

# Visibly Pushdown Transducers with Look-Ahead\*

Emmanuel Filiot<sup>1</sup> and Frédéric Servais<sup>2</sup>

<sup>1</sup> Université Libre de Bruxelles

<sup>2</sup> Hasselt University and Transnational University of Limburg

**Abstract.** Visibly Pushdown Transducers (VPT) form a subclass of pushdown transducers. In this paper, we investigate the extension of VPT with visibly pushdown look-ahead (VPT<sub>la</sub>). Their transitions are guarded by visibly pushdown automata that can check whether the well-nested subword starting at the current position belongs to the language they define. First, we show that VPT<sub>la</sub> are not more expressive than VPT, but are exponentially more succinct. Second, we show that the class of deterministic VPT<sub>la</sub> corresponds exactly to the class of functional VPT, yielding a simple characterization of functional VPT. Finally, we show that while VPT<sub>la</sub> are exponentially more succinct than VPT, checking equivalence of functional VPT<sub>la</sub> is, as for VPT, EXPT-C. As a consequence, we show that any functional VPT is equivalent to an unambiguous one.

## 1 Introduction

Visibly pushdown transducers (VPT) [17,9] form an interesting subclass of pushdown transducers (PT). Several problems that are undecidable for PT are decidable for VPT, noticeably: functionality is decidable in PTIME,  $k$ -valuedness in NPTIME and equivalence of functional VPT is EXPT-C [9].

Visibly pushdown machines [1], automata (VPA) or transducers, are pushdown machines such that the behavior of the stack, i.e. whether it pushes or pops, is visible in the input word. Technically, the input alphabet is partitioned into call, return and internal symbols. When reading a call the machine must push a symbol on the stack, when reading a return symbol it must pop and when reading an internal symbol it cannot touch the stack. The partitioning of the input alphabet induces a nesting structure of the input words [2]. A call symbol delimits an additional level of nesting, while a return symbol is a position in the word that ends a level of nesting. A word is well-nested if each call, respectively each return, has a matching return, respectively a matching call. Visibly pushdown transductions are transductions that can be defined by VPT.

Unranked trees in their linear form (such as XML documents) can be viewed as well-nested words. VPT are therefore a suitable formalism for unranked tree transformations. In particular, they can express operations such as node deletion, renaming and

---

\* This research was supported by the projects: Gasics: “Games for Analysis and Synthesis of Interactive Computational Systems”, <http://www.ulb.ac.be/di/gasics/>, and Moves: “Fundamental Issues in Modelling, Verification and Evolution of Software”, <http://moves.ulb.ac.be>, a PAI program funded by the Federal Belgian Government. Partially funded by the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under the FET-Open grant agreement FOX, No. FP7-ICT-233599.

insertion. Furthermore, over well-nested words, a simple and expressive subclass of VPT, the class of well-nested VPT [9], is closed under composition and has a decidable type checking problem. In the setting of XML documents, VPA, as they read the tree in a left-to-right depth-first traversal manner, are well-suited for streaming validation [11,15] or streaming XML queries [10]. In the same way well-nested VPT are amenable to define streaming transformations.

In this paper, one of our motivations is to give a simple characterization of functional VPT that can be checked easily. Deterministic VPT are not expressive enough to capture all functional VPT, as for instance swapping the first and last letters of a word cannot be done deterministically. Instead of non-determinism, we show that some limited inspection of the longest well-nested subword starting at the current position (called the *current well-nested prefix*) is required to capture (non-deterministic) functional VPT. More precisely, we show that functional VPT-transductions are captured by deterministic VPT extended with visibly pushdown look-aheads that inspect the current well-nested prefix. Moreover, inspecting the current well-nested prefix is somehow the minimal necessary information to capture all functional VPT.

In this paper, we therefore introduce and investigate the class of VPT with visibly pushdown look-ahead. A VPT with visibly pushdown look-ahead ( $VPT_{la}$ ) is a VPT such that call transitions are guarded with visibly pushdown automata (VPA). When reading a call at position  $i$ , a  $VPT_{la}$  can apply a call transition provided the longest well-nested word starting at position  $i$  is included in the language of the VPA of the transition. In the same way one can define VPA with look-ahead ( $VPA_{la}$ ). Our main contributions are the following:

1.  $VPT_{la}$  (resp.  $VPA_{la}$ ) are as expressive as VPT (resp. VPA), but exponentially more succinct.

For this we present an exponential construction that shows how a VPT can simulate look-aheads. Moreover we show this exponential blow-up is unavoidable.

2. Deterministic  $VPT_{la}$  and functional VPT are equally expressive.

This equivalence is obtained by a construction (which is also exponential) that replaces the non-determinism of the functional VPT with deterministic look-ahead. This also yields a simple characterization of functional VPT.

3. Functional VPT and unambiguous VPT are equally expressive.

As an application of look-aheads, we show that a nice consequence of the constructions involved in contributions 1 and 2 is that functional VPT are effectively characterized by unambiguous VPT. This result was already known for finite-state transducers [4,14,5] and here we extend it to VPT with rather simple constructions based on the concept of look-aheads. This characterization of functional finite-state transducers has been generalized to  $k$ -valued and  $k$ -ambiguous finite-state transducers [18] and recently with a better upper-bound [13] based on lexicographic decomposition of transducers.

4. Equivalence of functional  $VPT_{la}$  (resp  $VPA_{la}$ ) is, as for VPT (resp VPA), EXPT-C.

Therefore even though  $VPT_{la}$  are exponentially more succinct than VPT, testing equivalence of functional  $VPT_{la}$  is not harder than for functional VPT. This is done in two steps. First one checks equivalence of the domains. Then one checks that the union of the two transducers is still functional. We show that testing functionality is EXPT-C for  $VPT_{la}$ : get rid of the look-aheads with an exponential blow-up and test in PTIME

**Table 1.** Decision Problems for VPA, VPA<sub>la</sub>, VPT, VPT<sub>la</sub>

	VPA [1]	VPA <sub>la</sub>	VPT [9]	VPT <sub>la</sub>
Emptiness	PTIME	EXPT-C	PTIME	EXPT-C
Universality	EXPT-C	EXPT-C	NA	NA
Inclusion	EXPT-C	EXPT-C	EXPT-C	EXPT-C
Equivalence	EXPT-C	EXPT-C	EXPT-C (for fVPT)	EXPT-C (for fVPT)
Functionality	NA	NA	PTIME	EXPT-C

the functionality of the constructed VPT. To verify that the domains are equivalent, the naive technique (removing the look-aheads and then verifying the mutual inclusion of the domains) yields a doubly exponential algorithm. Instead, we show that the domains of VPT<sub>la</sub> are linearly reducible to alternating top-down tree automata. Testing the equivalence of such automata can be done in EXPT [3].

Table 1 summarizes the complexity of decision problems for VPA<sub>la</sub> and VPT<sub>la</sub>.

*Variants of look-ahead.* We discuss in [16] some variants of look-ahead. The closure by look-ahead (Theorem 1) and the equivalence between deterministic VPT<sub>la</sub> and functional VPT (Theorem 2) still hold when the look-ahead can inspect the whole suffix and can also be triggered on return transitions. However, when the look-ahead can inspect only the current well-nested prefix of the form *cwr* (corresponding to the first subtree of the current hedge in a tree), it is not sufficient to express all functional VPT with determinism.

*Related Works.* Regular look-aheads have been mainly considered for classes of tree transducers, where a transition can be fired provided the current subtree belongs to some regular tree language. For instance, regular look-aheads have been added to *top-down (ranked) tree transducers* in order to obtain a robust class of tree transducers that enjoys good closure properties wrt composition [6], or to *macro tree transducers* (MTT) [8]. For top-down tree transducers, adding regular look-ahead strictly increases their expressive power while MTT are closed by regular look-ahead [8]. Another strong result shows that every functional top-down tree transduction can be defined by a *deterministic top-down tree transducer with look-ahead* [7].

Trees over an alphabet  $\Sigma$  can be linearized as well-nested words over the structured alphabet  $\Sigma_c = \{c_a \mid a \in \Sigma\}$ ,  $\Sigma_r = \{r_a \mid a \in \Sigma\}$ . It is well-known that unranked trees can be represented by binary trees via the classical first-child next-sibling encoding (fcns). Top-down (ranked) tree transducers can thus be used as unranked tree transducers on fcns encodings of unranked trees. Inspecting a subtree in the fcns encoding corresponds to inspecting the first subtree and its next-sibling subtrees in an unranked tree, which in turn corresponds to inspecting the current longest well-nested prefix in their linearization. However top-down tree transducers and VPT are incomparable: top-down tree transducers can copy subtrees while VPT cannot, and VPT support concatenation of tree sequences while top-down tree transducers cannot. For example, the transformation that removes the *g* node in unranked trees of the form  $f(g(a, \dots, a), b, b, \dots, b)$  produces trees of the form  $f(a, a, \dots, a, b, \dots, b)$ . This transformation can easily be defined by a VPT, but not by a top-down ranked tree transducers with the fcns encoding [12,9]. Indeed, in the fcns encoding, this transformation maps any tree of the form  $f(g(t_a, t_b), \perp)$  to  $f(t_a.t_b, \perp)$ , where  $t_a, t_b, t_a.t_b$  are the binary encodings of the hedges

$(a, \dots, a), (b, \dots, b), (a, \dots, a, b, \dots, b)$  respectively:

$$\begin{aligned} t_a &= a(\perp, a(\perp, \dots a(\perp, \perp) \dots)) & t_b &= b(\perp, b(\perp, \dots b(\perp, \perp) \dots)) \\ t_a.t_b &= a(\perp, a(\perp, \dots a(\perp, b(\perp, b(\perp, \dots b(\perp, \perp) \dots))) \dots) \end{aligned}$$

Therefore, this transformation requires to move the subtree  $t_b$  (whose size may be unbounded) as a leaf of the subtree  $t_a$  (whose size may also be unbounded). This cannot be done by a top-down tree transducer, but can be defined by some MTT thanks to parameters (some parameter will store the entire subtree  $t_b$  while evaluating  $t_a$ ). A detailed comparison of VPT and tree transducers can be found in [16].

Modulo the former encodings, MTT subsume VPT [9] and as we said before, there is a correspondence between the two notions of look-aheads, for VPT and MTT respectively. However it is not clear how to derive our results on closure by look-aheads from the same result on MTT, as the latter highly relies on parameters and it would require back-and-forth encodings between the two models. The direct construction we give in this paper is self-contained and allows one to derive the characterization of functional VPT as unambiguous VPT by a careful analysis of the construction.

*An extended version of the paper with all proofs can be found in [16].*

## 2 Visibly Pushdown Languages and Transductions

All over this paper,  $\Sigma$  denotes a finite alphabet partitioned into two disjoint sets  $\Sigma_c, \Sigma_r$ , denoting respectively the *call* and *return* alphabets. We denote by  $\Sigma^*$  the set of (finite) words over  $\Sigma$  and by  $\epsilon$  the empty word. The length of a word  $u$  is denoted by  $|u|$ . The set of *well-nested* words  $\Sigma_{\text{wn}}^*$  is the smallest subset of  $\Sigma^*$  such that  $\epsilon \in \Sigma_{\text{wn}}^*$  and for all  $c \in \Sigma_c$ , all  $r \in \Sigma_r$ , all  $u, v \in \Sigma_{\text{wn}}^*$ ,  $cur \in \Sigma_{\text{wn}}^*$  and  $uv \in \Sigma_{\text{wn}}^*$ .

A *visibly pushdown automaton* (VPA) [1] on finite words over  $\Sigma$  is a tuple  $A = (Q, I, F, \Gamma, \delta)$  where  $Q$  is a finite set of states,  $I \subseteq Q$  the set of initial states,  $F \subseteq Q$  the set of final states,  $\Gamma$  the (finite) stack alphabet, and  $\delta = \delta_c \uplus \delta_r$  where  $\delta_c \subseteq Q \times \Sigma_c \times \Gamma \times Q$  are the *call transitions*,  $\delta_r \subseteq Q \times \Sigma_r \times \Gamma \times Q$  are the *return transitions*<sup>1</sup>.

On a call transition  $(q, a, \gamma, q') \in \delta_c$ ,  $\gamma$  is pushed onto the stack and the control goes from  $q$  to  $q'$ . On a return transition  $(q, a, \gamma, q') \in \delta_r$ ,  $\gamma$  is popped from the stack.

A *configuration* of a VPA is a pair  $(q, \sigma) \in Q \times \Gamma^*$ . A *run* of  $T$  on a word  $u = a_1 \dots a_l \in \Sigma^*$  from a configuration  $(q, \sigma)$  to a configuration  $(q', \sigma')$  is a finite sequence  $\rho = \{(q_k, \sigma_k)\}_{0 \leq k \leq l}$  such that  $q_0 = q, \sigma_0 = \sigma, q_l = q', \sigma_l = \sigma'$  and for each  $1 \leq k \leq l$ , there exists  $\gamma_k \in \Gamma$  such that either  $(q_{k-1}, a_k, \gamma_k, q_k) \in \delta_c$  and  $\sigma_k = \sigma_{k-1} \gamma_k$  or  $(q_{k-1}, a_k, \gamma_k, q_k) \in \delta_r$  and  $\sigma_{k-1} = \sigma_k \gamma_k$ . The run  $\rho$  is *accepting* if  $q_0 \in I, q_l \in F$  and  $\sigma_0 = \sigma_l = \perp$ . A word  $w$  is *accepted* by  $A$  if there exists an accepting run of  $A$  over  $w$ .  $L(A)$ , the *language* of  $A$ , is the set of words accepted by  $A$ . A language  $L$  over  $\Sigma$  is a *visibly pushdown language* if there is a VPA  $A$  over  $\Sigma$  such that  $L(A) = L$ .

As finite-state transducers extend finite-state automata with outputs, visibly pushdown transducers extend visibly pushdown automata with outputs [9]. To simplify notations, we suppose that the output alphabet is  $\Sigma$ , but our results still hold for an arbitrary

<sup>1</sup> In contrast to [1], we do not consider *internal* symbols  $i$ , as they can be simulated by a (unique) call  $c_i$  followed by a (unique) return  $r_i$ . We make this assumption to simplify proofs and notations. Moreover, we do not allow return transition on  $\perp$  and we require the final stack to be empty. This implies that all accepted words are well-nested. All our results extend easily to alphabets with internal symbols and to VPT that accept by final state only.

output alphabet. Informally, the stack behavior of a VPT is similar to the stack behavior of visibly pushdown automata (VPA). On a call symbol, the VPT pushes a symbol on the stack and produces some output word (possibly empty and not necessarily well-nested), on a return symbol, it must pop the top symbol of the stack and produce some output word (possibly empty) and on an internal symbol, the stack remains unchanged and it produces some output word.

**Definition 1.** A *visibly pushdown transducer* (VPT) on finite words over  $\Sigma$  is a tuple  $T = (Q, I, F, \Gamma, \delta)$  where  $Q$  is a finite set of states,  $I \subseteq Q$  is the set of initial states,  $F \subseteq Q$  the set of final states,  $\Gamma$  is the stack alphabet,  $\delta = \delta_c \uplus \delta_r$  the (finite) transition relation, with  $\delta_c \subseteq Q \times \Sigma_c \times \Sigma^* \times \Gamma \times Q$ ,  $\delta_r \subseteq Q \times \Sigma_r \times \Sigma^* \times \Gamma \times Q$ .

Configurations and runs are defined similarly as VPA. Given a word  $u = a_1 \dots a_l \in \Sigma^*$  and a word  $v \in \Sigma^*$ ,  $v$  is an *output* of  $u$  by  $T$  if there exists an accepting run  $\rho = \{(q_k, \sigma_k)\}_{0 \leq k \leq l}$  on  $u$  and  $l$  words  $v_1, \dots, v_l$  such that  $v = v_1 \dots v_l$  and for all  $0 \leq k < l$ , there is a transition of  $T$  from  $(q_k, \sigma_k)$  to  $(q_{k+1}, \sigma_{k+1})$  that produces the output  $v_{k+1}$  on input letter  $a_{k+1}$ . We write  $(q, \sigma) \xrightarrow{u/v} (q', \sigma')$  when there exists a run on  $u$  from  $(q, \sigma)$  to  $(q', \sigma')$  producing  $v$  as output. A transducer  $T$  defines the binary word relation  $\llbracket T \rrbracket = \{(u, v) \mid \exists q \in I, q' \in F, (q, \perp) \xrightarrow{u/v} (q', \perp)\}$ .

A *transduction* is a binary relation  $R \subseteq \Sigma^* \times \Sigma^*$ . We say that a transduction  $R$  is a VPT-transduction if there exists a VPT  $T$  such that  $R = \llbracket T \rrbracket$ . A transduction  $R$  is *functional* if for all  $u \in \Sigma^*$ , there exists at most one  $v \in \Sigma^*$  such that  $(u, v) \in R$ . A VPT  $T$  is *functional* if  $\llbracket T \rrbracket$  is functional, and we denote by fVPT the class of functional VPT. Two transducers  $T_1, T_2$  are *equivalent* if  $\llbracket T_1 \rrbracket = \llbracket T_2 \rrbracket$ . It is known [9] that functionality is decidable in PTIME for VPT and equivalence of functional VPT is EXPT-C. Finally, a VPT is *unambiguous* if there is at most one accepting run per input word. In particular, any unambiguous VPT is functional. Unambiguity can be checked in PTIME [9].

For any input word  $u \in \Sigma^*$ , we denote by  $R(u)$  the set  $\{v \mid (u, v) \in R\}$ . Similarly, for a VPT  $T$ , we denote by  $T(u)$  the set  $\llbracket T \rrbracket(u)$ . If  $R$  is functional, we confound  $R(u)$  (which is at most of cardinality 1) and the unique image of  $u$  if it exists. The *domain* of  $T$  (denoted by  $Dom(T)$ ) is the domain of  $\llbracket T \rrbracket$ . Note that the domain of  $T$  contains only well-nested words, which is not necessarily the case of the codomain.

*Example 1.* Let  $\Sigma_c = \{c, a\}$ ,  $\Sigma_r = \{r\}$  be the call and return symbols of the alphabet. The following VPT  $T$  transforms a word as follows: (i)  $a$  and  $r$  are mapped to  $a$  and  $r$  respectively; (ii)  $c$  is mapped either to  $c$  if no  $a$  appears in the longest well-nested word starting at  $c$ , and to  $a$  if an  $a$  appears. E.g.  $ccrrarcr$  is mapped to  $acrrarcr$ , and  $ccrrrcarr$  to  $aacrraraarr$ .

The VPT  $T = (Q, I, F, \Gamma, \delta)$  is defined by  $Q = \{q, q_a, q_{-a}\}$ ,  $I = \{q\}$ ,  $F = Q$ ,  $\Gamma = \{\gamma, \gamma_a, \gamma_{-a}\}$  and  $\delta$  contains the following transitions:

$$\begin{array}{lll}
 q \text{ or } q_a & \xrightarrow{c/a, \gamma} & q_a & q \text{ or } q_a & \xrightarrow{c/a, \gamma_a} & q & q & \xrightarrow{c/c, \gamma_{-a}} & q_{-a} \\
 q \text{ or } q_a & \xrightarrow{a/a, \gamma} & q & q_{-a} & \xrightarrow{c/c, \gamma_{-a}} & q_{-a} & & & \\
 q \text{ or } q_{-a} & \xrightarrow{r/r, \gamma_a} & q_a & q \text{ or } q_{-a} & \xrightarrow{r/r, \gamma} & q & q_{-a} & \xrightarrow{r/r, \gamma_{-a}} & q_{-a}
 \end{array}$$

The state  $q_a$ , resp.  $q_{-a}$ , means that there is, resp. is not, an  $a$  in the longest well-nested word that starts at the current position. The state  $q$  indicates that there is no

constraints on the appearance of  $a$ . If  $T$  is in state  $q$  and reads a  $c$ , there are two cases: it outputs an  $a$  or a  $c$ . If it chooses to output an  $a$ , then it must check that an  $a$  occurs later. There are again two cases: either  $T$  guesses there is an  $a$  in the well-nested word that starts just after  $c$  and takes the transitions  $q \xrightarrow{c/a, \gamma} q_a$ , or it guesses an  $a$  appears in the well-nested word that starts after the matching return of  $c$ , in that latter case it takes the transition  $q \xrightarrow{c/a, \gamma_a} q$  and uses the stack symbol  $\gamma_a$  to carry over this information. If on  $c$  it chooses to output  $c$ , it must check that there is no  $a$  later by using the transition  $q \xrightarrow{c/a, \gamma_{-a}} q_{-a}$ . Other cases are similar.

### 3 VPT with Visibly Pushdown Look-Ahead

Given a word  $w$  over  $\Sigma$  we denote by  $\text{pref}_{\text{wn}}(w)$  the longest well-nested prefix of  $w$ . E.g.  $\text{pref}_{\text{wn}}(ccrcr) = \epsilon$  and  $\text{pref}_{\text{wn}}(crc) = cr$ . We define a VPT  $T$  with visibly pushdown look-ahead (simply called look-ahead in the sequel) informally as follows. The look-ahead is given by a VPA  $A$  without initial state. On a call symbol  $c$ ,  $T$  can trigger the look-ahead from a state  $p$  of the VPA (which depends on the call transition). The look-ahead tests membership of the longest well-nested prefix of the current suffix (that starts by the letter  $c$ ) to  $L(A, p)$ , where  $(A, p)$  is the VPA  $A$  with initial state  $p$ . If the prefix is in  $L(A, p)$  then the transition of  $T$  can be fired. When we consider nested words that encode trees, look-aheads correspond to inspecting the subtree rooted at the current node and all right sibling subtrees (in other words, the current hedge). Formally:

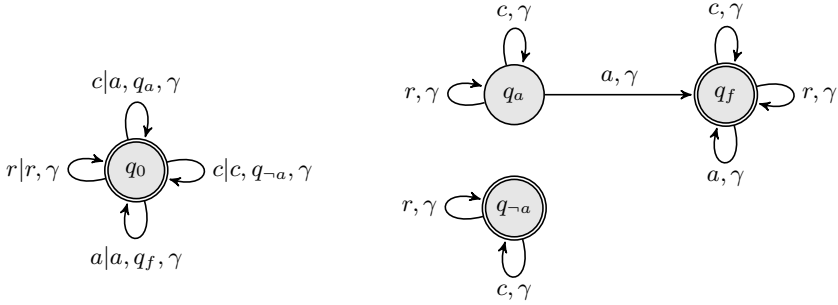
**Definition 2.** A VPT with look-ahead ( $\text{VPT}_{\text{la}}$ ) is a pair  $T_{\text{la}} = (T, A)$  where  $A$  is a VPA  $A = (Q^{\text{la}}, F^{\text{la}}, \Gamma^{\text{la}}, \delta^{\text{la}})$  without initial state and  $T$  is a tuple  $T = (Q, q_0, F, \Gamma, \delta)$  such that  $Q$  is a finite set of states,  $q_0 \in Q$  is an initial state,  $F \subseteq Q$  is a set of final states,  $\Gamma$  is a stack alphabet, and  $\delta = \delta_c \uplus \delta_r$  is a transition relation such that  $\delta_c \subseteq Q \times \Sigma_c \times \Sigma^* \times Q^{\text{la}} \times \Gamma \times Q$  and  $\delta_r \subseteq Q \times \Sigma_r \times \Sigma^* \times \Gamma \times Q$ .

A VPA with look-ahead ( $\text{VPA}_{\text{la}}$ ) is defined similarly.

Let  $u \in \Sigma^*$ . A run of  $T_{\text{la}}$  on  $u = a_1 \dots a_l$  is a sequence of configurations  $\rho = \{(q_k, \sigma_k)\}_{0 \leq k \leq l}$  such that, for all  $k < l$ , there exist  $\gamma \in \Gamma$  and  $v_{k+1} \in \Sigma^*$  such that (i) if  $a_{k+1} \in \Sigma_r$ , then  $\sigma_{k+1}\gamma = \sigma_k$  and  $(q_k, a_{k+1}, v_{k+1}, \gamma, q_{k+1}) \in \delta_r$ ; (ii) if  $a_{k+1} \in \Sigma_c$ , then  $\sigma_{k+1} = \sigma_k\gamma$ , and there exists  $p \in Q^{\text{la}}$  such that  $(q_k, a_{k+1}, v_{k+1}, p, \gamma, q_{k+1}) \in \delta_c$  and  $\text{pref}_{\text{wn}}(a_{k+1} \dots a_l) \in L(A, p)$ . The run  $\rho$  is accepting if  $\sigma_0 = \sigma_l = \perp$  and  $q_l \in F$ . The word  $v_1 \dots v_l$  is an output of  $u$ .

The  $\text{VPT}_{\text{la}}$   $T_{\text{la}}$  is *deterministic* if for all transitions  $(q, c, v_1, p_1, \gamma_1, q_1) \in \delta_c$  and  $(q, c, v_2, p_2, \gamma_2, q_2) \in \delta_c$ , if  $v_1 \neq v_2$  or  $\gamma_1 \neq \gamma_2$  or  $q_1 \neq q_2$  or  $p_1 \neq p_2$ , then  $L(A, p_1) \cap L(A, p_2) = \emptyset$ ; and for all transitions  $(q, r, v_1, \gamma_1, q_1) \in \delta_r$  and  $(q, r, v_2, \gamma_2, q_2) \in \delta_r$  we have  $v_1 = v_2$ ,  $\gamma_1 = \gamma_2$  and  $q_1 = q_2$ . Note that deciding whether some  $\text{VPT}_{\text{la}}$  is deterministic can be done in PTIME. One has to check that for each state  $q$  and each call symbol  $c$ , the VPL guarding transitions from state  $q$  and reading  $c$  are *pairwise* disjoint. The number of states of a  $\text{VPT}_{\text{la}}$  is the number of states of the transducer plus the number of states of the look-ahead.

*Example 2.* A  $\text{VPT}_{\text{la}}$  is represented in Figure 1. The look-ahead automaton is depicted on the right, while the transducer in itself is on the left. It defines the transduction



**Fig. 1.** A  $VPT_{1a}$  (left) and its look-ahead (right) on  $\Sigma_c = \{c, a\}$  and  $\Sigma_r = \{r\}$

of Example 1. When starting in state  $q_a$ , respectively  $q_{-a}$ , the look-ahead automaton accepts well-nested words that contains an  $a$ , respectively does not contain any  $a$ . When starting in state  $q_f$  it accepts any well-nested word. The transducer rewrites  $c$  symbols into  $a$  if the well-nested word starting at  $c$  contains an  $a$  (transition on the top), otherwise it just copy a  $c$  (transition on the right). This is achieved using the  $q_a$  and  $q_{-a}$  states of the look-ahead automaton. Other input symbols, i.e.  $a$  and  $r$ , are just copied to the output (left and bottom transitions).

The next theorem states that adding look-aheads to VPT does not add expressiveness. The main difficulty is to simulate an unbounded number of look-aheads at the same time. Indeed, a look-ahead is triggered at each call and is alive until the end of the well-nested subword starting at this call. To handle the simulation of the look-aheads that started at a *strictly less deeper* nesting level we use the notion of summaries. Recall that summaries were introduced in the context of the determinization of VPA ([1]), they are pairs of states. More precisely, for a given VPA, a pair  $(p, q)$  is a summary if there exists a well-nested word  $w$  such that the configuration  $(q, \perp)$  is accessible from  $(p, \perp)$  by reading  $w$ . We use a classical subset construction for the look-aheads that started at the *same* nesting level.

**Theorem 1.** *For any  $VPT_{1a}$ , resp.  $VPA_{1a}$ ,  $T_{1a}$  with  $n$  states, one can construct an equivalent VPT, resp. VPA,  $T'$  with  $O(n2^{n^2+1})$  states. Moreover, if  $T_{1a}$  is deterministic, then  $T'$  is unambiguous.*

*Proof.* We prove the result for  $VPT_{1a}$  only, this trivially implies the result for  $VPA_{1a}$ . Let  $T_{1a} = (T, A)$  with  $T = (Q, q_0, F, \Gamma, \delta)$  and  $A = (Q^{1a}, F^{1a}, \Gamma^{1a}, \delta^{1a})$ . We construct  $T' = (Q', q'_0, F', \Gamma', \delta')$  as follows (where  $Id_{Q^{1a}}$  denotes the identity relation on  $Q^{1a}$ ):  $Q' = Q \times 2^{Q^{1a} \times Q^{1a}} \times 2^{Q^{1a}}$ ,  $q'_0 = (q_0, Id_{Q^{1a}}, \emptyset)$ ,  $F' = \{(q, R, L) \in Q' \mid q \in F, L \subseteq F^{1a}\}$ ,  $\Gamma' = \Gamma \times 2^{Q^{1a} \times Q^{1a}} \times 2^{Q^{1a}} \times \Sigma_c$ .

The transducer  $T'$  simulates  $T$  and its running look-aheads. A state of  $T'$  is a triple  $(q, R, L)$ . The first component is the state of  $T$ . The second and third components are used to simulate the running look-aheads. When taking a call  $c$ ,  $T'$  non-deterministically chooses a new look-ahead triggered by  $T$ . This look-ahead is added to all running look-aheads that started at the same nesting level.  $T'$  ensures that the run will fail if the

longest well-nested prefix starting at  $c$  is not in the language of the triggered look-ahead. The  $L$  component contains the states of all running look-aheads triggered at the current nesting level. The  $R$  component is the summary necessary to update the  $L$ -component. When reading a call the  $L$  component is put on the stack. When reading a return,  $T'$  must check that all look-ahead states in  $L$  are final, i.e.  $T'$  ensures that the chosen look-aheads are successful.

After reading a well-nested word  $w$  if  $T'$  is in state  $(q, R, L)$ , with  $q \in Q$ ,  $R \subseteq Q^{la} \times Q^{la}$  and  $L \subseteq Q^{la}$ , we have the following properties. The pair  $(p, p') \in R$  iff there exists a run of  $A$  from  $p$  to  $p'$  on  $w$ . If some  $p''$  is in  $L$ , there exists a run of a look-ahead that started when reading a call symbol of  $w$  at depth 0 which is now in state  $p''$ . Conversely, for all look-aheads that started when reading a call symbol of  $w$  at depth 0, there exists a state  $p'' \in L$  and a run of this look-ahead that is in state  $p''$ .

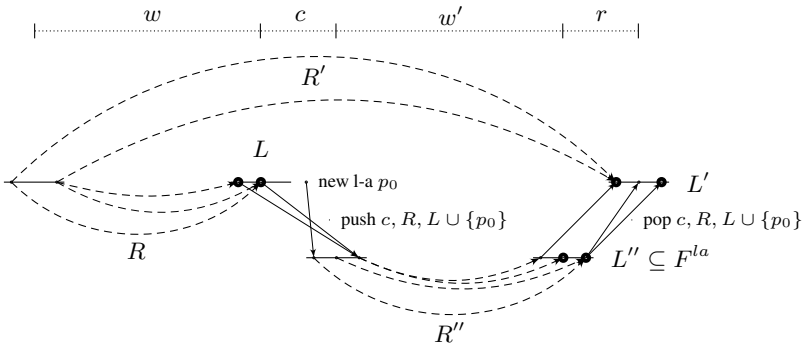


Fig. 2.

Let us consider a word  $wcw'r$  for some well-nested words  $w, w'$  (depicted on Fig. 2). Assume that  $T'$  is in state  $(q, R, L)$  after reading  $w$  (on the figure, the relation  $R$  is represented by dashed arrows and the set  $L$  by big points, and other states by small points). We do not represent the  $T$ -component of the states on the figure but rather focus on  $R$  and  $L$ . The information that we push on the stack when reading  $c$  is the necessary information to compute a state  $(q', R', L')$  of  $T'$  reached after reading  $wcw'r$ . After reading the call symbol  $c$ , we go in state  $(q', Id_{Q^{la}}, \emptyset)$  and produce the output  $v$  for some  $q', v$  such that  $q \xrightarrow{c|v, p_0, \gamma} q' \in \delta_c$ , where  $p_0 \in Q^{la}$  is the starting state of a new look-ahead. Note that determinism of  $T$  is preserved. On the stack we put the tuple  $(\gamma, R, L \cup \{p_0\}, c)$  where  $\gamma, R, L, p_0, c$  have been defined before.

Now, suppose that after reading  $wcw'$  the transducer  $T'$  is in state  $(q'', R'', L'')$ . It means that  $T$  is in state  $q''$  after reading  $wcw'$ , and  $(p, p') \in R''$  iff there exists a run of  $A$  from  $p$  to  $p'$  on  $w'$ , and  $L''$  is some set of states reached by the look-aheads that started at the same depth as  $w'$ . Therefore we first impose that any transition from  $(q'', R'', L'')$  reading  $r$  must satisfy  $L'' \subseteq F^{la}$ . Clearly,  $R'$  can be constructed from  $c, R$  and  $R''$ . Finally,  $L'$  is a set which satisfies for all  $p \in L \cup \{p_0\}$ , there exists  $p' \in L'$  such that there exists a run of  $A$  from  $p$  to  $p'$  on  $cw'r$ . If such an  $L'$  does not exist, there is no transition on  $r$ . The set  $L'$  can be constructed from  $L \cup \{p_0\}$  and  $R''$ .



We now define the transitions formally. First, for all  $q, R, L, c, \gamma$ , we have:

$$(q, R, L) \xrightarrow{c|u, (\gamma, R, L \cup \{p_0\}, c)} (q', Id_{Q^{la}}, \emptyset) \in \delta'_c \text{ whenever } q \xrightarrow{c|u, p_0, \gamma} q' \in \delta_c$$

Then, for all  $R, L, r, \gamma, q'', R'', L'', q', R', L'$  we have:

$$(q'', R'', L'') \xrightarrow{r|u, (\gamma, R, L, c)} (q', R', L') \in \delta'_r \text{ if the following conditions hold:}$$

- (i)  $q'' \xrightarrow{r|u, \gamma} q' \in \delta_r$ , (ii)  $L'' \subseteq F^{la}$
- (iii)  $R' = \{(p, p') \mid \exists s \xrightarrow{c, \gamma} s' \in \delta_c^{la} \cdot \exists (s', s'') \in R'' \cdot (p, s) \in R \text{ and } s'' \xrightarrow{r, \gamma} p' \in \delta_r^{la}\}$
- (iv) for all  $p \in L$ , there exist  $p' \in L', \gamma \in \Gamma, s, s' \in Q^{la}$  such that  $(s, s') \in R'', p \xrightarrow{c, \gamma} s \in \delta_c^{la}, s' \xrightarrow{r, \gamma} p' \in \delta_r^{la}$ .

If  $T$  is deterministic, then  $T'$  is unambiguous. Indeed, it is deterministic on return transitions. If there are two possible transitions  $q \xrightarrow{c|u_1, p_1, \gamma_1} q_1$  and  $q \xrightarrow{c|u_2, p_2, \gamma_2} q_2$  on a call symbol  $c$ , as  $T$  is deterministic, we know that either the look-ahead starting in  $p_1$  or the look-ahead starting in  $p_2$  will fail. In  $T'$ , there will be two transitions that will simulate both look-aheads respectively, and therefore at least one continuation of the two transitions will fail as well. Therefore there is at most one accepting computation per input word in  $T$ .  $\square$

*Succinctness.* The exponential blow-up in the construction of Theorem 1 is unavoidable. Indeed, it is obviously already the case for finite state automata with regular look-ahead. These finite state automata can be easily simulated by VPA on flat words (in  $(\Sigma_c \Sigma_r)^*$ ) (in that case the stack is useless). For example, consider for all  $n$  the language  $L_n = \{vuv \mid |v| = n\}$ . One can construct a finite state automaton with regular look-ahead with  $O(n)$  states that recognizes  $L_n$ . It is done by using look-aheads that check for all  $a \in \Sigma$  and  $i \leq n$  that the  $m - (n - i)$ -th letter is equal to  $a$ , where  $m$  is the length of the word. Without a regular look-ahead, any automaton has to store the  $n$ -th first letters of  $w$  in its states, then it guesses the  $m - n$ -th position and checks that the prefix of size  $n$  is equal to the suffix of size  $n$ . A simple pumping argument shows that the automaton needs at least  $|\Sigma|^n$  states.

## 4 Functional VPT and VPT<sub>la</sub>

While there is no known syntactic restriction on VPT that captures all functional VPT, we show that the class of deterministic VPT<sub>la</sub> captures all functional VPT. As there may be an unbounded number of accepting runs, the equivalent VPT<sub>la</sub> has to choose only one of them by using look-aheads. This is done by ordering the states and extending this order to runs. Similar ideas have been used in [7] to show the same result for top-down tree transducers. The main new difficulty with VPT is to cope with nesting. Indeed, when the transducer enters an additional level of nesting, its look-ahead cannot inspect the entire suffix but is limited to the current nesting level. When reading a call, choosing (thanks to some look-ahead) the smallest run on the current well-nested prefix is not correct because it may not be possible to extend this run to an accepting run on the entire word. Therefore the transducer has to pass some information from one to the

next level of nesting about the chosen global run, while for top-down tree transducers, as the evaluation is top-down, the transformation of the current subtree is independent of the transition choices that have been made at upper levels.

**Theorem 2.** *For all VPT  $T$ , one can construct a deterministic VPT<sub>1a</sub>  $T_{1a}$  with at most exponentially many more states such that  $\llbracket T_{1a} \rrbracket \subseteq \llbracket T \rrbracket$  and  $\text{Dom}(T_{1a}) = \text{Dom}(T)$ . If  $T$  is functional, then  $\llbracket T_{1a} \rrbracket = \llbracket T \rrbracket$ .*

*Proof.* We order the states of  $T$  and use look-aheads to choose the smallest runs wrt to an order on runs that depends on the structure of the word. Let  $T = (Q, q_0, F, \Gamma, \delta)$  be a VPT. Wlog we assume that for all  $q, q' \in Q$ , all  $\alpha \in \Sigma$ , there is at most one  $u \in \Sigma^*$  and one  $\gamma \in \Gamma$  such that  $(q, \alpha, u, \gamma, q') \in \delta$ . A transducer satisfying this property can be obtained by duplicating the states with transitions, i.e. by taking the set of states  $Q \times \Delta$ .

We construct a deterministic VPT<sub>1a</sub>  $T_{1a} = (T', A)$  such that  $\llbracket T_{1a} \rrbracket \subseteq \llbracket T \rrbracket$  and  $\text{Dom}(T_{1a}) = \text{Dom}(T)$  and where  $T' = (Q', q_0, F', \Gamma', \delta')$  with  $Q' = \{q_0\} \cup Q^2$ ,  $F' = F \times Q$  if  $q_0 \notin F$  otherwise  $F' = (F \times Q) \cup \{q_0\}$ . The look-ahead  $A$  is defined later. Before defining  $\delta'$  formally, let us explain it informally. There might be several accepting runs on an input word  $w$ ,  $T_{1a}$  has to choose exactly one. Furthermore, to ensure determinism, when reading a symbol,  $T_{1a}$  has to choose exactly one transition. The idea is to order the states by a total order  $<_Q$  and to extend this order to runs. The look-ahead will be used to choose the next transition of  $T$  that has to be fired, so that the choice will ensure that  $T$  follows the smallest accepting run on  $w$ . However the look-ahead can only visit the current longest well-nested prefix, and not the entire word. Therefore the “parent” of the call  $c$  has to pass some information about the global run to its child  $c$ . In particular, when  $T'$  is in state  $(q, q')$  for some state  $q'$ , it means that  $T$  is in state  $q$  and the state reached after reading the last return symbol of the longest-well nested current prefix must be  $q'$ .

Consider a word of the form  $w = c_1 w_1 r_1 w_2 c_3 w_3 r_3$  where  $w_i$  are well-nested, depicted on Fig. 3. Suppose that before evaluating  $w$ ,  $T'$  is in state  $(q_1, q_3)$ . It means that the last transition  $T$  has to fire when reading  $r_3$  has a target state  $q_3$ . When reading the call symbol  $c_1$ ,  $T'$  uses a look-ahead to determine the smallest triple of states  $(q'_1, q'_2, q_2)$  such that there exists a run on  $w$  that starts in  $q_1$  and such that after reading  $c_1$  it is in state  $q'_1$ , before reading  $r_1$  it is in state  $q'_2$ , after reading  $r_1$  it is in state  $q_2$  and after reading  $r_3$  it is in state  $q_3$ . Then,  $T'$  fires the call transition on  $c_1$  that with source and target states  $q_1$  and  $q'_1$  respectively (it is unique by hypothesis), put on the stack the states  $(q_2, q_3)$  and passes to  $w_1$  (in the state) the information that the chosen run on  $w_1$  terminates by the state  $q'_2$ , i.e. it goes to the state  $(q'_1, q'_2)$ . (see Fig. 3). On the figure, we do not explicit all the states and anonymous components are denoted by  $\dots$ . When reading  $r_1$ ,  $T'$  pops from the stack the tuple  $(\gamma, q_2, q_3)$  and therefore knows that the transition to apply on  $r_1$  has target state  $q_2$  and the transition to apply on  $r_3$  has target state  $q_3$ . Then it passes  $q_3$  to the current state.

When the computation starts in  $q_0$ , we do not know yet what return transition has to be fired at the end of the hedge. This case can be easily treated separately by a look-ahead on the first call symbol that determine the smallest 4-tuple of states  $(q_1, q'_2, q_2, q_3)$  which satisfies the conditions described before, but to simplify the proof, we assume that the VPT accepts only words of the form  $cwr$ , where  $w$  is well-nested, so that one only needs to consider triples of states.

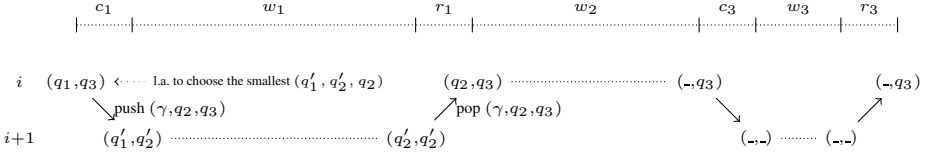


Fig. 3.

We now define the transition relation formally. Let  $<$  be a total order on states, extended lexicographically to tuples. For all states  $q_1, q'_1, q'_2, q_2, q_3 \in Q$ , it is easy to define a VPA  $A_{q_1, q'_1, q'_2, q_2, q_3}$  whose size is polynomial in the size of  $T$  that accepts a word  $w$  iff it is of the form  $c_1 w_1 r_1 w_3$  where  $w_1, w_3$  are well-nested and there exists a run of  $T$  on  $w$  that starts in state  $q_1$  and is state  $q'_1$  after reading  $c_1$ , in state  $q'_2$  before reading  $r_1$ , in state  $q_2$  after reading  $r_1$  and in state  $q_3$  after reading  $w_3$ . Note that if  $w_3 = \epsilon$  then if  $q_3 \neq q_2$ , then  $w \notin L(A_{q_1, q'_1, q'_2, q_2, q_3})$ . We denote by  $\overline{A_{q_1, q'_1, q'_2, q_2, q_3}}$  the complement of  $A_{q_1, q'_1, q'_2, q_2, q_3}$ .

We let  $B_{q_1, q'_1, q'_2, q_2, q_3}$  a VPA with initial state  $p_{q_1, q'_1, q'_2, q_2, q_3}$  that defines the language:

$$L(B_{q_1, q'_1, q'_2, q_2, q_3}) = L(A_{q_1, q'_1, q'_2, q_2, q_3}) \cap \bigcap_{\substack{(s_1, s'_2, s_2) \in Q^3 \\ (s_1, s_2, s_2) < (q_1, q'_2, q_2)}} L(\overline{A_{q_1, s_1, s'_2, s_2, q_3}})$$

Such a VPA exists as VPAs are closed by intersection and complement. Its size however may be exponential in  $|Q|$ . We define the look-ahead VPA as the union of all those VPAs,  $A_{la} = \bigcup B_{q_1, q'_1, q'_2, q_2, q_3}$ . We now define the call and return transitions of  $T'$  as follows, for all  $c \in \Sigma_c, r \in \Sigma_r, \gamma \in \Gamma, q_1, q'_1, q'_2, q_3, q \in Q, u \in \Sigma^*$ :

$$\begin{aligned} (q_1, q_3) & \xrightarrow{c|u, (\gamma, q_2, q_3), p_{q_1, q'_1, q'_2, q_2, q_3}} (q'_1, q'_2) \text{ if } (q_1 \xrightarrow{c|u, \gamma} q'_1) \in \delta_c \\ q_0 & \xrightarrow{c|u, (\gamma, q_3, q_3), p_{q_0, q'_1, q'_2, q_3, q_3}} (q'_1, q'_2) \text{ if } (q_0 \xrightarrow{c|u, \gamma} q'_1) \in \delta_c \\ (q'_2, q) & \xrightarrow{r|u, (\gamma, q_2, q_3)} (q_2, q_3) \text{ if } (q'_2 \xrightarrow{r|u, \gamma} q_2) \in \delta_r \end{aligned}$$

It can be shown that  $T'$  is deterministic [16]. Clearly, if  $T$  is functional then  $T'_{la}$  is equivalent.  $\square$

This construction, followed by the construction of Theorem 1 that removes the look-aheads, yields a nice characterization of functional VPT:

**Theorem 3.** *For all functional VPT  $T$ , one can effectively construct an equivalent unambiguous VPT  $T'$ .*

## 5 Decision Problems

In this section, we study the problems of functionality of  $\text{VPT}_{la}$  and equivalence of functional  $\text{VPT}_{la}$ . In particular, we prove that while being exponentially more succinct than VPT, the equivalence of functional  $\text{VPT}_{la}$  remains decidable in EXPT, as equivalence of functional VPT.

**Theorem 4.** *Functionality of  $VPT_{la}$  is EXPT-C, even for deterministic look-aheads.*

*Proof.* For the EXPT upper-bound, we first apply Theorem 1 to remove the look-aheads. This results in a VPT possibly exponentially bigger. Then functionality can be tested in PTIME [9]. For the lower-bound, we reduce the problem of deciding emptiness of the intersection of  $n$  deterministic top-down tree automata, which is known to be EXPT-C when  $n$  is part of the input [3].  $\square$

We know that the equivalence of two functional VPT is EXPT-C [9]. For equivalence of functional  $VPT_{la}$ , one can first remove the look-aheads, modulo an exponential blow-up, and use the procedure for VPT. This would yield a 2-EXPT procedure for the equivalence of functional  $VPT_{la}$ . However, it is possible to decide it in EXPT:

**Theorem 5.** *Emptiness of  $VPT_{la}$ , resp. of  $VPA_{la}$ , equivalence and inclusion of functional  $VPT_{la}$ , resp. of  $VPA_{la}$ , is EXPT-C, even if the transducers, resp. automata, and the look-aheads are deterministic.*

*Proof.* The lower bounds are obtained, as for functionality, by reduction of the emptiness of  $n$  (deterministic) tree automata.

Emptiness of  $VPA_{la}$  can be checked by first removing the look-aheads (modulo an exponential blow-up) and then check emptiness of the equivalent VPA (in PTIME). Checking emptiness of a  $VPT_{la}$  amounts to check emptiness of its domain, which is a  $VPA_{la}$ .

To show that equivalence and inclusion of two  $VPA_{la}$  is in EXPT, we construct two alternating (ranked) tree automata equivalent to the VPA modulo the first-child next-sibling encoding in PTIME. Look-aheads are encoding as universal transitions. Equivalence and inclusion of alternating tree automata is in EXPT [3].

Then, let us show how to check the equivalence, resp. inclusion, of two  $VPT_{la}$ : transform each  $VPT_{la}$  into an equivalent VPT with at most an exponential blow-up, take the union and verify (in PTIME) that the resulting VPT is still functional. Then check that their domains (which are  $VPA_{la}$  obtained by ignoring the output of the two  $VPT_{la}$ ) are equivalent, resp. included.  $\square$

*Acknowledgments.* We are very grateful to Sebastian Maneth for suggesting us to extend VPT with look-aheads, and to Pierre-Alain Reynier for simplifying the proof of Theorem 1.

## References

1. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: STOC, pp. 202–211 (2004)
2. Alur, R., Madhusudan, P.: Adding nesting structure to words. JACM 56(3), 1–43 (2009)
3. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications (2007)
4. Eilenberg, S.: Automata, Languages, and Machines. Academic Press, Inc. (1974)
5. Elgot, C.C., Mezei, J.E.: On relations defined by generalized finite automata. IBM Journal of Research and Development 9, 47–68 (1965)
6. Engelfriet, J.: Top-down tree transducers with regular look-ahead. Mathematical Systems Theory 10, 289–303 (1977)
7. Engelfriet, J.: On tree transducers for partial functions. Inf. Process. Lett. 7(4), 170–172 (1978)

8. Engelfriet, J., Vogler, H.: Macro tree transducers. *JCSS* 31(1), 71–146 (1985)
9. Filiot, E., Raskin, J.-F., Reynier, P.-A., Servais, F., Talbot, J.-M.: Properties of Visibly Pushdown Transducers. In: Hliněný, P., Kučera, A. (eds.) *MFCS 2010*. LNCS, vol. 6281, pp. 355–367. Springer, Heidelberg (2010)
10. Gauwin, O., Niehren, J., Tison, S.: Queries on XML streams with bounded delay and concurrency. *Inf. Comput.* 209(3), 409–442 (2011)
11. Kumar, V., Madhusudan, P., Viswanathan, M.: Visibly pushdown automata for streaming XML. In: *WWW*, pp. 1053–1062 (2007)
12. Perst, T., Seidl, H.: Macro forest transducers. *IPL* 89(3), 141–149 (2004)
13. Sakarovitch, J., de Souza, R.: Lexicographic decomposition of  $k$ -valued transducers. *TCS* 47(3), 758–785 (2010)
14. Schützenberger, M.P.: Sur les relations rationnelles entre monoïdes libres. *TCS* 3(2), 243–259 (1976)
15. Segoufin, L., Vianu, V.: Validating streaming XML documents. In: *PODS*, pp. 53–64 (2002)
16. Servais, F.: Visibly Pushdown Transducers. PhD thesis, Université Libre de Bruxelles (2011)
17. Staworko, S., Laurence, G., Lemay, A., Niehren, J.: Equivalence of Deterministic Nested Word to Word Transducers. In: Kutyłowski, M., Charatonik, W., Gębala, M. (eds.) *FCT 2009*. LNCS, vol. 5699, pp. 310–322. Springer, Heidelberg (2009)
18. Weber, A.: Decomposing finite-valued transducers and deciding their equivalence. *SIAM Journal on Computing* 22(1), 175–202 (1993)