Properties of Visibly Pushdown Transducers*

Emmanuel Filiot[1], Jean-François Raskin[1], Pierre-Alain Reynier[2], Frédéric Servais[1], and Jean-Marc Talbot[2]

[1] Université Libre de Bruxelles (U.L.B.)
[2] LIF, Université Aix-Marseille & CNRS

Abstract. Visibly pushdown transducers (VPTs) form a strict subclass of pushdown transducers (PTs) that extends finite state transducers with a stack. Like visibly pushdown automata, the input symbols determine the stack operations. It has been shown that visibly pushdown languages form a robust subclass of context-free languages. Along the same line, we show that word transductions defined by VPTs enjoy strong properties, in contrast to PTs. In particular, functionality is decidable in PTIME, k-valuedness is in NPTIME and equivalence of (non-deterministic) functional VPTs is EXPTIME-C. Those problems are undecidable for PTs. Output words of VPTs are not necessarily well-nested. We identify a general subclass of VPTs that produce well-nested words, which is closed by composition, and for which the type checking problem is decidable.

1 Introduction

Visibly pushdown languages (VPLs) form a robust subclass of context-free languages (CFLs) [2]. This class strictly extends the class of regular languages and still enjoys strong properties: closure under all Boolean operators and decidability of emptiness, universality, inclusion and equivalence. On the contrary, context-free languages are not closed under complement nor under intersection, moreover universality, inclusion and equivalence are all undecidable. Along the same line, we study the class of visibly pushdown transductions, a subclass of pushdown transductions, and we show that while extending regular transductions it also preserves desired properties.

Visibly pushdown automata (VPAs), that characterize VPLs, are obtained as a restriction of pushdown automata. In these automata the input symbols determine the stack operations. The input alphabet is partitioned into call, return and internal symbols: if a call is read, the automaton must push a symbol on the stack; if it reads a return, it must pop a symbol; and while reading an internal symbol, it can not touch, not even read, the stack. *Visibly pushdown transducers* (VPTs) are obtained by adding outputs to VPAs: each time the VPA reads an input symbol it also outputs a word. No restriction is imposed on the output word.

* Work supported by the projects: (i) QUASIMODO (FP7- ICT-STREP-214755), (ii) GASICS (ESF-EUROCORES LogiCCC), (iii) Moves: "Fundamental Issues in Modelling, Verification and Evolution of Software", `http://moves.ulb.ac.be`, a PAI program funded by the Federal Belgian Gouvernment, and (iv) ECSPER (ANR-JC09-472677) and SFINCS (ANR-07-SESU-012), two projects supported by the French National Research Agency.

P. Hliněný and A. Kučera (Eds.): MFCS 2010, LNCS 6281, pp. 355–367, 2010.

A transducer is k-valued if it transforms an input word into at most k (output) words. It is *functional* if it is 1-valued. The functionality problem for finite state (word) transducers has been extensively studied. The first proof of decidability was given in [19], and later in [4]. The first PTIME upper bound has been proved in [10], more generally this algorithm can be used for deciding k-valuedness. Also, this proof can be refined to get an NLOGSPACE upper bound [9]. An efficient procedure for testing functionality has been given in [3]. Those problems are undecidable for PTs.

We claim that VPTs form a rather robust model between finite-state transducers (FSTs) and pushdown transducers (PTs). Our main contribution is to show that interesting problems are decidable for VPTs: inclusion of the domain into a VPL, functionality (in PTIME), equivalence of functional transducers (EXPTIME-C), and most notably k-valuedness (in NPTIME). VPTs are not closed under composition and the type checking is undecidable. We exhibit the class of well-nested VPTs (wnVPTs), a subclass of VPTs, closed under composition and with decidable type checking (EXPTIME-C). As the output words are well-nested, this class is well-suited to model unranked tree transformations. To the best of our knowledge, this is the first class of unranked tree transducers that supports concatenation of tree sequences with decidable k-valuedness.

Visibly pushdown transducers have been first introduced in [18]. In that paper, VPTs allow for ϵ-transitions that can produce outputs and only a single letter can be produced by each transition. Using ϵ-transitions causes many interesting problems to be undecidable, such as functionality and equivalence (even of functional transducers). In contrast to [18], in this paper we consider visibly pushdown transducers where ϵ-transitions are not allowed, but where the transitions can output a word. Moreover, no visibly restriction is imposed on the output word. Therefore in the sequel we call the transducers of [18] ϵ-VPTs, and VPTs will denote the visibly pushdown transducers considered here. VPTs are exactly the so called *nested word to word transducers* of [23]. XML-DPDTs [14], equivalent to left-to-right attribute grammars, correspond to the deterministic VPTs.

Deciding equivalence of deterministic (and therefore functional) VPTs has been shown to be in PTIME [23]. However, functional VPTs can be exponentially more succinct and are strictly more expressive than deterministic VPTs. In particular, non-determinism is often needed to model functional transformations whose current production depends on some input which may be arbitrary far away from the current input. For instance, the transformation that swaps the first and the last input symbols is functional but non-determinism is needed to guess the last input. The proof of [23] is based on a reduction to the morphism equivalence problem on a context-free language, which is known to be decidable in PTIME [17]. In this paper, we show that the same reduction can be applied to prove that functionality is decidable in PTIME for VPTs, and as a consequence, equivalence of functional transducers is decidable. Moreover we extend this result by showing that the k-valuedness problem for VPTs can be reduced to the multiple morphism equivalence problem on a CFL which was proved to be decidable in [11]. Finally, we show this last problem can be decided in NPTIME.

While functionality and equivalence are decidable, the class of visibly pushdown transductions, characterized by VPTs, is not closed under composition and the type checking problem is undecidable. We identify a subclass of VPTs in which a connection is imposed between stack symbols and nesting of outputs. We call the resulting class

well-nested visibly pushdown transducers (wnVPTs), as it only accepts well-nested input words and the output condition ensures that output words are also well-nested. As a subclass of VPTs, it enjoys all the good properties stated above, and in addition it is closed under composition and type checking is EXPTIME-C. This class is of particular interest for XML document transformations as well-nested words can naturally model such documents.

Relation to tree transducers. We distinguish *ranked trees* from *unranked trees*, whose nodes may have an arbitrary number of ordered children. There is a strong connection between wnVPTs and unranked tree transducers. Indeed, unranked trees over an arbitrary finite alphabet Σ can be naturally represented by well-nested words over the structured alphabet $\Sigma \times \{c\} \cup \Sigma \times \{r\}$. wnVPTs are therefore well-suited to model tree transformations. To the best of our knowledge, wnVPTs consist in the first (nondeterministic) model of unranked tree transformations (that support concatenation of tree sequences) for which k-valuedness and equivalence of functional transformations are decidable. Finite (ranked) tree transducers [5] on binary encodings of unranked trees do not support concatenation, they are incomparable to wnVPTs, as they allow for copy, which is not the case of wnVPTs, but cannot define all context-free languages as codomain, what wnVPTs can do, as they support concatenation of tree sequences. Functionality is known to be decidable in PTIME for tree transducers [20]. More generally, finite-valuedness of (and equivalence of finite-valued) tree transducers is decidable [21]. However, those results cannot be lifted to unranked trees, as unranked tree transformations have to support concatenation of tree sequences, making usual binary encodings unsuitable. Considering finite tree transducers, their ability to copy subtrees is the main concern when dealing with k-valuedness. However for wnVPTs, it is more their ability to concatenate sequences of trees which makes this problem difficult. While concatenation of tree sequences cannot be modeled by ranked tree transducers on binary encodings, it can be simulated by *macro (ranked) tree transducers* (MTTs) by using parameters [8]. This makes wnVPTs strictly less expressive than MTTs. However, equivalence is decidable for the subclass of linear size increase *deterministic* MTTs, which are equivalent to deterministic MSO tree transductions [7,6]. In particular, k-valuedness is still open for MTTs. There have been several attempts to generalize ranked tree transducers to unranked tree transducers [15,16,14], but again, k-valuedness is still open.

Organization of the paper In Section 2, we define VPTs as an extension of VPAs. In Section 3, we give basic results about properties of VPTs. We prove in Section 4 the PTIME decidability of functionality for VPTs as well as the NPTIME result for deciding k-valuedness of VPTs. Finally, well-nested transducers are introduced in Section 5.

2 Visibly Pushdown Transducers

Let Σ be a finite alphabet partitioned into two disjoint sets Σ_c and Σ_r denoting respectively the *call* and *return* alphabets[1]. We denote by Σ^* the set of (finite) words over Σ

[1] In contrast to [2], 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.

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 $uv \in \Sigma^*_{\mathsf{wn}}$. The *height* of a well-nested word is inductively defined by $h(\epsilon) = 0$, $h(cur) = 1 + h(u)$, and $h(uv) = \max(h(u), h(v))$.

Visibly Pushdown Languages A *visibly pushdown automaton* (VPA) [2] on finite words over Σ is a tuple $A = (Q, I, F, \Gamma, \delta)$ where Q is a finite set of states, $I \subseteq Q$, respectively $F \subseteq Q$, the set of initial states, respectively 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 $(q, a, q', \gamma) \in \delta_c$, γ is pushed onto the stack and the control goes from q to q'. On a return transition $(q, \gamma, a, q') \in \delta_r$, γ is popped from the stack.

Stacks are elements of Γ^*, and we denote by $\perp$ the empty stack. A *run* of a VPA A on a word $w = a_1 \dots a_l$ is a sequence $\{(q_k, \sigma_k)\}_{0 \leq k \leq l}$, where q_k is the state and $\sigma_k \in \Gamma^*$ is the stack at step k, such that $q_0 \in I$, $\sigma_0 = \perp$, and for each $k < l$, we have either: (i) $(q_k, a_{k+1}, \gamma, q_{k+1}) \in \delta_c$ and $\sigma_{k+1} = \sigma_k\gamma$; or (ii) $(q_k, a_{k+1}, \gamma, q_{k+1}) \in \delta_r$, and $\sigma_k = \sigma_{k+1}\gamma$. A run is *accepting* if $q_l \in F$ and $\sigma_l = \perp$. A word w is *accepted* by A if there exists an accepting run of A over w. $L(A)$, the *language* of A, is the set of words accepted by A. A language L over Σ is a *visibly pushdown language* if there is a VPA A over Σ such that $L(A) = L$.

In contrast to [2] and to ease the notations, we do not allow return transitions on the empty stack and we accept on empty stack only. Therefore the words accepted by a VPA are well-nested (every call symbol has a matching return symbol and conversely).

Visibly Pushdown Transducers As finite-state transducers extend finite-state automata with outputs, visibly pushdown transducers extend VPAs with outputs. To simplify notations, we suppose that the output alphabet is Σ, but our results still hold for an arbitrary output alphabet.

Definition 1 (Visibly pushdown transducers). A *visibly pushdown transducer*[2] (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 $\delta_c \subseteq Q \times \Sigma_c \times \Sigma^* \times \Gamma \times Q$, $\delta_r \subseteq Q \times \Sigma_r \times \Sigma^* \times \Gamma \times Q$.

A *configuration* of a VPT 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 exist $v_k \in \Sigma^*$ and $\gamma_k \in \Gamma$ such that $(q_{k-1}, a_k, v_k, \gamma_k, q_k) \in \delta_c$ and either $a_k \in \Sigma_c$ and $\sigma_k = \sigma_{k-1}\gamma_k$, or $a_k \in \Sigma_r$ and $\sigma_{k-1} = \sigma_k\gamma_k$. The word $v = v_1 \dots v_l$ is called an *output* of ρ. 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. The transducer T defines a binary word relation $[\![T]\!] = \{(u, v) \mid \exists q \in I, q' \in F,\ (q, \perp) \xrightarrow{u/v} (q', \perp)\}$.

[2] In contrast to [18], there is no producing ϵ-transitions (inserting transitions) but a transition may produce a word and not only a single symbol. Moreover, the images of a word may not be necessarily well-nested.

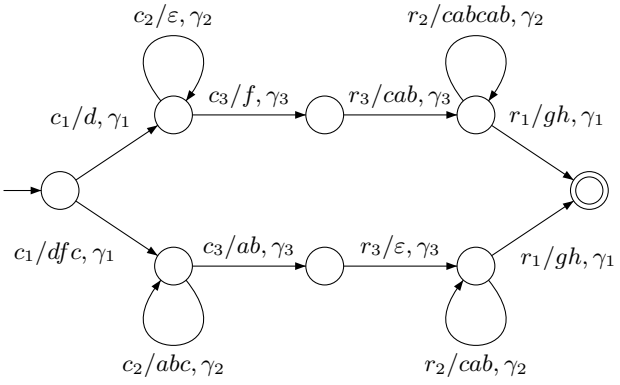

Fig. 1. A functional VPT on $\Sigma_c = \{c_1, c_2, c_3\}$ and $\Sigma_r = \{r_1, r_2, r_3\}$

The *domain* of T (denoted by $Dom(T)$), resp. the *codomain* of T (denoted by $CoDom(T)$), is the domain of $[\![T]\!]$, resp. the codomain of $[\![T]\!]$. Note that the domain of T contains only well-nested words, which is not necessarily the case of the codomain.

Consider the VPT T of Figure 1. Call (resp. return) symbols are denoted by c (resp. r). The domain of T is $Dom(T) = \{c_1(c_2)^n c_3 r_3 (r_2)^n r_1 \mid n \in \mathbb{N}\}$. For each word of $Dom(T)$, there are two accepting runs, corresponding respectively to the upper and lower part of T. For instance, when reading c_1, it pushes γ_1 and produces either d (upper part) or *dfc* (lower part). By following the upper part (resp. lower part), it produces words of the form $dfcab(cabcab)^n gh$ (resp. $dfc(abc)^n ab(cab)^n gh$). Therefore T is functional.

3 Properties of VPTs

In this section, we present results about expressiveness of VPTs and decision problems. We let VPLs, resp. CFLs, denote the class of visibly pushdown languages, resp. context-free languages, over Σ.

Proposition 1 (Domain and codomain). *Let T be a* VPT*, let L be a* VPL*. The domain $Dom(T)$ of T is a* VPL *and the language $T(L)$ is a* CFL*. Moreover, for any* CFL *L' over Σ, there exists a* VPT *whose codomain $CoDom(T)$ is L'. All these constructions can effectively be done in polynomial time.*

Proof (Sketch). By projecting the transitions of a VPT T on the input (resp. on the output), we obtain a VPA (resp. a pushdown automaton) which defines $Dom(T)$ (resp. $CoDom(T)$). As a consequence, given $L \in$ VPL, by restricting $Dom(T)$ to L which can be done by a classical product construction, we obtain $T(L)$ is a CFL. To produce as output a CFL L' on alphabet Σ, it is already known [2] that there exists a VPL L'' on a structured alphabet $\hat{\Sigma}$ and a renaming $\pi : \hat{\Sigma} \to \Sigma$ such that $\pi(L'') = L'$. The VPT implements π. □

As a consequence of Proposition 1 and of the fact that language inclusion for VPAs is EXPTIME-C, we can test whether a VPL is included in the domain of a given VPT

and conversely. This is of particular interest for XML transformations as it amounts to decide if any document valid for an input XML schema will be transformed. This is undecidable for ϵ-VPTs and PTs.

Thanks to non-determinism, transductions defined by VPTs are closed under *union*. This is not the case however for *composition* and *inverse*. Non-closure under composition can be simply proved by using Proposition 1 and by producing two VPTs whose composition transforms a VPL into a non CFL language. More formally, let $\Sigma = \{c_1, r_1, c_2, r_2, c_3, r_3\}$ be an alphabet where c_i's are call symbols and r_i's are return symbols. We let $l_i = c_i r_i$ for $i = 1, 2, 3$. First consider the following VPL language: $L_1 = (c_1)^n\ (r_2)^n\ (l_3)^*$. We can easily construct a VPT that transforms L_1 into the language $L_2 = (c_1)^n\ (l_2)^n\ (r_3)^*$. Applying the identity transducer on L_2 produces the non CFL language $L_3 = (c_1)^n\ (l_2)^n\ (r_3)^n$. This identity transducer has a domain which is a VPL and thus it extracts from L_2 the well-nested words which form the non CFL set L_3. Non closure under inverse is a consequence of the fact that, for any VPT T, for any word w, $T(w)$ is a finite set while a word w can be the image of an infinite number of input words. Finally, note that, as in the case of FSTs, VPTs are not closed under *intersection* (easy coding of PCP).

The following problem is known as the *translation membership problem* [13].

Proposition 2 (Translation Membership). *Let T be a* VPT *and $(u, v) \in \Sigma^* \times \Sigma^*$, the problem of deciding whether $(u, v) \in [\![T]\!]$ is in* PTIME.

Proof. We can first restrict T to a transducer $T|_u$ such that *Dom*$(T|_u) = \{u\}$ and $T|_u(u) = T(u)$. By Proposition 1, membership in *CoDom*$(T|_u)$ can be tested in PTIME. □

Theorem 1 (Type Checking). *Given a* VPT T *and two* VPAs A_1, A_2, *it is undecidable if $T(L(A_1)) \subseteq L(A_2)$.*

Proof. Given an instance $(u_1, v_1), (u_2, v_2), \ldots, (u_n, v_n)$ of PCP defined on the finite alphabet Σ, we associate with this instance a CFL and a VPL language defined on the alphabet $\Sigma_c = \Sigma$ and $\Sigma_r = \{1 \ldots n\}$. For all j, we let $l_j = |u_j|$. The CFL language is $L_1 = \{v_{i_1} \ldots v_{i_k} \# (i_k)^{l_k} \ldots (i_1)^{l_1} \mid i_1, \ldots, i_k \in \Sigma_r\}$. The VPL language is $L_2 = \{u_{i_1} \ldots u_{i_k} \# (i_k)^{l_k} \ldots (i_1)^{l_1} \mid i_1, \ldots, i_k \in \Sigma_r\}$. Clearly the PCP instance is negative if and only if L_1 is included in $\overline{L_2}$. By Proposition 1, there exists a VPT T whose image is L_1 and by definition, there exists a VPA that accepts $\overline{L_2}$, because VPAs are closed under complementation. □

4 On k-Valuedness of VPTs

Let $k \in \mathbb{N}$. A VPT T is *k-valued* if for all $u \in \Sigma^*$, $|\{v \mid (u, v) \in [\![T]\!]\}| \leq k$. T is *functional* if it is 1-valued. Two VPTs T_1, T_2 are *equivalent* if $[\![T_1]\!] = [\![T_2]\!]$. In this section, we prove that deciding if a VPT is k-valued is decidable in NPTIME (for a fixed k), and in PTIME for $k = 1$. The proof is done via a reduction to the *multiple morphism equivalence problem* on a context-free language, which was proved to be decidable in [11]. This reduction extends the one of [23], which was used to prove

the decidability of equivalence of deterministic (and therefore functional) VPTs ($k = 1$). By using a recent result of [24] on the complexity of constructing an existential Presburger formula for the Parikh image of a pushdown automaton, we give an upper bound for the multiple morphism equivalence problem. When there is only one pair of morphisms, this problem is known to be in PTIME [17].

Let Σ_1, Σ_2 be two finite alphabets. A *morphism* is a mapping $\Phi : \Sigma_1^* \to \Sigma_2^*$ such that $\Phi(\epsilon) = \epsilon$ and for all $u, v \in \Sigma_1^*$, $\Phi(uv) = \Phi(u)\Phi(v)$. A morphism can be finitely represented by its restriction on Σ_1, i.e. by the set of pairs $(a, \Phi(a))$ for all $a \in \Sigma_1$. Therefore its size is $|\Sigma| + \Sigma_{a \in \Sigma} |\Phi(a)|$.

Definition 2 (Multiple Morphism Equivalence on Context-Free Languages). *Given ℓ pairs of morphisms $(\Phi_1, \Psi_1), \ldots, (\Phi_\ell, \Psi_\ell)$ from Σ_1^* to Σ_2^* and a context free language L on Σ_1, $(\Phi_1, \Psi_1), \ldots, (\Phi_\ell, \Psi_\ell)$ are equivalent on L if for all $u \in L$, there exists i such that $\Phi_i(u) = \Psi_i(u)$.*

The next result was proved to be decidable in [11] on any class of languages whose Parikh images are effectively semi-linear. In the case of context-free languages, we show that it can be decided in NPTIME. The main part of our proof is to show that, for a fixed k, the emptiness of one-reversal pushdown machine with k counters is in NPTIME.

Theorem 2. *Let ℓ be fixed. Given ℓ pairs of morphisms and a pushdown automaton A, testing whether they are equivalent on $L(A)$ can be done in* NPTIME. *It is in* PTIME *if $\ell = 1$ and if the context-free language is given by a grammar in Chomsky normal form (Plandowski [17]).*

Proof. In order to prove this theorem, we briefly recall the procedure of [11] in the particular case of pushdown machines. It relies on the emptiness problem of reversal-bounded pushdown machines with a fixed number of counters. Let $k, m \in \mathbb{N}$, *an m-reversal pushdown machine with k counters (m-k-RBPM)* on an alphabet Σ is a pushdown automaton on Σ augmented with k counters. Each counter can be incremented or decremented by one and tested for zero, but the number of alternations between a nondecreasing and a non-increasing mode is bounded by m in any computation. The emptiness problem for such machines is decidable [12]. In order to decide the morphism equivalence problem of ℓ pairs of morphisms on a CFL L, the idea is to construct an 1-2ℓ-RBPM that accepts the language $L' = \{w \in L \mid \Phi_i(w) \neq \Psi_i(w) \text{ for all } i\}$. Clearly, $L' = \varnothing$ iff the morphisms are equivalent on L. Let A be the pushdown automaton that accepts L. We construct a pushdown automaton A' augmented with 2ℓ counters $c_{11}, c_{12}, \ldots, c_{\ell 1}, c_{\ell 2}$ that simulates A on the input word and counts the lengths of the outputs by the 2ℓ morphisms. For all $i \in \{1, \ldots, \ell\}$, A' guesses some position p_i where $\Phi_i(w)$ and $\Psi_i(w)$ differ: it increments in parallel (with ϵ-transitions) the counters c_{i1} and c_{i2} and non-deterministically decides to stop incrementing after p_i steps. Then when reading a letter $a \in \Sigma_1$, the two counters c_{i1} and c_{i2} are decremented by $|\Phi_i(a)|$ and $|\Psi_i(a)|$ respectively (by possibly several transitions as the counters can be incremented by at most one at a time). When one of the counter reaches zero A' stores the letter associated with the position (in the control state). At the end of the computation, for all $i \in \{1, \ldots, \ell\}$, one has to check that the two letters associated with the

position p_i in $\Phi_i(w)$ and $\Psi_i(w)$ are different. If n is the number of states of A and m is the maximal length of an image of a letter $a \in \Sigma_1$ by the 2ℓ morphisms, then A' has $O(n \cdot m \cdot |\Sigma_2|^{2\ell})$ states, because for all 2ℓ counters one has to store the letters at the positions represented by the counter values. This is polynomial as ℓ is fixed. Note that the resulting machine is 1-reversal bounded (counters are first set to zero and are incremented up to a position in the output word, and then are decremented to zero).

We now show that the emptiness of a one-reversal pushdown machine A with k counters on an alphabet Σ is in NPTIME. Wlog, we assume that each counter starts and ends with zero value, which is the case in the previous reduction. The NPTIME upper bound remains true without this assumption. For this, we recall the construction of [11] for testing emptiness of reversal-bounded machines with counters. The idea is to construct a semi-linear set for the Parikh image of A.[3] The emptiness of A then reduces to the emptiness of its Parikh image. Following [11], one extends the alphabet Σ with $3k$ letters $+_j, -_j, s_j \notin \Sigma$ intended to simulate the increasing, decreasing transitions of the j-th counter, and the transitions that do not change the j-th counter (skip). We denote by Σ_+ this alphabet. We construct a pushdown automaton B on Σ_+ that simulates A. When reading a letter $a \in \Sigma$, B tries to apply a transition of A, and passes into a mode in which it verifies that the next letters correspond to the increasing, decreasing or skip actions on the counters of the transition. Moreover, since A is 1-reversal bounded, B has to ensure that each counter does at most one reversal. The language of B is the set of words of the form $w = a_1 t_1 a_2 t_2 \dots a_n t_n$ where $a_i \in \Sigma$ and each t_i is a word of the form $b_1^i \dots b_k^i$ where $b_j^i \in \{+_j, -_j, s_j\}$, $j \in \{1, \dots, k\}$. Moreover, we require that (i) there exists a run of B on $a_1 \dots a_n$ ending up in a final state such that the counter actions of the transitions corresponds to $t_1 \dots t_n$ (ii) for all $j \in \{1, \dots, k\}$, $b_j^1 \dots b_j^n \in \{+_j, s_j\}^* \{-_j, s_j\}^*$ (one reversal). Let $\psi(w) = a_1 \dots a_n$ and $\psi_j(w) = b_j^1 \dots b_j^n$ for all $j \in \{1, \dots, k\}$. Condition (i) is enforced by a simple simulation of A, and condition (ii) is enforced by adding vectors of $\{+, -\}^k$ to the control states indicating whether the j-th counter is in increasing or decreasing mode. Note that $L(A) \subseteq \psi(L(B))$, but this inclusion may be strict, as we do not require that the counters end up in a zero value. More formally, we have $L(A) = \bigcap_{j=1}^k \{\psi(w) \mid w \in L(B) \text{ and } \psi_j(w) \in s_j^*(+_j.s_j^*)^\ell(-_j.s_j^*)^\ell, \ell \geq 0\}$.

As $L(B)$ is a context-free language, it is known by Parikh's theorem that the Parikh image of $L(B)$ is semi-linear. Therefore there exists an existential Presburger formula ϕ with $|\Sigma| + 3k$ free variables $(x_a)_{a \in \Sigma}$ and $(x_{+_j}, x_{-_j}, x_{s_j})_{j \in \{1,\dots,k\}}$ which defines the Parikh image of $L(B)$. Moreover, this formula can be constructed in time $O(|B|)$ [24]. Finally, the formula $\exists x_{+_1} \exists x_{-_1} \exists x_{s_1} \dots \exists x_{+_k} \exists x_{-_k} \exists x_{s_k} \phi \wedge \bigwedge_{j=1}^k x_{+_j} = x_{-_j}$ defines exactly the Parikh image of $L(A)$. Since B can be constructed in $O(|A| \cdot 2^k)$ (which is polynomial as k is fixed) and the satisfiability of existential Presburger formulas is in NP [24], one gets an NP algorithm to test the emptiness of A. We can conclude the proof by combining this result to the reduction of the multiple morphism equivalence problem described in the first part of the proof. □

[3] The Parikh image of a language $L \subseteq \Sigma^*$ over an ordered alphabet $\Sigma = \{a_1, \dots, a_n\}$ is the set $\{(\#_{a_1}(u), \dots, \#_{a_n}(u)) \mid u \in L\}$ where $\#_{a_i}(u)$ is the number of occurences of a_i in u.

Following ideas introduced in [3] for deciding functionality of FSTs, we define a notion of product for the class of VPTs. The k-power of T simulates k parallel executions on the same input. Note that this construction is possible for VPTs (but not for general PTs) because two runs along the same input have necessarily the same stack behavior. Let $T = (Q, I, F, \Gamma, \delta)$ be a VPT and O_T the set of outputs words occurring in the transitions of T, i.e. $O_T = \{u \mid \exists (p \xrightarrow{a/u} q) \in \delta\}$. As this set is finite, it can be regarded as an alphabet. The k-power of T is a VPT from words over Σ to words over $(O_T)^k$ defined as follows:

Definition 3 (k-Power). *The k-power of T, denoted T^k, is the* VPT *defined from Σ to $(O_T)^k$ by $T^k = (Q^k, I^k, F^k, \Gamma^k, \delta^k)$ where the transition relation $\delta^k = \delta_c^k \uplus \delta_r^k$ is defined for all $\alpha \in \{c, r\}$ and all $a \in \Sigma_\alpha$ by:*

$$(q_1, \ldots, q_k) \xrightarrow{a|(u_1,\ldots,u_k),(\gamma_1,\ldots,\gamma_k)} (q'_1, \ldots, q'_k) \in \delta_\alpha^k \text{ iff } q_i \xrightarrow{a|u_i,\gamma_i} q'_i \in \delta_\alpha \quad \forall 1 \leq i \leq k$$

For all $k \geq 0$, we define the morphisms $\Phi_1, \ldots, \Phi_k$ as follows:

$$\begin{aligned} \Phi_i : (O_T)^k &\to \Sigma^* \\ (u_1, \ldots, u_k) &\mapsto u_i \end{aligned}$$

Clearly, we obtain the following equivalence:

Proposition 3. *T is k-valued iff $(\Phi_i, \Phi_j)_{1 \leq i \neq j \leq k+1}$ are equivalent on CoDom(T^{k+1}).*

By Proposition 1 the language *CoDom*(T^k) is a context-free language. By Theorem 2, as *CoDom*(T^k) is represented by an automaton of polynomial size if k is fixed, we get:

Theorem 3 (k-valuedness). *Let $k \geq 0$ be fixed. The problem of deciding whether a* VPT *is k-valued is in* NPTIME. *It is in* PTIME *if $k = 1$.*

To get the PTIME bound when $k = 1$, one can construct a context-free grammar G_T in Chomsky normal form whose language is exactly the codomain of T^2.

Given two functional VPTs, they are equivalent iff their union is functional and they have the same domains. The domains being VPLs, testing their equivalence is EXPTIME-C. Therefore as a consequence of Theorem 3, we have:

Theorem 4 (Equivalence). *Testing equivalence of functional* VPTs *is* EXPTIME-C.

We end this section with a result on k-ambiguity of VPTs. A VPT is k-ambiguous if its underlying VPA is k-ambiguous, i.e. for each input word there are at most k accepting runs. The notion of k-ambiguity is stronger than k-valuedness. k-ambiguity can be tested in PTIME for tree automata. The standard construction to obtain a tree automaton (top-down or bottom-up) equivalent to a given VPA preserves the number of accepting runs, however it can yield an exponential blowup [1]. Therefore the PTIME bound cannot be obtained directly from this translation. For a given k, a PTIME construction to test k-ambiguity for VPAs can be obtained with a straightforward generalization of the construction for finite state automata. Basically, one constructs a VPA that simulates $k+1$ runs of the original VPA (this is possible because the stack are synchronized), and records which of these runs are different. It will accept any word that is accepted by the original VPA with $k+1$ different runs. If k is fixed, this construction can be done in PTIME, moreover testing emptiness of VPAs is in PTIME.

Proposition 4 (k-Ambiguity). *Let $k \in \mathbb{N}$ be fixed. Given a* VPA A*, resp. a* VPT T*, the problem of deciding whether A, resp. T, is k-ambiguous is in* PTIME.

5 Well-Nested VPTs

We have seen that VPTs are not closed under composition and their type checking problem is undecidable. In this section we introduce a natural subclass of VPTs that is closed for composition and for which the type checking is decidable.

The undecidability of the type checking is a consequence of the fact that the stack of the transducers and the stack of the output VPA are not synchronized, because no (visibly) restriction is imposed on the output words. Similarly, non-closure under composition is a consequence of the fact that the stack of both VPTs are not synchronized. To overcome those problems we introduce a restriction between the stack symbols and the output words.

Definition 4. *A* VPT $T = (Q, I, F, \Gamma, \delta)$ *is* well-nested *(*wnVPT*) if for all $(q_1, c, u, \gamma, q_1') \in \delta_c$ and $(q_2, r, v, \gamma, q_2') \in \delta_r$, we have $uv \in \Sigma^*_{\mathsf{wn}}$.*

This restriction ensures that all output words are well-nested.

Lemma 1. *For all* wnVPTs T *and all words $w \in \Sigma^*_{\mathsf{wn}}$, $T(w) \subseteq \Sigma^*_{\mathsf{wn}}$.*

We now show that this class of transducers is closed under composition and has a decidable type checking problem.

Proposition 5 (Closure properties). *The class of* wnVPTs *is effectively closed under union and composition.*

Proof (Sketch).
We first need an additional notion in order to present the construction of the composition of two such transducers. A word is *return-matched* (resp. *call-matched*) if there is no unmatched returns (resp. calls). Let $\mathsf{m}(w)$ be equal to the number of unmatched returns (resp. unmatched calls) if w is call-matched (resp. return-matched).

It is easy to show that a VPT $T = (Q, I, F, \Gamma, \delta)$ is well-nested iff (i) for all $(q, \alpha, w, \gamma, q') \in \delta_c$ (resp. δ_r), the word w is return-matched (resp. call-matched), and (ii) there exists a function $\mathsf{val} : \Gamma \to \mathbb{N}$ (called a *valuation*) such that for all $(q, \alpha, w, \gamma, q') \in \delta$, we have $\mathsf{val}(\gamma) = \mathsf{m}(w)$. This valuation is unique and can be computed in linear time.

Let $T_i = (Q_i, I_i, F_i, \Gamma_i, \delta_i)$, $i \in \{1, 2\}$, be two wnVPTs, and val_i their associated valuation. We define their composition T as the tuple $(Q_1 \times Q_2, I_1 \times I_2, F_1 \times F_2, \Gamma, \delta, \mathsf{val})$. Intuitively, it is a synchronized product in which the synchronization is not letter to letter, but is based on mappings val_i. More precisely, the stack alphabet Γ of T is defined as the finite set $\{(\gamma_1, \sigma_2) \in \Gamma_1 \times \Gamma_2^* \mid \mathsf{val}_1(\gamma_1) = |\sigma_2|\}$. The valuation val is defined by $\mathsf{val}(\gamma_1, \sigma_2) = \mathsf{val}_2(\sigma_2)$ where the extension of val_2 to Γ_2^* is defined as follows: if $\sigma_2 = \gamma_{2,1}\gamma_{2,2}\ldots\gamma_{2,n}$ then $\mathsf{val}_2(\sigma_2) = \sum_{i=1}^{n} \mathsf{val}_2(\gamma_{2,i})$. Call transitions are defined, for $c \in \Sigma_c$, by $(q_1, q_2) \xrightarrow{c/w,(\gamma_1,\sigma_2)} (q_1', q_2') \in \delta_c$ if and only if there exists

Table 1. Decision problems and closure properties

	Functionality / k-valuedness	Equivalence of functional	Type checking (against VPL)	$\cup$	$\circ$
FST	NL/P	PSPACE-C	EXP-C	Yes	Yes
dVPT	-	P [23]	Undec	No	No
VPT	**P/NP**	**Exp-c**	**Undec**	**Yes**	**No**
wnVPT	**P/NP**	**Exp-c**	**Exp-c**	**Yes**	**Yes**
ϵ-VPT [18]	Undec	Undec	EXP-C [18]	Yes	No
dPT	-	Dec[22]	Undec	No	No
PT	Undec	Undec	Undec	Yes	No

$v \in \Sigma^*$ such that $q_1 \xrightarrow{c/v,\gamma_1} q_1' \in \delta_c^1$, and $(q_2, \bot) \xrightarrow{v/w} (q_2', \sigma_2)$ is a run in the transducer T_2. Note that as T_1 is well-nested, we have $\mathsf{val}_1(\gamma_1) = \mathsf{m}(v)$, and then, as T_2 is a VPT, $\mathsf{val}_1(\gamma_1) = |\sigma_2|$. Return transitions are defined similarly and one can verify that T is a wnVPT and that the construction is correct. □

Theorem 5 (Type Checking). *Given a* wnVPT T*, two* VPAs A_1*,* A_2*, the problem of deciding if* $T(L(A_1)) \subseteq L(A_2)$ *is* EXPTIME-C*. It is in* PTIME *if* A_2 *is deterministic.*

Proof. For the EXPTIME-HARD part, first note that we can construct a wnVPT T_{id} whose domain is the set of well-nested words on the structured alphabet Σ and whose relation is the identity relation. Given any VPA A_1, A_2, we have that $T_{id}(L(A_1)) \subseteq L(A_2)$ if and only if $L(A_1) \subseteq L(A_2)$. This later problem is EXPTIME-C [2].

To prove it is in EXPTIME, we consider the wnVPT T_2 whose domain is $L(A_2)$ and whose relation is the identity relation. As wnVPTs are closed under composition, we can construct a wnVPT T' such that $T' = T \circ T_2$. Then we can note that $Dom(T') = T^{-1}(L(A_2))$. As $T(L(A_1)) \subseteq L(A_2)$ if and only if $L(A_1) \subseteq T^{-1}(L(A_2))$ and as all those transducers and automata can be constructed in polynomial time, we conclude that we can decide our problem in EXPTIME by checking the former inclusion using the algorithm for language inclusion between VPA. □

6 Conclusion

Table 1 summarizes the known results on several classes of word transducers. The results of this paper are in bold face. PTs denotes the class of pushdown transducers, and deterministic classes are denoted with a preceding d. Undecidability of ambiguity and functionality for PTs is well-known and can for example be proved by reduction of the emptiness problem for the intersection of two CFLs. Undecidability of the equivalence problem for two functional PTs is a direct consequence of the undecidability of the equivalence problem for CFLs. Undecidability of these problems for ϵ-VPTs can be proved in the exact same way since we can embed any CFL into the domain of such a transducer [18]. The undecidability of type checking for dPTs and PTs against VPLs can be proved as in Theorem 1. Finally, note that for all classes where equivalence of

functional transducers is decidable, the complexity depends on the complexity of testing equivalence of their domains.

As future works, we would like to investigate several problems. The first problem is the *sequentiality* problem for VPTs [3]. In particular, this problem asks whether a given VPT is equivalent to an *input-deterministic* VPT. Input-determinism is relevant to XML streaming transformations, as very large documents have to be processed on-the-fly without storing the whole document in memory. A second problem is to decide if two k-valued VPTs are equivalent.

Acknowledgments. We are grateful to the referees for their valuable comments and relevant questions, and we warmly thank Oscar H. Ibarra for pointing out some references.

References

1. Alur, R.: Marrying words and trees. In: PODS. LNCS, vol. 5140, pp. 233–242. Springer, Heidelberg (2007)
2. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: STOC, pp. 202–211 (2004)
3. Béal, M.-P., Carton, O., Prieur, C., Sakarovitch, J.: Squaring transducers: an efficient procedure for deciding functionality and sequentiality. TCS 292(1), 45–63 (2003)
4. Blattner, M., Head, T.: Single-valued a-transducers. JCSS 15(3), 310–327 (1977)
5. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications (2007), `http://www.grappa.univ-lille3.fr/tata`
6. Engelfriet, J., Maneth, S.: Macro tree translations of linear size increase are MSO definable. SICOMP 32, 950–1006 (2003)
7. Engelfriet, J., Maneth, S.: The equivalence problem for deterministic MSO tree transducers is decidable. IPL 100(5), 206–212 (2006)
8. Engelfriet, J., Vogler, H.: Macro tree transducers. JCSS 31(1), 71–146 (1985)
9. Filiot, E., Raskin, J.-F., Reynier, P.-A., Servais, F., Talbot, J.-M.: On functionality of visibly pushdown transducers. CoRR, abs/1002.1443 (2010)
10. Gurari, E.M., Ibarra, O.H.: A note on finite-valued and finitely ambiguous transducers. MST 16 (1983)
11. Harju, T., Ibarra, O.H., Karhumaki, J., Salomaa, A.: Some decision problems concerning semilinearity and commutation. JCSS 65 (2002)
12. Ibarra, O.H.: Reversal-bounded multicounter machines and their decision problems. JACM 25(1), 116–133 (1978)
13. Inaba, K., Maneth, S.: The complexity of translation membership for macro tree transducers. CoRR, abs/0910.2315 (2009)
14. Koch, C., Scherzinger, S.: Attribute grammars for scalable query processing on XML streams. VLDB 16(3), 317–342 (2007)
15. Maneth, S., Neven, F.: Structured document transformations based on XSL. In: Connor, R.C.H., Mendelzon, A.O. (eds.) DBPL 1999. LNCS, vol. 1949, pp. 80–98. Springer, Heidelberg (2000)
16. Perst, T., Seidl, H.: Macro forest transducers. IPL 89(3), 141–149 (2004)
17. Plandowski, W.: Testing equivalence of morphisms on context-free languages. In: van Leeuwen, J. (ed.) ESA 1994. LNCS, vol. 855, pp. 460–470. Springer, Heidelberg (1994)
18. Raskin, J.-F., Servais, F.: Visibly pushdown transducers. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 386–397. Springer, Heidelberg (2008)

19. Schützenberger, M.P.: Sur les relations rationnelles. In: Automata Theory and Formal Languages. LNCS, vol. 33, pp. 209–213. Springer, Heidelberg (1975)
20. Seidl, H.: Single-valuedness of tree transducers is decidable in polynomial time. TCS 106(1), 135–181 (1992)
21. Seidl, H.: Equivalence of finite-valued tree transducers is decidable. MST 27(4), 285–346 (1994)
22. Sénizergues, G.: T(A) = T(B)? In: Wiedermann, J., Van Emde Boas, P., Nielsen, M. (eds.) ICALP 1999. LNCS, vol. 1644, pp. 665–675. Springer, Heidelberg (1999)
23. Staworko, S., Laurence, G., Lemay, A., Niehren, J.: Equivalence of deterministic nested word to word transducers. In: Gębala, M. (ed.) FCT 2009. LNCS, vol. 5699, pp. 310–322. Springer, Heidelberg (2009)
24. Verma, K.N., Seidl, H., Schwentick, T.: On the complexity of equational horn clauses. In: Nieuwenhuis, R. (ed.) CADE 2005. LNCS (LNAI), vol. 3632, pp. 337–352. Springer, Heidelberg (2005)