Visibly Pushdown Transducers with Look-Ahead

Emmanuel Filiot[1] Frédéric Servais[1]

[1] Université Libre de Bruxelles

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 ($\mathsf{VPT}_{\mathsf{la}}$). 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 $\mathsf{VPT}_{\mathsf{la}}$ are not more expressive than VPT, but are exponentially more succinct. Second, we show that the class of deterministic $\mathsf{VPT}_{\mathsf{la}}$ corresponds exactly to the class of functional VPT, yielding a simple characterization of functional VPT. Finally, we show that while $\mathsf{VPT}_{\mathsf{la}}$ are exponentially more succinct than VPT, checking equivalence of functional $\mathsf{VPT}_{\mathsf{la}}$ is, as for VPT, EXPTIME-C. As a consequence of these results, we show that for any functional VPT there is an equivalent unambiguous one.

1 Introduction

Visibly pushdown transducers (VPT) [12, 8] 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 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 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 can be defined by VPT [1].

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.

[1] In this paper when it is clear from the context, we also use VPT to denote visibly pushdown transductions

In this paper, we therefore introduce and investigate the class of VPT with visibly pushdown look-ahead. A VPT with visibly pushdown 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 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. Our main contributions are the following:

1. $\mathsf{VPT}_{\mathsf{la}}$ *are as expressive as* VPT*, 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* $\mathsf{VPT}_{\mathsf{la}}$ *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 an elegant and simple characterization of functional VPT.

3. *Equivalence of functional* $\mathsf{VPT}_{\mathsf{la}}$ *is, as for* VPT*,* EXPTIME-C.

Therefore even though $\mathsf{VPT}_{\mathsf{la}}$ are exponentially more succinct than VPT, testing equivalence of functional $\mathsf{VPT}_{\mathsf{la}}$ 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 EXPTIME-C for $\mathsf{VPT}_{\mathsf{la}}$: get rid of the look-aheads with an exponential blow-up and test in PTIME 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 $\mathsf{VPT}_{\mathsf{la}}$ are linearly reducible to alternating top-down tree automata. Testing the equivalence of such automata can be done in EXPTIME [3].

4. *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 3 is that functional VPT are effectively characterized by unambiguous VPT. This result was already known for finite-state transducers [4, 11, 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 [13] and recently with a better upper-bound [10] based on lexicographic decomposition of transducers.

Finally, we discuss slightly different look-aheads. First, we consider look-aheads that are allowed to inspect the whole current prefix until the end. We show this does not add expressivity nor succinctness. Second, we show that restricting the look-ahead to the current prefix in between the current call and its matching return is not sufficient to have a characterization of functional VPT by deterministic $\mathsf{VPT}_{\mathsf{la}}$. All these results are new indications that the class of VPT is robust, and they show that the class of $\mathsf{VPT}_{\mathsf{la}}$ 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 [6], or to *macro tree transducers*

(MTT) [7]. For top-down tree transducers, adding regular look-ahead strictly increases their expressive power while MTT are closed by regular look-ahead [7].

Trees over an alphabet Σ 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 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 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.

Modulo those encodings, MTT subsume VPT [8] 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.

2 Visibly Pushdown Languages and Transductions

All over this paper, Σ denotes a finite alphabet partitioned into two disjoint sets[2] Σ_c, Σ_r, denoting respectively the *call* and *return* alphabets. We denote by Σ^* the set of (finite) words over Σ and by ϵ the empty word. The length of a word u is denoted by $|u|$. The set of *well-nested* words Σ^*_{wn} is the smallest subset of Σ^* 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 $uv \in \Sigma^*_{\mathsf{wn}}$.

A *visibly pushdown automaton* (VPA) [1] on finite words over Σ 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, Γ 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 $(q, a, \gamma, q') \in \delta_c$, γ is pushed onto the stack and the control goes from q to q'. On a return transition $(q, a, \gamma, q') \in \delta_r$, γ 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, σ) to a configuration (q', σ') 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 ρ is *accepting* if $q_0 \in I$, $q_l \in F$ and $\sigma_0 = \sigma_l = \perp$ [3]. 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 Σ is a *visibly pushdown language* if there is a VPA A over Σ such that $L(A) = L$. Finally, a

[2] 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.

[3] 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.

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 [8].

As finite-state transducers extend finite-state automata with outputs, visibly pushdown transducers extend visibly pushdown automata with outputs [8]. To simplify notations, we suppose that the output alphabet is Σ, 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 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 Σ 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, Γ 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 $\rho = \{(q_k, \sigma_k)\}_{0 \le k \le l}$ on u and l words $v_1, \dots, v_l$ such that $v = v_1 \dots v_l$ and for all $0 \le k < l$, there is a transition of T from (q_k, σ_k) to (q_{k+1}, σ_{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, σ) to (q', σ') 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 $[\![T_1]\!] = [\![T_2]\!]$. It is known [8] that functionality is decidable in PTIME for VPT and equivalence of functional VPT is EXPTIME-C.

The class of functional VPT 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 by $Q = \{q, q_a\}$, $I = \{q\}$, $F = Q$, $\Gamma = \{(a, a), (a, \neg a), (\neg a, a), (\neg a, \neg a), (-, a)\}$ and δ 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 that 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 (α, β) 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 β, carry over the fact that there is or not an a in the well nested word ending at the current call.

3 VPT with Visibly Pushdown Look-Ahead

Given a word w over Σ 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-ahead 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 (*VPT_{la}*) is a pair* $T_{la} = (T, A)$ *where* A *is a* VPA $A = (Q^{la}, F^{la}, \Gamma^{la}, \delta^{la})$ *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,* Γ *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 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 $p \in Q^{la}$ 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 ρ 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 VPT_{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 some VPT_{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 the transition from state q and reading c are *pairwise* disjoint. The number of states of a VPT_{la} is the number of states of the transducer plus the number of states of the look-ahead.

Example 2. A VPT_{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 of Example 1. When starting in state q_a, respectively $q_{\neg a}$, 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

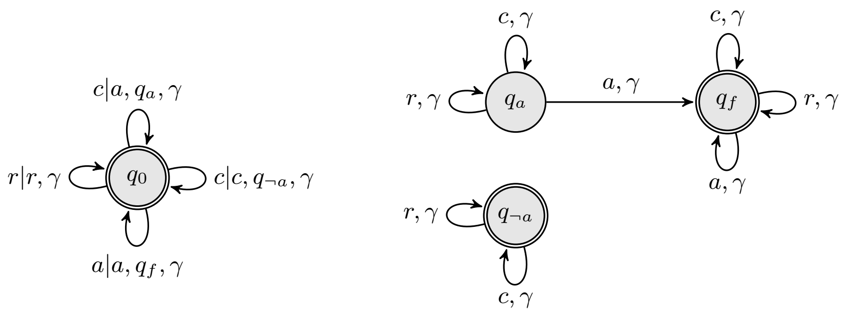

Fig. 1. A $\mathsf{VPT}_{\mathsf{la}}$ (left) and its look-ahead (right) on $\Sigma_c = \{c, a\}$ and $\Sigma_r = \{r\}$

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 $q_{\neg a}$ states of the look-ahead automaton. Other input symbols, i.e. a and r, are just copied to the output (left and bottom transitions).

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 life until the end of the well-nested subword starting at this call. We use summaries [1] to handle look-aheads that started at a strictly less deeper nesting level and a subset construction for those that started at the same nesting level.

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

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

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

The transducers 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 (see [1]) 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-ahead 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''.

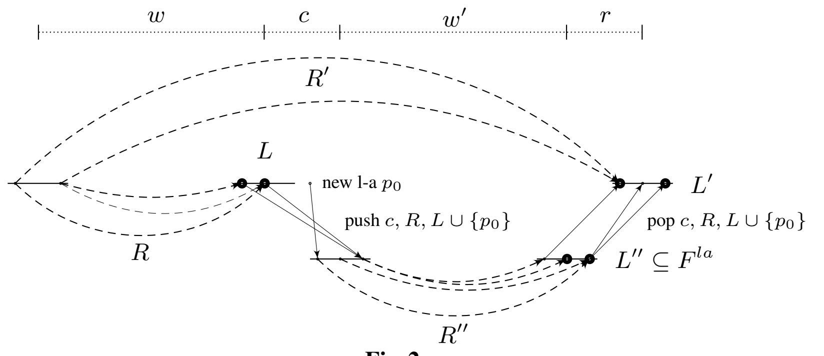

Fig. 2.

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 $(q', Id_{Q^{la}}, \varnothing)$ 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 γ, 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, γ, 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) $R' = \{(p, p') \mid \exists s \xrightarrow{c,\gamma} s' \in \delta^{la}_c \cdot \exists (s', s'') \in R'' \cdot (p, s) \in R \text{ and } s'' \xrightarrow{r,\gamma} p' \in \delta^{la}_r\}$
(iv) for all $p \in L$, there exist $p' \in L'$, $\gamma \in \varGamma$, $s, s' \in Q^{la}$ such that $(s, s') \in R''$, $p \xrightarrow{c,\gamma} s \in \delta^{la}_c$, $s' \xrightarrow{r,\gamma} p' \in \delta^{la}_r$.

The proof of correctness is sketched in Appendix. If T is deterministic, then T' is unambigous. 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. □

Proposition 1. *Let Σ be a finite alphabet with at least two letters. There exists a family of transductions $(T_n)_n$ over Σ such that for all $n \geq 0$, T_n is definable by a* $\mathsf{VPT}_{\mathsf{la}}$ *with $O(n)$ states and any* VPT *defining L_n has at least $O(2^n)$ states.*

Proof (sketch). We show that the result already holds for finite state automata with regular look-ahead, which yields a construction for $\mathsf{VPT}_{\mathsf{la}}$. VPT on flat words (in $(\Sigma_c\Sigma_r)^*$) can indeed simulate finite state automata (in that case the stack is useless). For all n, we define the language $L_n = \{vuv \mid |v| = n\}$. We show that L_n can be recognized by a finite state automaton with regular look-ahead with $O(n)$ states while without regular look-ahead any automaton would need at least $|\Sigma|^n$ states. □

4 Functional VPT and $\mathsf{VPT}_{\mathsf{la}}$

While there is no known syntactic restriction on VPT that captures all functional VPT, we show that the class of deterministic $\mathsf{VPT}_{\mathsf{la}}$ captures all functional VPT. As there may be an unbounded number of accepting runs, the equivalent $\mathsf{VPT}_{\mathsf{la}}$ has to choose only one of them by using look-aheads. This is done by ordering the states and extending this order to runs. The main difficulty 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.

Theorem 2. *For all functional* VPT *T, one can construct a deterministic* $\mathsf{VPT}_{\mathsf{la}}$ *T_{la} with exponentially many more states such that $[\![T]\!] = [\![T_{la}]\!]$.*

Proof. It is clear that deterministic $\mathsf{VPT}_{\mathsf{la}}$ are functional VPT. For the converse, 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 functional 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 an equivalent deterministic $\mathsf{VPT}_{\mathsf{la}}$ (T', A) 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 δ' 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 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 _. When reading r_1, T' pops from the stack the tuple (γ, 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.

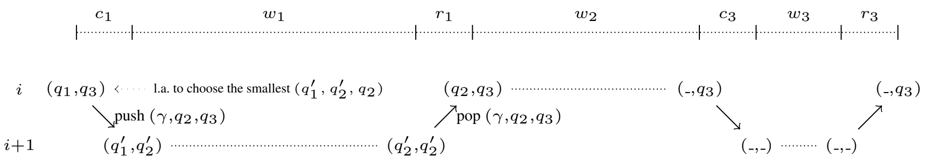

Fig. 3.

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.

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 VPA 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 VPA, $A_{la} = \biguplus B_{q_1,q_1',q_2',q_2,q_3}$. We now define the call and return transitions of T' as follows, for all $c \in \Sigma_c, r \in \Sigma_r, \gamma \in \Gamma, q_1, q_1', q_2', q_3, q \in Q, u \in \Sigma^*$:

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

The transducer T' is deterministic: return transitions are fully determined by the states q_2', q_2, q_3 and the input letter r (by our first assumption there is at most one transition in T from q_2' to q_2). For call transitions, suppose that from (q_1, q_3) there are two possible look-aheads $p_{q_1,q_1',q_2',q_2,q_3}$ and $p_{q_1,s_1',s_2',s_2,s_3}$. By definition of the look-aheads, we have $L(B_{q_1,q_1',q_2',q_2,q_3}) \cap L(B_{q_1,q_1',q_2',q_2,q_3}) = \varnothing$. Moreover, there cannot be two transitions with the same look-ahead as transitions are fully determinied by $q_1, q_3, q_2, q_2', q_1'$ (there is at most one call transition by our assumption with source and target states q_1 and q_1' respectively). A simple analysis of the complexity shows that the look-ahead A has exponentially many more states than T (the exponentiation comes from the complement in the definition of $B_{q_1,q_1',q_2',q_2,q_3}$). □

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 unambigous* VPT T'.

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

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

Theorem 4. *Functionality of* VPT$_{\mathsf{la}}$ *is* EXPTIME-C*, even for deterministic look-aheads.*

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 [8]. 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]. The full proof is in Appendix.

We know that the equivalence of two functional VPT is EXPTIME-C [8]. To decide the equivalence of two functional VPT$_{\mathsf{la}}$, one can first remove the look-aheads, modulo an exponential blow-up, and use the procedure for VPT. This would yield a 2-EXPTIME procedure for the equivalence of functional VPT$_{\mathsf{la}}$. However, it is possible to decide it in EXPTIME:

Theorem 5. *Equivalence of functional* VPT$_{\mathsf{la}}$ *is* EXPTIME-C*, even if the transducers and the look-aheads are deterministic.*

Proof. The lower bound is obtained similarly as for functionality (see Appendix). For the upper-bound, in a first step we check equivalence of the respective domain of two $\mathsf{VPT}_{\mathsf{la}}$ (we show how to do it in EXPTIME). Then we check the equivalence as follows: transform each $\mathsf{VPT}_{\mathsf{la}}$ 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.

We now show how to check the equivalence of the domain of two $\mathsf{VPT}_{\mathsf{la}}$. Let T_1, T_2 be two $\mathsf{VPT}_{\mathsf{la}}$. Their domains are defined by VPA extended with visibly pushdown look-aheads: it suffices to project away the output words in T_1 and T_2 respectively. We do not formally define VPA with look-aheads, but they can be defined as $\mathsf{VPT}_{\mathsf{la}}$ without considering output words. Let us denote A_1, A_2 the VPA with look-ahead that define $\mathit{Dom}(T_1)$ and $\mathit{Dom}(T_2)$ 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]. VPA over Σ can be translated into equivalent top-down tree automata over first-child next-sibling encodings on $\tilde{\Sigma}$ of well-nested words over Σ in PTIME [9]. Look-aheads of VPA inspect the longest well-nested prefix of the current suffix. This corresponds to subtrees in first-child next-sibling encodings of unranked trees. Therefore VPA with look-aheads can be translated into top-down tree automata with look-aheads that inspect the current subtree. 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 $\mathsf{VPT}_{\mathsf{la}}$. $\square$

6 Discussion and Conclusion

Discussion We discuss several variants of visibly pushdown 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 VPT would still be closed by such look-aheads, we would not have a correspondence between deterministic $\mathsf{VPT}_{\mathsf{la}}$ and functional VPT anymore. For instance, the class of transductions defined in Prop. 1 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 $\mathsf{VPT}_{\mathsf{la}}$ and functional VPT.

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 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 VPT are still closed by such look-aheads. The idea is to 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, VPT with such look-aheads still satisfy the correspondence between functional VPT and deterministic VPT with look-ahead. However, the proof of the EXPTIME upper-bound for functional equivalence cannot be adapted 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 VPT with visibly pushdown look-ahead that inspect the whole suffix as future work.

Finally, note that most of our results also hold for VPA extended with visibly pushdown look-aheads: closure of VPA by visibly pushdown look-aheads, succinctness of VPA with look-aheads, and EXPTIME-completeness of equivalence of VPA with look-ahead (even for deterministic VPA and look-aheads).

Future Work It is unknown whether k-valued VPT are equivalent to k-ambiguous $\mathsf{VPT}_{\mathsf{la}}$. This question seems more difficult than for functional VPT. 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 VPT, 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 VPT, 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.

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, 2007.
4. S. Eilenberg. Automata, Languages, and Machines. Academic Press, Inc., 1974.
5. C. C. Elgot and J. E. Mezei. On relations defined by generalized finite automata. IBM Journal of Research and Development, 9:47–68, 1965.
6. J. Engelfriet. Top-down tree transducers with regular look-ahead. Mathematical Systems Theory, 10:289–303, 1977.
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. J. Sakarovitch and R. de Souza. Lexicographic decomposition of k -valued transducers. TCS, 47(3):758–785, 2010.
11. M. P. Schützenberger. Sur les relations rationnelles entre monoides libres. TCS, 3(2):243–259, 1976.
12. 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.
13. A. Weber. Decomposing finite-valued transducers and deciding their equivalence. SIAM Journal on Computing, 22(1):175–202, 1993.

A VPT with Visibly Pushdown Look-Ahead

Proof (Correctness of the construction of Theorem 1). We sketch the proof of 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 $w = c_1c_2r_2c_3r_3$, then $sh(w) = c_2r_2c_3r_3$. However if $w = c_1c_2$, 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 $w = c_1w_1r_1c_2w_2r_2 \dots c_nw_nr_n$ 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. □

Proof (Proof of Proposition 1). We show that the exponential blow-up already holds with finite state automaton with regular look-ahead. This same blow-up is then obtained for $\mathsf{VPT}_{\mathsf{la}}$ accepting only flat words, i.e. words in $(\Sigma_c\Sigma_r)^*$ (in that case the stack is useless). On such words each look-ahead visits the suffix of the whole word starting at the position it is triggered. More precisely, we can construct a $\mathsf{VPT}_{\mathsf{la}}$ such that any VPT needs an exponential number of states to have the same domain of the given VPT. We now show how the exponential blow-up is obtained on finite state automata with look-ahead.

Let $n \in \mathbb{N}$ and $L_n = \{vuv \mid |v| = n\}$. L_n is definable by a finite state automaton with look-ahead A_n with $O(n)$ states. Let $w = a_1 \dots a_nw'$ and $m = |w|$. 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 w is equal to 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. A_n as $O(n)$ states.

Without a regular look-ahead, an 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. □

.

B Closure by Visibly Pushdown Look-Ahead Inspecting the Whole Suffix

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

We denote by $\mathsf{VPT}_{\mathsf{la}}^{+}$ the set of visibly pushdown transducers with look-aheads that can inspect the whole suffix. We prove that VPT 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 $L_\bot \subseteq Q^{la}$ 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, $L_\bot$ 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:

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

The transitions are defined as follows:

First, for all $q, R, L, L_\bot, c, \gamma$, we have:

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

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

$$(q'', R'', L'', L_\bot'') \xrightarrow{r|u,(\gamma,R,L,L_\bot,c)} (q', R', L', L_\bot') \in \delta_r'$$

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. $L_\bot$), there exist $p' \in L'$ (resp. $L_\bot'$), $\gamma \in \Gamma$, $s, s' \in Q^{la}$ 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 $p \in L_\bot''$, there exist $p' \in L_\bot'$, such that $p \xrightarrow{r} p' \in \delta_\bot^{la}$.

C Decisions Problems

Proof of Theorem 4, Lower Bound Given n deterministic top-down binary tree automata $T_1, \dots, T_n$ over an alphabet Δ, one can construct in linear-time n deterministic VPA $A_1, \dots, A_n$ 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 $\tilde{\Delta} = \{c_a \mid a \in \Delta\} \uplus \{r_a \mid a \in \Delta\}$ [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 Δ. It is easy to define a VPA that accepts such words and whose size is polynomial in n and Δ. 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 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. □

Proof of the lower bound for Theorem 5 **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 $T_1, \dots, T_n$. We construct two deterministic $\mathsf{VPT}_{\mathsf{la}}$ T_1, T_2 with deterministic look-aheads as in the proof of the lower-bound of Theorem 4, except that the output words are produced differently. On the word $c_1 r_1 \dots c_n r_n enc(t)$, 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 $\bigcap L(T_i) = \varnothing$.