Visibly Pushdown Transducers*

Jean-François Raskin[1] and Frédéric Servais[2]

[1] Computer Science Department,
[2] Department of Computer & Decision Engineering (CoDE),
Université Libre de Bruxelles (U.L.B.)

Abstract. Visibly pushdown automata have been recently introduced by Alur and Madhusudan as a subclass of pushdown automata. This class enjoys nice properties such as closure under all Boolean operations and the decidability of language inclusion. Along the same line, we introduce here visibly pushdown transducers as a subclass of pushdown transducers. We study properties of those transducers and identify subclasses with useful properties like decidability of type checking as well as preservation of regularity of visibly pushdown languages.

1 Introduction

Visibly pushdown languages (VPL) have recently been proposed by Alur and Madhusudan in [3] as a subclass of *context-free languages* (CFL) with interesting closure and decidability properties. While CFL are not closed under intersection nor under complementation, VPL are closed under all Boolean operations, and the language inclusion problem is decidable. VPL are expressive enough to model a large number of relevant problems, for example those related to the analysis of programs with procedure calls or to the formalization of structured documents (like XML documents). As a consequence, VPL offer an appropriate theoretical framework to unify many known decidability results in those fields as well as opportunities to solve new problems. In [3], visibly pushdown automata (VPA) are defined as a subclass of the pushdown automata where stack operations are restricted by the input word. VPA operate on words over a *tagged alphabet* $\hat{\Sigma} = \Sigma^c \uplus \Sigma^r \uplus \Sigma^i$ where Σ^c are *call* symbols, Σ^r are *return* symbols, and Σ^i are *internal* symbols. Each time a call symbol is read, the automaton has to push a symbol on the stack; each time a return symbol is read, the automaton has to pop a symbol from the stack; and each time an internal symbol is read, the automaton must leave the stack unchanged. VPA exactly recognize VPL.

Transducers are machines that model relations between words, i.e. they recognize sets of pairs of words. Transducers transform languages into languages: let L be a set of words, T a transducer then $T(L) = \{w \mid \exists v \in L : T \text{ accepts the pair } (v, w)\}$. There are many important applications of transducers. For example, while languages are useful to formalize sets of XML documents (i.e. XML document types), transducers are useful to formalize XML document transformations (e.g., XSLT) [9]. Motivated by

* This research was supported by the Belgian FNRS grant 2.4530.02 of the FRFC project "Centre Fédéré en Vérification" and by the project "MoVES", an Interuniversity Attraction Poles Programme of the Belgian Federal Government.

L. Aceto et al. (Eds.): ICALP 2008, Part II, LNCS 5126, pp. 386–397, 2008.

this application, the *type checking problem* asks if all the words of L_1 are translated into words of L_2 under a transducer T, i.e. whether $T(L_1) \subseteq L_2$. Transducers have also been intensively used in the so-called regular model-checking [1,5]. In that setting, the states of a system are modeled by words, state sets by languages and state transitions by transducers. So far, the concept of regular model-checking has only been applied to regular languages (with the notable exception of [6]). Unfortunately some parametric systems cannot be modeled in this setting and more powerful classes of transducers with good decidability and closure properties are needed.

In this paper, we study several subclasses of *pushdown transducers*. In the spirit of [3], we define subclasses of pushdown transducers by imposing restrictions on the use of the stack and the transition relation. We study three main classes of pushdown transducers. First, *visibly pushdown transducers* are pushdown transducers that operate over pairs of words defined on a tagged alphabet $\hat{\Sigma}$. Those transducers respect two restrictions: (i) along the reading of a pair of words, either the head is moved only in one of the two words (allowing deletion and insertion), or it is moved over a pair of symbols of the same type (two calls, two returns, or two internals), (ii) when reading internals the transducer leaves the stack unchanged, when reading calls it pushes a symbol on the stack, when reading returns it pops a symbol from the stack. We show here that unfortunately the type checking is undecidable for this class even if L_1 and L_2 are VPL. They are not closed under composition and they do not preserve VPL, i.e. the transduction of a VPL is not necessarily a VPL. Second, *synchronized visibly pushdown transducers* are obtained from visibly pushdown transducers by imposing the following additional restrictions: (i) when a call is deleted then the *matching* return is deleted, (ii) when a call is inserted then a matching return is inserted, and (iii) when a call is copied then the matching return is also copied. We show that this class of pushdown transducers has a decidable type checking problem for VPL. This result is not trivial as we also show that the transduction of a VPL with a synchronized visibly pushdown transducer is not necessarily a VPL. This class of transducers is well suited to formally validate XML document transformations. Indeed, opening and closing tags are modeled by calls and returns respectively, and a transformation that inserts (respectively deletes) a new opening tag will usually also insert (respectively delete) the corresponding closing tag. The synchronized restriction to our transducer is therefore very natural in that context. Finally, we define the class of *fully synchronized visibly pushdown transducers* as a subclass of synchronized visibly pushdown transducers that, in addition to having a decidable type checking problem, preserve VPL, and are closed under composition and inverse. This class of transducers has all the properties required to extend the techniques used in regular model-checking from regular languages to VPL.

2 Preliminaries

Basics. An *alphabet* Σ is a finite set of symbols[1], we note Σ_ϵ for $\Sigma \cup \{\epsilon\}$ (the alphabet Σ together with the empty word symbol ϵ). The *tagged alphabet* over Σ is an alphabet, noted $\hat{\Sigma}$, which is equal to $\Sigma^c \uplus \Sigma^r \uplus \Sigma^i$, where $\Sigma^c = \{\overline{a} \mid a \in \Sigma\}$, $\Sigma^r = \{\underline{a} \mid$

[1] For technical reasons, we assume that all alphabets Σ in this paper are such that $|\Sigma| \geq 2$.

$a \in \Sigma\}$ and $\Sigma^i = \{a \mid a \in \Sigma\}$.[2] A *word* over Σ is a finite sequence of symbols in Σ. A *language* over Σ is a set of words over Σ. In the rest of the paper, given any alphabet Σ, we note $\mathsf{RL}(\Sigma)$, respectively $\mathsf{CFL}(\Sigma)$, the set of *regular*, respectively *context-free*, languages over Σ. Let π be the function from $\hat{\Sigma}$ into Σ defined as follows: $\pi(a) = \pi(\overline{a}) = \pi(\underline{a}) = a$. We extend π to words: for $w = a_1 a_2 \dots a_n$, $\pi(w) = \pi(a_1)\pi(a_2)\dots\pi(a_n)$, and to languages: $\pi(L) = \{\pi(w) \mid w \in L\}$. Let $\Sigma_1 \subseteq \Sigma_2$, for $w \in \Sigma_2^*$, $\downarrow^{\Sigma_1}(w) \in \Sigma_1^*$ returns the word w where the occurences of symbols in $\Sigma_2 \setminus \Sigma_1$ have been erased. Finally, a *stack alphabet* Γ is a finite set of symbols that contains a special symbol, noted $\bot$, called the *bottom-of-stack symbol*.

Visibly pushdown languages. A *visibly pushdown automaton* (VPA) [3] on finite words over the tagged alphabet $\hat{\Sigma} = \Sigma^c \uplus \Sigma^r \uplus \Sigma^i$ is a tuple $A = (Q, Q_0, Q_f, \Gamma, \delta)$ where Q is a finite set of states, $Q_0 \subseteq Q$, respectively $Q_f \subseteq Q$, the set of initial states, respectively final states, Γ the stack alphabet, and $\delta = \delta_c \uplus \delta_r \uplus \delta_i$ where $\delta_c \subseteq Q \times \Sigma^c \times Q \times (\Gamma \setminus \{\bot\})$ are the *call transitions*, $\delta_r \subseteq Q \times \Gamma \times \Sigma^r \times Q$ are the *return transitions*, and $\delta_i \subseteq Q \times \Sigma^i \times Q$ are the *internal 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 (note that if $\bot$ is the top of the stack then it is read but not popped). Finally, on an internal transition $(q, a, q') \in \delta_i$, there is no stack operation. Accordingly, a *run* of a visibly pushdown automaton A over the 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 $q_0 \in Q_0$, $\sigma_0 = \bot$, and for each $k < l$, we have either: (i) $(q_k, a_{k+1}, q_{k+1}, \gamma) \in \delta_c$ and $\sigma_{k+1} = \gamma\sigma_k$; (ii) $(q_k, \gamma, a_{k+1}, q_{k+1}) \in \delta_r$ and if $\gamma \neq \bot$ then $\sigma_k = \gamma\sigma_{k+1}$ else $\sigma_k = \sigma_{k+1} = \bot$; or (iii) $(q_k, a_{k+1}, q_{k+1}) \in \delta_i$ and $\sigma_k = \sigma_{k+1}$. A run is *accepting* if $q_l \in Q_f$. 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 a tagged alphabet $\hat{\Sigma}$ is a *visibly pushdown language* if there is a VPA A over $\hat{\Sigma}$ such that $L(A) = L$. We note $\mathsf{VPL}(\hat{\Sigma})$ for the set of VPL over the tagged alphabet $\hat{\Sigma}$.

Example 1. $V_{2n} = \{\overline{a}^n \underline{b}^n \mid n \geq 0\}$ is a $\mathsf{VPL}(\hat{\Sigma})$, while $C_{2n} = \{a^n b^n \mid n \geq 0\}$ is not.

Proposition 1 ([3]). *Here are the main properties of* VPL *and* VPA.

1. *The class of* VPL *is closed under all Boolean operations.*[3] *In particular, given* $A, A_1, A_2 \in$ VPA *we can compute in polynomial time a* VPA B *such that* $L(B) = L(A_1) \cap L(A_2)$*, and in exponential time a* VPA C *such that* $L(C) = \overline{L(A)}$.
2. *Given* $A_1, A_2 \in$ VPA*, the problem of deciding whether* $L(A_1) \subseteq L(A_2)$ *is* EXPTIME-COMPLETE*, when* A_2 *is deterministic the problem is* PTIME-COMPLETE.
3. *Given* $A \in$ VPA*, we can decide, in polynomial time, whether* $L(A) = \emptyset$.
4. *Let* $C \in \mathsf{CFL}(\Sigma)$*, then there exists* $V \in \mathsf{VPL}(\hat{\Sigma})$ *such that* $\pi(V) = C$.

[2] We sometimes write a^c for $\overline{a}$, a^r for $\underline{a}$ and a^i for a. We may also write a when the type of a is clear from the context.

[3] This is in sharp contrast with CFL that are not closed under intersection nor complement.

The following result states the undecidability of checking inclusion between a CFL and VPL. To the best of our knowledge, the direction CFL into VPL is not established in the literature. We give a proof of the theorem in [10].

Theorem 1. *Let $C \in \mathsf{CFL}$ and $V \in \mathsf{VPL}$ then checking whether $C \subseteq V$, and checking whether $V \subseteq C$ are undecidable problems.*

Transduction relations and the type-checking problem. A relation $R \subseteq \Sigma^* \times \Sigma^*$ is a *transduction relation*, or simply a *transduction*, over Σ, i.e. a set of pairs of words over Σ. When $R(v, w)$ holds, we sometimes call v the *input* and w the *output* of the transduction. The *transduction of a word* v over Σ by a transduction relation $R \subseteq \Sigma^* \times \Sigma^*$ is the language $\{w \mid R(v, w)\}$, noted $R(v)$. The *transduction of a language* L over Σ by a transduction relation $R \subseteq \Sigma^* \times \Sigma^*$ is the language $\{w \mid \exists v \in L : R(v, w)\}$, noted $R(L)$. The *type checking problem* asks, given an effective representation of two languages L_1 and L_2, and an effective representation of a transduction relation R, to establish if $R(L_1) \subseteq L_2$.

3 Visibly Pushdown Transducers

VPA are pushdown automata such that the input restrict the stack operations. Similarly we define *visibly pushdown transducers* as pushdown transducers such that input and output restrict the stack operations. Such a transducer will push, respectively pop, onto the stack when it reads and/or write a call, respectively a return.

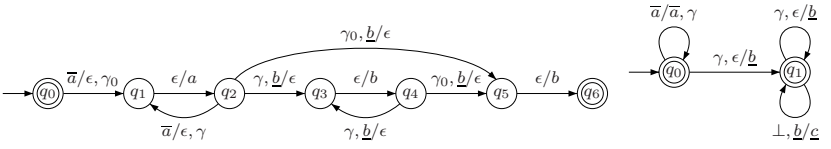
Definition 1 (VPT). A *visibly pushdown transducer* on finite words over $\hat{\Sigma}$ is a tuple $T = (Q, Q_0, Q_f, \Gamma, \delta)$ where Q is a finite set of states, $Q_0 \subseteq Q$, respectively $Q_f \subseteq Q$, the set of initial states, respectively final states, Γ the stack alphabet, and $\delta = \delta_c \uplus \delta_r \uplus \delta_i$, with $\delta_c \subseteq Q \times \Sigma_\epsilon^c \times \Sigma_\epsilon^c \times Q \times (\Gamma \setminus \{\bot\})$, $\delta_r \subseteq Q \times \Gamma \times \Sigma_\epsilon^r \times \Sigma_\epsilon^r \times Q$, $\delta_i \subseteq Q \times \Sigma_\epsilon^i \times \Sigma_\epsilon^i \times Q$. Moreover if $(q, \alpha, \beta, q', \gamma) \in \delta_c$, $(q, \gamma, \alpha, \beta, q') \in \delta_r$ or $(q, \alpha, \beta, q') \in \delta_i$ then $\alpha \neq \epsilon$ or $\beta \neq \epsilon$. The class of visibly pushdown transducer is noted VPT.[4]

Definition 2 (Run of a VPT). A *run* of a VPT T over (v, w), where $v = a_1 \dots a_l$ and $w = b_1 \dots b_m$ are words on $\hat{\Sigma}$, is a sequence $\{(q_k, i_k, j_k, \sigma_k)\}_{0 \leq k \leq n}$, where q_k is the state at step k, i_k, respectively j_k, are the index of the last letter of v, respectively w, the transducer has reached, and $\sigma_k \in \Gamma^*$ is the stack, such that $q_0 \in Q_0, i_0 = 1, j_0 = 1$, $\sigma_0 = \bot$, and for all $k < n$, let $\alpha = \epsilon$ or $\alpha = a_{i_k}$ and $\beta = \epsilon$ or $\beta = b_{j_k}$, $i_{k+1} = i_k + |\alpha|$, $j_{k+1} = j_k + |\beta|$, and we have either: (i) $(q_k, \alpha, \beta, q_{k+1}, \gamma) \in \delta_c$ and $\sigma_{k+1} = \gamma\sigma_k$, (ii) $(q_k, \gamma, \alpha, \beta, q_{k+1}) \in \delta_r$ and if $\gamma \neq \bot$ then $\sigma_k = \gamma\sigma_{k+1}$, else $\sigma_k = \sigma_{k+1} = \bot$, (iii) $(q_k, \alpha, \beta, q_{k+1}) \in \delta_i$ and $\sigma_k = \sigma_{k+1}$. A run is *accepting* if $q_n \in Q_f$, $i_n = |v| + 1$, and $j_n = |w| + 1$.

[4] Note that we define transducers that operate over pairs of words defined on the same alphabet. This is not restrictive: a transducer from words on an alphabet Σ_1 to words on an alphabet Σ_2 can be seen as a transducer from $\Sigma_1 \cup \Sigma_2$ to $\Sigma_1 \cup \Sigma_2$. In the following, we will abuse notations and sometimes we will define transducers where the input and output alphabets differ.

We note $[\![T]\!]$ the transduction induced by T, it is the set of pairs $(v, w) \in \hat{\Sigma}^* \times \hat{\Sigma}^*$ such that there exists an accepting run of T on (v, w)[5]. A transduction relation $R \subseteq \hat{\Sigma}^* \times \hat{\Sigma}^*$ is a *visibly pushdown transduction* if there exists $T \in \mathsf{VPT}$ such that $R = [\![T]\!]$.

Example 2. The transducer T_{del} of Fig. 1(a) deletes the calls $\overline{a}$, respectively the returns $\underline{b}$, and replaces them with the internals a, respectively b, it further verifies that the number of deleted calls is equal to the number of deleted returns. Clearly, T_{del} is a VPT that transduces V_{2n} into C_{2n} (defined in Example 1), which is also obtained when T_{del} is applied on $\hat{\Sigma}^*$. The transducer T_{ins} of Fig. 1(b) copies the calls $\overline{a}$ it encounters and then inserts the same number of returns $\underline{b}$, finally it renames the remaining returns $\underline{b}$ into $\underline{c}$. Then T_{ins} is a VPT that transduces V_{2n} into the language $S_{3n} = \{\overline{a}^n \underline{b}^n \underline{c}^n \mid n \geq 0\}$.



(a) T_{del} (b) T_{ins}

Fig. 1. Examples of VPT

Definition 3 (Inverse transducer). Given a VPT $T = (Q, Q_0, Q_f, \Gamma, \delta)$, we define its *inverse* $T^{-1} = (Q, Q_0, Q_f, \Gamma, \delta')$ with (i) $(q_1, \beta, \alpha, q_2, \gamma) \in \delta'_c \Leftrightarrow (q_1, \alpha, \beta, q_2, \gamma) \in \delta_c$, (ii) $(q_1, \gamma, \beta, \alpha, q_2) \in \delta'_r \Leftrightarrow (q_1, \gamma, \alpha, \beta, q_2) \in \delta_r$, and (iii) $(q_1, \beta, \alpha, q_2) \in \delta'_i \Leftrightarrow (q_1, \alpha, \beta, q_2) \in \delta_i$.

Proposition 2 (Inverse transduction). *Let* $T \in \mathsf{VPT}$*, then* $[\![T^{-1}]\!] = [\![T]\!]^{-1}$.

Proof. Any run of T on (v, w) can easily be transformed in a run of T^{-1} on (w, v) by interchanging α with β and i_k with j_k. □

Lemma 1. *For all* $C \in \mathsf{CFL}(\hat{\Sigma})$*, there exist* $T \in \mathsf{VPT}(\hat{\Sigma})$ *and* $V \in \mathsf{VPL}(\hat{\Sigma})$ *such that* $T(V) = C$.

Proof. In this proof we use the alphabet $\hat{\hat{\Sigma}}$ which is the set $\{(a^x)^y \mid a \in \Sigma \wedge x, y \in \{c, r, i\}\}$ and we make the hypothesis that Σ contains the letters c, r, and i. This is without lost of generality as we make the hypothesis that our alphabets always contain at least two letters.

First, by Proposition 1, there exists $V' \in \mathsf{VPL}(\hat{\hat{\Sigma}})$ such that $\pi(V') = C$. With the notations above, π is defined as follows: $\pi((a^x)^y) = a^x$. Second, let us consider the function $\tau_1 : \hat{\hat{\Sigma}} \to \hat{\Sigma} \times \hat{\Sigma}$ defined as $\tau_1((a^x)^y) = x^i a^y$. This function codes any character of $\hat{\hat{\Sigma}}$ into a sequence of two characters of $\hat{\Sigma}$. We extend the function τ_1 to words as follows: let $w = a_1 \dots a_n \in \hat{\hat{\Sigma}}^*$, $\tau_1(w) = \tau_1(a_1) \dots \tau_1(a_n)$. Given A' a VPA on $\hat{\hat{\Sigma}}$ for V', it is easy to construct A a VPA on $\hat{\Sigma}$ such that $L(A) = \tau_1(L(A'))$, since

[5] In the sequel, we sometimes say that the transducer read the input v and write the output w.

τ_1 maps a call on an internal followed by a call, maps a return on an internal followed by a return, and maps an internal on two internals.

Third, let us consider the function $\tau_2 : \{c^i, r^i, i^i\} \times \hat{\Sigma} \to \hat{\Sigma}$ defined by: $\tau_2(x^i a^y) = a^x$. Clearly, for any word $w \in \hat{\hat{\Sigma}}^*$, $\pi(w) = \tau_2(\tau_1(w))$. We are left to show that τ_2 can be defined as a VPT T. Here is the construction. First, T, when in state q, reads an internal x^i which determines the type of the ouput: a call if $x = c$, a return if $x = r$, and an internal if $x = i$. Accordingly, it goes into q^c, q^r or q^i respectively using the transitions $(q, c^i, \epsilon, q^c) \in \delta_i$, $(q, r^i, \epsilon, q^r) \in \delta_i$ or $(q, i^i, \epsilon, q^i) \in \delta_i$. Note that those transitions do not move the head on the output (so erasing the internal x^i). Then, T reads the next letter a^y and rewrites it into the output type defined by its current state, that is if the state is q^c then it writes (imposes to read) a^c on the output, etc. There are nine cases to consider, (i) read a call write a call, (ii) read a call write a return, (iii) read a call write an internal, (iv) read a return write a call, and so on. For translation of one type of character to another, we need to use two transitions that use first epsilon on output and then epsilon on input. Here are two representative cases over the nine cases: (i) for a^c into a^c: $(q^c, a^c, a^c, q, \gamma) \in \delta_c$, (ii) for a^c into a^r: $(q^r, a^c, \epsilon, q^r_a, \gamma) \in \delta_c$, $(q^r_a, \perp, \epsilon, a^r, q) \in \delta_r$ and $(q^r_a, \gamma, \epsilon, a^r, q) \in \delta_r$. Clearly, T is a VPT. To complete the proof, a simple induction shows that $T(V) = \tau_2(V) = \pi(V') = C$. Note that the VPT is not using the stack: only one character is pushed on the stack and return transitions can always use this character or the bottom character. As a matter of fact, the transduction τ_2 is definable by a finite state transducer on $\hat{\Sigma}$. □

In the next proposition, we establish that the transduction and inverse transduction of a VPL by a VPT is not necessarily a VPL nor even a CFL, and that the transduction and inverse transduction of a RL by a VPT is not necessarily a VPL but it is always a CFL. We note $\mathsf{VPT}(\mathsf{RL}) = \{T(R) \mid T \in \mathsf{VPT}, R \in \mathsf{RL}\}$ and $\mathsf{VPT}(\mathsf{VPL}) = \{T(V) \mid T \in \mathsf{VPT}, V \in \mathsf{VPL}\}$.

Proposition 3. $\mathsf{VPL} \subsetneq \mathsf{VPT}(\mathsf{RL}) \subseteq \mathsf{CFL} \subsetneq \mathsf{VPT}(\mathsf{VPL})$.

Proof. First, we know that $\mathsf{VPT}(\mathsf{RL}) \subseteq \mathsf{CFL}$ since it is true for the class of pushdown transducers (which contains VPT). Second, to show that $\mathsf{VPL} \subseteq \mathsf{VPT}(\mathsf{RL})$, for any $V \in \mathsf{VPL}$ we construct a VPT that first ignores the input (taking only transitions that are labelled by ϵ for the input), checks that the output is in V by simulating the VPA that accepts V, and when it reaches the end of the output, it reads the input without constraining the output using ϵ transitions on the output. When executing this transducer on $\hat{\Sigma}^*$, we get V. Third, to show that $\mathsf{VPL} \neq \mathsf{VPT}(\mathsf{RL})$, we consider T_{del} of Example 2: when executed on $\hat{\Sigma}^*$ it returns C_{2n}, a CFL which is not a VPL. Fourth, to prove $\mathsf{CFL} \subsetneq \mathsf{VPT}(\mathsf{VPL})$, first by Lemma 1 we get $\mathsf{CFL} \subseteq \mathsf{VPT}(\mathsf{VPL})$, second we consider the transducer T_{ins} of Example 2, it transduces $V_{2n} \in \mathsf{VPL}$ into $S_{3n} \notin \mathsf{CFL}$. □

In the next result states that the class of VPT is not closed under composition.

Corollary 1 (Composition). *There exists $T, T' \in \mathsf{VPT}$ such that $[\![T]\!] \circ [\![T']\!]$ is not a visibly pushdown transduction.*

Proof. From Proposition 3, there are $V \in \mathsf{VPL}$ and $T \in \mathsf{VPT}$ such that $T(V) \notin \mathsf{CFL}$. Also there exist $R \in \mathsf{RL}$ and $T' \in \mathsf{VPT}$ such that $T'(R) = V$ since $\mathsf{VPL} \subseteq \mathsf{VPT}(\mathsf{RL})$. So $[\![T]\!] \circ [\![T']\!](R) \notin \mathsf{CFL}$ but then it cannot be a VPT as $\mathsf{VPT}(\mathsf{RL}) \subseteq \mathsf{CFL}$. □

The next theorem shows that the type checking problem of VPT against VPL is undecidable.

Theorem 2. *For $A_1, A_2 \in \mathsf{VPA}$ and $T \in \mathsf{VPT}$, it is undecidable whether $T(L(A_1)) \subseteq L(A_2)$.*

Proof. Let $C \in \mathsf{CFL}(\hat{\Sigma})$, by Lemma 1 there exist $V \in \mathsf{VPL}(\hat{\Sigma})$ and $T \in \mathsf{VPT}$ such that $T(V) = C$. Therefore we have that $T(V) \subseteq V'$ iff $C \subseteq V'$ which is undecidable as established in Theorem 1. □

4 Synchronized Visibly Pushdown Transducers

We define here a restricted class of transducers that allow typechecking. The idea is to synchronize the insertion, respectively the deletion, of a call with the insertion, respectively the deletion, of the matching return.

Definition 4 (SVPT)**.** A *synchronized visibly pushdown transducer* is a VPT such that $\Gamma = \Gamma_{copy} \uplus \Gamma_{del} \uplus \Gamma_{ins} \uplus \{\bot\}$ and such that if $(q, \alpha, \beta, q', \gamma) \in \delta_c$ or $(q, \gamma, \alpha, \beta, q') \in \delta_r$ then either: (i) $\alpha = \epsilon, \beta \neq \epsilon$ and $\gamma \in \Gamma_{ins} \cup \{\bot\}$, (ii) $\alpha \neq \epsilon, \beta = \epsilon$ and $\gamma \in \Gamma_{del} \cup \{\bot\}$, or (iii) $\alpha \neq \epsilon, \beta \neq \epsilon$ and $\gamma \in \Gamma_{copy} \cup \{\bot\}$.[6] The set of synchronized visibly pushdown transducer is noted SVPT.

Example 3. T_{del} of Example 2 is a SVPT with $\Gamma_{del} = \Gamma, \Gamma_{ins} = \emptyset$ and $\Gamma_{copy} = \emptyset$. On the other hand, T_{ins} is not a SVPT since γ is used for inserting, see transition $(q_0, \gamma, \epsilon, \underline{b}, q_1)$, and for copying, see transition $(q_0, \overline{a}, \overline{a}, q_0, \gamma)$.

The next proposition states the class SVPT is closed by inverse.

Proposition 4. *Let $T \in \mathsf{SVPT}$ then $T^{-1} \in \mathsf{SVPT}$.*

Proof. T^{-1} is a VPT (Proposition 2). Moreover, with $\Gamma = \Gamma'_{copy} \uplus \Gamma'_{del} \uplus \Gamma'_{ins} \uplus \{\bot\}$ where $\Gamma'_{copy} = \Gamma_{copy}$, $\Gamma'_{del} = \Gamma_{ins}$ and $\Gamma'_{ins} = \Gamma_{del}$, this transducer is synchronized. □

In the next proposition, we establish that the transduction or inverse transduction of a VPL by a SVPT is not always a VPL. We note $\mathsf{SVPT}(\mathsf{RL}) = \{S(R) \mid S \in \mathsf{SVPT}, R \in \mathsf{RL}\}$ and $\mathsf{SVPT}(\mathsf{VPL}) = \{S(V) \mid S \in \mathsf{SVPT}, V \in \mathsf{VPL}\}$.

Proposition 5. $\mathsf{VPL} \subsetneq \mathsf{SVPT}(\mathsf{RL}) = \mathsf{SVPT}(\mathsf{VPL}) \subsetneq \mathsf{CFL}$.

Proof. First, for $\mathsf{VPL} \subsetneq \mathsf{SVPT}(\mathsf{RL})$, consider T_{del} of Example 2, it transduces a RL into a CFL that is not a VPL (see Proposition 3), T_{del} is a SVPT. Second, for $\mathsf{SVPT}(\mathsf{RL}) = \mathsf{SVPT}(\mathsf{VPL})$, consider any $S \in \mathsf{SVPT}$ and $A \in \mathsf{VPA}$. Let $V = L(A)$. We construct $S' \in \mathsf{SVPT}$ such that $S'(\hat{\Sigma}^*) = S(V)$. More concretely, we impose that, for all $w \in \hat{\Sigma}^*$ we have that $S'(w) = S(w)$ when $w \in V$ and $S'(w) = \emptyset$ otherwise. To achieve that, S' simulates S and A: it translates w as S does and, in parallel, it simulates A on w. A run of S' is accepting if the corresponding runs in S and A are accepting. It is crucial to note that the parallel simulation of the stacks of S and A is only possible because S is a SVPT: each time that it copies, respectively deletes or inserts, a call, it will copy, respectively delete or insert the *matching* return. As a consequence the content of the stack of S and A can be represented as pairs of symbols as follows:

[6] As SVPT are VPT, call transitions are not allowed to push $\bot$.

- *call-return copy*: when A and S are moving and pushing a symbol γ and $\gamma' \in \Gamma_{copy}$ on their respective stack, S' pushes the symbol (γ, γ'). As S is a SVPT and $\gamma' \in \Gamma_{copy}$, this ensures that when we reach the *matching* return, S copies the return, and the pair (γ, γ') will be popped from the stack. This simulates the behavior of the stacks of A and S. From there, S' continues the parallel simulation of A and S.
- *call-return delete*: when A and S are moving and pushing a symbol γ and $\gamma' \in \Gamma_{del}$ on their respective stack, S' pushes the symbol (γ, γ'). As S is a SVPT and $\gamma' \in \Gamma_{del}$, this ensures that when we reach the *matching* return, S will delete the return, and the pair (γ, γ') will be popped from the stack. This simulates the behavior of the stacks of A and S. From there, S' continues the parallel simulation of A and S.
- *call-return insert*: on a call-return insert, only S is moving. It pushes a symbol $\gamma' \in \Gamma_{ins}$ on its stack. To simulate this, S' pushes the pair $(\gamma_\epsilon, \gamma')$ on its stack, γ_ϵ being a new stack symbol that does not belong to the stack symbols of A. As γ' belongs to Γ_{ins}, we know that the *matching* return will be inserted (so no input will be read and A will not move), at that time S' will pop the pair $(\gamma_\epsilon, \gamma')$, not moving on the input. This simulates the behavior of the stacks of A and S. From there, S' continues the parallel simulation of A and S.
- Other cases are treated similarly.

Third, $\mathsf{SVPT}(\mathsf{VPL}) \subseteq \mathsf{CFL}$ is a consequence of the facts that $\mathsf{SVPT}(\mathsf{RL}) \subseteq \mathsf{VPT}(\mathsf{RL}) \subseteq \mathsf{CFL}$ and $\mathsf{SVPT}(\mathsf{RL}) = \mathsf{SVPT}(\mathsf{VPL})$. Finally, $\mathsf{SVPT}(\mathsf{VPL}) \neq \mathsf{CFL}$ is a consequence of the fact that typechecking SVPT against VPL is decidable (Theorem 3, see below) and the undecidability of checking the inclusion of a CFL into a VPL (Theorem 1). □

Non-deleting and non-inserting transducers. Two important subclasses of SVPT are the class of transducers that do not insert and the ones that do not delete.

Definition 5. A *non-inserting* SVPT $T = (Q, Q_0, Q_f, \Gamma, \delta)$ is a SVPT such that (i) $\delta_c \subseteq Q \times \Sigma^c \times \Sigma^c_\epsilon \times Q \times \Gamma$, (ii) $\delta_r \subseteq Q \times \Gamma \times \Sigma^r \times \Sigma^r_\epsilon \times Q$ and (iii) $\delta_i \subseteq Q \times \Sigma^i \times \Sigma^i_\epsilon \times Q$ (and thus $\Gamma_{ins} = \emptyset$). This class is noted $\mathsf{SVPT}_{\mathsf{ni}}$. A *non-deleting* SVPT $T = (Q, Q_0, Q_f, \Gamma, \delta)$ is a SVPT such that (i) $\delta_c \subseteq Q \times \Sigma^c_\epsilon \times \Sigma^c \times Q \times \Gamma$, (ii) $\delta_r \subseteq Q \times \Gamma \times \Sigma^r_\epsilon \times \Sigma^r \times Q$ and (iii) $\delta_i \subseteq Q \times \Sigma^i_\epsilon \times \Sigma^i \times Q$ (and thus $\Gamma_{del} = \emptyset$). This class is noted $\mathsf{SVPT}_{\mathsf{nd}}$.

Proposition 6. *Let* $T \in \mathsf{SVPT}$,

1. $T \in \mathsf{SVPT}_{\mathsf{nd}}$ *iff* $T^{-1} \in \mathsf{SVPT}_{\mathsf{ni}}$,
2. *if* $T \in \mathsf{SVPT}_{\mathsf{nd}}$ *and* $V \in \mathsf{VPL}$ *then* $T(V) \in \mathsf{VPL}$,
3. *if* $T \in \mathsf{SVPT}_{\mathsf{ni}}$ *and* $V \in \mathsf{VPL}$ *then* $T^{-1}(V) \in \mathsf{VPL}$.

Proof. The first assertion is a direct consequence of Proposition 2 stating that T^{-1} is also a VPT and the fact that the inverse transducer of a non-inserting, respectively non-deleting, transducer is obviously a non-deleting, respectively non-inserting, transducer.

Our proof for the second assertion is constructive. Given the $\mathsf{SVPT}_{\mathsf{nd}}$ T, and the VPA A^{in} for V, we construct a VPA A^{out} that accepts $T(V)$. We sketch here the main arguments of the proof, the full detailed proof is given in [10]. On a word w, A^{out} guesses a word v and checks that the pair $(v, w) \in [\![T]\!]$ and $v \in V$. For that, the VPA

A^{out} simulates in parallel the execution of A^{in} on v and the execution of T on the pair (v, w), its run is accepting if the simulated runs in A^{in} and T are accepting. The main delicate part of the proof is to show that A^{out} can simulate the two stacks while respecting the restrictions imposed to a VPA. The parallel simulation of the stack of A^{in} and T is possible because T is a $\mathsf{SVPT}_{\mathsf{nd}}$: each time that it copies, respectively inserts, a call, it will copy, respectively insert, the *matching* return. As a consequence, the content of the stacks of A^{in} and T can be represented as pairs of symbols as follows:

- *call-return copy*: when A^{in} and T are moving and pushing a symbol γ^{in} and $\gamma^T \in \Gamma^T_{copy}$ on their respective stack, A^{out} pushes the symbol (γ^{in}, γ^T) and reads in w the same symbol as written by T. As T is a $\mathsf{SVPT}_{\mathsf{nd}}$ and $\gamma^T \in \Gamma^T_{copy}$, this ensures that when we reach the *matching* return in v (and A^{in} pop γ^{in}), T copies the return in w and pop γ^T. At that time, A^{out} pops the pair (γ^{in}, γ^T) from its stack and reads in w the same symbol as written by T. This simulates the behavior of the stacks of A^{in} and T. From there, A^{out} continues the parallel simulation of A^{in} and T.
- *call-return insert*: on a call-return insert, only T is moving. It pushes a symbol $\gamma^T \in \Gamma^T_{ins}$ on its stack and write β. To simulate this, A^{out} reads β and pushes the pair $(\gamma_\epsilon, \gamma^T)$ on its stack, γ_ϵ being a new stack symbol that does not belong to the stack symbols of A^{in}. As γ^T belongs to Γ^T_{ins}, we know that the *matching* return in w, say β', will be inserted (no letter will be read by T and so A^{in} will not move), at that time A^{out} will pop the pair $(\gamma_\epsilon, \gamma^T)$ when reading β'. This simulates the behavior of the stacks of A^{in} and T. From there, A^{out} continues the parallel simulation of A^{in} and T.

Note that A^{out} could not simulate a transducer that deletes matching calls and returns as it would have to modify its stack while not reading any letter, which is not allowed in a VPA. The last assertion is a direct consequence of the first and the second. □

We can now prove that type checking is decidable for SVPT.

Theorem 3. *Let* $A_1, A_2 \in$ VPA *and* $T \in$ SVPT*, the problem of checking if* $T(L(A_1)) \subseteq L(A_2)$ *is* ExpTime-Complete*, the problem is* PTime-Complete *when* A_2 *is deterministic.*

Proof. We know that checking inclusion between two VPL is ExpTime-Hard (Proposition 1), if we choose T to be the identity transducer (which is a SVPT), we obtain the hardness part. For the easiness part, we first show that T is equivalent to the composition of two transducers: $[\![T]\!] = [\![T_{ni}]\!] \circ [\![T_{nd}]\!]$, which are respectively *non-inserting* and *non-deleting*.

T_{nd} will behave as T with the essential difference that whenever T deletes a *call*, a *return*, respectively an *internal*, T_{nd} replaces it with ϵ^c, ϵ^r, respectively ϵ^i which are new call, return, respectively internal symbols that do not belong to the alphabet $\hat{\Sigma}$. More formally, let $T = (Q, Q_0, Q_f, \Gamma, \delta)$ over $\hat{\Sigma}$, T_{nd} is a transducer from $\hat{\Sigma}$ into $\hat{\Sigma}_{nd} = \Sigma^c_{nd} \uplus \Sigma^r_{nd} \uplus \Sigma^i_{nd}$ where $\Sigma^c_{nd} = \Sigma^c \uplus \{\epsilon^c\}$, $\Sigma^r_{nd} = \Sigma^r \uplus \{\epsilon^r\}$, $\Sigma^i_{nd} = \Sigma^i \uplus \{\epsilon^i\}$. We define $T_{nd} = (Q, Q_0, Q_f, \Gamma^{nd}, \delta_{nd})$, such that $\Gamma^{nd} = \Gamma^{nd}_{copy} \uplus \Gamma^{nd}_{ins} \uplus \{\bot\}$, where $\Gamma^{nd}_{copy} = \Gamma_{copy} \uplus \Gamma_{del}$, and $\Gamma^{nd}_{ins} = \Gamma_{ins}$, $\delta_{nd} = \delta'_c \cup \delta'_r \cup \delta'_i$, $\delta'_c =$

$\{(q,\alpha,b,q',\gamma) \in \delta_c \mid \alpha \in \hat{\Sigma}^c_\epsilon, b \in \hat{\Sigma}^c\} \cup \{(q,a,\epsilon^c,q',\gamma) \mid (q,a,\epsilon,q',\gamma) \in \delta_c, a \in \hat{\Sigma}^c\}$, $\delta'_r = \{(q,\gamma,\alpha,b,q') \in \delta_r \mid \alpha \in \hat{\Sigma}^r_\epsilon, b \in \hat{\Sigma}^r\} \cup \{(q,\gamma,a,\epsilon^r,q') \mid (q,\gamma,a,\epsilon,q') \in \delta_r, a \in \hat{\Sigma}^r\}$, $\delta'_i = \{(q,\alpha,b,q') \in \delta_i \mid \alpha \in \hat{\Sigma}^i_\epsilon, b \in \hat{\Sigma}^i\} \cup \{(q,a,\epsilon^i,q') \mid (q,a,\epsilon,q') \in \delta_i, a \in \hat{\Sigma}^i\}$. Note that T_{nd} does not erase and so the partition of the stack symbols is different from the one for T, and clearly we have $T_{nd} \in \mathsf{SVPT}_{\mathsf{nd}}$. Furthermore, by construction, we have that for all $(w_1,w_2) \in [\![T]\!]$, there exists w_3 such that $(w_1,w_3) \in [\![T_{nd}]\!]$ and $w_2 = \downarrow^{\hat{\Sigma}}(w_3)$, and conversely, for all $(w_1,w_3) \in [\![T_{nd}]\!]$, $(w_1,\downarrow^{\hat{\Sigma}}(w_3)) \in [\![T]\!]$. As T is a SVPT, for all $w \in T_{nd}(\hat{\Sigma}^*)$, w is such that the matching return of every ϵ^c is an ϵ^r, and conversely. This property is easily proved by induction on the length of runs of T_{nd}. We say that those words are *synchronized* on the pair (ϵ^c,ϵ^r).

We define the transducer T_{ni}. For all w_1 that are synchronized on the pair (ϵ^c,ϵ^r), the transducer accepts the pairs $(w_1,\downarrow^{\hat{\Sigma}}(w_1)) \in \hat{\Sigma}_{nd} \times \hat{\Sigma}$. Clearly, this transduction relation is realized by the following transducer $T_{ni} = (\{q\},\{q\},\{q\},\{\gamma_{copy},\gamma_{del},\bot\},\delta_{ni})$, on $(\hat{\Sigma} \cup \{\epsilon^c,\epsilon^r,\epsilon^i\}) \times \hat{\Sigma}$, such that $\delta_{ni} = \delta''_c \cup \delta''_r \cup \delta''_i$, $\delta''_c = \{(q,a,a,q,\gamma_{copy}) \mid a \in \Sigma^c\} \cup \{(q,\epsilon^c,\epsilon,q,\gamma_{del})\}$, $\delta''_r = \{(q,\gamma_{copy},a,a,q) \mid a \in \Sigma^r\} \cup \{(q,\gamma_{del},\epsilon^r,\epsilon,q)\} \cup \delta''_r = \{(q,\bot,a,a,q) \mid a \in \Sigma^r\} \cup \{(q,\bot,\epsilon^r,\epsilon,q)\}$, and $\delta''_i = \{(q,a,a,q) \mid a \in \Sigma^i\} \cup \{(q,\epsilon^i,\epsilon,q)\}$ which is in the class $\mathsf{SVPT}_{\mathsf{ni}}$.[7] Clearly, $[\![T]\!] = [\![T_{ni}]\!] \circ [\![T_{nd}]\!]$.

To finish the proof, we consider the following equivalence: $T(L(A_1)) \subseteq L(A_2) \Leftrightarrow T_{nd}(L(A_1)) \cap T^{-1}_{ni}(\overline{L(A_2)}) = \emptyset$. The proof of Proposition 6 tells us that we can construct, in deterministic polynomial time in the size of T_{nd} and of A_1, a VPA B_1 that accepts the language $T_{nd}(L(A_1))$. Also, Proposition 1 tells us that we can compute, in deterministic exponential time in the size of A_2, an automaton B_2 that accepts $\overline{L(A_2)}$ (in polynomial time if A_2 is deterministic), and we can construct, in deterministic polynomial time in the size of B_2 and T^{-1}_{ni}, a VPA B_3 that accepts $T^{-1}_{ni}(\overline{L(A_2)})$. Finally, checking emptiness of intersection between two VPA can be done in deterministic polynomial time (Proposition 1). This concludes our proof of ExpTime-Easiness (PTime-Easiness if A_2 is deterministic). □

The following proposition states that any CFL can be obtained by applying two SVPT on a VPL.

Proposition 7. *For all $C \in \mathsf{CFL}(\hat{\Sigma})$, there exist $V \in \mathsf{VPL}(\hat{\Sigma})$, $T_1,T_2 \in \mathsf{SVPT}$ such that $T_2(T_1(V)) = C$.*

Proof. First, the proof of Lemma 1 tells us that there exists $V \in \mathsf{VPL}(\hat{\Sigma})$ such that $\tau_2(V) = C$, where $\tau_2 : \{c^i,r^i,i^i\} \times \hat{\Sigma} \to \hat{\Sigma}$ is defined as: $\tau_2(x^i a^y) = a^x$. We now show that τ_2 can be expressed as the composition of two SVPT. We decompose τ_2 into the following two functions. First, $\tau_3 : \{c^i,r^i,i^i\} \times \hat{\Sigma} \to \{c^i,r^i,i^i\} \times \hat{\Sigma}^i$ defined as: $\tau_3(x^i a^y) = x^i a^i$. Second, $\tau_4 : \{c^i,r^i,i^i\} \times \hat{\Sigma}^i \to \hat{\Sigma}$ defined as: $\tau_4(x^i a^i) = a^x$. Clearly, those two functions can be expressed as SVPT and $\tau_2 = \tau_4 \circ \tau_3$. □

As a consequence of Proposition 7 and Theorem 1, we cannot type check the composition of two SVPT against VPL.

[7] Note that without the hypothesis of *synchronized* on the pair (ϵ^c,ϵ^r), there is no $\mathsf{SVPT}_{\mathsf{ni}}$ that realizes $\downarrow^{\hat{\Sigma}}$, that is the reason why this construction can not be generalized when T is a VPT.

Theorem 4. *Let* $A_1, A_2 \in \mathsf{VPA}$ *and* $T_1, T_2 \in \mathsf{SVPT}$*, it is undecidable whether* $T_1(T_2(L(A_1))) \subseteq L(A_2)$.

Fully synchronized visibly pushdown transducers. We finish this section by introducing a class of VPT that maintain regularity, are closed under inverse and under composition and for which type checking is decidable.

Definition 6 (FSVPT)**.** A *fully synchronized visibly pushdown transducer* is a synchronized visibly pushdown transducer which is both non-inserting and non-deleting. This class is noted FSVPT.

Theorem 5. *Let* $T \in \mathsf{FSVPT}$*, then:*

1. VPL *preservation: for any* $V \in \mathsf{VPL}$, $T(V) \in \mathsf{VPL}$*;*
2. *Inverse:* $T^{-1} \in \mathsf{FSVPT}$*;*
3. *Composition: for any* $T_1 \in \mathsf{FSVPT}$ *there exists* $T_2 \in \mathsf{FSVPT}$ *such that* $[\![T_2]\!] = [\![T_1]\!] \circ [\![T]\!]$*;*
4. *Decidable type-checking: given two* VPA A_1, A_2*, deciding* $T(L(A_1)) \subseteq L(A_2)$ *is* EXPTIME-COMPLETE.

Note that $\mathsf{FSVPT} = \mathsf{SVPT}_{\mathsf{nd}} \cap \mathsf{SVPT}_{\mathsf{ni}}$, this class is exactly the class of VPT that do not delete nor insert. Moreover, we could define *finite state transducers* to transduce words on $\hat{\Sigma}$, such that calls are mapped on calls, returns on returns, and internals on internals. This class would be a strict subclass of FSVPT as such automata would translate languages from $\mathsf{RL}(\hat{\Sigma})$ into $\mathsf{RL}(\hat{\Sigma})$ while FSVPT can transduce languages from $\mathsf{RL}(\hat{\Sigma})$ into languages that are not in $\mathsf{RL}(\hat{\Sigma})$. Finally, if T is a FSVPT then it can be seen as a VPA that works on pairs of symbols (of the same type), and so, equivalence between FSVPT is EXPTIME-COMPLETE.

5 Conclusion

In this paper, we have identified two interesting sub-classes of pushdown transducers. SVPT (synchronized visibly pushdown transducer) is a powerful subclass with a decidable (EXPTIME-COMPLETE) type checking problem against VPL. This positive result is surprising as we have shown that SVPT do not preserve VPL. Also, the class of SVPT is not closed under composition. This has triggered the definition of FSVPT (fully synchronized visibly pushdown transducers), this class of transducers enjoys nice properties like preservation of VPL, closure to composition and decidable (EXPTIME-COMPLETE) type checking problem against VPL.[8]

Alur and Madushudan have shown in [4] that VPL are equivalent to regular languages of *nested words*. Our results can be rephrased in this setting as well. In [2], Alur has studied the relation between VPL and tree languages. In future work, we will study in details the relation between the transducers defined on regular tree languages, as

[8] A. Thomo et al. defined in [11] a class of visibly pushdown transducers equivalent to ours. However, their article does not study this class of transducers per se and they incorrectly states that VPT maintains VPL in contradiction with our Proposition 3.

defined in [7,8], and our transducers. It seems pretty clear that their expressive power are incomparable but a fine comparison requires a large effort of formalization and is beyond the subject of this paper. As already said, those works on transducers were often motivated by the application in XML, we will study the practical advantages and drawbacks of our transducers for that application in future work.

In [6], Fisman and Pnueli use context-free languages for extending regular model-checking. CFL are used to model the set of initial states of the system, the transition relation as well as the specification (the set of *good* states) are given by finite state transducers and automata, respectively. We conjecture that FSVPT can be used to rephrase and extend those results by offering an unified framework for regular model-checking in the context of VPL, as FSVPT are preserving VPL. We will investigate this important application in future work.

Acknowledgement. We want to thank Laurent Van Begin for suggesting the main idea of the proof for Theorem 1 and Ahmed Bouajjani for pointing us the paper of Fisman and Pnueli.

References

1. Abdulla, P., Legay, A., d'Orso, J., Rezine, A.: Tree regular model checking: A simulation-based approach. J. Log. Algebr. Program. 69(1-2), 93–121 (2006)
2. Alur, R.: Marrying words and trees. In: PODS, pp. 233–242 (2007)
3. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: STOC, pp. 202–211 (2004)
4. Alur, R., Madhusudan, P.: Adding nesting structure to words. In: H. Ibarra, O., Dang, Z. (eds.) DLT 2006. LNCS, vol. 4036, pp. 1–13. Springer, Heidelberg (2006)
5. Bouajjani, A., Jonsson, B., Nilsson, M., Touili, T.: Regular model checking. In: CAV, pp. 403–418 (2000)
6. Fisman, D., Pnueli, A.: Beyond regular model checking. In: Hariharan, R., Mukund, M., Vinay, V. (eds.) FSTTCS 2001. LNCS, vol. 2245, pp. 156–170. Springer, Heidelberg (2001)
7. Maneth, S., Berlea, A., Perst, T., Seidl, H.: XML type checking with macro tree transducers. In: PODS, pp. 283–294 (2005)
8. Martens, W., Neven, F.: Typechecking top-down uniform unranked tree transducers. In: Calvanese, D., Lenzerini, M., Motwani, R. (eds.) ICDT 2003. LNCS, vol. 2572, pp. 64–78. Springer, Heidelberg (2002)
9. Milo, T., Suciu, D., Vianu, V.: Typechecking for XML transformers. In: PODS, pp. 11–22 (2000)
10. Raskin, J.-F., Servais, F.: Visibly pushdown transducers. Technical Report 2008.100, Federated Center for Verification - Université Libre de Bruxelles (2008), http://www.ulb.ac.be/di/ssd/cfv/publications.html
11. Thomo, A., Venkatesh, S., Ying Ye, Y.: Visibly pushdown transducers for approximate validation of streaming XML. In: FoIKS, pp. 219–238 (2008)