Visibly Pushdown Transducers

Jean-François Raskin, Frédéric Servais

This work was partially supported by a FRFC grant: 2.4530.02

http://www.ulb.ac.be/di/ssd/cfv
Visibly Pushdown Transducers *

Jean-François Raskin1 and Frédéric Servais2

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

Abstract. Visibly pushdown automata have recently been 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 \( \hat{\Sigma} = \Sigma^c \sqcup \Sigma^r \sqcup \Sigma^i \) where \( \Sigma^c \) are call symbols, \( \Sigma^r \) are return symbols, and \( \Sigma^i \) 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

* 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.
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 \( \Sigma \) is a finite set of symbols\(^1\), we note \( \Sigma_e \) for \( \Sigma \cup \{ \epsilon \} \) (the alphabet \( \Sigma \) together with the empty word symbol \( \epsilon \)). The tagged alphabet over \( \Sigma \) is an alphabet, noted \( \Sigma_T \), which is equal to \( \Sigma^{\epsilon} \cup \Sigma^{\eta} \cup \Sigma^{\iota} \), where \( \Sigma^{\epsilon} = \{ \pi \mid a \in \Sigma \}, \Sigma^{\eta} = \{ \eta \mid a \in \Sigma \} \) and \( \Sigma^{\iota} = \{ \iota \mid a \in \Sigma \} \).

\(^1\) For technical reasons, we assume that all alphabets \( \Sigma \) in this paper are such that \( |\Sigma| \geq 2 \).
A word over $\Sigma$ is a finite sequence of symbols in $\Sigma$. A language over $\Sigma$ is a set of words over $\Sigma$. In the rest of the paper, given any alphabet $\Sigma$, we note $RL(\Sigma)$, respectively $CFL(\Sigma)$, the set of regular, respectively context-free, languages over $\Sigma$. Let $\pi$ be the function from $\hat{\Sigma}$ into $\Sigma$ defined as follows: $\pi(a) = \pi(\pi(\pi(\ldots \pi(a) \ldots ))) = a$. We extend $\pi$ to words: for $w = a_1 a_2 \ldots a_n$, $\pi(w) = \pi(a_1) \pi(a_2) \ldots \pi(a_n)$, and to languages: $\pi(L) = \{ \pi(w) \mid w \in L \}$. Let $\Sigma_1 \subseteq \Sigma_2$, for $w \in \Sigma_2^n$, $\hat{\pi}(w) \in \Sigma_1^*$ returns the word $w$ where the occurrences of symbols in $\Sigma_2 \setminus \Sigma_1$ have been erased. Finally, a stack alphabet $\Gamma$ 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 $\hat{\Sigma} = \Sigma^* \sqcup \Sigma^* \sqcup \Sigma^*$ 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, $\Gamma$ the stack alphabet, and $\delta = \delta_c \sqcup \delta_r \sqcup \delta_i$ 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 $(q, a, q', \gamma) \in \delta_c$, $\gamma$ is pushed onto the stack and the control goes from $q$ to $q'$. On a return transition $(q, \gamma, a, q') \in \delta_r$, $\gamma$ is popped from the stack (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 $w = a_1 \ldots a_l$ 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) $(q_k, \sigma_{k+1}, q_{k+1}, \gamma) \in \delta_c$ and $\sigma_{k+1} = \gamma \sigma_k$; (ii) $(q_k, \gamma, a_{k+1}, q_{k+1}) \in \delta_r$ and $\gamma \neq \bot$ then $\sigma_k = \gamma \sigma_{k+1}$ else $\sigma_k = \sigma_{k+1} = \bot$; or (iii) $(q_k, \sigma_{k+1}, q_{k+1}) \in \delta_i$ 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 $VPL(\hat{\Sigma})$ for the set of VPL over the tagged alphabet $\hat{\Sigma}$.

Example 1. $V_{2n} = \{ \pi^* b^n \mid n \geq 0 \}$ is a 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. 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 $L(C) = L(A_1) \cup L(A_2)$.
2. Given $A_1, A_2 \in VPA$, the problem of deciding whether $L(A_1) \subseteq L(A_2)$ is EXPSPACE-COMPLETE, when $A_2$ is deterministic the problem is PSPACE-COMPLETE.
3. Given $A \in VPA$, we can decide, in polynomial time, whether $L(A) = \emptyset$.
4. Let $C \in CFL(\Sigma)$, then there exists $V \in VPL(\hat{\Sigma})$ such that $\pi(V) = C$.

We sometimes write $a^\omega$ for $\pi$, $a^\omega$ for $a$ and $a^\omega$ for $a$. We may also write $a$ when the type of $a$ is clear from the context.

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 full proof in the Appendix.

**Theorem 1.** Let $C \subseteq \text{CFL}$ and $V \subseteq \text{VPL}$ 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 $\Sigma$, i.e. a set of pairs of words over $\Sigma$. 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 $\Sigma$ 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 $\Sigma$ 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 $T = (Q, Q_0, F, \Gamma, \delta)$ where $Q$ is a finite set of states, $Q_0 \subseteq Q$, respectively $F \subseteq Q$, the set of initial states, respectively final states, $\Gamma$ the stack alphabet, and $\delta = \delta_0 \cup \delta_1 \cup \delta_2$, with $\delta_0 \subseteq Q \times \Sigma \times Q \times (\Gamma \setminus \{\bot\})$, $\delta_1 \subseteq Q \times \Gamma \times \Sigma \times Q \times Q$, $\delta_2 \subseteq Q \times \Sigma \times \Sigma \times Q$. Moreover if $(q, \alpha, \beta, q' \gamma) \in \delta_0$, $(q, \gamma, \alpha, \beta, q') \in \delta_1$ or $(q, \alpha, \beta, q') \in \delta_2$ then $\alpha \neq \epsilon$ or $\beta \neq \epsilon$. The class of visibly pushdown transducer is noted VPT.

**Definition 2 (Run of a VPT).** A run of a VPT $T$ over $(v, w)$, where $v = a_1 \ldots a_l$ and $w = b_1 \ldots b_m$ 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 $\alpha = a_i$, and $\beta = \epsilon$ or $\beta = b_j$, $i_{k+1} = i_k + |\alpha|$, $j_{k+1} = j_k + |\beta|$, and we have either: (i) $(q_k, \alpha, \beta, q_{k+1}, \gamma) \in \delta_0$ and $\sigma_{k+1} = \gamma \sigma_k$, (ii) $(q_k, \gamma, \alpha, \beta, q_{k+1}) \in \delta_1$ and if $\gamma \neq \bot$ then $\sigma_k = \gamma \sigma_{k+1}$, else $\sigma_k = \sigma_{k+1} = \bot$, (iii) $(q_k, \alpha, \beta, q_{k+1}) \in \delta_2$ and $\sigma_k = \sigma_{k+1}$. A run is accepting if $q_n \in F, i_n = |v| + 1$, and $j_n = |w| + 1$.

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 $\Sigma_1$ to words on an alphabet $\Sigma_2$ 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 $(v, w) \in \Sigma^* \times \Sigma^*$ such that there exists an accepting run of $T$ on $(v, w)$\(^5\). A transduction relation $R \subseteq \Sigma^* \times \Sigma^*$ is a visibly pushdown transduction if there exists $T \in \text{VPT}$ such that $R = [T]$.

Example 2. The transducer $T_{\text{del}}$ of Fig. 1(a) deletes the calls $\pi$, respectively the returns $\epsilon$, 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, $T_{\text{del}}$ is a VPT that transduces $V_{2n}$ into $C_{2n}$ (defined in Example 1), which is also obtained when $T_{\text{del}}$ is applied on $\Sigma\hat{=}$. The transducer $T_{\text{ins}}$ of Fig. 1(b) copies the calls $\pi$ it encounters and then inserts the same number of returns $\epsilon$, finally it renames the remaining returns $\epsilon$ into $\gamma$. Then $T_{\text{ins}}$ is a VPT that transduces $V_{2n}$ into the language $S_{3n} = \{\pi^n \gamma^n \epsilon^n \mid n \geq 0\}$.

**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) $(q_1, \gamma, \beta, \alpha, q_2) \in \delta'_r \Leftrightarrow (q_1, \beta, \alpha, q_2, \gamma) \in \delta_r$, and (iii) $(q_1, \gamma, \beta, \alpha, q_2) \in \delta'_e \Leftrightarrow (q_1, \beta, \alpha, q_2) \in \delta_e$.

**Proposition 2 (Inverse transduction).** Let $T \in \text{VPT}$, 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 $\alpha$ with $\beta$ and $i_k$ with $j_k$. \hfill $\Box$

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

**Proof.** In this proof we use the alphabet $\hat{\Sigma}$ which is the set $\{(a^x)^y \mid a \in \Sigma \times x, y \in \{c, r, t\}\}$ and we make the hypothesis that $\Sigma$ contains the letters $c$, $r$, and $t$. 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 $V' \in \text{VPL}(\hat{\Sigma})$ such that $\pi(V') = C$. With the notations above, $\pi$ is defined as follows: $\pi((a^x)^y) = a^x$. Second, let us consider the function $\tau_1 : \hat{\Sigma} \rightarrow \hat{\Sigma} \times \hat{\Sigma}$ defined as $\tau_1((a^x)^y) = x^a y^a$. This function codes any character of $\hat{\Sigma}$ into a sequence of two characters of $\hat{\Sigma}$. We extend the function $\tau_1$ to words as follows: let $w = a_1 \ldots a_n \in \hat{\Sigma}^*$, $\tau_1(w) = \tau_1(a_1) \ldots \tau_1(a_n)$. Given $A'$ a VPA on $\hat{\Sigma}$ for $V'$, it is easy to construct $A$ a VPA on $\hat{\Sigma}$ such that $L(A) = \tau_1(L(A'))$, since

\[^5\text{In the sequel, we sometimes say that the transducer read the input } v \text{ and write the output } w.\]
Also there exist

\[ \tau_1 \text{ maps a call on an internal followed by a call, maps a return on an internal followed by a return, and maps an internal on two internals.} \]

Third, let us consider the function \( \tau_2 : \{ c', r', i' \} \times \hat{\Sigma} \rightarrow \hat{\Sigma} \) defined by: \( \tau_2(a^i a^j) = a^k \). Clearly, for any word \( w \in \hat{\Sigma}^* \), \( \pi(w) = \tau_2(\tau_1(w)) \). We are left to show that \( \tau_2 \) 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 output: a call if \( x = \epsilon \), a return if \( x = \tau \), and an internal if \( x = i \). Accordingly, it goes into \( q', q'' \) or \( q' \) respectively using the transitions \( (q, c', \epsilon, q') \in \delta_i \), \( (q, r', \epsilon, q') \in \delta_i \) or \( (q, i', \epsilon, q') \in \delta_i \). Note that those transitions do not move the head on the output (so erasing the internal \( x^i \)). Then, \( T \) reads the next letter \( a^k \) and rewrites it into the output type defined by its current state, that is if the state is \( q' \) then it writes (imposes to read) \( a^k \) 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 \( a^c \): \( (q, a^c, \epsilon, q', \gamma) \in \delta_i \), (ii) for \( a^c \) into \( a^c \): \( (q, a^c, \epsilon, q', \gamma) \in \delta_i \), \( (q, a^c, \epsilon, a^c, q) \in \delta_i \) and \( (q, a^c, \epsilon, a^c, q) \in \delta_i \). 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 \( \tau_2 \) is definable by a finite state transducer on \( \hat{\Sigma} \). \( \square \)

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 \( \text{VPT}(\text{RL}) = \{ T(R) \mid T \in \text{VPT}, R \in \text{RL} \} \) and \( \text{VPT}(\text{VPL}) = \{ T(V) \mid T \in \text{VPT}, V \in \text{VPL} \} \).

**Proposition 3.** \( \text{VPL} \subseteq \text{VPT}(\text{RL}) \subseteq \text{CFL} \subseteq \text{VPT}(\text{VPL}) \).

**Proof.** First, we know that \( \text{VPT}(\text{RL}) \subseteq \text{CFL} \) since it is true for the class of pushdown transducers (which contains VPT). Second, to show that \( \text{VPL} \subseteq \text{VPT}(\text{RL}) \), for any \( V \in \text{VPL} \) we construct a VPT that first ignores the input (taking only transitions that are labelled by \( \epsilon \) 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 \( \epsilon \) transitions on the output. When executing this transducer on \( \hat{\Sigma}^* \), we get \( V \). Third, to show that \( \text{VPL} \neq \text{VPT}(\text{RL}) \), we consider \( T_{\text{vis}} \) of Example 2: when executed on \( \hat{\Sigma}^* \) it returns \( C_{2n} \), a CFL which is not a VPL. Fourth, to prove \( \text{CFL} \subseteq \text{VPT}(\text{VPL}) \), first by Lemma 1 we get \( \text{CFL} \subseteq \text{VPT}(\text{VPL}) \), second we consider the transducer \( T_{\text{vis}} \) of Example 2, it transduces \( V_{2n} \in \text{VPL} \) into \( S_{3n} \notin \text{CFL} \). \( \square \)

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

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

**Proof.** From Proposition 3, there are \( V \in \text{VPL} \) and \( T \in \text{VPT} \) such that \( T(V) \notin \text{CFL} \). Also there exist \( R \in \text{RL} \) and \( T' \in \text{VPT} \) such that \( T'(R) = V \) since \( \text{VPL} \subseteq \text{VPT}(\text{RL}) \). So \( [T] \circ [T'](R) \notin \text{CFL} \) but then it cannot be a VPT as \( \text{VPT}(\text{RL}) \subseteq \text{CFL} \). \( \square \)
The next theorem shows that the type checking problem of VPT against VPL is undecidable.

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

**Proof.** Let $C \in CFL(\Sigma)$, by Lemma 1 there exist $V \in VPL(\Sigma)$ and $T \in VPT$ 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. \qed

### 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 $\Gamma = \Gamma_{\text{copy}} \uplus \Gamma_{\text{del}} \uplus \Gamma_{\text{ins}} \uplus \{ \bot \}$ and such that if $(q, \alpha, \beta, q', \gamma) \in \delta$, or $(q, \gamma, \alpha, \beta, q') \in \delta$, then either: (i) $\alpha = \epsilon, \beta \neq \epsilon$ and $\gamma \in \Gamma_{\text{ins}} \cup \{ \bot \}$, (ii) $\alpha \neq \epsilon, \beta = \epsilon$ and $\gamma \in \Gamma_{\text{del}} \cup \{ \bot \}$, or (iii) $\alpha \neq \epsilon, \beta \neq \epsilon$ and $\gamma \in \Gamma_{\text{copy}} \cup \{ \bot \}$. The set of synchronized visibly pushdown transducer is noted SVPT.

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

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

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

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

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 $SVPT(\text{RL}) = \{ S(R) \mid S \in SVPT, R \in \text{RL} \}$ and $SVPT(\text{VPL}) = \{ S(V) \mid S \in SVPT, V \in \text{VPL} \}$.

**Proposition 5.** $VPL \subseteq SVPT(\text{RL}) = SVPT(\text{VPL}) \subseteq CFL$.

**Proof.** First, for $VPL \subseteq SVPT(\text{RL})$, consider $T_{\text{del}}$ of Example 2, it transduces a RL into a CFL that is not a VPL (see Proposition 3). $T_{\text{del}}$ is a SVPT. Second, for $SVPT(\text{RL}) = SVPT(\text{VPL})$, consider any $S' \in SVPT$ and $A \in VPA$. Let $V = L(A)$. We construct $S' \in SVPT$ such that $S'(\Sigma^*) = S(V)$. More concretely, we impose that, for all $w \in \Sigma^*$ 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:

---

\* 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 \( \gamma \) and \( \gamma' \in \Gamma_{\text{copy}} \) on their respective stack, \( S' \) pushes the symbol \( (\gamma, \gamma') \). As \( S \) is a SVPT and \( \gamma' \in \Gamma_{\text{copy}} \), this ensures that when we reach the matching return, \( S \) copies the return, and the pair \( (\gamma, \gamma') \) 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 \( \gamma \) and \( \gamma' \in \Gamma_{\text{del}} \) on their respective stack, \( S' \) pushes the symbol \( (\gamma, \gamma') \). As \( S \) is a SVPT and \( \gamma' \in \Gamma_{\text{del}} \), this ensures that when we reach the matching return, \( S \) will delete the return, and the pair \( (\gamma, \gamma') \) 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 \( \gamma' \in \Gamma_{\text{ins}} \) on its stack. To simulate this, \( S' \) pushes the pair \( (\gamma, \gamma') \) on its stack, \( \gamma \) being a new stack symbol that does not belong to the stack symbols of \( A \). As \( \gamma' \) belongs to \( \Gamma_{\text{ins}} \), 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 \( (\gamma, \gamma') \), 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, \( \text{SVPT}(\text{VPL}) \subseteq \text{CFL} \) is a consequence of the facts that \( \text{SVPT}(\text{RL}) \subseteq \text{VPT}(\text{RL}) \subseteq \text{CFL} \) and \( \text{SVPT}(\text{VPL}) = \text{SVPT}(\text{VPL}) \). Finally, \( \text{SVPT}(\text{VPL}) \neq \text{CFL} \) is a consequence of the fact that typechecking \( \text{SVPT} \) against \( \text{VPL} \) is decidable (Theorem 5, see below) and the undecidability of checking the inclusion of a CFL into a VPL (Theorem 1). \( \square \)

### 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) \( \delta_e \subseteq Q \times \Sigma^e \times \Sigma^e \times Q \times \Gamma \), (ii) \( \delta_e \subseteq Q \times \Gamma \times \Sigma^e \times \Sigma^e \times Q \) and (iii) \( \delta_e \subseteq Q \times \Sigma^e \times \Sigma^e \times Q \) (and thus \( \Gamma_{\text{ins}} = \emptyset \)). This class is noted \( \text{SVPT}_{\text{ni}} \). A non-deleting SVPT \( T = (Q, Q_0, Q_f, \Gamma, \delta) \) is a SVPT such that (i) \( \delta_e \subseteq Q \times \Sigma^e \times \Sigma^e \times Q \times \Gamma \), (ii) \( \delta_e \subseteq Q \times \Sigma^e \times \Sigma^e \times Q \times \Gamma \), (ii) \( \delta_e \subseteq Q \times \Gamma \times \Sigma^e \times \Sigma^e \times Q \) and (iii) \( \delta_e \subseteq Q \times \Sigma^e \times \Sigma^e \times Q \) (and thus \( \Gamma_{\text{del}} = \emptyset \)). This class is noted \( \text{SVPT}_{\text{nd}} \).

**Proposition 6.** Let \( T \in \text{SVPT} \).

1. \( T \in \text{SVPT}_{\text{rd}} \) iff \( T^{-1} \in \text{SVPT}_{\text{ni}} \).
2. If \( T \in \text{SVPT}_{\text{rd}} \) and \( V \in \text{VPL} \) then \( T(V) \in \text{VPL} \).
3. If \( T \in \text{SVPT}_{\text{ni}} \) and \( V \in \text{VPL} \) then \( T^{-1}(V) \in \text{VPL} \).

**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 \( \text{SVPT}_{\text{rd}} \) \( T \), and the VPA \( A^{\text{ini}} \) for \( V \), we construct a VPA \( A^{\text{out}} \) that accepts \( T(V) \). We sketch here the main arguments of the proof, the full detailed proof is given in Appendix. On a word \( w \), \( A^{\text{out}} \)
guesses a word $v$ and checks that the pair $(v, w) \in [T]$ and $v \in V$. For that, the VPA $A^{\text{out}}$ simulates in parallel the execution of $A^{\text{in}}$ on $v$ and the execution of $T$ on the pair $(v, w)$, its run is accepting if the simulated runs in $A^{\text{in}}$ and $T$ are accepting. The main delicate part of the proof is to show that $A^{\text{out}}$ can simulate the two stacks while respecting the restrictions imposed to a VPA. The parallel simulation of the stack of $A^{\text{in}}$ and $T$ is possible because $T$ is a SVPT$_{nd}$; 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 $A^{\text{in}}$ and $T$ can be represented as pairs of symbols as follows:

- **call-return copy**: when $A^{\text{in}}$ and $T$ are moving and pushing a symbol $\gamma^{\text{in}}$ and $\gamma^{T} \in \Gamma^{\text{copy}}_\text{in}$ on their respective stack, $A^{\text{out}}$ pushes the symbol $(\gamma^{\text{in}}, \gamma^{T})$ and reads in $w$ the same symbol as written by $T$. As $T$ is a SVPT$_{nd}$ and $\gamma^{T} \in \Gamma^{\text{copy}}_\text{in}$, this ensures that when we reach the matching return in $v$ (and $A^{\text{in}}$ pop $\gamma^{\text{in}}$), $T$ copies the return in $w$ and pop $\gamma^{T}$. At that time, $A^{\text{out}}$ pops the pair $(\gamma^{\text{in}}, \gamma^{T})$ from its stack and reads in $w$ the same symbol as written by $T$. This simulates the behavior of the stacks of $A^{\text{in}}$ and $T$. From there, $A^{\text{out}}$ continues the parallel simulation of $A^{\text{in}}$ and $T$.

- **call-return insert**: on a call-return insert, only $T$ is moving. It pushes a symbol $\gamma^{T} \in \Gamma^{\text{ins}}_\text{in}$ on its stack and write $\beta$. To simulate this, $A^{\text{out}}$ reads $\beta$ and pushes the pair $(\gamma^{\text{in}}, \gamma^{T})$ on its stack, $\gamma^{\text{in}}$ being a new stack symbol that does not belong to the stack symbols of $A^{\text{in}}$. As $\gamma^{T} \in \Gamma^{\text{ins}}_\text{in}$, we know that the matching return in $w$, say $\beta'$, will be inserted (no letter will be read by $T$ and so $A^{\text{in}}$ will not move), at that time $A^{\text{out}}$ will pop the pair $(\gamma^{\text{in}}, \gamma^{T})$ when reading $\beta'$. This simulates the behavior of the stacks of $A^{\text{in}}$ and $T$. From there, $A^{\text{out}}$ continues the parallel simulation of $A^{\text{in}}$ and $T$.

Note that $A^{\text{out}}$ 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 $A_1, A_2 \in \text{VPA}$ and $T \in \text{SVPT}$, 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: $[T] = [T_{\text{in}}] \circ [T_{\text{nd}}]$, which are respectively non-inserting and non-deleting.

$T_{\text{nd}}$ will behave as $T$ with the essential difference that whenever $T$ deletes a call, a return, respectively an internal, $T_{\text{nd}}$ replaces it with $e^c$, $e^r$, respectively $e^i$ which are new call, return, respectively internal symbols that do not belong to the alphabet $\Sigma$. More formally, let $\hat{T} = (Q, Q_0, Q_f, \Gamma, \delta)$ over $\hat{\Sigma}$, $T_{\text{nd}}$ is a transducer from $\hat{\Sigma}$ into $\hat{\Sigma}_{\text{nd}} = \Sigma_{\text{nd}} \sqcup \Sigma_{\text{nd}}^c \sqcup \Sigma_{\text{nd}}^r \sqcup \Sigma_{\text{nd}}^i$ where $\Sigma_{\text{nd}} = \Sigma \cup \{e^c\}, \Sigma_{\text{nd}}^c = \Sigma \cup \{e^c\}, \Sigma_{\text{nd}}^r = \Sigma \cup \{e^r\}, \Sigma_{\text{nd}}^i = \Sigma \cup \{e^i\}$. We define $T_{\text{nd}} = (Q, Q_0, Q_f, \Gamma_{\text{nd}}, \delta_{\text{nd}})$, such that $\Gamma_{\text{nd}} = \Gamma_{\text{copy}} \sqcup \Gamma_{\text{ins}} \sqcup \Gamma_{\text{nd}}$.
As a consequence of Proposition 7 and Theorem 1, we cannot type check the composition of two functions. First, the proof of Lemma 1 tells us that there exists $w_1$, $w_2 \in [T]$, such that for all $(w_1, w_2) \in [T]$, there exists $w_3$ such that $(w_1, w_3) \in [T_{nd}]$, and conversely, for all $(w_1, w_3) \in [T_{nd}]$, $(w_1, 1_{\Sigma}(w_3)) \in [T]$. As $T$ is a SVPT, for all $w \in [T_{nd}(A^*)]$, $w$ is such that the matching return of every $e^c$ is an $e^c$, 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 $(e^c, e^c)$.

We define the transducer $T_{nd}$. For all $w_1$ that are synchronized on the pair $(e^c, e^c)$, the transducer accepts the pairs $(w_1, 1_{\Sigma}(w_1)) \in \Sigma_{nd} \times \Sigma$. Clearly, this transduction relation is realized by the following transducer $T_{nd} = \{(\hat{q}, q), \{\text{copy}, \gamma \text{del}, \bot, \delta_{ni}\}, (\Sigma \cup \{e^c, e^c\}) \times \Sigma$, such that $\delta_{ni} = \delta_i^c \cup \delta_i^c \cup \delta_i^c$, $\delta_{n\gamma} = \{(q, a, a, q, \gamma \text{copy}) \mid a \in \Sigma^* \} \cup \{(q, e^c, e^c, \gamma \text{del})\}$, $\delta_i^c = \{(q, \gamma \text{copy}, a, a, q) \mid a \in \Sigma^* \} \cup \{(q, e^c, e^c, e, q)\}$, $\delta_i^c = \{(q, \bot, a, a, q) \mid a \in \Sigma^* \} \cup \{(q, e^c, e^c, e, q)\}$, and $\delta_i^c = \{(q, a, a, q) \mid a \in \Sigma^* \} \cup \{(q, e^c, e^c, e, q)\}$ which is in the class SVPT$_{n}$. Clearly, $[T] = [T_{nd}]$. To prove the problem, we consider the following equivalence: $T(L(A_1)) \subseteq L(A_2) \Leftrightarrow T_{nd}(L(A_1)) \cap T_{nd}(L(A_2)) = \emptyset$. 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 $T_{nd}(L(A_2))$ (in polynomial time if $A_2$ is deterministic), and we can construct, in deterministic polynomial time in the size of $B_2$ and $T_{nd}^{-1}$, a VPA $B_3$ that accepts $T_{nd}^{-1}(L(A_2))$. Finally, checking emptiness of intersection between two VPA can be done in deterministic polynomial time (Proposition 1). This concludes our proof of 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 $C \in \text{CFL}(\hat{\Sigma})$, there exist $V \in \text{VPL}(\hat{\Sigma})$, $T_1, T_2 \in \text{SVPT}$ such that $T_2(T_1(V)) = C$.

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

---

\footnote{Note that without the hypothesis of synchronized on the pair $(e^c, e^c)$, there is no SVPT$_{n}$ that realizes $1_{\Sigma}$, that is the reason why this construction can not be generalized when $T$ is a VPT.}
Theorem 4. Let $A_1, A_2 \in \mathcal{VPA}$ and $T_1, T_2 \in \mathcal{SVPT}$, 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 $\mathcal{FSVPT}$.

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

1. VPL preservation: for any $V \in \mathcal{VPL}$, $T(V) \in \mathcal{VPL}$;
2. Inverse: $T^{-1} \in \mathcal{FSVPT}$;
3. Composition: for any $T_1 \in \mathcal{FSVPT}$ there exists $T_2 \in \mathcal{FSVPT}$ 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 $\text{Exptime-Complete}$.

Note that $\mathcal{FSVPT} = \mathcal{SVPT}_{\text{sd}} \cap \mathcal{SVPT}_{\text{nd}}$, 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 $\mathcal{FSVPT}$ as such automata would translate languages from $\mathcal{RL}(\hat{\Sigma})$ into $\mathcal{RL}(\hat{\Sigma})$ while $\mathcal{FSVPT}$ can transduce languages from $\mathcal{RL}(\hat{\Sigma})$ into languages that are not in $\mathcal{RL}(\hat{\Sigma})$. Finally, if $T$ is a $\mathcal{FSVPT}$ then it can be seen as a VPA that works on pairs of symbols (of the same type), and so, equivalence between $\mathcal{FSVPT}$ is $\text{Exptime-Complete}$.

5 Conclusion

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

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 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 $\mathcal{FSVPT}$ as such automata would translate languages from $\mathcal{RL}(\hat{\Sigma})$ into $\mathcal{RL}(\hat{\Sigma})$ while $\mathcal{FSVPT}$ can transduce languages from $\mathcal{RL}(\hat{\Sigma})$ into languages that are not in $\mathcal{RL}(\hat{\Sigma})$. Finally, if $T$ is a $\mathcal{FSVPT}$ then it can be seen as a VPA that works on pairs of symbols (of the same type), and so, equivalence between $\mathcal{FSVPT}$ is $\text{Exptime-Complete}$.

\[ \text{Exptime-Complete} \]

\[ \text{Exptime-Complete} \]

\[ \text{Exptime-Complete} \]

\[ \text{Exptime-Complete} \]

\[ \text{Exptime-Complete} \]

\[ \text{Exptime-Complete} \]

\[ \text{Exptime-Complete} \]

\[ \text{Exptime-Complete} \]
in details the relation between the transducers defined on regular tree languages, as 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

6 Appendix

6.1 Undecidability of inclusion between CFL and VPL

Lemma 2. Let $L \subseteq \text{CFL}$ and $L' \subseteq \text{VPL}$ then $L \subseteq L'$ and $L' \subseteq L$ are undecidable.

Proof. The undecidability $L' \subseteq L$ is a direct consequence of the undecidability of the universality of context-free languages and the fact that $\Sigma^* \in \text{VPL}$.

On the other hand, we reduce the reachability problem of a 2CM to $L \cap L' = \phi$ for VPL and CFL. Then, we can deduce that $L \subseteq L'$ is undecidable since VPL can be complemented.

A two-counter machine $C$, 2CM for short, is a tuple $\langle c_1, c_2, \text{Instr} \rangle$ where (i) $c_1, c_2$ are two counters taking their values in $\mathbb{N}$, (ii) $L = \{i_1, i_2, \ldots, i_a\}$ is a finite non-empty set of $a$ locations (iii) $\text{Instr}$ is a function that labels each location $l \in L$ with an instruction that has one of the three following forms where $j \in \{1, 2\}$ and $l' \in L$:

- $1: c_j := c_j + 1; \text{goto } l'$, is an increment and $\text{inst}_{\text{type}}(l) = \text{inc}_j$
- $1: c_j := c_j - 1; \text{goto } l'$, is a decrement and $\text{inst}_{\text{type}}(l) = \text{dec}_j$
- $1: \text{if } c_j = 0; \text{goto } l'; \text{else goto } l''$, is a zero-test and $\text{inst}_{\text{type}}(l) = \text{jz}_j$

A configuration of $C$ is a tuple $\langle \text{loc}, v_1, v_2 \rangle$ where $\text{loc} \in L$ and $v_1, v_2$ are natural numbers giving the valuation of $c_1$ and $c_2$. A computation, $\gamma$, is a finite or infinite sequence of configurations $\{\langle \text{loc}_0, v_1^0, v_2^0 \rangle, \langle \text{loc}_1, v_1^1, v_2^1 \rangle, \ldots\}$ such that: (i) $\text{loc}_0 = l_0, v_1^0 = 0$ and $v_2^0 = 0$ and (ii) for each $i \in \mathbb{N}, i < |\gamma|$ we have $\langle \text{loc}_{i+1}, v_1^{i+1}, v_2^{i+1} \rangle$ is the configuration obtained by applying $\text{Instr}(\text{loc}_i)$ to the configuration $\langle \text{loc}_i, v_1^i, v_2^i \rangle$. Minsky proved the undecidability of the reachability problem for 2CM [10].

Given the above defined 2CM, we define $\Sigma^e = \{\text{inc}_1, \text{nok}_1\}, \Sigma^r = \{\text{dec}_1, \text{jz}_1\}, \Sigma^i = \{\text{ok}_1, \text{dec}_2, \text{inc}_2, \text{jz}_2, \text{ok}_2, \text{nok}_2\}, \Sigma = \Sigma_e \cup \Sigma_r \cup \Sigma_i$. We associate to any finite run of $C$, $\rho = \langle \langle \text{loc}_1, v_1^1, v_2^1 \rangle, \ldots, \langle \text{loc}_k, v_1^k, v_2^k \rangle \rangle$, a word on $\Sigma$, $w(\rho) = w_1 \cdots w_{k-1}$, where $w_i$ is defined as:

$$w_i = \begin{cases} \text{ins}_i & \text{if } \text{inst}_{\text{type}}(\text{loc}_i) = \text{ins}_i \neq \text{jz}_j \\ \text{jz}_j \text{ ok}_j & \text{if } \text{inst}_{\text{type}}(\text{loc}_i) = \text{jz}_j \text{ and test is true} \\ \text{jz}_j \text{ nok}_j & \text{if } \text{inst}_{\text{type}}(\text{loc}_i) = \text{jz}_j \text{ and test is false} \end{cases}$$

where $j \in \{1, 2\}$.

We construct a VPL, $A = \langle Q, Q_0, I, \delta, Q_F \rangle$, that accept all words $w(\rho)$ associated with a run $\rho$ that reaches $l_f$ (it may accept other words as well). This VPL on $\Sigma$ simulates the first counter of $C$ but guesses the outcome of zero tests on the second counter:

- $I = \{\bot, 1\}$
- $Q = L \cup \{(l, a) | l \in L, a \in \{\bot, \bot_2, 1, 2\}\}$ - States $(l, \bot_1)$ allows to read the internal ok$_1$ and $(l, 1)$ allow only to push 1 and read the call nok$_1$. States $(l, \bot_2)$ and $(l, 2)$ allow only to go to state $l$ when reading the internals ok$_2$ and nok$_2$, respectively.
- \( Q_0 = \{ l_1 \} \)

- \( Q_F = \{ f \} \)

- \( \delta \subseteq (Q \times \Sigma_r \times Q) \cup (Q \times \Sigma_r \times \{ \bot \}) \cup (Q \times Q \times \{ \bot \}) \) such that for any \( l \in L \) if \( \text{inst\_type}(l) =: \)
  - \( \text{Inc}_1 \) then \( (l, \text{Inc}_1, l', 1) \) in \( \delta ; \)
  - \( \text{Dec}_1 \) then \( (l, 1, \text{Dec}_1, l') \) in \( \delta ; \)
  - \( jz_2 \) then \( (l, jz_2, \perp, (l', 1, 1)) \) in \( \delta ; \) and \( ((l', 1), \text{ok}_2, l') \) in \( \delta ; \) (test evaluates to true iff stack is empty), \( (l, 1, jz_2, (l'', 1)) \) in \( \delta ; \) and \( ((l'', 1), \text{ok}_1, l'', 1) \) in \( \delta ; \) (test evaluates to false iff stack not empty);
  - \( \text{Inc}_2 \) then \( (l, \text{Inc}_2, l') \) in \( \delta ; \)
  - \( \text{Dec}_2 \) then \( (l, \text{Dec}_2, l') \) in \( \delta ; \)
  - \( jz_2 \) then \( (l, jz_2, (l', 1, 2)) \) in \( \delta ; \) and \( ((l', 2), \text{ok}_2, l') \) in \( \delta ; \) (guessing test evaluates to true), \( (l, jz_2, (l'', 2)) \) in \( \delta ; \) and \( ((l'', 2), \text{ok}_2, l'') \) in \( \delta ; \) (guessing test evaluates to false).

Finally, the second counter is simulated by a PDA \( B \) in the same spirit as \( A \) simulates the first counter. As the PDA does not have to comply with partition of the alphabet, it pushes when it reads a \( \text{Inc}_2 \), it pops when it reads \( \text{Dec}_2 \), and it leaves the stack unchanged on all other symbols. The conjunction of the constraints expressed by the VPA and the PDA encodes exactly the definition of the executions of the two-counter machine.

### 6.2 SVPT\(_{nd}(L) \in \text{VPL}\)

**Definition 7.** Let \( \sigma \in \Gamma^* \) and \( \sigma' \in (\Gamma \cup \{\gamma_i\})^* \). We write \( \sigma \equiv \sigma' \iff \text{inst\_type}(\sigma) =: l \) \( \overset{l}{\Rightarrow} \) (\( \sigma \).

**Definition 8.** Let \( \sigma = (\gamma_1, \gamma_2, \ldots, \gamma_{n-1}, \gamma_n) \), then we define \( \pi_1(\sigma) = \gamma_1 \cdots \gamma_n \) and \( \pi_2(\sigma) = \gamma_1' \cdots \gamma_n' \).

We can now prove the following property.

**Proposition 8.** Let \( T \in \text{SVPT}\(_{nd}\) and \( L \in \text{VPL} \) then \( T(L) \in \text{VPL} \).

**Proof.** Let \( A_{in} = (Q^{in}, Q^{in}_0, Q^{in}_f, \Gamma^{in}, \delta^{in}) \) a VPA on \( \Sigma = \Sigma_r \sqcup \Sigma_f \sqcup \Sigma_i \) with \( L(A_{in}) = L, T = (Q^T, Q^T_0, Q^T_f, \Gamma^T, \delta^T) \) a SVPT\(_{nd}\), and let us define \( A_{out} = (Q^{out}, Q^{out}_0, Q^{out}_f, \Gamma^{out}, \delta^{out}) \) as follow:

- \( Q^{out} = Q^{in} \times Q^T, Q^{out}_0 = Q^{in}_0 \times Q^T_0, Q^{out}_f = Q^{in}_f \times Q^T_f, \Gamma^{out} = (\Gamma^{in} \setminus \{\bot\}) \times \Gamma^T, \delta^{out} \) the special bottom-of-stack element is noted \( (\bot, \bot) \).

- Call transitions: \( ((q_1^{in}, q_1^T), b, (q_2^{in}, q_2^T), (\gamma^{in}, \gamma^T)) \in \delta^{out} \) iff \( \exists a \in \Sigma : (q_1^{in}, q_2^{in}, a, q_2^{in} \in \delta^{in}_a, (q_1^T, \gamma^T, a, b, q_2^T) \in \delta^T_a) \) and \( (q_1^{in}, q_2^{in}) \in \Gamma^{in} \) and \( (\gamma^{in}, \gamma^T) \in \Gamma^{out} \). Observe that we always have \( (\gamma^{in}, \gamma^T) \in \Gamma^{out} \setminus \{ (\bot, \bot) \} \). Specially, never \( (\bot, \gamma^T) \) nor \( (\gamma^{in}, \bot) \) are put on the stack.

- Return transitions: \( ((q_1^{in}, q_1^T), (\gamma^{in}, \gamma^T), b, (q_2^{in}, q_2^T)) \in \delta^{out} \) iff one of the 3 cases is true: (i) \( \exists a \in \Sigma : (q_1^{in}, q_1^T, a, q_2^{in} \in \delta^{in}_a, (q_1^T, \gamma^T, a, b, q_2^T) \in \delta^T_a) \) and \( (\gamma^{in}, \gamma^T) \in \Gamma^{out} \), (ii) \( q_1^{in} = q_2^{in}, \gamma^{in} = \gamma_\epsilon, (q_1^T, \gamma^T, a, b, q_2^T) \in \delta^T_a \) and \( \gamma^T \neq \bot, \) or (iii) \( (q_1^T, \bot, a, b, q_2^T) \in \delta^T_a \) and \( (\gamma^{in}, \gamma^T) = ((\bot, \bot) \).
Internal transitions: if the top of the stack such that, (i) \( (\gamma^m, \gamma^T) \) is a run on \( \rho \) and \( \gamma^m = \gamma^m \) a run of \( T \) on \( u_k \) a word of length \( k \) then there exist

- \( u_k \in \Sigma^* \) a word of length \( j \),
- \( \rho = (q_0^m, \sigma_0^m) \ldots (q_j^m, \sigma_j^m) \) a run of \( A_{in} \) on \( u_k \),
- \( \rho^T = (q_0^T, j_0, j_0, \sigma_0^T) \ldots (q_k^T, j_k, j_k, \sigma_k^T) \) a run of \( T \) on \( (u_k, v_k) \).

such that, (i) \( q_k^T = (q_k^m, q_k^T) \), (ii) \( \sigma_k^m \equiv \pi_1(\sigma_k^m) \) and (iii) \( \sigma_k^m = \pi_2(\sigma_k^m) \). Note that if the top of the stack \( \sigma_k^m \) is in \((T^m(\bot) \times T^m)\) then (ii) implies that the top of \( \sigma_k^m \) is equal to the top of \( \pi_2(\sigma_k^m) \).

[Induction Basis] If \( |\rho^m| = 1 \), then it is of the form \( \rho^m = ((q^m, q^T), (\bot, \bot)) \), which is a run on \( v_0 = \epsilon \), we have:

- \( u_0 = \epsilon \),
- \( \rho = (q^m, 1, 1, \bot) \) is a run of \( A_{in} \) on \( u_0 \),
- \( \rho^T = (q^T, 1, 1, \bot) \) is a run of \( T \) on \( (u_0, v_0) \).

Moreover, (i) \( q_0^m = (q_0^m, q_0^T) \), (ii) \( \pi_1(\sigma^m_0) = \pi_1((\bot, \bot)) \equiv \bot = \sigma^m \) and (iii) \( \pi_2(\sigma^m_0) = \pi_2((\bot, \bot)) = \bot = \sigma^T \).

[Inductive Step] We suppose it is true for all \( |\rho^m| = k \) and let’s show it is true if \( |\rho^m| = k + 1 \). Let \( \rho^m_{k+1} = (q_0^m, \sigma_0^m) \ldots (q_k^m, \sigma_k^m)(q_{k+1}^m, \sigma_{k+1}^m) \) be a run of \( A_{out} \) on \( v_{k+1} = b_1 \ldots b_k \).

We know by induction that there are \( u_k = a_1 \ldots a_j \in \Sigma^* \), \( \rho^m_k = (q_0^m, \sigma_0^m) \ldots (q_j^m, \sigma_j^m) \) a run of \( A_{in} \) on \( u_k \), and \( \rho^T_k = (q_0^T, j_0, j_0, \sigma_0^T) \ldots (q_j^T, j_k, j_k, \sigma_j^T) \) a run of \( T \) on \( (u_k, v_k) \) with \( v_k = b_1 \ldots b_k \), such that, (i) \( q_k^m = (q_k^m, q_k^T) \), (ii) \( \sigma_k^m \equiv \pi_1(\sigma_k^m) \) and (iii) \( \sigma_k^T \equiv \pi_2(\sigma_k^m) \).

There are six types of transition to consider:

1. Call transition: \( \sigma_{k+1}^m = (\gamma^m, \gamma^T) \) and \( (q_{k+1}^m, b_{k+1}, \sigma_{k+1}^m, (\gamma^in, \gamma^T)) \in \delta_m \), with \( q_{k+1}^m = (q_{k+1}^m, q_{k+1}^T) \), then:

   - Call copy: \( (\gamma^m, \gamma^T) \) in \((T^m(\bot) \times T^m)\), \( \exists a \in \Sigma^* : (q_j^m, a, q_{j+1}^m, \gamma^in) \in \delta_m^a \) and \( (q_k^T, a, b_{k+1}, q_{k+1}^T, \gamma^T) \in \delta_v^a \), in that case:

     - \( u_{k+1}^m = u_k^m = a_k \)
     - \( \rho^m_{k+1} = \rho^m_k(q_{j+1}^m, \gamma^m \gamma^in) \) is a run of \( A_{in} \) on \( u_{k+1}^m \).
     - \( \rho^T_{k+1} = \rho^T_k(q_{j+1}^T, j_{k+1}, j_{k+1}, \gamma^T) \) is a run of \( T \) on \( (u_{k+1}^m, v_{k+1}^m) \).

If we note \( \sigma_{k+1}^m = \gamma^m \gamma^in, \sigma_{k+1}^T = \gamma^T \gamma^T \), we can check that the required induction properties are preserved: (i) \( q_{k+1}^m = (q_{j+1}^m, q_{j+1}^T) \), (ii) \( \pi_1(\sigma_{k+1}^m) = \gamma^m \gamma^in, \pi_1(\sigma_{k+1}^m) = \gamma^T \gamma^T, (iii) \pi_2(\sigma_{k+1}^m) = \gamma \pi_2(\sigma_{k+1}^m) = \gamma \sigma_{k+1}^T = \sigma_{k+1}^T \), or
- Call insert: \((\gamma'^{in}, \gamma^T) \in (\{\gamma_k\} \times \Gamma_{trans}^{T}), q_j^{in} = q_j^{i+1}\) and 
  \((q_k^{in}, \gamma^T, \epsilon, b_{k+1}, q_k^{T+1}) \in \delta^T\), in that case:
  - \(u_{k+1} = u_k\),
  - \(\rho_{k+1}^{in} = \rho_k^{in}\) is a run of \(A_{in}\) on \(u_{k+1}\),
  - \(\rho_{k+1}^{in} = \rho_k^{in} (q_k^{in} + k, j_k + 1, \gamma^T, \sigma_k^{T+1})\) is a run of \(T\) on \((u_{k+1}, v_{k+1})\).

If we note \(\sigma_{j+1}^{in} = \sigma_j^{in}, \sigma_{k+1}^{T+1} = \gamma^T \sigma_k^{T+1}\), we can check that the required induction properties are preserved: (i) \(q_k^{in} = q_j^{i+1}\), (ii) \(\pi_1(\sigma_k^{T+1}) = \pi_1(\sigma_k^{T+1}) \equiv \sigma_j^{in} \equiv \sigma_j^{i+1}\), (iii) \(\pi_2(\sigma_k^{T+1}) = \gamma^T \pi_2(\sigma_k^{T+1}) = \gamma^T \sigma_k^{T+1}\).

2. Return transition: \(\sigma_{k+1}^{out} = (\gamma'^{in}, \gamma^T) \sigma_k^{T+1}\) and 
  \((q_k^{out}, (\gamma'^{in}, \gamma^T), b_{k+1}, q_k^{out+1}) \in \delta^{out}\),
  with \(q_k^{out+1} = (q_j^{i+1}, q_k^{T+1})\), then either:

- Matched return copy: \((\gamma'^{in}, \gamma^T) \in (\Gamma^{in} \setminus \{\perp\}) \times \Gamma^{T}_{copy}\), \(\exists a \in \Sigma^* : (q_j^{in}, \gamma'^{in}, a, q_j^{out+1}) \in \delta^T\), in that case, if we pose \(\gamma'^{in} \sigma_j^{i+1} = \sigma_j^T\) and \(\gamma^T \sigma_{k+1}^{T+1} = \gamma^T \sigma_k^{T+1}\), then we have:
  - \(u_{k+1} = u_k a\),
  - \(\rho_{k+1}^{in} = \rho_k^{in}(q_k^{in} + 1, \sigma_j^{i+1})\) is a run of \(A_{in}\) on \(u_{k+1}\) (indeed, by induction properties (i) and (ii) before the last transition the automaton is in state \(q_j^{in}\) and the stack as a \(\gamma'^{in}\) on top by induction hypothesis),
  - \(\rho_{j+1}^{in} = \rho_k^{in} (q_k^{in} + 1, j_k + 1, \sigma_{k+1}^{T+1})\) is a run of \(T\) on \((u_{k+1}, v_{k+1})\) (indeed, by induction properties (i) and (iii) before the last transition the transducer is in state \(q_k^{in}\) and the stack as a \(\gamma^T\) on top by induction hypothesis).

We can check that the required induction properties are preserved: (i) \(q_k^{out+1} =
  (q_j^{i+1}, q_k^{T+1})\), (ii) \(\pi_1(\sigma_k^{T+1}) = \pi_1(\gamma'^{in} \gamma^T \sigma_k^{T+1}) \equiv \sigma_j^{i+1}\) and \(\gamma'^{in} \sigma_k^{T+1}\), and so \(\pi_1(\sigma_k^{T+1}) \equiv \sigma_j^{i+1}\) (iii) \(\pi_2(\sigma_k^{T+1}) = \gamma^T \pi_2(\sigma_k^{T+1}) = \gamma^T \sigma_k^{T+1}\), and so \(\pi_1(\sigma_k^{T+1}) = \sigma_{k+1}\).

- Unmatched return copy: \((\gamma'^{in}, \gamma^T) = (\perp, \perp), \exists a \in \Sigma^* : (q_j^{in}, \perp, a, q_j^{out+1}) \in \delta^T\), and 
  \((q_k^{out}, (\gamma'^{in}, \perp), b_{k+1}, q_k^{out+1}) \in \delta^{out}\), in that case, note that by definition of run of \(A_{in}\) and of \(T\) we must have \(\sigma_j^{in} = \perp\) and \(\sigma_j^{T+1} = \perp\), so if we pose \(\sigma_j^{i+1} = \sigma_j^{in} = \perp\) and \(\sigma_{k+1}^{T+1} = \sigma_k^{T+1} = \perp\), then we have:
  - \(u_{k+1} = u_k a\),
  - \(\rho_{k+1}^{in} = \rho_k^{in}(q_k^{in} + 1, \sigma_j^{i+1})\) is a run of \(A_{in}\) on \(u_{k+1}\) (indeed, by induction properties (i) and (ii) before the last transition the automaton is in state \(q_j^{in}\) and the stack as a \(\perp\) on top by induction hypothesis),
  - \(\rho_{j+1}^{in} = \rho_k^{in} (q_k^{in} + 1, j_k + 1, \sigma_{k+1}^{T+1})\) is a run of \(T\) on \((u_{k+1}, v_{k+1})\) (indeed, by induction properties (i) and (iii) before the last transition the transducer is in state \(q_k^{out}\) and the stack as a \(\perp\) on top by induction hypothesis).

We can check that the required induction properties are preserved: (i) \(q_k^{out+1} =
  (q_j^{i+1}, q_k^{T+1})\), (ii) \(\pi_1(\sigma_k^{T+1}) = \pi_1(\perp, \perp) \equiv \perp = \sigma_j^{i+1}\), (iii) \(\pi_2(\sigma_k^{T+1}) = \pi_2(\perp, \perp) \equiv \perp = \sigma_{k+1}\).

- Matched return insert: \((\gamma'^{in}, \gamma^T) \in (\{\gamma_k\} \times \Gamma^{in}_{trans}), q_j^{in} = q_j^{i+1}\) and 
  \((q_k^{in}, \gamma^T, \epsilon, b_{k+1}, q_k^{out+1}) \in \delta^T\), in that case, if we pose \(\gamma^T \sigma_{k+1}^{T+1} = \sigma_k^{T+1}\) then we have:
\begin{itemize}
  \item $u_{k+1} = u_k$,
  \item $\rho_{k+1}^n = \rho_k^n$ is a run of $A_{in}$ on $u_{k+1}$,
  \item $\rho_k^T = \rho_k^T(q_{k+1}^T, i, j_k + 1, \sigma_k^T)$ is a run of $T$ on $(u_{k+1}, v_{k+1})$ (indeed, by induction properties (i) and (iii) before the last transition the transducer is in state $q_k^T$ and $\gamma^T$ is on top of its stack).
\end{itemize}

We can check that the required induction properties are preserved: (i) $q_{k+1}^{out} = (q_{k+1}^n, q_{k+1}^T)$, (ii) $\pi_1(\sigma_{k+1}^{out}) = \pi_1((\gamma, \gamma^T)\sigma_{k+1}^{out}) = \pi_1(\sigma_{k+1}^{out}) \equiv \sigma_{j+1}^n$, and so $\pi_1(\sigma_{k+1}^{out}) \equiv \sigma_{j+1}^n$. (iii) $\gamma^T_2(\sigma_{k+1}^{out}) = \pi_2(\gamma^T_2(\sigma_{k+1}^{out})) = \pi_2(A_k) = \sigma_k^T_1 = \gamma^T_2\sigma_{k+1}^T$, and so $\pi_1(\sigma_{k+1}^{out}) = \sigma_k^T_1$.

- Unmatched return insert: $(\gamma^m, \gamma^T) = (\perp, \perp), q_{j+1}^{in} = q_{j+1}^{in}$ and $(q_{k+1}^T, i, e, b, q_{k+1}^T) \in \delta_k^T$, in that case, note that $\sigma_k^T = \perp$, and if we pose $\sigma_k^{out} = \sigma_k^T$ then we have:
  \begin{itemize}
    \item $u_{k+1} = u_k$,
    \item $\rho_{k+1}^n = \rho_k^n$ is a run of $A_{in}$ on $u_{k+1}$,
    \item $\rho_k^T = \rho_k^T(q_{k+1}^T, i, j_k + 1, \sigma_k^T)$ is a run of $T$ on $(u_{k+1}, v_{k+1})$ (indeed, by induction properties (i) and (iii) before the last transition the transducer is in state $q_k^T$ and $\perp$ is on top of its stack).
  \end{itemize}

We can check that the required induction properties are preserved: (i) $q_{k+1}^{out} = (q_{k+1}^n, q_{k+1}^T)$, (ii) $\pi_1(\sigma_{k+1}^{out}) = \pi_1((\perp, \perp)\sigma_{k+1}^{out}) = \perp = \sigma_{j+1}^n$, and so $\pi_1(\sigma_{k+1}^{out}) \equiv \sigma_{j+1}^n$. (iii) $\gamma^T_2(\sigma_{k+1}^{out}) = \pi_2(\gamma^T_2(\sigma_{k+1}^{out})) = \pi_2(A_k) = \sigma_k^T_1 = \gamma^T_2\sigma_{k+1}^T$, and so $\pi_1(\sigma_{k+1}^{out}) = \sigma_k^T_1$.

3. Internal transition: $(q_{k+1}^{out}, b_{k+1}, q_{k+1}^{out}) \in \delta_q^{out}$, with $q_{k+1}^{out} = (q_{k+1}^n, q_{k+1}^T)$ then either:

- Internal copy: $\exists a \in \Sigma^i: (q_{j+1}^m, a, q_{j+1}^n) \in \delta_k^m \land (q_{k+1}^m, a, b_{k+1}, q_{k+1}^T) \in \delta_k^T$. If we note $\sigma_{j+1}^n = \sigma_{j+1}^T = \sigma_k^T$ and $\sigma_{k+1}^{out} = \sigma_{k+1}^{out}$, in that case:
  \begin{itemize}
    \item $u_{k+1} = u_k a$,
    \item $\rho_{k+1}^n = \rho_k^n(q_{k+1}^n, \sigma_{j+1}^n)$ is a run of $A_{in}$ on $u_{k+1}$,
    \item $\rho_k^T = \rho_k^T(q_{k+1}^T, i, j_k + 1, \sigma_k^T)$ is a run of $T$ on $(u_{k+1}, v_{k+1})$.
  \end{itemize}

We can check that the required induction properties are preserved: (i) $q_{k+1}^{out} = (q_{k+1}^n, q_{k+1}^T)$, (ii) $\pi_1(\sigma_{k+1}^{out}) = \pi_1((\perp, \perp)\sigma_{k+1}^{out}) = \perp = \sigma_{j+1}^n$, and so $\pi_1(\sigma_{k+1}^{out}) \equiv \sigma_{j+1}^n$. (iii) $\gamma^T_2(\sigma_{k+1}^{out}) = \pi_2(\gamma^T_2(\sigma_{k+1}^{out})) = \pi_2(A_k) = \sigma_k^T_1 = \gamma^T_2\sigma_{k+1}^T$, and so $\pi_1(\sigma_{k+1}^{out}) = \sigma_k^T_1$.

- Internal insert: $(q_{j+1}^m, e, b, q_{k+1}^T) \in \delta_k^T$, in that case, if we pose $\sigma_{j+1}^n = \sigma_j^T$, $\sigma_{k+1}^T = \sigma_k^T$ and $\sigma_{k+1}^{out} = \sigma_{k+1}^{out}$, then we have:
  \begin{itemize}
    \item $u_{k+1} = u_k$,
    \item $\rho_{k+1}^n = \rho_k^n$ is a run of $A_{in}$ on $u_{k+1}$,
    \item $\rho_k^T = \rho_k^T(q_{k+1}^T, i, j_k + 1, \sigma_k^T)$ is a run of $T$ on $(u_{k+1}, v_{k+1})$.
  \end{itemize}

We can check that the required induction properties are preserved: (i) $q_{k+1}^{out} = (q_{k+1}^n, q_{k+1}^T)$, (ii) $\pi_1(\sigma_{k+1}^{out}) = \pi_1((\perp, \perp)\sigma_{k+1}^{out}) = \perp = \sigma_{j+1}^n$, and so $\pi_1(\sigma_{k+1}^{out}) \equiv \sigma_{j+1}^n$. (iii) $\gamma^T_2(\sigma_{k+1}^{out}) = \pi_2(\gamma^T_2(\sigma_{k+1}^{out})) = \pi_2(A_k) = \sigma_k^T_1 = \gamma^T_2\sigma_{k+1}^T$, and so $\pi_1(\sigma_{k+1}^{out}) = \sigma_k^T_1$.

So we showed that if we have a run of $A_{out}$ on $v$, then $v$ is indeed the result of the transduction by $T$ of a word $u$ which is accepted by $A_{in}$. Let’s prove the reverse is also true.
\[ T(L) \subseteq L(A_{\text{out}}) \]. If there exist \( u, v \in \Sigma^* \), \( \rho^{in} \) a run of \( A_{in} \) on \( u \), and \( \rho^T \) a run of \( T \) on \( (u, v) \), we must have a run \( \rho^{out} \) of \( A_{\text{out}} \) on \( v \). Note that we always have \( |\rho^{in}| \leq |\rho^T| \).

We now prove by induction on \( |\rho^T| \) the following assertion. Given \( u, v \in \Sigma^* \), \( \rho^{in} = (q_0^{in}, \sigma_0^{in}) \ldots (q_n^{in}, \sigma_n^{in}) \) a run of \( A_{in} \) on \( u \), and \( \rho^T = (q_0^T, j_0, \beta_0, \sigma_0^T) \ldots (q_m^T, j_m, \beta_m, \sigma_m^T) \) a run of \( T \) on \( (u, v) \), we have a run \( \rho^{out} = (q_0^{out}, \sigma_0^{out}) \ldots (q_n^{out}, \sigma_n^{out}) \) of \( A_{\text{out}} \) on \( v \) such that:

(i) \( q_0^{out} = (q_0^T, \sigma_0^T) \),
(ii) \( \pi_1(\sigma_{out}^{m}) \equiv \sigma_{in}^T \), and (iii) \( \pi_2(\sigma_{out}^{m}) = \sigma_{out}^T \).

[Induction Basis] If \( |\rho^T| = 1 \) then \( \rho^T = (q_0^T, 1, 1, 1) \) is a run of \( T \) on \( (u, v) \) with \( u = v = \lambda \), then \( |\rho^{in}| = 1 \) and \( \rho^T = (q_0^T, 1) \) is the given run on \( u \), we have:
\( \rho^{out} = ((q_0^T, 1, 1, 1)) \) is a run of \( A_{\text{out}} \) on \( v \). Moreover the inductive properties are satisfied:
(i) \( q_0^{out} = (q_0^T, \beta_0^T) \),
(ii) \( \pi_1(\sigma_{out}^{m}) \equiv \pi_1((1, 1, 1)) \equiv \beta_0 \), and (iii) \( \pi_2(\sigma_{out}^{m}) = \pi_2((1, 1, 1)) = \sigma_{out}^T \).

[Inductive Step] We suppose it is true for all \( |\rho_k^T| = k \) and let’s show it is true if \( |\rho_{k+1}^T| = k + 1 \).

Let \( \rho_{k+1}^T = (q_0^T, \sigma_0^T) \ldots (q_k^T, \sigma_k^T)(q_{k+1}^T) \) a run of \( T \) on \( (u_{k+1}, v_{k+1}) \) with \( u_{k+1} = a_1 \ldots a_j \) and \( v_{k+1} = b_1 \ldots b_k+1 \) and \( \rho^{in} = (q_0^{in}, \sigma_0^{in}) \ldots (q_k^{in}, \sigma_k^{in}) \) the given run of \( A_{in} \) on \( s_{k+1} \).

We have several possible cases:

1. Call copy:
\( (q_j^T, a_j, b_{j+1}, q_{j+1}^T, \gamma^T) \) in \( \delta_{in}^T \) with \( \sigma_k^{in} = \gamma^T \sigma_k^T \) then we must have \( x_j = \Delta_{in}^T \) and thus \( (q_j^{in}, a_j, q_j^{out}, \gamma^{out}) \) in \( \delta_{out}^T \) for \( \gamma^{out} \) such that \( \sigma_j^{in} = \gamma^{out} \). By definition of \( A_{\text{out}} \), \( ((q_j^{in}, q_j^{out}), b_{j+1}, (q_{j+1}^{in}, q_{j+1}^{out})), (\gamma^{out}, \gamma^T)) \) in \( \delta_{out}^T \). Moreover by induction hypothesis we have that there exists a run \( \rho_{out}^T = (q_0^{out}, \sigma_0^{out}) \ldots (q_k^{out}, \sigma_k^{out}) \) of \( A_{\text{out}} \) on \( v_k = b_1 \ldots b_k \) with:
(i) \( q_k^{out} = (q_k^{in}, q_k^{out}) \),
(ii) \( \pi_1(\sigma_{out}^{m}) \equiv \gamma^{out} \), and (iii) \( \pi_2(\sigma_{out}^{m}) = \sigma_k^T \).

But then if we note \( \sigma_{out}^{m+1} = (\gamma^{out}, \gamma^T) \sigma_k^{out} \), then \( \rho^{out} = (q_0^{out}, \sigma_0^{out}) \ldots (q_k^{out}, \sigma_k^{out})((q_{k+1}^{in}, q_{k+1}^{out}), (\gamma^{out}, \gamma^T)) \) is a run of \( A_{\text{out}} \) in \( v_{k+1} \). Indeed, \( (\sigma_0^{out}, \sigma_0^{out}) \ldots (q_k^{out}, \sigma_k^{out}) \) is a run of \( A_{\text{out}} \) on \( v_k = b_1 \ldots b_k \). By definition of \( A_{\text{out}} \), \( ((q_j^{in}, q_j^{out}), b_{j+1}, (q_{j+1}^{in}, q_{j+1}^{out})), (\gamma^{out}, \gamma^T)) \) in \( \delta_{out}^T \). Moreover by induction hypothesis we have that there exists a run \( \rho_{out}^T = (q_0^{out}, \sigma_0^{out}) \ldots (q_k^{out}, \sigma_k^{out})((q_{k+1}^{in}, q_{k+1}^{out}), (\gamma^{out}, \gamma^T)) \) is a run of \( A_{\text{out}} \) on \( v_k = b_1 \ldots b_k \).
Finally the induction properties are verified: (i) \( \delta_{k+1}^{\text{out}} = (q^{i_n}_{k}, q^{T}_{k+1}) \), (ii) \( \pi_1(\gamma, \sigma^{\text{out}}_k) = \pi_1(\gamma, \sigma^{\text{out}}_{k+1}) \equiv \sigma^{\text{out}}_{j+1} \), and (iii) \( \pi_2(\sigma^{\text{out}}_k) = \gamma \cdot \pi_2(\sigma^{\text{out}}_{k+1}) = \gamma \cdot \sigma^{\text{T}}_k \). 

3. Matched return copy: \((q^{T}_{k}, \gamma, a_j, b_{k+1}, q^{T}_{k+1}) \) \( \in \delta_{r}^{T} \) with \( \gamma \neq \perp, \sigma^{T}_k \neq \perp, \sigma^{T}_{k+1} \) and \( \gamma \in \Gamma_{\text{in}} \). But then we must have \( a_j \in \Sigma^{r} \) and thus \( (q^{i_n}_{k+1}, \gamma, a_j, q^{i}_k) \in \delta^{i_n} \) for some \( \gamma^{i_n} \) such that \( \sigma^{i_n}_{j-1} = \gamma^{i_n} \sigma^{i_n}_{j} \) or \( \sigma^{i_n}_{j-1} = \perp \) (definition of run of \( A_{in} \)). Also, we have \((\gamma^{i_n}_{k+1}, q^{T}_{k}, \gamma, a_j, b_{k+1}, q^{T}_{k+1}) \) \( \in \delta^{r} \) by definition of \( A_{out} \).

Moreover by induction hypothesis there exists a run \( \rho^{\text{out}}_k = (q^{0}_{k}, \sigma^{\text{out}}_0) \ldots (q^{0}_{k+1}, \sigma^{\text{out}}_{k+1}) \) of \( A_{out} \) on \( v_k = b_k \ldots b_1 \) with: (i) \( q^{0}_{k+1} = (q^{i_n}_{k+1}, q^{T}_{k}) \). (ii) \( \pi_1(\sigma^{\text{out}}_{k+1}) \equiv \sigma^{i_n}_{j+1} \), and (iii) \( \pi_2(\sigma^{\text{out}}_{k+1}) = \sigma^{T}_{k} \). 

This implies that \( \gamma \in \Gamma^{i_n} \) is on top of \( \sigma^{\text{out}} \) for some \( \gamma \). By definition of \( \Gamma^{i_n} \) and the fact that \( \gamma \in \Gamma_{\text{in}} \setminus \{\perp\} \), so \( \gamma = \gamma^{i_n} \) by induction property (ii).

But then if we note \( \sigma^{\text{out}}_{k+1} \) defined by \( \delta^{\text{out}}_k \equiv (\sigma^{i_n}, \gamma)_{\sigma^{\text{out}}_{k+1}} \), then \( \rho^{\text{out}}_k = (q^{0}_{k+1}, \sigma^{\text{out}}_0) \ldots (q^{0}_{k+1}, \sigma^{\text{out}}_{k+1}) \) is a run of \( A_{out} \) on \( v_k + 1 \).

Indeed, by induction property (i), \((q^{0}_{k}, \sigma^{\text{out}}_0) \ldots (q^{0}_{k+1}, \sigma^{\text{out}}_{k+1}) \) is a run of \( A_{out} \) on \( v_k = b_k \ldots b_1 \) that finishes in state \( q^{i_n}_{k+1} = (q^{i_n}_{k+1}, q^{T}_{k}) \) and the stack has \( (\gamma^{i_n}, \gamma^{T}) \) on top.

Finally the induction properties are verified: (i) \( \delta^{\text{out}}_{k+1} = (q^{i_n}_{k+1}, q^{T}_{k+1}) \), (ii) \( \gamma^{i_n} \pi_2(\sigma^{\text{out}}_{k+1}) = \pi_1((\gamma^{i_n}, \gamma^{T})_{\sigma^{\text{out}}_{k+1}}) = \pi_1(\sigma^{\text{out}}_{j+1}) \equiv \gamma^{i_n} \sigma^{\text{out}}_{j+1} \) so we have \( \pi_1(\sigma^{\text{out}}_{k+1}) \equiv \gamma^{i_n} \sigma^{\text{out}}_{j} \), and (iii) \( \gamma^{T} \pi_2(\sigma^{\text{out}}_{k+1}) = \pi_2((\gamma^{i_n}, \gamma^{T})_{\sigma^{\text{out}}_{k+1}}) = \pi_2(\sigma^{\text{out}}_{k+1}) = \gamma^{i_n} \sigma^{T}_{k} \). 

We have \( a_j \in \Sigma^{r} \).

4. Unmatched return copy: \((q^{T}_{k}, \perp, a_j, b_{k+1}, q^{T}_{k+1}) \) \( \in \delta_{r}^{T} \) with \( \sigma^{T}_k = \sigma^{T}_{k+1} = \perp \). 

By induction hypothesis there exists a run \( \rho^{\text{out}}_k = (q^{0}_{k}, \sigma^{\text{out}}_0) \ldots (q^{0}_{k+1}, \sigma^{\text{out}}_{k+1}) \) of \( A_{out} \) on \( v_k = b_k \ldots b_1 \) with: (i) \( q^{0}_{k+1} = (q^{i_n}_{k+1}, q^{T}_{k}) \). (ii) \( \pi_1(\sigma^{\text{out}}_{k+1}) \equiv \sigma^{i_n}_{j+1} \), and (iii) \( \pi_2(\sigma^{\text{out}}_{k+1}) = \sigma^{T}_{k} \). Properties (ii) and (iii) and the fact that \( \sigma^{i_n}_{j-1} = \perp \) implies that \( \sigma^{i_n}_{j} = \perp \).

Thus \( (q^{i_n}_{j+1}, \perp) \) \( \in \delta^{i_n} \) with \( \sigma^{i_n}_{j-1} = \perp \) (definition of run of \( A_{in} \)). Also, we have \((q^{i_n}_{j}, q^{T}_{j}), (\perp, \perp) \) \( \in \delta^{r} \) by definition of \( A_{out} \). 

But then if we note \( \sigma^{\text{out}}_{k+1} = \sigma^{\text{out}}_{k} \) \( \equiv (\perp, \perp) \), then \( \rho^{\text{out}}_k = (q^{0}_{k}, \sigma^{\text{out}}_0) \ldots (q^{0}_{k+1}, \sigma^{\text{out}}_{k+1}) \) \( \in \delta^{r} \). But (i) \( q^{0}_{k+1} = (q^{i_n}_{k+1}, q^{T}_{k}) \). Indeed, by induction property (i), \((q^{0}_{k}, \sigma^{\text{out}}_0) \ldots (q^{0}_{k+1}, \sigma^{\text{out}}_{k+1}) \) is a run of \( A_{out} \) on \( v_k = b_k \ldots b_1 \) that finishes in state \( q^{i_n}_{k+1} = (q^{i_n}_{k+1}, q^{T}_{k}) \) and the stack has \( (\perp, \perp) \) on top.

Finally the induction properties are verified: (i) \( \delta^{\text{out}}_{k+1} = (q^{i_n}_{k+1}, q^{T}_{k+1}) \), (ii) \( \pi_1(\sigma^{\text{out}}_{k+1}) \equiv \perp = \sigma^{i_n}_{j} \), and (iii) \( \pi_2(\sigma^{\text{out}}_{k+1}) = \pi_2((\perp, \perp)) = \perp = \sigma^{T}_{k} \).

5. Matched Return insert: \((q^{T}_{k}, \gamma, a_j, b_{k+1}, q^{T}_{k+1}) \) \( \in \delta_{r}^{T} \) with \( \gamma \neq \perp, \sigma^{T}_k \neq \perp, \sigma^{T}_{k+1} \) and \( \gamma \in \Gamma_{\text{in}} \). We have \((q^{i_n}_{k+1}, \perp), (\gamma, \gamma^{T}), b_{k+1}, (q^{T}_{k}, q^{T}_{k+1}) \) \( \in \delta^{r} \).

Moreover by induction hypothesis there exists a run \( \rho^{\text{out}}_k = (q^{0}_{k}, \sigma^{\text{out}}_0) \ldots (q^{0}_{k+1}, \sigma^{\text{out}}_{k+1}) \) of \( A_{out} \) on \( v_k = b_k \ldots b_1 \) with: (i) \( q^{0}_{k+1} = (q^{i_n}_{k+1}, q^{T}_{k}) \). (ii) \( \pi_1(\sigma^{\text{out}}_{k+1}) \equiv \sigma^{i_n}_{j} \), and (iii) \( \pi_2(\sigma^{\text{out}}_{k+1}) = \sigma^{T}_{k} \).
But then if we note $\sigma_{k+1}^{\text{out}}$ defined by $\sigma_{k+1}^{\text{out}} = (\gamma, \gamma^T)\sigma_k^{\text{out}}$, then $\rho^{\text{out}} = (q_0^{\text{out}}, \sigma_0^{\text{out}})\ldots(q_k^{\text{out}}, \sigma_k^{\text{out}})((q_j^{\text{in}}, q_{k+1}^T), \sigma_{k+1}^{\text{out}})$ is a run of $A_{\text{out}}$ on $v_{k+1}$.

Indeed, $(q_0^{\text{out}},\sigma_0^{\text{out}})\ldots(q_k^{\text{out}}, \sigma_k^{\text{out}})$ is a run of $A_{\text{out}}$ on $v_k = b_1\ldots b_k$ that finishes in state $q_k^{\text{out}} = (q_j^m, q_k^m)$ (by induction property (i)) and the stack has $(\gamma, \gamma^T)$ on top because by induction property (ii) it has $(\gamma, \gamma^T) \in \Gamma_{\text{out}}$ on top, but $\gamma^T \in \Gamma_{\text{ins}}$ and $\Gamma_{\text{out}} = (\Gamma_{\text{in}} \times \Gamma_{\text{copy}}) \cup (\{\gamma\} \times \Gamma_{\text{ins}})$ thus $\gamma = \gamma^T$. Finally the induction properties are verified: (i) $\pi_1(\sigma_{k+1}^{\text{out}}) = \pi_1((\gamma, \gamma^T)\sigma_{k+1}^{\text{out}}) = \pi_1(\sigma_{k+1}^{\text{out}}) \equiv \sigma_{j+1}^{\text{in}} \equiv \sigma_{j+1}$, and so $\pi_1(\sigma_{k+1}^{\text{out}}) \equiv \sigma_{j+1}$, and (iii) $\gamma^T\pi_2(\sigma_{k+1}^{\text{out}}) = \pi_2((\gamma, \gamma^T)\sigma_{k+1}^{\text{out}}) = \pi_2(\sigma_{k+1}^{\text{out}}) = \sigma_k^T = \gamma^T\sigma_{k+1}^T$, and so $\pi_1(\sigma_{k+1}^{\text{out}}) = \sigma_{k+1}^T$.

6. Unmatched return insert: Similar as the previous one.

7. The case of internal transitions is straightforward because the stack is not involved.