# Regularity Problems for Visibly Pushdown Languages

Vince Bárány[1], Christof Löding[1], and Olivier Serre[2⋆]

[1] RWTH Aachen, Germany
[2] LIAFA, Université Paris VII & CNRS, France

**Abstract.** Visibly pushdown automata are special pushdown automata whose stack behavior is driven by the input symbols according to a partition of the alphabet. We show that it is decidable for a given visibly pushdown automaton whether it is equivalent to a visibly counter automaton, i.e. an automaton that uses its stack only as counter. In particular, this allows to decide whether a given visibly pushdown language is a regular restriction of the set of well-matched words, meaning that the language can be accepted by a finite automaton if only well-matched words are considered as input.

## 1 Introduction

The class of context-free languages (CFL) plays an important role in several areas of computer science. Besides its definition using context-free grammars it has various other characterizations, the most prominent being the one via nondeterministic pushdown automata. It is well known that CFL does not enjoy good closure properties, e.g. it is not closed under complement and intersection, and that several interesting problems are undecidable, e.g. checking whether a context free language is regular, or whether it contains all words. This situation only slightly improves when considering the subclass of deterministic context free languages, i.e. languages accepted by deterministic pushdown automata (see [10] for an overview).

Another subclass of CFL that has recently been defined in [2] is the class of visibly pushdown languages. These are languages that are accepted by pushdown automata whose stack behavior (i.e. whether to execute a push, a pop, or no stack operation) is completely determined by the input symbol according to a fixed partition of the input alphabet. These automata are called visibly pushdown automata (VPA). As shown in [2, 3] this class of visibly pushdown languages enjoys many good properties similar to those of the class of regular languages, the main reason for this being that each nondeterministic VPA can be transformed into an equivalent deterministic one. Visibly pushdown automata have turned out to be useful in various context, e.g. as specification formalism for verification

⋆ Supported by the European Community Research Training Network "Games and Automata for Synthesis and Validation" (GAMES). Most of this work was done when the third author was a postdoctoral researcher at RWTH Aachen.

and synthesis problems for pushdown systems [1, 11], and as automaton model for processing XML streams [14, 12].

As each nondeterministic VPA can be determinized, all problems that concern the accepted language and that are decidable for deterministic pushdown automata are also decidable for VPAs. For example, in [15] and later with improved complexity in [16] it is shown that for a given deterministic pushdown automaton it is decidable whether its accepted language is regular. Hence, this problem is also decidable for VPAs.

In the context of validating streaming XML documents a similar question has been addressed in [14]. Phrased in the terminology of finite automata on words and trees the problem of validating streaming documents is the following: given the coding of a tree by a word using opening and closing tags around each subtree, check whether the corresponding tree belongs to a given regular tree language. It is rather simple to see that this task can be solved by a deterministic pushdown automaton that pushes a symbol onto the stack for each opening tag and pops a symbol for each closing tag. One of the questions raised and analyzed in [14] is whether one can decide for a given tree language if the streaming validation task can be solved by a finite automaton. As such an automaton has to verify that the input codes a tree, the class of these tree languages is rather restricted. The question gets more involved under the assumption that the input indeed codes a tree.

Coming back to VPAs, this assumption on the input being the coding of a tree corresponds to the assumption that the input is well-matched in the sense that each symbol that is pushed is popped eventually (each opening tag has a matching closing tag). The question of regularity of the accepted language then becomes: given a VPA, is there an equivalent finite automaton, where equivalence is restricted to the set of well-matched words? Restricting the equivalence to well-matched words can also be seen as allowing the finite automaton to count the difference between opening and closing tags to know in the end if the input was well-matched. This model is what we refer to as a visibly counter automaton (VCA). The main result of this paper is that it is decidable for a given VPA whether it is equivalent to a VCA. This problem is mentioned in [16] for deterministic pushdown automata and deterministic one-counter automata, and is to our knowledge still open.

The remainder of this paper is organized as follows. In Section 2 we provide the basic definitions of visibly pushdown and counter automata and state the main questions that we address. In Section 3 we give some basic concepts and constructions on which the decidability proofs are based. In Section 4 we show that it is decidable for a given visibly pushdown automaton whether it is equivalent to a visibly counter automaton that is allowed to test its counter value up to a certain threshold, and in Section 5 we prove that it is decidable whether such a threshold can be reduced.

We thank Victor Vianu and Luc Segoufin for drawing our attention to this topic.

## 2 Definitions

For a finite set $X$ we denote the set of finite words over $X$ by $X^*$. We denote by $\varepsilon$ the empty word. For $u \in X^*$, we write $u(n)$ for the $n$th letter in $u$ and $u{\restriction}_n$ for the prefix of length $n$ of $u$, i.e., $u{\restriction}_0 = \varepsilon$ and $u{\restriction}_n = u(0)\cdots u(n-1)$ for $n \geq 1$.

A pushdown alphabet is a tuple $\widetilde{\Sigma} = \langle \Sigma_c, \Sigma_r, \Sigma_{\text{int}} \rangle$ that comprises three disjoint finite alphabets: $\Sigma_c$ is a finite set of *calls*, $\Sigma_r$ is a finite set of *returns*, and $\Sigma_{\text{int}}$ is a finite set of *internal actions*. For any such $\widetilde{\Sigma}$, let $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_{\text{int}}$.

We define *visibly pushdown automata* over $\widetilde{\Sigma}$. Intuitively, a visibly pushdown automaton is a pushdown automaton restricted such that it pushes onto the stack only when it reads a call, it pops the stack only on reading a return, and it does not use the stack when reading an internal action.

**Definition 1 (Visibly pushdown automaton [2]).** *A visibly pushdown automaton (*VPA*) over $\widetilde{\Sigma}$ is a tuple $\mathcal{A} = (Q, \Sigma, \Gamma, Q_{in}, F, \Delta)$ where $Q$ is a finite set of states, $Q_{in} \subseteq Q$ is a set of initial states, $F \subseteq Q$ is a set of final states, $\Gamma$ is a finite stack alphabet, and $\Delta \subseteq (Q \times \Sigma_c \times Q \times \Gamma) \cup (Q \times \Sigma_r \times \Gamma \times Q) \cup (Q \times \Sigma_{\text{int}} \times Q)$ is the transition relation.*

To represent stacks we use a special bottom-of-stack symbol $\perp$ that is not in $\Gamma$. A *stack* is a finite sequence from the set $\perp \cdot \Gamma^*$ starting with the special symbol $\perp$ on the left, and ending with the top symbol on the right.[3] The *empty stack* is the one that only contains the symbol $\perp$.

A transition $(q, a, q', \gamma)$ with $a \in \Sigma_c$ is a push-transition where on reading $a$, $\gamma$ is pushed onto the stack and the control changes from state $q$ to $q'$. Similarly, $(q, a, \gamma, q')$ with $a \in \Sigma_r$ is a pop-transition where $\gamma$ is read from the top of the stack and popped (if the top of stack is $\perp$, then no pop-transition can be applied), and the control state changes from $q$ to $q'$. Our model (in contrast to the original definition from [2]) is therefore inherently restricted to input words having no prefix of negative stack height (to be defined below). Note that on internal actions, there is no stack operation.

A *configuration* of a VPA $\mathcal{A}$ is a pair $(\sigma, q)$, where $q \in Q$ and $\sigma \in \perp \cdot \Gamma^*$. There is an *a-transition* from a configuration $(\sigma, q)$ to $(\sigma', q')$, denoted $(\sigma, q) \xrightarrow{a} (\sigma', q')$ ($\mathcal{A}$ will be clear from context), if the following are satisfied.

- If $a$ is a call, then $\sigma' = \sigma\gamma$ for some $(q, a, q', \gamma) \in \Delta$.
- If $a$ is a return, then $\sigma = \sigma'\gamma$ for some $(q, a, \gamma, q') \in \Delta$.
- If $a$ is an internal action, then $\sigma = \sigma'$ and $(q, a, q') \in \Delta$ .

For a finite word $u = a_0 a_1 \cdots a_n$ in $\Sigma^*$, a *run* of $\mathcal{A}$ on $u$ is a sequence of configurations $(\sigma_0, q_0)(\sigma_1, q_1)\cdots(\sigma_{n+1}, q_{n+1})$, where $q_0 \in Q_{in}$, $\sigma_0 = \perp$ and for every $0 \leq i \leq n$, $(\sigma_i, q_i) \xrightarrow{a_i} (\sigma_{i+1}, q_{i+1})$ holds. In this case we also use the notation $(\sigma_0, q_0) \xrightarrow{u} (\sigma_{n+1}, q_{n+1})$. A word $u \in \Sigma^*$ is *accepted* by a VPA if there is a run over $u$ which ends in a final configuration, that is a configuration with

[3] Note that we are using here the reverse of the more common notation of stacks, having the top symbol on the left and the bottom on the right.

empty stack and a control state, which is final. The language $L(\mathcal{A})$ of a VPA $\mathcal{A}$ is the set of words accepted by $\mathcal{A}$.

A VPA is *deterministic* if it has a unique initial state $q_{in}$, and for each input letter and configuration there is at most one successor configuration. For deterministic VPAs (DVPAs) we denote the transition relation by $\delta$ instead of $\Delta$ and write $\delta(q,a) = (q',\gamma)$ instead of $(q,a,q',\gamma) \in \delta$ if $a \in \Sigma_c$, $\delta(q,a,\gamma) = q'$ instead of $(q,a,\gamma,q') \in \delta$ if $a \in \Sigma_r$, and $\delta(q,a) = q'$ instead of $(q,a,q') \in \delta$ if $a \in \Sigma_{\text{int}}$.

Let us stress, that during the run of any VPA $\mathcal{A}$ on a given word $u \in \Sigma^*$ the automaton $\mathcal{A}$ controls only which symbols are pushed on the stack, but not when a symbol is pushed or popped. At each step, the height of the stack is pre-determined by the prefix of $u$ read thus far. Let $\chi(a)$ be the *sign* of the symbol $a \in \Sigma$ defined as $\chi(a) = 1$ if $a \in \Sigma_c$, $\chi(a) = 0$ if $a \in \Sigma_{\text{int}}$, and $\chi(a) = -1$ if $a \in \Sigma_r$. We define the *stack height* $\text{sh}(u)$ of a word $u \in \Sigma^*$ as the sum of the signs of its constituent symbols, with $\text{sh}(\varepsilon) = 0$. Furthermore, let $\text{minsh}(u) = \min\{\text{sh}(u\restriction_n) \mid 0 \le n \le |u|\}$ and $\text{maxsh}(u) = \max\{\text{sh}(u\restriction_n) \mid 0 \le n \le |u|\}$. A word $u$ is *well matched* if $\text{sh}(u) = \text{minsh}(u) = 0$.

Given a DVPA $\mathcal{A}$ with control states $Q$ each well-matched word $u \in \Sigma^*$ induces a transformation $T_u^{\mathcal{A}} : Q \to Q$ defined as $\{(q,q') \mid (\bot,q) \xrightarrow{u} (\bot,q')\}$, which completely describes the behavior of $\mathcal{A}$ on reading $u$ in any context. The set of all transformations induced by a well-matched word is denoted $\mathcal{T}_{\text{wm}}^{\mathcal{A}}$. In the following we write just $\mathcal{T}_{\text{wm}}$ and $T_u$ when $\mathcal{A}$ is understood.

The fact that VPAs control only the content of their stack but not its height allows one to determinize every VPA as shown in [2]. In the rest of the paper we will therefore assume that all VPAs considered are deterministic.

Note that we have defined acceptance with empty stack. This implies, together with the noted implicit restriction imposed by the visibility condition, that only well-matched words can be accepted. Therefore, we are considering only languages that are subsets of the language $L_{\text{wm}} = \{u \in \Sigma^* \mid \text{sh}(u) = \text{minsh}(u) = 0\}$ of well-matched words. Observe that $L_{\text{wm}}$ is accepted by a trivial single state DVPA $\mathcal{A}_{\text{wm}}$ having a single stack symbol, hence using its stack solely as a counter to keep track of the stack height of the word being read. The following definition generalizes this concept.

**Definition 2 (Visibly counter automaton).** *A visibly counter automaton with threshold $m$ ($m$-VCA) over $\widetilde{\Sigma}$ is a tuple $\mathcal{A} = (Q, \Sigma, q_{in}, F, \delta_0, \ldots, \delta_m)$ where $Q$ is a finite set of states, $q_{in} \in Q$ is the initial state, $F \subseteq Q$ is a set of final states, $m \ge 0$ is a* threshold*, and $\delta_i : Q \times \Sigma \to Q$ is a transition function for every $i = 0, \ldots, m$.*

A configuration of $\mathcal{A}$ is a pair $(k,q)$ of counter value $k \in \mathbb{N}$ and state $q \in Q$. For $a \in \Sigma$, there is an $a$-transition from $(k,q)$ to $(k',q')$, denoted $(k,q) \xrightarrow{a} (k',q')$, if $k' = k + \chi(a)$, and $q' = \delta_k(q,a)$ if $k < m$ and $q' = \delta_m(q,a)$ if $k \ge m$.

For a finite word $u = a_0 a_1 \cdots a_n$ in $\Sigma^*$, the *run* of $\mathcal{A}$ on $u$ is the sequence $(k_0,q_0)(k_1,q_1)\cdots(k_{n+1},q_{n+1})$ of configurations, where $q_0 = q_{in}$, $k_0 = 0$, and $(k_i,q_i) \xrightarrow{a_i} (k_{i+1},q_{i+1})$ for every $0 \le i \le n$. A word $u \in \Sigma^*$ is *accepted* by a VCA $\mathcal{A}$ if the run of $\mathcal{A}$ over $u$ ends in a final configuration, that is a configuration

with counter value 0 and control state from $F$. The language $L(\mathcal{A})$ of a VCA $\mathcal{A}$ is the set of words accepted by $\mathcal{A}$.

Observe that a 0-VCA has absolutely no access to its counter, which can be perceived as an auxiliary device ensuring that only well-matched words are accepted. Other than that, a zero threshold VCA is essentially a finite automaton. Indeed, it is easy to see that a language $L$ is accepted by some 0-VCA if and only if $L = L' \cap L_{\mathrm{wm}}$ for some regular language $L'$. The next example shows that $(m+1)$-VCAs are more powerful than $m$-VCAs.

*Example 1.* Consider the languages $L_m = \{\Sigma_c^n \Sigma_r^{n-m} \Sigma_c^{l-m} \Sigma_r^l \mid m \leq l, n \in \mathbb{N}\}$ defined for each $m \in \mathbb{N}$. Each $L_m$ consists of well-matched words and is clearly accepted by an appropriate $(m+1)$-VCA. Moreover, it is easy to show that there is no $m$-VCA accepting $L_m$.

Note that we have defined VCAs to be deterministic. In Section 5 we also use nondeterministic VCAs with the natural definition. The standard subset construction that is used to determinize finite automata can also be used to determinize VCAs.

Based on the preceding definitions we can now state the problems that we address:

(1) Given a DVPA $\mathcal{A}$ and $m \in \mathbb{N}$, is there an $m$-VCA that accepts $L(\mathcal{A})$?
(2) Given a DVPA $\mathcal{A}$, is there $m \in \mathbb{N}$ and an $m$-VCA that accepts $L(\mathcal{A})$?
(3) Given an $m$-VCA $\mathcal{A}$ and $m' \in \mathbb{N}$, is there an $m'$-VCA that accepts $L(\mathcal{A})$?

Note that decidability of the two last questions implies decidability of the first one. The following example illustrates that for (2) and (3) an exponential blow-up in the size of the automaton is unavoidable.

*Example 2.* Let $\Sigma$ be the alphabet with $\Sigma_c = \{c_a, c_b\}$, $\Sigma_r = \{r_a, r_b\}$ and $\Sigma_{\mathrm{int}} = \emptyset$. For a given $m \in \mathbb{N}$ let $L_m = \{c_{x_1} \cdots c_{x_m} w r_{x_m} \cdots r_{x_1} \mid x_1, \ldots, x_m \in \{a, b\}$ and $w \in L_{\mathrm{wm}}\}$ be the set of well-matched words starting with $m$ initial calls and ending with $m$ corresponding returns. For each $m$, it is easily seen that $L_m$ is accepted by a DVPA with $\mathcal{O}(m)$ states that stores the first $m$ calls on its stack and then compares them to the $m$ final returns. Instead of storing the initial calls on the stack it is also possible to memorize them in the control state, leading to an $(m+1)$-VCA with $\mathcal{O}(2^m)$ states. A pumping argument shows that this exponential blow-up is unavoidable.

For each $m$, let $L'_m$ be the set of well-matched words that end with a sequence of $m$ returns, where the first return in this sequence is $r_a$: such a language is easily accepted by an $m$-VCA with two states. It can also be accepted by an exponentially larger 0-VCA that remembers in its control states the last $m$ returns. Again, a pumping argument shows that this exponential blow-up is unavoidable.

The rest of the paper is devoted to the proof of the following result (cf. Theorem 2 in Section 4 and Theorem 3 in Section 5).

**Theorem 1.** *Questions (1), (2) and (3) are decidable and lead to effective constructions.*

From a prior remark concerning languages accepted by 0-VCAs and from the decidability of (1) for $m = 0$ we obtain the following result.

**Corollary 1.** *It is decidable, whether a given* VPA $\mathcal{A}$ *accepts a regular restriction of the set of well-matched words, i.e. whether* $L(\mathcal{A}) = L \cap L_{\mathrm{wm}}$ *for some regular language* $L$*. When so, then a finite automaton recognizing* $L$ *can be effectively constructed.*

Concerning the restriction that we only consider languages that are subsets of $L_{\mathrm{wm}}$, note that the case where acceptance is defined only via final states can be reduced to our setting as follows. By adding a fresh symbol to $\Sigma_r$ used to close unmatched calls, one can pass to a language consisting of well-matched words only. This new language can be recognized by a VCA (accepting with final states and counter value 0) iff the original language can be recognized by a VCA accepting with final states only.

## 3 Basic Tools and Constructions

We shall now introduce the basic concepts and tools that we are using. Throughout the rest of the paper let $\mathcal{A} = (Q, \Sigma, \Gamma, q_{in}, F, \delta)$ be a given DVPA.

We use finite single-tape and multi-tape letter-to-letter automata to represent sets and relations of configurations respectively. Therefore we assume w.l.o.g. that $Q$ and $\Gamma$ are disjoint, and identify each configuration $(\sigma, q)$ of $\mathcal{A}$ with the word $\sigma q$. Letter-to-letter 2-tape finite automata accept precisely the length-preserving rational relations. Basic results on length-preserving and synchronized rational relations can be found in [9]. Letter-to-letter multi-tape finite automata can be seen as classical single-tape finite automata over the product alphabet. Hence, all classical constructions and results of automata theory apply. Below we often use this fact without explicit reference. In various estimates we use the binary function $\exp(k, n)$ denoting a tower of exponentials of height $k$ defined inductively by letting $\exp(0, n) = n$ and $\exp(k + 1, n) = 2^{\exp(k,n)}$ for all $k$ and $n$.

When considering language acceptance only those configurations of $\mathcal{A}$ are of concern that are *reachable* from the initial configuration. Accordingly, in our constructions we restrict our attention to the set $V_{\mathcal{A}}$ of configurations of $\mathcal{A}$ reachable from the initial configuration. The fact, first observed by Büchi [7], that $V_{\mathcal{A}}$ is regular is therefore essential. Moreover, an obvious adaptation of the construction of [6] (see also [8]) shows that a non-deterministic finite automaton recognizing $V_{\mathcal{A}}$ with $\mathcal{O}(|Q|)$ states can be constructed in polynomial time. From now on by configuration we always mean reachable configuration, unless explicitly stated otherwise.

First we define equivalence (denoted $\sim$) of configurations of $\mathcal{A}$ in a standard way according to the languages they accept, and observe a necessary condition

(2') for a positive answer for question (2). Next we show that $\sim$, when considered as a binary relation on words describing the configurations, can be accepted by a letter-to-letter two-tape automaton. This allows us not only to decide (2') but also to prove its sufficiency.

The *configuration graph* of $\mathcal{A}$ is the edge-labelled graph $G_{\mathcal{A}} = (V_{\mathcal{A}}, E_{\mathcal{A}})$, where $V_{\mathcal{A}}$ is, as above, the set of reachable configurations of $\mathcal{A}$ and $E_{\mathcal{A}}$ is the set that contains all triples of the form $((\sigma, q), a, (\sigma', q'))$ such that $(\sigma, q), (\sigma', q') \in V_{\mathcal{A}}$, $a \in \Sigma$, and $(\sigma, q) \xrightarrow{a} (\sigma', q')$. Below we often suppress the index $\mathcal{A}$.

**Definition 3 (Equivalence of configurations).** *Two configurations $\sigma q, \sigma' q'$ of $\mathcal{A}$ are equivalent, in symbols $\sigma q \sim \sigma' q'$, if $|\sigma| = |\sigma'|$ and for every word $u \in \Sigma^*$ there is an accepting run of $\mathcal{A}$ labelled by $u$ from $(\sigma, q)$ to a final configuration iff there is one from $(\sigma', q')$.*

Because $\mathcal{A}$ is deterministic $\sim$ is in fact a congruence with respect to the transition relations $\xrightarrow{a}$ $(a \in \Sigma)$ restricted to the set of reachable configurations. This allows us to define the quotient graph $G/_\sim$ as follows.

**Definition 4 (Quotient of the configuration graph).** *We define the quotient of the configuration graph $G = (V, E)$ with respect to the congruence $\sim$ as $G/_\sim = (V/_\sim, E/_\sim)$, where $V/_\sim$ consists of equivalence classes of $V$ under $\sim$ and for all $C_1, C_2 \in V/_\sim$, and for any letter $a \in \Sigma$, $(C_1, a, C_2) \in E/_\sim$ if and only if there are some (equivalently for all) $v_1 \in C_1$ and $v_2 \in C_2$ such that $(v_1, a, v_2) \in E$.*

Note that by definition $\sigma q \sim \sigma' q'$ implies that $|\sigma| = |\sigma'|$. In other words, $\sim$ refines the equivalence defined according to stack height, i.e. $\sim$ is length-preserving. If we denote by $V|_n$ the set of reachable configurations that contain $n$ stack symbols, i.e. $V|_n = V \cap (\bot \cdot \Gamma^n Q)$, then $\sim$ induces a certain number of equivalence classes on each set $V|_n$. In case $\mathcal{A}$ is equivalent to some $m$-VCA, this number of equivalence classes must be bounded by a bound independent of $n$, because configurations of a VCA that have the same counter value can only be distinguished by finitely many control states.

**Proposition 1.** *The following is necessary for (2) to have a positive answer.*

(2') *$\exists K \, \forall n$ $V|_n$ is partitioned into at most $K$ $\sim$-equivalence classes.*

It is, however, not immediate that the above condition is also sufficient. Both to show equivalence of (2) and (2') and to prove their decidability the following observation is crucial.

**Lemma 1.** *One can effectively construct a letter-to-letter 2-tape automaton $\mathcal{A}_\sim$ having at most $2^{\mathcal{O}(|Q|^2)}$ states and recognizing $\sim$.*

This lemma can be shown by noting that an automaton can guess a separating word for two configurations of the same length $n$. Such a word consists of $n$ returns interleaved with well-matched words. As not the particular well-matched

words $u$ but only the transformations $T_u$ (from the finite set $\mathcal{T}_{\mathrm{wm}}$) induced by them are interesting, a finite automaton can check whether two configurations are *not* equivalent. Then one can conclude using the closure properties of finite letter-to-letter 2-tape automata.

We are interested in the number of equivalence classes of $\sim$ for each stack height and therefore want to elect representatives for these classes. For this purpose we fix some linear ordering of the symbols of $\Gamma$ and $Q$, thus determining the lexicographic ordering $<_{\mathrm{lex}}$ of all configurations. Note that $<_{\mathrm{lex}}$ is synchronized rational, hence, its restriction to words of equal length is recognized by a letter-to-letter automaton. Using the automata recognizing $<_{\mathrm{lex}}$, $\sim$, and $V$ we can further construct an automaton recognizing the set $\mathrm{Rep} = \{\sigma q \in V \mid \neg\exists \sigma' q' \in V(\sigma q \sim \sigma' q' \wedge \sigma' q' <_{\mathrm{lex}} \sigma q)\}$ of lexicographically smallest representatives of each $\sim$-class as follows: One can construct a letter-to-letter automaton recognizing pairs of equivalent reachable configurations, such that the first component precedes the second one in the lexicographic ordering. After projection onto the second component, determinization, and complementation (with respect to $V$) one obtains a deterministic automaton $\mathcal{A}_{\mathrm{Rep}}$ recognizing Rep. The largest one of the components is the automaton $\mathcal{A}_\sim$ and the costliest operation is, of course, determinization potentially causing an exponential increase in the number of states. Thus, we obtain $\exp(2, \mathcal{O}(|Q|^2))$ as an upper bound on the size of $\mathcal{A}_{\mathrm{Rep}}$.

We now observe that (2') is equivalent to the slenderness of Rep. Following [13] and [4] we say that a language $L \subseteq \Gamma^*$ is *slender* if there is a constant $K$ such that $|L \cap \Gamma^n| \leq K$ for all $n \in \mathbb{N}$, in which case we may also say that $L$ is *$K$-thin*. Let us therefore introduce the notation $\mathrm{Rep}_n = \mathrm{Rep} \cap V|_n$. Analogously, we say that the graph $G/_\sim$ is *slender* if there is a constant $K$ such that $|(V|_n)/_\sim| \leq K$ for all $n \geq 0$. Relying on results of [4] and [13] we immediately obtain the following.

**Proposition 2.** *Condition (2') is decidable, moreover, if* Rep *is $K$-thin, then* $K \leq |\Gamma|^{N-2} \cdot |Q| = \exp(3, \mathcal{O}(|Q|^2))$, *where $N$ is the number of states of the minimal deterministic automaton recognizing* Rep.

Let us assume that $G/_\sim$ is slender. We identify each of its nodes $C$ with the pair $(\mathrm{sh}(C), \mathrm{index}(C)) \in \mathbb{N} \times \{1, \cdots, K\}$, where the stack height of a class $C$ is the stack height of any (hence all) of the configurations belonging to $C$ and the index of $C$ is the position of its representative $w \in C \cap \mathrm{Rep}$ with respect to $<_{\mathrm{lex}}$ among $\mathrm{Rep}_{\mathrm{sh}(C)}$. In the next section we will show that in case $G/_\sim$ is slender it is (in the above representation) actually the configuration graph of a VCA. The following lemma constitutes an important step in the proof of this result.

**Lemma 2.** *Assuming condition (2') holds with slenderness bound $K$ we can effectively construct an automaton $\mathcal{C}_\sim$ reading stack contents and whose states $q_\sim$ encode mappings $\rho_{q_\sim} : Q \to \{0, \ldots, K\}$ where $K$ is the slenderness index of $G/_\sim$. After reading a stack content $\sigma$ the automaton $\mathcal{C}_\sim$ is in a state $q_\sim$ such that* $\mathrm{index}((\sigma, q)) = \rho_{q_\sim}(q)$ *for all $q \in Q$. Moreover* $\exp(5, \mathcal{O}(|Q|^2))$ *is an upper bound on the number of states of $\mathcal{C}_\sim$.*

## 4 From Pushdown to Counter Automata: Decidability of Question (2)

In this section we prove that slenderness is actually a sufficient condition for (2) to hold. As it is also necessary and decidable, it shows the decidability of question (2). Effectiveness follows from the proof.

Assume that $\mathcal{A}$ is a DVPA (with the usual components) such that $G_{\mathcal{A}}/_{\sim}$ is slender, and let $K$ be a slenderness bound, i.e. there are at most $K$ classes on each level of $G_{\mathcal{A}}/_{\sim}$.

The proof and the construction are split in two steps. First we show that $G_{\mathcal{A}}/_{\sim}$ can be effectively described by an ultimately periodic word. Then, in the second step, it easily follows that $\mathcal{A}$ is equivalent to an $m$-VCA with $m$ being the offset of the ultimately periodic word.

The infinite word describing $G_{\mathcal{A}}/_{\sim}$ is such that the $n$th letter codes the edges of $E_{\mathcal{A}}/_{\sim}$ that leave the vertices from the $n$th level, i.e., the outgoing edges from the vertex set $\{(n,i) \mid i \in \{1,\dots,K\}\}$. These edges are fully described by a (partial) mapping assigning to each pair $(i,a)$ of class index and input letter the index of the class reached from class $i$ on level $n$ when reading an $a$. If there are less than $i$ classes on level $n$, then the value for $(i,a)$ is undefined.

More formally, the description $\tau_n : \{1,\dots,K\} \times \Sigma \to \{1,\dots,K\}$ of the $n$th level of $G_{\mathcal{A}}/_{\sim}$ is defined by $\tau_n(i,a) = j$ iff $((n,i),a,(n+\chi(a),j)) \in E_{\mathcal{A}}/_{\sim}$ and $\tau_n(i,a)$ is undefined if $(n,i)$ is not a vertex of $G_{\mathcal{A}}/_{\sim}$.

The sequence $\alpha := \tau_0\tau_1\dots$ completely describes $G_{\mathcal{A}}/_{\sim}$. Using the automaton $\mathcal{C}_{\sim}$ (cf. Lemma 2) it is possible to construct a finite state machine that outputs this sequence. This implies the main technical result of this section, namely that $\alpha$ is ultimately periodic.

**Lemma 3.** *The description $\alpha = \tau_0\tau_1\tau_2,\dots$ of $G_{\mathcal{A}}/_{\sim}$ is an ultimately periodic sequence that can be constructed effectively.*

As $\alpha$ is ultimately periodic there are numbers $m$ and $k$ such that $\alpha = \tau_0 \cdots \tau_{m-1}(\tau_m \cdots \tau_{m+k-1})^{\omega}$. We call $m$ the offset and $k$ the period of $\alpha$. It is not difficult to verify that a VCA that knows whether it is in the offset part of $\alpha$ (using its threshold) or in the periodic part (using a modulo $k$ counter to keep track of the position) can simulate $\mathcal{A}$. This is established in the following proposition.

**Proposition 3.** *If the description $\alpha = \tau_0\tau_1\dots$ of $G_{\mathcal{A}}/_{\sim}$ is ultimately periodic with offset $m$ and period $k$, then one can build an $m$-VCA $\mathcal{B}$ such that $L(\mathcal{A}) = L(\mathcal{B})$.*

Combining Propositions 1, 2, and 3 we get the following theorem answering question (2) from Section 3.

**Theorem 2.** *It is decidable if for a given* VPA *there exists an equivalent* VCA*. If such a* VCA *exists it can be effectively constructed and has $\mathcal{O}((|\Gamma|\cdot|Q_{\mathcal{C}_{\sim}}|\cdot K)^{2K}\cdot K)$ states and its threshold is bounded by $\mathcal{O}((|\Gamma|\cdot|Q_{\mathcal{C}_{\sim}}|\cdot K)^{2K})$.*

## 5 Reducing the Threshold: Decidability of Question (3)

In all this section, we assume that $\mathcal{A} = (Q, \Sigma, q_{in}, F, \delta_0, \ldots, \delta_m)$ is an $m$-VCA for some threshold $m$. Given $m' < m$, we want to decide whether there is an $m'$-VCA $\mathcal{B}$ such that $L(\mathcal{A}) = L(\mathcal{B})$. If such a $\mathcal{B}$ exists, we want to provide an effective construction of it.

The decision procedure that we present consists of two steps. First, we build an $m'$-VCA $\mathcal{A}'$ and show that if $\mathcal{A}$ is equivalent to some $m'$-VCA then $L(\mathcal{A}) = L(\mathcal{A}')$. Intuitively, $\mathcal{A}'$ is a canonical candidate to be equivalent to $\mathcal{A}$. Then, we have to check whether $L(\mathcal{A}) = L(\mathcal{A}')$ holds, which is known to be decidable [2].

As the technical details of the construction of $\mathcal{A}'$ and the correctness proofs are quite involved we restrict ourselves in the following to an explanation of the underlying ideas.

The difference between $\mathcal{A}$ and an $m'$-VCA is that for a word $w$ of stack height $h$ with $m' \leq h < m$ the automaton $\mathcal{A}$ exactly knows the current stack height because it uses $\delta_h$ to compute the next configuration, whereas the $m'$-VCA only knows that the stack height is at least $m'$. Such a situation is depicted in Figure 1 (where for now we ignore all annotations except $m$ and $m'$).

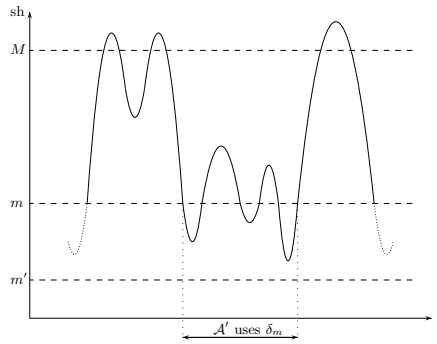


**Fig. 1.** A critical situation when simulating threshold $m$ by threshold $m'$

The main idea is to show that, under the assumption that $\mathcal{A}$ is indeed equivalent to some $m'$-VCA, this additional information gained by $\mathcal{A}$ when using $\delta_h$ is not used (under certain conditions) so that instead of using $\delta_h$ to compute the next configuration one could also have used $\delta_m$. The conditions under which it is possible to use $\delta_m$ instead of the correct transition function $\delta_h$ are also illustrated in Figure 1. If the input exceeds a certain stack height (denoted by $M$, a parameter depending on the size of $\mathcal{A}$), then comes back into the area between $m$ and $m'$, and then again goes beyond $M$, then one can also use $\delta_m$ when the stack height is between $m$ and $m'$, without changing the acceptance behavior of $\mathcal{A}$. The condition on the stack height is needed for the correctness proof to be

able to apply pumping arguments without changing the transformation on the state space that is induced by the input.

This allows to construct a nondeterministic $m'$-VCA $\mathcal{A}'$ that maintains in its state space a counter up to $M$ that is updated according to the stack height. As long as the stack height stays below $M$, $\mathcal{A}'$ can exactly simulate $\mathcal{A}$. If the stack height exceeds $M$, $\mathcal{A}'$ starts using $\delta_m$ for its transitions, and it guesses the points where it can switch back to exact simulation of $\mathcal{A}$. These are the points where the stack height falls below $M$ and reaches a value less than $m'$ before exceeding $M$ again. These guesses can be verified as correct by $\mathcal{A}'$ at the moment where the stack height goes below $m'$ (because then it can compare the counter value maintained in the state space with the real stack height).

As nondeterministic VCAs can be determinized as explained in Section 2, we obtain the following lemma.

**Lemma 4.** *From $\mathcal{A}$ one can construct an $m'$-VCA $\mathcal{A}'$ such that $L(\mathcal{A}) \neq L(\mathcal{A}')$ implies that there is no $m'$-VCA that is equivalent to $\mathcal{A}$.*

Finally, using the fact that equivalence for VPAs (hence for VCAs) is decidable, we obtain the following result answering question (3) from Section 2.

**Theorem 3.** *It is decidable, given an $m$-VCA $\mathcal{A}$ and $m' < m$, whether $\mathcal{A}$ is equivalent to some $m'$-VCA, in which case such an $m'$-VCA $\mathcal{A}'$ can be constructed effectively.*

Concerning complexity, we note that the number of states of (the nondeterministic) $\mathcal{A}'$ is in $\mathcal{O}(|Q|^{2|Q|})$ (stemming from the definition of $M$). To check equivalence of $\mathcal{A}'$ with $\mathcal{A}$, one determinizes $\mathcal{A}'$ (exponential blow-up) and transforms it into a VPA: hence the complexity is doubly exponential in $|Q|$.

## 6 Conclusion

We have introduced the notion of visibly counter automaton as a direct adaption of standard one-counter automata to the framework of visibly pushdown automata. We have shown that it is decidable for a given VPA if it is equivalent to some VCA, even if we allow the counter to be tested up to a certain threshold, and provided an algorithm to construct such a counter automaton if it exists. This solves a special case of a problem that was posed in [16] for general deterministic pushdown automata.

A drawback of the presented proof is the high complexity of the resulting construction. The upper bound on the size of the VCA that we construct is 6-fold exponential in the size of the given visibly pushdown automaton, whereas the lower bound (Example 2) that we can prove is only singly exponential.

As a corollary of our main result we obtain that it is decidable for a given VPA whether it accepts a regular restriction of the set of well-matched words, i.e. whether its language is of the form $L \cap L_{\mathrm{wm}}$ for a regular language $L$. To answer the question from [14] one would have to solve the corresponding problem

with $L_{\text{wm}}$ replaced by another language: If we consider inputs as obtained when coding trees by words using opening and closing tags for the subtrees, then $L_{\text{wm}}$ describes those words for which each opening tag is closed by *some* closing tag. To be a valid coding of a tree (in the sense of [14]) each opening tag has to be closed by a unique corresponding tag, i.e. the word has to be *strongly* well matched (see also [5]). Hence, to decide whether membership for a set $L(\mathcal{A})$ of coded trees can be tested by a finite automaton under the assumption that the input is well formed in the above sense, one has to check if $L(\mathcal{A})$ is of the form $L \cap L_{\text{swm}}$ for some regular language $L$ and for $L_{\text{swm}}$ being the set of strongly well-matched words.

Currently, we are working on the following generalization of these problems: Given two VPAs $\mathcal{A}$ and $\mathcal{B}$, is the language accepted by $\mathcal{A}$ a regular restriction of the language accepted by $\mathcal{B}$, i.e. $L(\mathcal{A}) = L \cap L(\mathcal{B})$ for some regular language $L$?

## References


1. R. Alur, K. Etessami, and P. Madhusudan. A temporal logic of nested calls and returns. In *TACAS'04*, *LNCS* 2988, 467–481. Springer, 2004.
2. R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proceedings of STOC'04*, pages 202–211. ACM, 2004.
3. R. Alur, P. Madhusudan, V. Kumar, and M. Viswanatha. Congruences for visibly pushdown languages. In *ICALP'05*, *LNCS* 3580, pages 1102–1114, 2005.
4. M. Andraşiu, G. Păun, J. Dassow, and A. Salomaa. Language-theoretic problems arising from Richelieu cryptosystems. *Theor. Comp. Sci.*, 116(2):339–357, 1993.
5. Jean Berstel and Luc Boasson. Formal properties of XML grammars and languages. *Acta Informatica*, 38(9):649–671, 2002.
6. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR'97*, *LNCS* 1243, pages 135–150. Springer, 1997.
7. J. R. Büchi. Regular canonical systems. *Archiv für Mathematische Grundlagenforschung*, 6:91–111, 1964.
8. J. Esparza, D. Hansel, P. Rossmanith, and S Schwoon. Efficient algorithms for model checking pushdown systems. In *CAV'00*, *LNCS* 1855, pp. 232–247. Springer.
9. C. Frougny and J. Sakarovitch. Synchronized rational relations of finite and infinite words. *Theoretical Computer Science*, 108(1):45–82, 1993.
10. J. E. Hopcroft and J. D. Ullman. *Formal Languages and their Relation to Automata*. Addison-Wesley, 1969.
11. C. Löding, P. Madhusudan, and O. Serre. Visibly pushdown games. In *FST&TCS'04*, *LNCS* 3328, pages 408–420. Springer, 2004.
12. C. Pitcher. Visibly pushdown expression effects for XML stream processing. In *Programming Language Technologies for XML, PLAN-X'05*, pages 5–19, 2005.
13. G. Păun and A. Salomaa. Thin and slender languages. *Discrete Applied Mathematics*, 61(3):257–270, 1995.
14. L. Segoufin and V. Vianu. Validating streaming XML documents. In *Proceedings of PODS'02*, pages 53–64. ACM, 2002.
15. R. E. Stearns. A regularity test for pushdown machines. *Information and Control*, 11(3):323–340, 1967.
16. L. G. Valiant. Regularity and related problems for deterministic pushdown automata. *Journal of the ACM*, 22(1):1–10, 1975.