longrightarrow (t, w)$ are respecting q's type that coincides with x^c's type. Therefore rewrite rules of Δ have the form required for a *vtars*. Condition (iii) ensures that the child node sequence of a t's node n with type c and label x is associated with a word of states of $L_{\mathcal{A}}(x, q)$ and therefore that n can be associated with state q;
- form (c) deals with the root: $x^c((q_1, t_1)(w_1), (q_\perp, t_\perp)(\perp)) \longrightarrow (q, t_\perp)(w)$ with (i) $q \in Q_c$, (ii) $t_1 \in T^{f}_{(x,q)}$ and (iii) if $c = up$ then $w = U$ else $w = \perp$.

Conversely let us suppose that $Enc(L) = L(\mathcal{B})$ for some *vtars* $\mathcal{B} = (\Gamma, Q, Q_f, \Delta)$. Following the same intuition as above, we construct a *nsta* $\mathcal{A} = (\Sigma, \widetilde{T}, \eta, F)$ with $\widetilde{T} = (T, (T_{up}, T_{down}, T_{fix}))$ recognizing L as follows:

- $T = \tilde{\Sigma} \times Q$ with $T_c = \Sigma_c \times Q$ for $c \in \{up, down, fix\}$. Given a *tulab* tree t, the simulation by $\mathcal{A}$ of a $\mathcal{B}$'s run r on $Enc(t)$ will associate with a node n of t, with type c and label x, a state (x^c, q) iff the corresponding node in $Enc(t)$ labeled with x^c is associated with state q during the $\mathcal{B}$'s run r;
- $F = \tilde{\Sigma} \times Q_f$;
- the set of transitions η is defined as follows: for each $(x, t) \in \Sigma \times T$, $L_{\mathcal{A}}(x, t)$ is non empty iff $t = (x^c, q)$ for some $q \in Q$ and $L_{\mathcal{A}}(x, (x^c, q)) = L(\mathcal{B}_{(x,c,q)})$ where $\mathcal{B}_{(x,c,q)}$ is a *vpa* over strings of T^*. Roughly speaking $L(\mathcal{B}_{(x,c,q)})$ is the language of strings $(x_1^{c_1}, q_1) \dots (x_n^{c_n}, q_n)$ built with labels encountered (from top to down) on particular paths of $Enc(t)$, each label $x_i^{c_i}$ being coupled with the state q_i associated with the corresponding node during a $\mathcal{B}$'s run. Such paths have to be maximal right branches of $Enc(t)$ whose top node (labeled with $x_1^{c_1}$ and associated with q_1) has to be the left son of a node labeled with x^c and associated with q. More formally, $\mathcal{B}_{(x,c,q)} = (\Gamma, T, T_{(x,c,q)}, \delta)$. So $\mathcal{B}_{(x,c,q)}$ uses, as state alphabet, its input alphabet T itself and, as stack alphabet, $\mathcal{B}$'s stack alphabet. Furthermore:
  - the set δ of rewrite rules is obtained by translating each rule of Δ with the form $x^c(q_1(w_1), q_2(w_2)) \longrightarrow q(w)$ into the set of rules $\{((x^c, q), (x_2^{c_2}, q_2), w_2) \longrightarrow (x^c, q)(w) | x_2^{c_2} \in \tilde{\Sigma}\}$; δ's transitions perform Δ's transitions by forgetting the left parts of these transitions, simulating a pushdown automaton over strings of T^*. Because the relationship between stack words w_2 and w is respecting x^c's type in Δ's rules, it is also respecting (x^c, q)'s type in δ's rules. Therefore $\mathcal{B}_{(x,c,q)}$ is a *vpa*.
  - $T_{(x,c,q)}$ is defined from states associated during a $\mathcal{B}$'s run with nodes that are left sons of nodes labeled with x^c and associated with q (such left sons are therefore tops of maximal right branches). Formally: $T_{(x,c,q)} = \{(x_1^{c_1}, q_1) | x_1^{c_1} \in \tilde{\Sigma}, \exists w_1, w_2, w \in \Gamma^*, q_2 \in Q, x^c(q_1(w_1), q_2(w_2)) \longrightarrow q(w) \in \Delta\}$.

By construction $\mathcal{A}$ is clearly a *nested sibling tree automaton*. One verifies without major difficulties that $\mathcal{A}$'s runs on t effectively simulate $\mathcal{B}$'s runs on $Enc(t)$. Therefore L is a language of $NSTL(\Sigma)$. □

Proposition 5.3 shows the equivalence (up to the firstChild-nextSibling encoding) between the two classes of tree languages recognized, on the one hand by a *nsta*, and on the other hand by a *vtars*.

This equivalence delivers more precise results about robustness properties of *vtars* languages. Indeed it is shown in [10] that the class of *vtam* languages is closed under boolean operations and that the emptiness and inclusion problems are decidable for *vtam*s. Because a *vtars* automaton is a particular case of a *vtam* automaton, the decidability of the emptiness and inclusion problems for *vtars*

automata directly follows from [10] but it is not the case for the closure properties of this strict subclass of the *vtam* languages. Therefore Proposition 5.3 delivers (a) the closure under boolean operations of the class of *vtars* languages and (b) better complexities of the emptiness and inclusion problems for *vtars* automata: the emptiness problem is decidable in PTIME for *vtars* automata while, in [10] it is proved decidable in EXPTIME for *vtam* languages, and the inclusion problem is decidable in EXPTIME for *vtars* automata while, in [10] it is proved decidable in 2-EXPTIME for *vtam* languages.

6. Conclusion

In this paper we have proposed a new framework to express and validate some context-free style structural constraints on unranked labeled trees. More precisely this framework introduces (i) *typed unranked labeled* trees that are trees defined on a typed tree domain and (ii) *nested sibling tulab tree automata* (*nsta*) to deal with them. *Nsta*s are defined by replacing regular constraints on sibling nodes of regular unranked labeled trees languages by visibly pushdown constraints. This leads to a new class of labeled trees languages, *NSTL*, that:

- is strictly more expressive that the regular languages class;
- is closed under boolean operations;
- has decision procedures for the classical decision problems;
- has a characterization in terms of expressed logic properties;
- can be related, through adequate encodings, with the class of regular sets of 3-ary trees and with the class of sets of binary trees accepted by particular visibly tree automata with memory.

To our knowledge, no such class of unranked labeled trees possessing all these good properties has been exhibited beyond the class of regular ones. Therefore we think that *NSTL* is a quite interesting class as well as *nsta*s that provide efficient tools to validate structural constraints beyond regular ones.

7. Appendix. Proposition 5.1 *(Equivalence with MSO_μ)*

Formula $\phi_M(S, U_1, \ldots, U_{m_u}, F_1, \ldots, F_{m_f}, D_1, \ldots, D_{m_d}))$

Recall that M is a visibly pushdown word automaton $M = (\widetilde{X}, P, P_{in}, , \Gamma, \delta, F)$ where P is a finite set of states and $P_{in}, F \subseteq P = \{p_1, \ldots, p_n\}$ being respectively the set of initial and final states, and $\Gamma = \{\gamma_1, \ldots, \gamma_m\} \cup \{\bot\}$ is a finite stack alphabet. Formula $\phi_M(S, U_1, \ldots, U_{m_u}, F_1, \ldots, F_{m_f}, D_1, \ldots, D_{m_d}))$ is as follows:

$$\exists X_{p_1} \dots X_{p_n} \quad \exists X_{\gamma_1} \dots X_{\gamma_m}$$

$$\forall x \in S, (\bigvee_{1 \leq i \leq n} x \in X_{p_i}) \wedge (\bigwedge_{1 \leq i,j \leq n,\, i \neq j} x \in X_{p_i} \Rightarrow \neg(x \in X_{p_j})) \ \wedge$$

$$\forall x \in S, (\exists y \in S, \mu(x,y)) \Rightarrow$$
$$((\bigvee_{1 \leq i \leq m} x \in X_{\gamma_i}) \wedge (\bigwedge_{1 \leq i,j \leq m,\, i \neq j} x \in X_{\gamma_i} \Rightarrow \neg(x \in X_{\gamma_j}))) \ \wedge$$

$$\forall x \in S, (\bigvee_{1 \leq i \leq m_f} x \in F_i) \Rightarrow$$
$$(\bigvee_{(p_i,f_j,p_k) \in \delta} x \in F_j \wedge x \in X_{p_i} \wedge \exists y\, \mathsf{sibling}(x,y) \wedge y \in X_{p_k}) \ \wedge$$
$$\forall x \in S, (\bigvee_{1 \leq i \leq m_u} x \in U_i) \Rightarrow$$
$$(\bigvee_{(p_i,u_j,p_k,\gamma_l) \in \delta} x \in U_j \wedge x \in X_{p_i} \wedge \exists y\, \mathsf{sibling}(x,y) \wedge y \in X_{p_k} \wedge x \in X_{\gamma_l}) \wedge$$

$$\forall x \in S, (\bigvee_{1 \leq i \leq m_d} x \in D_i) \Rightarrow (\bigvee_{(p_i,d_j,\gamma_l,p_k) \in \delta} x \in D_j \wedge x \in X_{p_i} \wedge$$
$$\exists y\, \mathsf{sibling}(x,y) \wedge y \in X_{p_k} \wedge \exists z\, \mu(z,x) \wedge z \in X_{\gamma_l}) \ \wedge$$
$$\forall x \in S (\forall y \in S, \neg(\mathsf{sibling}(y,x)) \Rightarrow \bigvee_{p \in P_{in}} x \in X_p) \wedge (\forall y \in S, \neg(\mathsf{sibling}(x,y)) \Rightarrow$$
$$\bigvee_{p \in F} x \in X_p).$$

The first part of the formula says that each position is labeled by exactly one state; the second one that positions of type *up* (with a matching position of type *down*) are labeled with a symbol push on the stack at this position. The following three parts state the existence of some transition between each position and its successor (in $\leq$). Finally, the last part states that the first position is labeled with some initial state and the last one by a final state.

Construction of an automaton accepting models of $\mu(x,y)$

We recall that the bottom of the stack contains a special symbol $\perp$ that cannot be pop out of the stack. The automaton accepting trees that model this formula is $\mathcal{A} = (\Sigma \times \{0,1\}^{n+m}, \widetilde{Q}, \delta, F)$ where

- $\widetilde{Q} = (\{q_o^u, q_o^f, q_o^d, q_x, q_y, q_e^u, q_e^f, q_e^d\}, (\{q_o^u, q_x, q_e^u\}, \{q_o^f, q_e^f\}, \{q_o^d, q_y, q_e^d\}))$;
- $F = \{q_e^u, q_e^f, q_e^d\}$;
- δ satisfies:
  - $(\alpha, q_o^u, L_0), (\alpha, q_o^f, L_0), (\alpha, q_o^d, L_0) \in \delta$ with α has its positions x and y set to 0;
  - $(\alpha, q_e^u, L_1), (\alpha, q_e^f, L_1), (\alpha, q_e^d, L_1) \in \delta$ with α has its positions x and y set to 0;
  - $(\alpha, q_x, L_0) \in \delta$ with α has its position x set to 1 and its positions y set to 0;
  - $(\alpha, q_y, L_0) \in \delta$ with α has its position y set to 1 and its positions x set to 0;
  - $(\alpha, q_e^u, L_t), (\alpha, q_e^f, L_t), (\alpha, q_e^d, L_t) \in \delta$ with α has its positions x and y set to 0.

The three visibly pushdown word languages L_0, L_1 and L_t are defined respectively by:

- for L_0: (q_0^u, p, p, γ), (q_0^f, p, p), (q_0^d, p, γ, p), $(q_0^d, p, \bot, p)$ where p is the unique state, initial and final;
- for L_1: (q_0^u, p, p, γ), (q_0^f, p, p), (q_0^d, p, γ, p), $(q_0^d, p, \bot, p)$, (q_e^u, p, p', γ), (q_e^f, p, p'), (q_e^d, p, γ, p'), $(q_e^d, p, \bot, p')$, (q_0^u, p', p', γ), (q_0^f, p', p'), (q_0^d, p', γ, p'), $(q_0^d, p', \bot, p')$ where the states are $\{p, p'\}$, p being initial and p' final and γ is the unique stack symbol;
- for L_t:
  (q_0^u, p, p, γ), (q_0^f, p, p), (q_0^d, p, γ, p), $(q_0^d, p, \bot, p)$, (q_x, p, p', γ_x), (q_0^u, p', p', γ), (q_0^f, p', p'), (q_0^d, p', γ, p'), $(q_0^d, p', \bot, p')$, (q_y, p', γ_x, p''), $(q_0^u, p'', p'', \gamma)$, (q_0^f, p'', p''), $(q_0^d, p'', \gamma, p'')$, $(q_0^d, p'', \bot, p'')$ where the states are $\{p, p', p''\}$, p being initial and p'' final and γ, γ_x are the stack symbols.

Note first that x and y have to be sibling nodes. With the nodes corresponding to x (resp. y) will be associated the state q_x (resp. q_y). The path from the father node of x,y to the root is labeled with q_e (in fact, with q_e^u, q_e^f, q_e^d depending on the type of the node). The other nodes will be labeled with q_o (in fact, with q_o^u, q_o^f, q_o^d depending on the type of the node).

The language L_o contains only words of q_0 (q_o^u, q_o^f, q_o^d): the x and y are below none of these nodes. The language L_1 contains words made of the sequence of q_o, followed by a q_e and terminated by an another sequence of q_o. The nodes x, y are below the nodes labeled by q_e. Finally, the language L_t contains words made of a sequence of q_o, followed by a q_x, then a sequence of q_o, then q_y and is terminated by an another sequence of q_o. The stack symbol γ_x is used to ensure the matching between the node labeled by q_x and the node labeled by q_y.

References

[1] S. Abiteboul and P. Buneman, *Data on the Web: From Relations to Semi-structured Data and XML* (1999).

[2] R. Alur, S. Chaudhuri and P. Madhusudan, A fixpoint calculus for local and global program flows, Symposium on Principles of Programming Languages, POPL 2006 (2006) 153–165.

[3] R. Alur, S. Chaudhuri and P. Madhusudan, Languages of nested trees, Computer-Aided Verification, CAV 2006 (2006) 329–342.

[4] Rajeev Alur and P. Madhusudan, Visibly pushdown languages, Chicago, USA. 36th ACM symposium on Theory of Computing (2004) 202–211.

[5] M. Bojanczyck and T. Colcombet, Tree-walking automata cannot be determinized. ICALP (2004) 246–256.

[6] T. Bray, J.P. Paoh and C.M. Sperberg-McQueen, Extensible markup language (xml) 1.0, http://www.w3org/TR/1998/REC-xml-19980210/ (1998).

[7] A. Bruggemann, M. Murata and D. Wood, *Regular tree and regular hedge languages over unranked alphabets*, Technical Report 2001–05. HKUST TCS Center (1998).

[8] H. Comon and V. Cortier, Tree automata with one memory set constraints and cryptographic protocols. *Theoret. Comput. Sci.* **331** (2005) 143–214.

[9] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison and M. Tommasi, Tree automata techniques and applications, available on: http://www.grappa.univ-lille3.fr/tata (2007), release October, 12th (2007).

[10] H. Comon-Lundh, F. Jacquemard and N. Perrin, Tree automata with memory, visibility and structural constraints, In *Foundations of Software Science and Computational Structures, 10th International Conference, FOSSACS 2007. Lect. Notes Comput. Sci.* **4423** (2007) 168–182.
[11] D. Lugiez and S. DalZilio, *Xml schema, tree logic and sheaves automata*, Technical Report RR-4631, Inria (2002).
[12] F. Neven, *Automata, logic and xml*. CSL (2002) 2–26.
[13] C. Pitcher, Visibly pushdown expression effects for xml stream processing, Programming Languages Technologies for XML. PLAN-X (2005) 5–19.
[14] T. Schwentick, *Tree, automata and xml*. Paris, PODS (2004).
[15] L. Segoufin, *Typing and quering xml documents: some complexity bounds*. San Diego CA, PODS (2003) 167–178.
[16] H. Seidl, T. Schwentick and A. Muscholl, *Numerical document queries*. San Diego CA, PODS (2003).
[17] J.W. Thatcher and J.B. Wright, Generalized finite automata with an application to a decision problem of second-order logic. *Mathematical System Theory* **2** 57–82 (1968).

Communicated by S. Tison.
Received April 8, 2008. Accepted January 19, 2009.