# Visibly Pushdown Transducers with Look-Ahead

Emmanuel Filiot, Frédéric Servais

# Visibly Pushdown Transducers with Look-Ahead

Emmanuel Filiot[1] Frédéric Servais[1]

[1] Université Libre de Bruxelles

**Abstract.** Visibly Pushdown Transducers (VPTs) form a subclass of pushdown transducers. In this paper, we investigate the extension of VPTs with look-ahead ($\mathsf{VPTs}_{\mathsf{la}}$). Intuitively they are VPTs that can check that the well-nested subword starting at the current position belongs to a given visibly pushdown language. First, we show that $\mathsf{VPTs}_{\mathsf{la}}$ are not more expressive than VPTs, but they are exponentially more succinct. Second, we show that the class of deterministic $\mathsf{VPTs}_{\mathsf{la}}$ corresponds exactly to the class of functional VPTs. Finally, we show that, while $\mathsf{VPTs}_{\mathsf{la}}$ may be exponentially more succinct than VPTs, checking equivalence of functional $\mathsf{VPTs}_{\mathsf{la}}$ is, as for VPTs, ExpTime-c. These results add to previous one, mainly decidability of functionality and equivalence of functional VPTs, and show that VPTs form a robust class of transducers.

## 1 Introduction

Visibly pushdown transducers (VPTs) [15, 8] form an interesting subclass of pushdown transducers (PTs). Several problems that are undecidable for PTs are decidable for VPTs, noticeably: functionality is decidable in PTime, $k$-valuedness in NPTime and equivalence of functional VPTs is ExpTime-c [8].

Visibly pushdown machines [1], automata 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 can not 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.

In this paper, we introduce and investigate the class of VPTs with visibly look-ahead. A VPT with look-ahead ($\mathsf{VPT}_{\mathsf{la}}$) is a VPT such that call transitions are guarded with visibly pushdown automata (VPA). When reading a call at position $i$, a $\mathsf{VPT}_{\mathsf{la}}$ can take a matching call transition provided the longest well-nested word starting at position $i$ is included in the language of the VPA of the transition.

First, we show that $\mathsf{VPTs}_{\mathsf{la}}$ are no more expressive than VPTs. We present an exponential construction that remove the look-aheads which are simulated by the equivalent VPT. Moreover we show this exponential blow-up is unavoidable.

Second, we show that the class of deterministic $\mathsf{VPTs}_{\mathsf{la}}$ is exactly the class of functional VPTs. 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 an elegant and simple characterization of functional VPTs.

Third, we show that, even though $\mathsf{VPTs}_{\mathsf{la}}$ are exponentially more succinct than VPTs, testing the equivalence of functional $\mathsf{VPTs}_{\mathsf{la}}$ is not harder than for functional VPTs, these problems are ExpTime-c. Testing the equivalence of functional transducers is done in two steps. First one must check that the domains are equivalent. Then one checks that the union of the two transducers is still functional. We show that testing functionality is ExpTime-c for $\mathsf{VPTs}_{\mathsf{la}}$: get rid of the look-aheads with an exponential blow-up and test in PTime the functionality of the equivalent 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 $\mathsf{VPTs}_{\mathsf{la}}$ are linearly reducible to alternating top-down tree automata. Testing the equivalence of such automata can be done in ExpTime [3].

Finally, we consider slightly different look-aheads. First, we discuss look-aheads that are allowed to inspect the word until the end. We show this does not add expressivity nor succinctness. Second, we show that restricting the look-ahead to the word in between the current call and its matching return is not sufficient to have a characterization of functional VPTs by deterministic $\mathsf{VPTs}_{\mathsf{la}}$. All these results are new indications that the class of VPTs is robust, and they show that the class of $\mathsf{VPTs}_{\mathsf{la}}$ is interesting in itself.

*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 [4], or to *macro tree transducers* (MTTs) [7]. For top-down tree transducers, adding regular look-ahead strictly increases their expressive power while macro tree transducers are closed by regular 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 therefore be used as unranked tree transducers on fcns encodings of unranked trees. Inspecting a subtree in the fncs 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, as explained in [8], top-down tree transducers and VPTs are incomparable: top-down tree transducers can copy subtrees while VPTs cannot, and VPTs support concatenation of tree sequences while top-down tree transducers cannot.

Modulo those encodings, MTTs subsume VPTs [8] and as we said before, there is a correspondence between the two notions of look-aheads, for VPTs and

MTTs respectively. However it is not clear how to derive our results on closure by look-aheads from the same result on MTTs, as the latter highly relies on parameters.

## 2 Visibly Pushdown Languages and Transductions

All over this paper, $\Sigma$ denotes a finite alphabet partitioned into two disjoint sets[1] $\Sigma_c$, $\Sigma_r$, denoting respectively the *call* and *return* alphabets.

*Words and nested words* 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 $\Sigma^*_{\mathsf{wn}}$ is the smallest subset of $\Sigma^*$ such that $\epsilon \in \Sigma^*_{\mathsf{wn}}$ and for all $c \in \Sigma_c$, all $r \in \Sigma_r$, all $u, v \in \Sigma^*_{\mathsf{wn}}$, $cur \in \Sigma^*_{\mathsf{wn}}$ and $uv \in \Sigma^*_{\mathsf{wn}}$.

*Visibly Pushdown Languages* 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, $I \subseteq Q$, respectively $F \subseteq Q$, the set of initial states, respectively 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*.

On a call transition $(q, a, q', \gamma) \in \delta_c$, $\gamma$ is pushed onto the stack and the control goes from $q$ to $q'$. On a return transition $(q, \gamma, a, q') \in \delta_r$, $\gamma$ 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 $\sigma_{k-1} = \sigma_k\gamma_k$, or $(q_{k-1}, a_k, q_k) \in \delta_i$ and $\sigma_k = \sigma_{k-1}$. The run $\rho$ is *accepting* if $q_0 \in I$, $q_l \in F$ and $\sigma_0 = \sigma_l = \perp$ [2]. 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 $L(A) = L$.

*Visibly pushdown transducers (*VPTs*)* As finite-state transducers extend finite-state automata with outputs, visibly pushdown transducers extend visibly pushdown automata with outputs [**?**]. To simplify notations, we suppose that the output alphabet is $\Sigma$, but our results still hold for an arbitrary 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), on a return symbol, it

[1] 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 $r_i$. All our results extend trivially to alphabets with internal symbols. We make this assumption to simplify notations.

[2] Note that, in contrast to [1] and to ease notations, 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.

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. Formally:

**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 $[\![T]\!] = \{(u, v) \mid \exists q \in I, q' \in F,\ (q, \bot) \xrightarrow{u/v} (q', \bot)\}$.

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 $R = [\![T]\!]$. 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 $[\![T]\!]$ is functional. Two transducers $T_1, T_2$ are *equivalent* if $[\![T_1]\!] = [\![T_2]\!]$. The following is known:

**Proposition 1 ([8]).** *Functionality is decidable in* PTIME *for* VPTs. *Equivalence of functional* VPTs *is* EXPTIME-C.

The class of functional VPTs is denoted by fVPT. 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 $[\![T]\!](u)$. 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 $[\![T]\!]$. 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 just after $c$, and to $a$ if an $a$ appears. E.g. $ccrarrcr$ is mapped to $acrarrcr$, and $cccarrcrcarrrr$ to $aaaarrcraarrrr$.

The VPT $T = (Q, I, F, \Gamma, \delta)$ is defined as follows. $Q = \{q, q_a\}$, $I = \{q\}$, $F = Q$, $\Gamma = \{(a, a), (a, \neg a), (\neg a, a), (\neg a, \neg a), (-, a)\}$ and $\delta$ contains the following transitions:

$$
\begin{array}{lll}
q \xrightarrow{c/c,(\neg a,\neg a)} q & q \xrightarrow{c/a,(a,\neg a)} q & q \xrightarrow{a/a,(-,a)} q \\
q_a \xrightarrow{c/c,(\neg a,a)} q & q_a \xrightarrow{c/a,(a,a)} q & q_a \xrightarrow{a/a,(-,a)} q \\
q \xrightarrow{r/r,(\neg a,\neg a)} q & q \xrightarrow{r/r,(a,\neg a)} q_a & q \xrightarrow{r/r,(-,a)} q_a \\
q_a \xrightarrow{r/r,(a,\neg a)} q_a & q_a \xrightarrow{r/r,(a,a)} q_a & q_a \xrightarrow{r/r,(-,a)} q_a
\end{array}
$$

If $T$ is in state $q_a$ it means there is an $a$ in the longest well-nested word that ends at the current position, otherwise it is in state $q$. Consider an element $(\alpha, \beta)$ of the stack. If $\alpha = a$, respectively $\neg a$, it means that the call was a $c$ and the automaton translated it into an $a$, respectively translated into a $c$, i.e. it guesses there is an $a$, respectively there is no $a$, in the well-nested word starting just after $c$. If $\alpha = -$ then the call was an $a$. The second component $\beta$, carry over the fact that there is or not an $a$ in the well nested word ending at the current call.

## 3 VPTs with Regular 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 look-ahead 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 test 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-ahead correspond to inspecting the subtree rooted at the current node and all right sibling subtrees. Formally:

**Definition 2.** *A* VPT *with look-ahead (*$\mathsf{VPT}_{\mathsf{la}}$*) is a pair* $T_{la} = (T, A)$ *where* $A$ *is a* VPA $A = (Q^{la}, \delta_c^{la}, \delta_r^{la}, \Gamma^{la}, F^{la})$ *without initial state and* $T$ *is a tuple* $T = (Q, q_0, F, \Gamma, \delta = \delta_c \uplus \delta_r)$ *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* $\delta_c \subseteq Q \times \Sigma_c \times \Sigma^* \times Q^{la} \times \Gamma \times Q$ *and* $\delta_r \subseteq Q \times \Sigma_r \times \Sigma^* \times \Gamma \times Q$.

Let $u \in \Sigma^*$. A run of $T_{la}$ on $u = a_1 \dots a_l$ is a sequence of configurations $\rho = \{(q_k, \sigma_k)\}_{0 \leq k \leq l}$ such that there are $\gamma \in \Gamma$ and $v_{k+1} \in \Sigma^*$ such that $(i)$ if $a_{k+1} \in \Sigma_r$, then $\sigma_{k+1}\gamma = \sigma_k$ and $(q_k, a_{k+1}, v_{k+1}, \gamma, q_{k+1}) \in \Sigma_r$; $(ii)$ if $a_{k+1} \in \Sigma_c$, then $\sigma_{k+1} = \sigma_k\gamma$, and there $p \in Q^{la}$ such that $(q_k, a_{k+1}, v_{k+1}, \gamma, q_{k+1}) \in \Sigma_r$ 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 $\mathsf{VPT}_{\mathsf{la}}$ $T^{la}$ 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 $L(A, p_1) \cap L(A, p_2) = \varnothing$; 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 a $\mathsf{VPT}_{\mathsf{la}}$ is deterministic can be done in PTIME. One has to check that for each state $q$ and each call symbol $c$, the VPLs guarding the transition from state $q$ and reading $c$ are *pairwise* disjoint.

*Example 2.* A $\mathsf{VPT}_{\mathsf{la}}$ is represented in Figure 1. The look-ahead automaton is depicted on the right, while the transducers in itself is on the left. When starting in state $q_a$, respectively $q_{\neg a}$, the look-ahead automaton accepts well-nested words

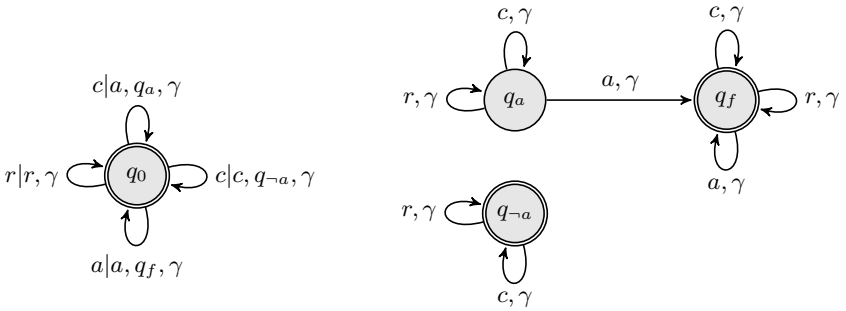


**Fig. 1.** A $\mathsf{VPT}_{\mathsf{la}}$ (left) and its look-ahead (right) on $\Sigma_c = \{c, a\}$ and $\Sigma_r = \{r\}$

that contains an $a$, respectively does not contain any $a$. When starting in state $q_f$ it accepts any well-nested word. The transducers will rewrite $c$ symbols into an $a$ 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 $q_\neg a$ states of the look-ahead automaton. Other input symbols, i.e. $a$ and $r$, are just copied to the output (left and bottom transitions). This transduction is the same as in Example 1.

The next Theorem states that adding look-ahead to $\mathsf{VPTs}$ does not improves their expressive power.

**Theorem 1.** *For all* $\mathsf{VPT}_{\mathsf{la}}$ $T$ *with* $n$ *states, one can effectively construct an equivalent* $\mathsf{VPT}$ $T'$ *with* $O(n2^{n^2+1})$ *states.*

*Proof.* Let $T = (Q, F, q_0, \delta_c, \delta_r)$ and $A = (Q^{la}, F^{la}, \delta_c^{la}, \delta_r^{la})$. We construct $T' = (Q', F', q_0', \delta_c', \delta_r')$ as follows (where $Id_{Q^{la}}$ denotes the identity relation on $Q^{la}$):

- $Q' = Q \times 2^{Q^{la} \times Q^{la}} \times 2^{Q^{la}}$;
- $\Gamma' = \Gamma \times 2^{Q^{la} \times Q^{la}} \times 2^{Q^{la}} \times \Sigma_c$;
- $F' = \{(q, R, L) \in Q' \mid q \in F, L \subseteq F^{la}\}$.
- $q_0' = (q_0, Id_{Q^{la}}, \varnothing)$.

Before defining the transition relations, we first explain the semantics of the states. 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 current state of $T$ is $q$, $(p, p') \in R$ iff there exists a run of $A$ from $p$ to $p'$ on $w$ and if $p'' \in L$, there exists a look-ahead that started when reading a call symbol of $w$ at depth 0 which can be in state $p''$ (since look-ahead are non-deterministic there are several choices for $p''$).

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 $(q', Id_{Q^{la}}, \varnothing)$ and produce the output $v$ for some $q', v$ such that

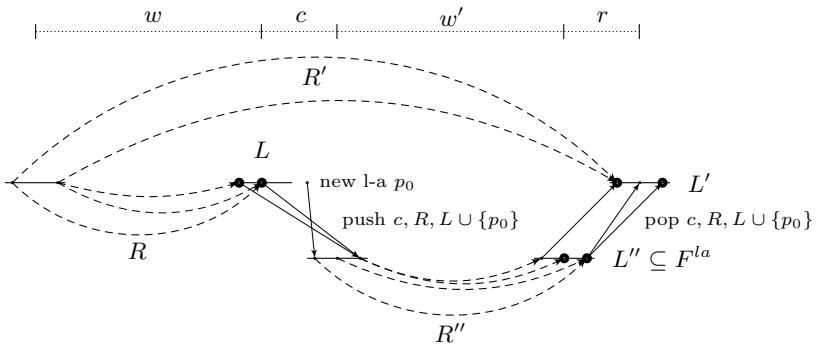


**Fig. 2.**

$q \xrightarrow{c|v,p_0,\gamma} q' \in \delta_c$, where $p_0 \in Q^{la}$ is the starting state of a new look-ahead. You can notice that we preserve the non-determinism of $T$. 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 the set of states reached by the look-aheads that started at the depth of $w'$. Therefore we first impose that any transition from $(q'', R'', L'')$ reading $r$ must satisfy $L'' \subseteq F^{la}$. Then $q'$ is a state such that $q'' \xrightarrow{r|u,\gamma} q' \in \delta_r$. 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:

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

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

$$(q'', R'', L'') \xrightarrow{r|u,(\gamma,R,L,c)} (q', R', L') \in \delta'_r$$

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 $(s, s') \in R''$, $p \xrightarrow{c,\gamma} s \in \delta_c^{la}$, $s' \xrightarrow{r,\gamma} p' \in \delta_r^{la}$.

We sketch the proof of the correctness of the construction. Let $w \in \Sigma^*$ such that $w$ is a prefix of well-nested word. We define $sh(w)$ as the longest well-nested suffix of $w$, we call $sh(w)$ the *subhedge* of $w$. For instance, if $w = c_1 c_2 r_2 c_3 r_3$, then $sh(w) = c_2 r_2 c_3 r_3$. However if $w = c_1 c_2$, then $sh(w) = \epsilon$.

First, one can check (e.g. by induction on the length of $w$) that the successive computations of the $R$ component of the state ensures that the following property holds: for all words $w \in \Sigma^*$ prefix of a well-nested word, if there is a run of $T'$

from $q'_0$ to $(q, R, L)$ on $w$, then for all $p, p' \in Q^{la}$, $(p, p') \in R$ iff there is a run of $A$ on $sh(w)$ from $p$ to $p'$.

With this last property it is easy to show that the following property also holds: let $w = c_1 w_1 r_1 c_2 w_2 r_2 \dots c_n w_n r_n$ where all $w_i$ are well-nested. A run of $T$ on $w$ will trigger a new look-ahead at each call $c_i$, all these look-ahead will still be 'live' until $r_n$. These look-aheads are simulated by the $L$ component of the state of $T'$. If there is a run of $T$ on $w$, it means that all look-aheads accepts the respective remaining suffixes of $w$, and therefore after reading $r_i$ there are $i$ accepting runs of the previous look-aheads. Suppose that those accepting runs are in the states $Q_i$ after reading $r_i$. By suitable choices of $L$-components ($T'$ is non-deterministic on $L$-components), we can ensure that there is an accepting run of $T'$ such that after reading $r_i$ the $L$-component of the states is $Q_i$, for all $i$. Conversely, if there is an accepting run of $T'$ on $w$, then one can easily reconstruct accepting runs of the look-aheads. □

**Proposition 2.** *Let $\Sigma$ be a finite alphabet with at least two letters. There exists a family of transductions $(T_n)_n$ over $\Sigma$ such that for all $n \geq 0$, $T_n$ is definable by a $\mathsf{VPT}_{\mathsf{la}}$ with $O(n)$ states and any $\mathsf{VPT}$ defining $L_n$ has at least $O(2^n)$ states.*

*Proof.* [sketch] We show that the exponential blow-up may occur with flat words, i.e. words in $(\Sigma_c \Sigma_r)^*$. On such words each look-ahead visits the suffix of the whole word starting at the position it is triggered. In the sequel all words are flat words.

Let $n \in \mathbb{N}$ and $T_n = \{(uv, vu) \mid |v| = n\}$. $L_n$ is definable by a $\mathsf{VPT}_{\mathsf{la}}$ $A$ with $O(n)$ states. Let $a_1 \dots a_n$ be the $n$ first letters of $u$ and $m = |u|$. For all $1 \leq i \leq n$, when reading $a_i$, $A$ uses a look-ahead to check that the $(m-n+i)$-th letter of $u$ is equal to the $a_i$. The look-ahead starts in a state $q_{a_i, n-i}$, non-deterministically guesses that the current position is the $m-n+i$-th letter of $u$, verifies that it is equal to $a_i$ and decreases a counter from $n-i$ to 0 (the counter is implemented with $n-i$ states) to check that the guess was correct.

Without a regular look-ahead, an automaton has to store the $n$-th first letters of $u$ in its states, then non-deterministically 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 $\mathsf{VPTs}$ and $\mathsf{VPTs}_{\mathsf{la}}$

While there is no syntactic restriction on $\mathsf{VPTs}$ that captures all functional $\mathsf{VPTs}$, we show that deterministic $\mathsf{VPTs}_{\mathsf{la}}$ captures all functional $\mathsf{VPTs}$.

**Theorem 2.** *Deterministic* $\mathsf{VPTs}_{\mathsf{la}}$ = *functional* $\mathsf{VPTs}$

*Proof.* It is clear that deterministic $\mathsf{VPT}_{\mathsf{la}}$ are functional $\mathsf{VPTs}$. For the converse, we order the transitions and the states and use look-ahead to choose the smallest transition that can be continued to an accepting run.

Let $T = (Q, F, q_0, \Gamma, \delta)$ be a functional VPT. We construct an equivalent deterministic $\mathsf{VPT}_{\mathsf{la}}$ $T' = (Q', F', q'_0, \Gamma', \delta')$ with $Q' = \{q_0\} \cup (Q \times (\delta_r \cup \{\bot\}))$, $F' = F \times \delta_r$, $q'_0 = q_0$. Before defining $\delta'$ formally, let us explain it informally.

There might be several accepting runs on an input word $w$, each of them producing the same output, as $T$ is functional. To ensure determinism, $T'$ has to choose exactly one transition when reading a symbol. The idea is to order the transitions by a total order $<_\delta$ 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. However the look-ahead can only visit the current longest well-nested prefix, and not the whole 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 $(q, t_r)$ for some return transition $t_r$, it means that $T$ is in state $q$ and the return transition on the last return symbol of the longest-well nested current prefix must be $t_r$.

Consider a word of the form $w = c_1 w_1 c_2 w_2 r_2 r_1 w_3 c_3 w_4 r_3$ where $w_i$ are well-nested, depicted on Fig. 3. Suppose that before evaluating $w$, $T'$ is in state $(q, t_{r_3})$. It means that the last transition $T$ has to fire when reading $r_3$ is $t_{r_3}$. When reading the call symbol $c_1$, $T'$ uses a look-ahead to determine the smallest triple of transitions $(t_{c_1}, t_{r_2}, t_{r_1})$, where $t_{c_1}$ is a call transition, $t_{r_1}, t_{r_2}$ are return transitions, such that there exists a run on $w$ that starts with transition $t_{c_1}$ on $c_1$, and uses transition $t_{r_2}$ on $r_2$, transition $t_{r_1}$ on $r_1$ and transition $t_{r_3}$ on $r_3$. Then, $T'$ follows transitions $t_{c_1}$ on $c_1$, put on the stack the transitions $t_{r_1}, t_{r_3}$ and passes to $w_1$ (through the states) the information that the chosen run on $w_1 c_2 w_2 r_2$ terminates by the transition $t_{r_2}$ on $r_2$, i.e. $T'$ goes to some state $(q', t_{r_2})$ (see Fig. 3). On the figure, we do not explicit all the states and anonymous components are denoted by $\_$. When reading $r_1$, $T'$ pops from the stack the tuple $(\gamma, t_{r_1}, t_{r_3})$ and therefore knows that the transition to apply on $r_1$ is $t_{r_1}$ and the transition to apply on $r_3$ is $t_{r_3}$. Then it passes $t_{r_3}$ to the current state.

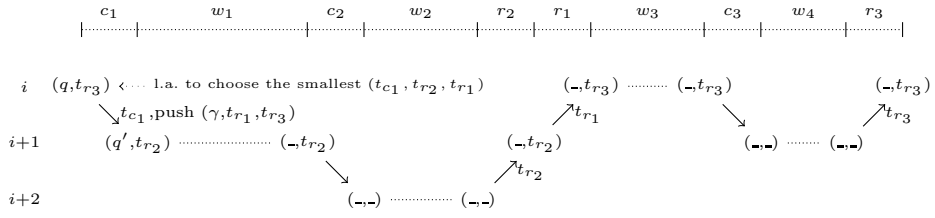


**Fig. 3.**

It might be the case that the current longest well-nested prefixe has nested depth 1 at most (it is the case for instance for the word $c_1 r_1 c_3 r_3$. In that case one does not need to consider triples of transitions when reading $c_1$ but rather pairs of transitions $(t_{c_1}, t_{r_1})$. We have to treat this case separately by using the symbol $\bot$ instead of a transition in the next state.

Finally, 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 also be easily treated separately but to simplify the proof, we assume that the VPTs accepts only words of the form $cwr$, where $w$ is well-nested.

We now define the transition relation formally. Let $<$ be a total order on transitions, extended lexicographically to tuples. For all push and pop transitions $t_{c_1} \in \delta_c$ and $t_{r_2}, t_{r_1}, t_{r_3} \in \delta_r$, it is easy to define a VPA $A_{t_{c_1},t_{r_2},t_{r_1},t_{r_3}}$ whose size is polynomial in the size of $T$ that accepts a word $w$ iff it is of the form $c_1w_1c_2w_2r_2r_1w_3$ where $w_1, w_2, w_3$ are well-nested and there exists a run of $T$ on $w$ that uses transition $t_{c_1}$ on $c_1$, $t_{r_2}$ on $r_2$, $t_{r_1}$ on $r_1$ and $t_{r_3}$ on the last return symbol of $r_1w_3$ (which may be $r_1$ when $w_3 = \epsilon$). We denote by $\overline{A_{t_{c_1},t_{r_2},t_{r_1},t_{r_3}}}$ the complement of $A_{t_{c_1},t_{r_2},t_{r_1},t_{r_3}}$. For any transition $t \in \delta$, we denote by $st(t)$ the starting state of $t$ and by $in(t)$ the input letter of $t$. We let $B_{t_{c_1},t_{r_2},t_{r_1},t_{r_3}}$ a VPA with initial state $p_{t_{c_1},t_{r_2},t_{r_1},t_{r_3}}$ that defines the language:

$$L(B_{t_{c_1},t_{r_2},t_{r_1},t_{r_3}}) = L(A_{t_{c_1},t_{r_2},t_{r_1},t_{r_3}}) \cap \bigcap_{\substack{(t^*,t^*_{r_2},t^*_{r_1}) \in \delta_c \times \delta_r \times \delta_r \\ (t^*,t^*_{r_2},t^*_{r_1}) < (t_{c_1},t_{r_2},t_{r_1}) \\ in(t^*) = in(t_{c_1})\ st(t^*) = st(t_{c_1})}} L(\overline{A_{t^*,t^*_{r_2},t^*_{r_1},t_{r_3}}})$$

Such a VPA exists as VPAs are closed by intersection. Its size however may be exponential in $|\delta|$. Similarly we can define $C_{t_{c_1},t_{r_1},t_{r_3}}$ a VPA with initial state $p_{t_{c_1},t_{r_1},t_{r_3}}$ which accepts words of the form $c_1r_1w_3$ for which $(t_{c_1}, t_{r_1})$ is the minimal pair of transitions such that there exists a run of $T$ that uses $t_{c_1}$ on $c_1$, $t_{r_1}$ on $r_1$ and $t_{r_3}$ on the last return symbol of $r_1w_3$ (which may be $r_1$ when $w_3 = \epsilon$).

We define the look-ahead vpa as the union of all those VPAs, $A_{la} = \biguplus B_{t_{c_1},t_{r_2},t_{r_1},t_{r_3}} \uplus \biguplus C_{t_{c_1},t_{r_1},t_{r_3}}$. We now define the call and return transitions of $T'$ as follows, for all $c \in \Sigma_c, r \in \Sigma_r, \gamma \in \Gamma, t_c \in \delta_c, t_r, t'_r, t''_r \in \delta_r, \alpha \in \delta_r \cup \{\bot\}, u \in \Sigma^*$:

$$\begin{array}{rcll}
(q, t_r) & \xrightarrow{c|u,\ (\gamma,t_{r'},t_r),\ p_{t_c,t_{r''},t_{r'},t_r}} & (q', t_{r''}) & \text{if } t_c = q \xrightarrow{c|u,\gamma} q' \\
(q, t_r) & \xrightarrow{c|u,\ (\gamma,t_{r'},t_r),\ p_{t_c,t_{r'},t_r}} & (q', \bot) & \text{if } t_c = q \xrightarrow{c|u,\gamma} q' \\
q'_0 & \xrightarrow{c|u,\ (\gamma,t_r,t_r),\ p_{t_c,t_{r''},t_r,t_r}} & (q', t_{r''}) & \text{if } t_c = q \xrightarrow{c|u,\gamma} q' \\
q'_0 & \xrightarrow{c|u,\ (\gamma,t_r,t_r),\ p_{t_c,t_r,t_r}} & (q', \bot) & \text{if } t_c = q \xrightarrow{c|u,\gamma} q' \\
(q, \alpha) & \xrightarrow{r|u,(\gamma,t'_r,t''_r)} & (q', t''_r) & \text{if } t'_r = q \xrightarrow{r|u,\gamma} q'
\end{array}$$

Note that $T'$ is deterministic. Indeed, it is clear for return transitions as the next transition is fully determined by the top of the stack. For call transitions, one can check that the look-aheads are disjoint, i.e. for any $t_c, t'_c \in \delta_c$ such that $in(t_c) = in(t'_c)$ and $st(t_c) = st(t_c)$ and $t_1, t_2, t_3, t'_1, t'_2, t'_3 \in \delta_r$ we have $B_{t_c,t_1,t_2,t_3} \cap B_{t'_c,t'_1,t'_2,t'_3} = \emptyset$ (by definition of $B$s), $C_{t_c,t_1,t_2} \cap C_{t'_c,t'_1,t'_2} = \emptyset$ (by definition of $C$s), and $B_{t_c,t_1,t_2,t_3} \cap C_{t'_c,t'_1,t'_2} = \emptyset$ (because $B_{t_c,t_1,t_2,t_3} \subseteq \Sigma_c\Sigma_{wn}\Sigma_c\Sigma_{wn}\Sigma_r\Sigma_r\Sigma_{wn}$ and $C_{t'_c,t'_1,t'_2} \subseteq \Sigma_c\Sigma_r\Sigma_{wn}$). This ensures that $T'$ is deterministic. □

## 5 Functionality and Equivalence for $\mathsf{VPT}_{\mathsf{la}}$

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

**Theorem 3.** *Functionality of* $\mathsf{VPT}_{\mathsf{la}}$ *is* ExpTime-c*, even if the look-aheads are deterministic.*

*Proof.* For the ExpTime 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 by Proposition 1.

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 ExpTime-c when $n$ is part of the input [3]. Given $n$ deterministic top-down binary tree automata $T_1, \dots, T_n$ over an alphabet $\Delta$, one can construct in linear-time $n$ deterministic VPAs $A_1, \dots, A_n$ that define the same languages as $T_1, \dots, T_n$ respectively, modulo the natural encoding of trees as nested words over the structured alphabet $\tilde{\Delta} = \{c_a \mid a \in \Sigma\} \uplus \{r_a \mid a \in \Sigma\}$ [9]. The encoding corresponds to a depth-first left-to-right traversal of the tree. For instance, $enc(f(f(a,b),c)) = c_f c_f c_a r_a c_b r_b r_f c_c r_c r_f$. We now construct a $\mathsf{VPT}_{\mathsf{la}}$ $T$ over the alphabet $\tilde{\Delta}$ such that $T$ is functional iff $\bigcap_i L(T_i) = \varnothing$. The domain of $T$ are words of the form $w_{n,t} = c_1 r_1 \dots c_n r_n enc(t)$ for some ranked tree $t$ over $\Delta$. It is easy to define a VPA that define such words and whose size is polynomial in $n$ and $\Delta$. The $n$ first call symbols are used to run $n$ look-aheads. When the $i$-th call $c_i$ is read, a look-ahead $B_i$ checks that $enc(t) \in L(A_i)$: it first count that $2(n-i+1)$ symbols have been read and go to the initial state of $A_i$. The output words of $T$ are produced as follows: when reading $c_i$, there are two possible transitions, that both push the same symbol on the stack, launch the same look-ahead, and goes to the same state, but output two different words, let say $c_a$ and $r_a$ respectively. If the look-ahead is not accepting the suffix, then none of these transitions can be fired and the computation stops. Any run of $T$ that on the word $w_{n,t}$ is accepting. Therefore there is an accepting run of $T$ on $w_{n,t}$ iff $enc(t) \in \bigcap_i L(A_i)$, and in that case there are $2^n$ accepting runs whose output words are of the form $\alpha_1 \dots \alpha_n$ respectively where $\alpha_i \in \{c_a, r_a\}$. Therefore $T$ is not functional iff there is a tree $t$ such that $enc(t) \in \bigcap_i L(A_i)$, iff there is a tree $t$ such that $t \in \bigcap_i L(T_i)$. Note that the look-aheads are deterministic. □

We know by Proposition 1 that the functional equivalence of two VPTs is ExpTime-c. To decide the functional equivalence of two $\mathsf{VPTs}_{\mathsf{la}}$, one can first remove the look-aheads, modulo an exponential blow-up, and apply Proposition 1. This would yield a 2-ExpTime procedure for the equivalence of functional $\mathsf{VPTs}_{\mathsf{la}}$. However, it is possible to decide it in ExpTime:

**Theorem 4.** *Equivalence of functional* $\mathsf{VPT}_{\mathsf{la}}$ *is* ExpTime-c*, even if the transducers and the look-aheads are deterministic.*

*Proof.* **Lower bound** The lower-bound in the case where both the transducer and the look-ahead are deterministic is obtained similarly as the lower-bounds for functionality, i.e. by reduction of the emptiness of $n$ deterministic top-down tree automata $T_1, \ldots, T_n$. We construct two deterministic $\mathsf{VPTs_{la}}$ $T_1, T_2$ with deterministic look-aheads as in the proof of the lower-bound of Theorem 3, except that the output words are produced differently. On the word $c_1 r_1 \ldots c_n r_n enc(t)$, the transducer $T_1$ (resp. $T_2$) outputs $c_a$ (resp. $r_a$) when reading $c_i$ and only if the look-ahead is accepting, i.e. $t \in L(T_i)$. Both transducers are deterministic and as we saw, the look-aheads are also deterministic. Now, the image of $T_1$ (resp. $T_2$) is non-empty iff $\bigcap L(T_i) \neq \varnothing$ iff there exists a tree $t$ such that $(w_{n,t}, c_a^n) \in [\![T_1]\!]$ (resp. $(w_{n,t}, r_a^n) \in [\![T_2]\!]$). Therefore $T_1$ and $T_2$ are equivalent iff $\bigcap L(T_i) = \varnothing$.

**Upper bound** We now proceed to the proof of the ExpTime upper-bound. In a first step, we check equivalence of the respective domains of two $\mathsf{VPTs_{la}}$ (we show how to do it in ExpTime). Then we check the equivalence as follows: transform each $\mathsf{VPT_{la}}$ into an equivalent $\mathsf{VPT}$ with at most an exponential blow-up, take the union and verify (in PTime) that the resulting $\mathsf{VPT}$ is still functional.

We now show how to check the equivalence of he domains of two $\mathsf{VPTs_{la}}$. Let $T_1, T_2$ be two $\mathsf{VPTs_{la}}$. Their domains are defined by $\mathsf{VPAs}$ extended with regular look-aheads: it suffices to project away the output words in $T_1$ and $T_2$ respectively. We do not formally define $\mathsf{VPAs}$ with look-aheads, but they can be defined as $\mathsf{VPTs_{la}}$ without considering output words. Let us denote $A_1, A_2$ the $\mathsf{VPAs}$ with look-ahead that define $Dom(T_1)$ and $Dom(T_2)$ respectively. We show how to check $L(A_1) = L(A_2)$ in ExpTime. The idea is to reduce the problem to equivalence of finite alternating tree automata, which is known to be in ExpTime [3]. Well-nested words over the alphabet $\Sigma = \Sigma_c \uplus \Sigma_r$ can be translated as unranked trees over the alphabet $\tilde{\Sigma} = \Sigma_c \times \Sigma_r$. Those unranked trees can be again translated as binary trees via the classical first-child next-sibling encoding [3]. $\mathsf{VPAs}$ over $\Sigma$ can be translated into equivalent top-down tree automata over first-child next-sibling encodings on $\tilde{\Sigma}$ of well-nested words over $\Sigma$ in PTime [9]. Look-aheads of $\mathsf{VPAs}$ inspect the longest well-nested prefix of the current suffix. This corresponds to subtrees in the first-child next-sibling encodings. Therefore $\mathsf{VPAs}$ with look-aheads can be translated into top-down tree automata with look-aheads that inspect the current subtrees. Top-down tree automata with such look-aheads can be again translated into alternating tree automata: triggering a new look-ahead corresponds to a universal transition towards two states: the current state of the automaton and the initial state of the look-ahead. This again can be done in PTime. Since equivalence of finite alternating tree automata is in ExpTime [3], one gets an ExpTime upper-bound for testing equivalence of the domains of two $\mathsf{VPTs_{la}}$. □

## 6 Discussion and Conclusion

*Discussion* We discuss several variants of visibly look-aheads. Instead of inspecting the longest well-nested current prefix, one could visit only the current well-nested prefix of the form $cwr$. This would correspond to the first subtree

of the current hedge in encodings of unranked trees as well-nested words. While VPTs would still be closed by such look-aheads, we would not have a correspondence between deterministic $\mathsf{VPTs}_{\mathsf{la}}$ and functional VPTs anymore. For instance, the class of transductions defined in Prop. 2 would not be definable by a deterministic $\mathsf{VPT}_{\mathsf{la}}$, although it is a functional transduction. In some sense, our definition of look-ahead is the minimal requirement to get the equivalence between deterministic $\mathsf{VPTs}_{\mathsf{la}}$ and functional VPTs.

One could also allow look-aheads on return transitions. Such look-aheads would inspect the longest well-nested prefix starting just after the current return symbol. This could be easily simulated by look-aheads on call transitions only in PTime, and it would preserve determinism. Therefore our results still hold in this setting.

Another way of adding look-aheads is to allow them to inspect the whole current suffix. Such look-aheads can be defined by visibly pushdown automata where return transitions on empty stack are allowed, as originally defined in [1]. The construction of Theorem 1 can be slightly modified to show that VPTs are still closed by such look-aheads. The idea is extend the states with a new component $L_\perp \subseteq Q^{la}$ that corresponds to states of look-aheads that started at a deeper position than the current position. For those states we apply only return transitions on empty stack when reading a return symbol. See Appendix for a formal construction. Obviously, VPTs with such look-aheads still satisfy the correspondence between functional VPTs and deterministic VPTs with look-ahead. However, the proof of the ExpTime upper-bound for functional equivalence is not correct anymore as we cannot reduce the problem of testing equivalence of the domains to equivalence of alternating tree automata. We let the question of finding the exact complexity of functional equivalence for VPTs with regular look-ahead that inspect the whole suffix as future work.

*Conclusion* We have introduced visibly pushdown transducers with visibly look-aheads. We have shown that while they are exponentially more succinct than VPTs, they have the same complexity as VPTs wrt functional equivalence. Moreover, we have shown that deterministic VPTs with visibly look-aheads capture functional VPTs. This yields a simple characterization of functional VPTs. It is unknown whether this result extends to $k$-valued VPTs, i.e. whether $k$-valued VPTs are equivalent to $k$-ambiguous $\mathsf{VPTs}_{\mathsf{la}}$. This question seems more difficult than for functional VPTs. Indeed, let us assume that the look-aheads can visit the whole suffix (this setting is more powerful but already allows us to give some insights on the difficulty of this problem). For functional VPTs, when several transitions are possible, we know that any of them that can be completed into an accepting run can be triggered. Therefore one just has to order the transitions and, by using a look-ahead, trigger the smallest satisfying this property. For $k$-valued VPTs, among a set of possible transitions it is necessary to choose for each output word (among at most $k$ output words) exactly one transition, in order to turn $k$-valuedness into $k$-ambiguity. It is not clear how to use look-aheads to make such choices.

It is known that finite alternating word automata are doubly exponentially more succinct than finite deterministic automata. This question is open for VPTs with look-aheads and is interesting, as look-aheads are somehow particular universal transitions.

Finally, we would like to investigate those questions on more powerful models of transductions, where *parameters* are added to VPTs, in the spirit of macro tree transducers [7].

## References


1. R. Alur and P. Madhusudan. Visibly pushdown languages. In STOC, pages 202–211, 2004.
2. R. Alur and P. Madhusudan. Adding nesting structure to words. JACM, 56(3):1–43, 2009.
3. H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. http://www.grappa.univ-lille3.fr/tata, 2007.
4. J. Engelfriet. Top-down tree transducers with regular look-ahead. Mathematical Systems Theory, 10:289–303, 1977.
5. J. Engelfriet and S. Maneth. Macro tree translations of linear size increase are MSO definable. SICOMP, 32:950–1006, 2003.
6. J. Engelfriet and S. Maneth. The equivalence problem for deterministic MSO tree transducers is decidable. IPL, 100(5):206–212, 2006.
7. J. Engelfriet and H. Vogler. Macro tree transducers. JCSS, 31(1):71–146, 1985.
8. E. Filiot, J.-F. Raskin, P.-A. Reynier, F. Servais, and J.-M. Talbot. Properties of visibly pushdown transducers. In MFCS, pages 355–367, 2010.
9. O. Gauwin. Streaming Tree Automata and XPath. PhD thesis, Université Lille 1, 2009.
10. C. Koch and S. Scherzinger. Attribute grammars for scalable query processing on XML streams. VLDB, 16(3):317–342, 2007.
11. S. Maneth and F. Neven. Structured document transformations based on XSL. In DBPL, volume 1949 of LNCS, pages 80–98, 2000.
12. T. Perst and H. Seidl. Macro forest transducers. IPL, 89(3):141–149, 2004.
13. H. Seidl. Single-valuedness of tree transducers is decidable in polynomial time. TCS, 106(1):135–181, 1992.
14. H. Seidl. Equivalence of finite-valued tree transducers is decidable. MST, 27(4):285–346, 1994.
15. S. Staworko, G. Laurence, A. Lemay, and J. Niehren. Equivalence of deterministic nested word to word transducers. In FCT, volume 5699 of LNCS, pages 310–322, 2009.

# A Closure by Visibly Look-Heads Visiting the Whole Suffix

We consider in this section VPAs that can trigger return transitions on empty stack. Such a VPA is a tuple $(Q, q_0, F, \Gamma, \delta = \delta_r \uplus \delta_c \uplus \delta_\perp)$ where $\delta_\perp \subseteq Q \times \Sigma_r \times Q$ denotes the set of those new transitions.

We denote by $\mathsf{VPTs}^+_{\mathsf{la}}$ the set of visibly pushdown transducers with look-aheads that can inspect the whole suffix. We prove that VPTs are closed by such look-aheads. The construction is done by a slight modification of the construction given in the proof of Theorem 1.

The idea is extend the states a the constructed VPT with a new component $L_\perp \subseteq Q^{la}$ that corresponds to states of look-aheads that started at a deeper position than the current position., and such that any position in between the position at which they started and the current position is not above the current position. For instance, let us consider a well-nested word of the form $wcc_1w_1r_1 \dots c_nw_nr_nrw'$ where $w_i$ is well-nested. After reading $r_i$, $L_\perp$ contains current states of look-aheads that started when reading the words $w_1, \dots, w_n$. We therefore have two $L$-components: the look-aheads that started when reading $c_1, \dots, c_n$ and the new look-ahead component. When reading $c$, we push on the stack this new component. Let us formally define the construction. From a $\mathsf{VPT}^+_{\mathsf{la}}$ $T = (Q, q_0, F, \Gamma, \delta)$ with look-ahead $A = (Q^{la}, q_0^{la}, F^{la}, \Gamma^{la}, \delta^{la})$, we construct an equivalent VPT $T' = (Q', q_0', F', \Gamma', \delta')$ possibly exponentially bigger as follows:

- $Q' = Q \times 2^{Q^{la} \times Q^{la}} \times 2^{Q^{la}} \times 2^{Q^{la}}$;
- $\Gamma' = \Gamma \times 2^{Q^{la} \times Q^{la}} \times 2^{Q^{la}} \times 2^{Q^{la}} \times \Sigma_c$;
- $F' = \{(q, R, L, L_\perp) \in Q' \mid q \in F, L \subseteq F^{la}, L_\perp \subseteq F^{la}\}$.
- $q_0' = (q_0, Id_{Q^{la}}, \varnothing, \varnothing)$.

The transitions are defined as follows:
First, for all $q, R, L, L_\perp, c, \gamma$, we have:

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

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

$$(q'', R'', L'', L''_\perp) \xrightarrow{r|u,(\gamma,R,L,L_\perp,c)} (q', R', L', L'_\perp) \in \delta'_r$$

if the following conditions hold:

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