Congruences for Visibly Pushdown Languages*

Rajeev Alur[1], Viraj Kumar[2], P. Madhusudan[2], and Mahesh Viswanathan[2]

[1] University of Pennsylvania, Philadelphia, PA, USA,
alur@cis.upenn.edu
[2] University of Illinois at Urbana-Champaign, Urbana, IL, USA,
{kumar,madhu,vmahesh}@cs.uiuc.edu

Abstract. We study congruences on words in order to characterize the class of visibly pushdown languages (VPL), a subclass of context-free languages. For any language L, we define a natural congruence on words that resembles the syntactic congruence for regular languages, such that this congruence is of finite index if, and only if, L is a VPL. We then study the problem of finding canonical minimal deterministic automata for VPLs. Though VPLs in general do not have unique minimal automata, we consider a subclass of VPAs called k-module single-entry VPAs that correspond to programs with recursive procedures without input parameters, and show that the class of well-matched VPLs do indeed have unique minimal k-module single-entry automata. We also give a polynomial time algorithm that minimizes such k-module single-entry VPAs.

1 Introduction

The class of *visibly pushdown languages* (VPL), introduced in [1], is a subclass of context-free languages accepted by pushdown automata in which the input letter determines the type of operation permitted on the stack. Visibly pushdown languages are closed under all boolean operations, and problems such as inclusion, that are undecidable for context-free languages, are decidable for VPL. VPLs are relevant to several applications that use context-free languages such as the model-checking of software programs using their pushdown models [1–3]. Recent work has shown applications in other contexts: in modeling semantics of effects in processing XML streams [4], in game semantics for programming languages [5], and in identifying larger classes of pushdown specifications that admit decidable problems for infinite games on pushdown graphs [6].

Our main result in this paper is a characterization of the class of VPLs in terms of congruences on strings. It is well known that the *syntactic congruence*, which is defined as $w_1 \approx w_2$ when for every u, v, $uw_1v \in L$ if and only if $uw_2v \in L$, has finite index precisely for languages L that are regular. Our central thesis is that for VPLs L, when we restrict our attention to *well-matched* words w_1 and w_2 (i.e., words where every push transition has a corresponding pop

* This research was partially supported by ARO URI award DAAD19-01-1-0473, NSF awards CCR-0306382 and CCF 04-29639, and DARPA/AFOSR MURI award F49620-02-1-0325.

transition and vice versa), the syntactic congruence has finite index. Moreover, for languages consisting only of well-matched words, if the syntactic congruence on well-matched words has finite index then the language is a VPL. For languages containing strings that are not well-matched, we need some additional conditions only because no congruence on well-matched words can *saturate* such a language. Our characterization of VPLs is a natural generalization of the Myhill-Nerode theorem for regular languages— when restricted to languages that do not require any push or pop operations, our congruence coincides with the right congruence defined by Myhill and Nerode [7, 8].

One important consequence of the congruence based characterization of regular (word) languages and regular tree languages is that for any regular language there is a unique minimum state deterministic automaton recognizing the language, which can also be constructed efficiently [8, 9]. For VPLs, however, we show that in general there is no unique minimum state recognizer. Thus, while our characterization yields the construction of a *canonical* deterministic acceptor for VPLs, it may not in general be minimal. An implicit consequence of the results in [1] is that VPLs have canonical deterministic pushdown automata. It is shown in [1] that with any language L, a language of trees called *stack trees*, can be associated such that L is a VPL exactly when the corresponding set of stack trees form a regular tree language. The unique minimal bottom-up tree automaton accepting the language of stack trees can then be translated to a canonical deterministic visibly pushdown automaton. However, since bottom-up tree automata can only be translated into deterministic pushdown automata with exponentially more states, the implicit construction in [1] does not result in necessarily small deterministic VPAs.

Visibly pushdown automata are a natural model for programs with recursive procedure calls and finite data types. Such programs are called Boolean programs in the literature on software model checking [3]. When modeling a program as a visibly pushdown automaton, the natural structure the model assumes is one where the machine's states are partitioned into k modules, one for each procedure in the program. As one expects, these modules are such that from a state in a module, a sequence of calls and returns to other modules results in a state of the same module. Moreover, if the programs modeled are such that the calls to modules have no input parameters (or if a function is modeled separately for each possible value of its input parameters), then the visibly pushdown automaton assumes additional structure, namely that every call results in going to a unique state in the module corresponding to the call. We call such structured VPAs k-module single-entry VPAs (k-SEVPAs). They correspond roughly to the model of *recursive state machines* with a single entry per module [10].

Though visibly pushdown languages in general do not have unique minimum-state recognizers, partitioning the calls into the modules they correspond to fixes enough additional structure that there is a minimum-state k-SEVPA that respects the partition and accepts the language. More precisely, we show that for any partition of the call-alphabet into k-sets, there is a unique minimum-state k-SEVPA accepting any well-matched VPL L. If $k = 0$ (that is, there

are no calls), the result is equivalent to the Myhill-Nerode theorem for regular languages. The characterization of this unique minimal k-SEVPA is done via a set of $k+1$ congruences on words. We also present an algorithm which, given any deterministic k-SEVPA accepting a well-matched language, minimizes it in $O(n^3)$ time, where n is the size of the original machine.

The rest of the paper is organized as follows. We first recall the definitions of visibly pushdown languages and visibly pushdown automata in Section 2. Our main result characterizing visibly pushdown languages in terms of language theoretic congruences is presented in Section 3. We also show that VPLs, in general, do not have unique minimum state deterministic recognizers. In Section 4, we define the notion of how partitions on calls define k-module single-entry VPAs and prove that every (well-matched) VPL has a unique minimal k-SEVPA accepting it. We also present an example of a family of languages for which the minimal 1-module machine is super polynomial in the size of the smallest visibly pushdown automaton recognizing it. Conclusions and open problems are presented in Section 5.

2 Preliminaries

In this section, we recall definitions of *visibly pushdown automata* and *visibly pushdown languages*, and introduce some notation that we will use in the rest of the paper.

Pushdown Alphabet. A *pushdown alphabet* is a tuple $\widehat{\Sigma} = (\Sigma_{\text{call}}, \Sigma_{\text{ret}}, \Sigma_{\text{int}})$ that comprises three *disjoint* finite alphabets— Σ_{call} is a finite set of *calls*, Σ_{ret} is a finite set of *returns*, and Σ_{int} is a finite set of *internal actions*. For any such $\widehat{\Sigma}$, let $\Sigma = \Sigma_{\text{call}} \cup \Sigma_{\text{ret}} \cup \Sigma_{\text{int}}$. In the paper we will use $u, v, u_1, \ldots$ for strings in Σ^*, $c, c_1, c_i, \ldots$ for elements of Σ_{call}, $r, r_1, r_i, \ldots$ for elements of Σ_{ret}, and $i, i_1, i_j, \ldots$ for elements of Σ_{int}.

Visibly Pushdown Automata. For visibly pushdown automata, unlike the case of pushdown automata, it turns out that deterministic VPAs are as powerful as a non-deterministic VPAs [1]. In light of this, we will only consider deterministic VPAs. A *visibly pushdown automaton* (VPA) on finite strings over $\widehat{\Sigma} = (\Sigma_{\text{call}}, \Sigma_{\text{ret}}, \Sigma_{\text{int}})$ is a tuple $M = (Q, q_0, \Gamma, \delta, Q_F)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, Γ is a finite stack alphabet that contains a special bottom-of-stack symbol $\perp$, $\delta = \delta_{\text{call}} \cup \delta_{\text{ret}} \cup \delta_{\text{int}}$ is the transition function, where $\delta_{\text{call}} : Q \times \Sigma_{\text{call}} \to Q \times (\Gamma \setminus \{\perp\})$, $\delta_{\text{ret}} : Q \times \Sigma_{\text{ret}} \times \Gamma \to Q$, and $\delta_{\text{int}} : Q \times \Sigma_{\text{int}} \to Q$, and $Q_F \subseteq Q$ is a set of final states.

If $\delta_{\text{call}}(q, c) = (q', \gamma)$, where $c \in \Sigma_{\text{call}}$ and $\gamma \neq \perp$, there is a *push-transition* from q on input c where on reading c, γ is pushed onto the stack and the control changes from state q to q'; we denote such a transition by $q \xrightarrow{c/\gamma} q'$. Similarly, if $\delta_{\text{ret}}(q, r, \gamma) = q'$, there is a *pop-transition* from q on input r where γ is read from the top of the stack and popped (if the top of the stack is $\perp$, then it is read but not popped), and the control changes from q to q'; we denote such a transition

by $q \xrightarrow{r/\gamma} q'$. If $\delta_{\text{int}}(q, i) = q'$, there is an *internal-transition* from q on input i where on reading i, the state changes from q to q'; we denote such a transition by $q \xrightarrow{i} q'$. Note that there are no stack operations on internal transitions.

Acceptance. A *stack* is a non-empty finite sequence over Γ ending in the bottom-of-stack symbol $\perp$. The set of all stacks is denoted as $St = (\Gamma \setminus \{\perp\})^* \cdot \{\perp\}$. A *configuration* is a pair (q, σ) such that q is a state and $\sigma \in St$. The transition function of a VPA can be used to define how the configuration of the machine changes in a single step: we say $\delta((q, \sigma), a) = (q', a')$ [3] if one of the following holds:

1. If $a \in \Sigma_{\text{call}}$ then there exists $\gamma \in \Gamma$ such that $\delta_{\text{call}}(q, a) = (q', \gamma)$ and $\sigma' = \gamma \cdot \sigma$
2. If $a \in \Sigma_{\text{ret}}$, then there exists $\gamma \in \Gamma$ such that $\delta_{\text{ret}}(q, a, \gamma) = q'$ and either $\gamma \neq \perp$ and $\sigma = \gamma \cdot \sigma'$, or $\gamma = \perp$ and $\sigma = \sigma' = \perp$
3. If $a \in \Sigma_{\text{int}}$ is an internal action, then $\delta_{\text{int}}(q, a) = q'$ and $\sigma' = \sigma$

The transitive closure of the single-step transition function, which we also denote by δ, can be easily defined in the standard inductive manner. For a stack $\sigma \in St$, we define the function $\delta_\sigma : Q \times \Sigma^* \to Q$ as $\delta_\sigma(q, u) = q'$ whenever $\delta((q, \sigma), u) = (q', \sigma')$ for some $\sigma' \in St$.

A string $u \in \Sigma^*$ is *accepted* by VPA M if $\delta_\perp(q_0, u) \in Q_F$. The *language* of M, $L(M)$, is the set of strings accepted by M.

Visibly Pushdown Languages. A language over finite strings $L \subseteq \Sigma^*$ is a *visibly pushdown language* (VPL) with respect to $\widehat{\Sigma}$ (a $\widehat{\Sigma}$-VPL) if there is a VPA M over $\widehat{\Sigma}$ such that $L(M) = L$.

Matched calls and returns. Let $MR(\widehat{\Sigma})$ denote the set of all strings where every return has a matched call before it, i.e. $u \in MR(\widehat{\Sigma})$ if for every prefix u' of u, the number of return symbols in u' is at most the number of call symbols in u'. Similarly, let $MC(\widehat{\Sigma})$ denote the set of all strings where every call has a matching return after it, i.e. $u \in MC(\widehat{\Sigma})$ if for every suffix u' of u, the number of call symbols in u' is at most the number of return symbols in u'. The set of *well-matched* strings over $\widehat{\Sigma}$ is $WM(\widehat{\Sigma}) = MR(\widehat{\Sigma}) \cap MC(\widehat{\Sigma})$.

A $\widehat{\Sigma}$-VPL L is said to be *well-matched* if $L \subseteq WM(\widehat{\Sigma})$.

Remark 1. For every $w \in WM(\widehat{\Sigma})$, there is a unique matching between call and return symbols such that every call-symbol always precedes its matching return-symbol and the substring w' between a matching pair of call and return symbols is a well-matched string.

[3] We abuse notation and use δ for both the transition function of the automaton and the single step transition function on configurations.

3 Congruence based characterization of VPLs

In this section we present a congruence based characterization of when a language over $\widehat{\Sigma}$ is a visibly pushdown language. Before presenting the characterization for general VPLs, we first consider the case of VPLs that have only well-matched words.

3.1 Well-matched visibly pushdown languages

For a language L over the pushdown alphabet $\widehat{\Sigma} = (\Sigma_{\text{call}}, \Sigma_{\text{ret}}, \Sigma_{\text{int}})$, consider the following congruence on well-matched words:

$$w_1 \approx w_2 \text{ iff } \forall u, v \in \Sigma^*,\ uw_1v \in L \text{ iff } uw_2v \in L$$

Recall that this is the standard syntactic congruence restricted to well-matched words over $\widehat{\Sigma}$. For example, if $\widehat{\Sigma} = (\{c\}, \{r\}, \emptyset)$ and $L = \{c^n.r^n \mid n \geq 0\}$, then there are only two equivalence classes that $\approx$ defines: $\{c^nr^n \mid n \geq 0\}$ and the complement of this set with respect to $WM(\widehat{\Sigma})$.

Analogous to the case of regular languages, the finiteness of the number of equivalence classes of the syntactic congruence (on well-matched words) provides a precise characterization of *well-matched* VPLs.

Theorem 1. *L is a well-matched $\widehat{\Sigma}$-VPL iff $\approx$ (as defined above) has finitely many equivalence classes.*

Proof. Suppose L is a $\widehat{\Sigma}$-VPL and $M = (Q, q_0, \Gamma, \delta, Q_F)$ is a VPA over $\widehat{\Sigma}$ with (unique) initial state q_0 such that $L(M) = L$. Every well-matched string w defines a function $f_w : Q \to Q$ as follows: $f_w(q) = \delta_\perp(q, w)$. Define the following equivalence on well-matched strings:

$$w_1 \approx_M w_2 \text{ iff } f_{w_1} = f_{w_2}$$

Observe that $\approx_M$ has finitely many equivalence classes (bounded by $|Q|^{|Q|}$). We will show that $\approx_M$ is a refinement of $\approx$, thus establishing that $\approx$ is also of finite index. Consider $w_1 \approx_M w_2$. Then for any $u, v \in \Sigma^*$, we know

$$\begin{aligned}\delta((q_0, \perp), uw_1v) &= \delta(\delta(\delta((q_0, \perp), u), w_1), v)\\ &= \delta(\delta(\delta((q_0, \perp), u), w_2), v) \text{ since } f_{w_1} = f_{w_2}\\ &= \delta((q_0, \perp), uw_2v)\end{aligned}$$

Hence $uw_1v \in L$ iff $uw_2v \in L$, and so $w_1 \approx w_2$. Thus $\approx_M$ is a refinement of $\approx$. Observe that this proof does not rely on L being a well-matched language.

To prove the converse, consider a language L such that $\approx$ is of finite index. We construct a deterministic (but incomplete[4]) VPA that recognizes L and whose

[4] A VPA is incomplete if the transition function δ is not total. An incomplete VPA can be easily modified to yield a VPA with at most one extra "dead" state to which all undefined transitions go.

states are the equivalence classes of $\approx$. Consider a string with no unmatched returns $u = w_1 c_1 w_2 c_2 \cdots c_k w_{k+1} \in MR(\widehat{\Sigma})$, where $c_1, \ldots c_k$ are the unmatched call symbols in u, and $w_1, \ldots w_{k+1}$ are well-matched strings between the unmatched call symbols. The automaton we construct will maintain the following invariant: after reading the string $u \in MR(\widehat{\Sigma})$, the state of the machine will be $[w_{k+1}]_\approx$ and the stack will be $([w_k]_\approx, c_k)([w_{k-1}]_\approx, c_{k-1}) \cdots ([w_1]_\approx, c_1)\bot$.

The formal construction of VPA $M = (Q, q_0, \Gamma, \delta, Q_F)$ is as follows: $Q = \{[w]_\approx | w \in WM(\widehat{\Sigma})\}$, $q_0 = [\epsilon]_\approx$, $\Gamma = \{\bot\} \cup (Q \times \Sigma_{\text{call}})$, and $Q_F = \{[w]_\approx | w \in L\}$. The transition function δ is defined as follows.

- $[w]_\approx \xrightarrow{i} [wi]_\approx$ for every $i \in \Sigma_{\text{int}}$
- $[w]_\approx \xrightarrow{c/([w]_\approx, c)} [\epsilon]_\approx$ for every $c \in \Sigma_{\text{call}}$
- $[w]_\approx \xrightarrow{r/([w']_\approx, c)} [w'cwr]_\approx$ for every $r \in \Sigma_{\text{ret}}$

The above machine has no pop transitions when $\bot$ is the only symbol on the stack. Observe that the definitions of Q_F and δ are sound because $\approx$ saturates L [5] and $\approx$ is a congruence with respect to well-matched words. Further, it is easy to verify that the above invariant is maintained. Thus, after reading a *well-matched* word w, the automaton will be in the state $[w]_\approx$ and hence $L = L(M) \cap WM(\widehat{\Sigma})$. Since $WM(\widehat{\Sigma})$ is a VPL, and VPLs are closed under intersection, the result follows. □

3.2 General visibly pushdown languages

For visibly pushdown languages that are not necessarily well-matched, $\approx$ being of finite index is not sufficient. This is because $\approx$ is no longer a congruence that saturates the VPL. We need to define two additional congruences on strings—one that will capture the behavior of a state when the stack only has $\bot$, and one that will capture the behavior when the stack has more than one element. The reason we need to distinguish the cases of the stack having only $\bot$ and that of the stack having additional elements, is because symbols in Σ_{ret} behave differently. In the first case, elements of Σ_{ret} are like internal actions which leave the stack unchanged, and in the second case they result in the stack being popped.

For a language L over $\widehat{\Sigma}$, define the following congruences.

$$\text{For } u_1, u_2 \in \Sigma^*,\ u_1 \equiv u_2 \text{ iff } \forall v \in MR(\widehat{\Sigma}).\ u_1 v \in L \text{ iff } u_2 v \in L$$
$$\text{For } u_1, u_2 \in MC(\widehat{\Sigma}),\ u_1 \sim_0 u_2 \text{ iff } \forall v \in \Sigma^*.\ u_1 v \in L \text{ iff } u_2 v \in L$$

Intuitively, the congruence $\equiv$ says that the two strings u_1 and u_2 cannot be distinguished by experiments ($v \in MR(\widehat{\Sigma})$) that do not examine the stacks reached on u_1 and u_2. The congruence $\sim_0$ is only defined on strings where every call is matched. Thus, after reading such a word, any VPA will only have $\bot$ on the stack. Starting from such configurations, as was observed earlier, return symbols behave like internal actions, and the congruence is the usual Myhill-Nerode right congruence. We now present the main theorem of this paper.

[5] An equivalence $\equiv$ saturates L iff either $[w]_\equiv \cap L = \emptyset$ or $[w]_\equiv \subseteq L$, for any equivalence class $[w]_\equiv$ of $\equiv$.

Theorem 2. *L is a $\widehat{\Sigma}$-VPL iff $\approx$, $\equiv$ and $\sim_0$ all have finite index.*

Proof. For a VPL L, let $M = (Q, q_0, \Gamma, \delta, Q_F)$ be a VPA recognizing L. In the proof of Theorem 1, we already showed that $\approx$ will have finite index. Define the following two equivalences over words in Σ^*:

$$u_1 \equiv^M u_2 \text{ iff } \delta_\perp(q_0, u_1) = \delta_\perp(q_0, u_2)$$
$$u_1 {\sim_0}^M u_2 \text{ iff } \delta_\perp(q_0, u_1) = \delta_\perp(q_0, u_2)$$

It can be shown that $\equiv^M$ refines $\equiv$, and $\sim_0^M$ refines $\sim_0$ when restricted to $MC(\widehat{\Sigma})$ (proof skipped in the interests of space). Hence, both $\equiv$ and $\sim_0$ have finitely many equivalence classes.

For the converse, we show that L is a VPL by once again constructing a VPA M whose states are equivalence classes of the congruences we have defined, but the construction is a bit more involved. The main intuition behind the construction is to ensure that the following invariant is maintained after M has read a string $u \in \Sigma^*$

- If $u \in MC(\widehat{\Sigma})$ then the state of M is $[u]_{\sim_0}$ and the stack is $\perp$.
- If $u = vc_1w_1 \cdots c_kw_k$, where $v \in MC(\widehat{\Sigma})$, each $w_j \in WM(\widehat{\Sigma})$, and each $c_j \in \Sigma_{\text{call}}$, then M is in state $([u]_\equiv, [w_k]_\approx)$ and the stack is $([w_{k-1}]_\approx, c_k) \cdots ([w_1]_\approx, c_2)([v]_{\sim_0}, c_1)\perp$.

The formal construction of M is as follows. $M = (Q, q_0, \Gamma, \delta, Q_F)$ where $Q = \{[u]_{\sim_0} \mid u \in MC(\widehat{\Sigma})\} \cup \{([u]_\equiv, [w]_\approx) \mid u \in \Sigma^*, w \in WM(\widehat{\Sigma})\}$; $q_0 = [\epsilon]_{\sim_0}$; $\Gamma = Q \times \Sigma_{\text{call}} \cup \{\perp\}$; $Q_F = \{[u]_{\sim_0} \mid u \in L\} \cup \{([u]_\equiv, [w]_\approx) \mid u \in L\}$; and δ is defined as follows:

- $[u]_{\sim_0} \xrightarrow{i} [ui]_{\sim_0}$ for every $i \in \Sigma_{\text{int}}$
- $[u]_{\sim_0} \xrightarrow{c/([u]_{\sim_0}, c)} ([uc]_\equiv, [\epsilon]_\approx)$ for every $c \in \Sigma_{\text{call}}$
- $[u]_{\sim_0} \xrightarrow{r/\perp} [ur]_{\sim_0}$ for every $r \in \Sigma_{\text{ret}}$
- $([u]_\equiv, [w]_\approx) \xrightarrow{i} ([ui]_\equiv, [wi]_\approx)$ for every $i \in \Sigma_{\text{int}}$
- $([u]_\equiv, [w]_\approx) \xrightarrow{c/(([u]_\equiv, [w]_\approx), c)} ([uc]_\equiv, [\epsilon]_\approx)$ for every $c \in \Sigma_{\text{call}}$
- $([u]_\equiv, [w]_\approx) \xrightarrow{r/([u']_{\sim_0}, c)} [u'cwr]_{\sim_0}$ for every $r \in \Sigma_{\text{ret}}$
- $([u]_\equiv, [w]_\approx) \xrightarrow{r/(([u']_\equiv, [w']_\approx), c)} ([u'cwr]_\equiv, [w'cwr]_\approx)$ for every $r \in \Sigma_{\text{ret}}$

The correctness of the construction relies on the intuition outlined earlier and is skipped in the interests of space. □

Remark 2. Note that in the case where $\Sigma_{\text{call}} = \Sigma_{\text{ret}} = \emptyset$ (i.e. for regular languages), the machine M constructed in Theorem 2 is the unique minimum-state automaton for L because the only reachable states will be of the form $[w]_{\sim_0}$, where $w \in \Sigma^*$.

Despite the above remark, the VPA M constructed in Theorem 2 need not be a *minimum-state* $\widehat{\Sigma}$-VPA accepting L. Furthermore,

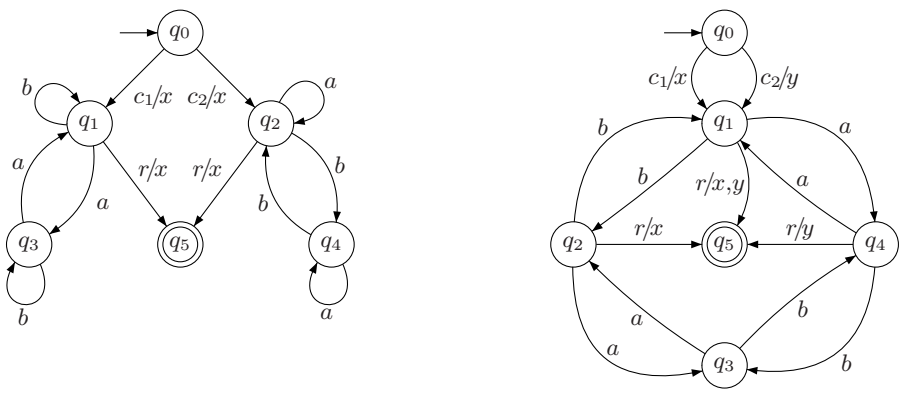

Fig. 1. Two non-isomorphic minimum-state VPAs

Proposition 1. *There are* VPLs *that have no unique minimum-state VPA accepting them.*

To illustrate the above proposition, consider the VPAs in Figure 1. Let $\widehat{\Sigma} = (\{c_1, c_2\}, \{r\}, \{a, b\})$. Let $L = c_1 L_1 r + c_2 L_2 r$, where L_1 is the regular language over $\{a, b\}$ such that the number of a's is even, and L_2 is the regular language over $\{a, b\}$ such that the number of b's is even. The figure shows two *non-isomorphic* minimum-state $\widehat{\Sigma}$-VPAs accepting L. In both machines, the initial state is q_0 and the following transitions have been omitted in the figure for readability: in both machines, every call-transition not shown is of the form $q \xrightarrow{c_j/z} q_2$, and every other transition not shown goes to state q_2.

Notice that the first machine consists of two distinct "modules", one recognizing L_1 and one recognizing L_2, and the call symbol c_1 or c_2 determines which module is "invoked". In contrast, the second machine consists of a single recognizer for both L_1 and L_2, and this module is invoked regardless of the call symbol. As this example illustrates, it is not clear when splitting the task of recognition into distinct modules reduces the total number of states in the VPA. In the following section, we consider a restricted class of VPAs for which the partition of the VPA into modules has already been provided and the call-symbol determines which module is to be invoked. The task then is to minimize the number of states of the automaton, while preserving the given partition of states into modules.

4 *k*-module single-entry visibly pushdown automata

In this section we show that the class of well-matched VPLs have *unique* minimum-state *k-module* single-entry automata (k-SEVPA). As mentioned in the introduction, these automata are motivated by models of programs with finite data-types,

and are similar to single-entry recursive state machines [10] (see [11] for a precise comparison).

k-SEVPAs. Let $\{\Sigma^j_{\text{call}}\}_{j=1}^k$ be a partition of Σ_{call}. A VPA $M = (Q, q_0, \Gamma, \delta, Q_F)$ is a k-module single-entry VPA with respect to $\{\Sigma^j_{\text{call}}\}_{j=1}^k$ if there is a partition $\{Q_j\}_{j=0}^k$ of Q and distinguished states $q_j \in Q_j$ for every $j = 1, \ldots, k$ such that:

1. $Q_F \subseteq Q_0$, $q_0 \in Q_0$;
2. $\Gamma = \{\bot\} \cup (Q \times \Sigma_{\text{call}})$;
3. if $q \xrightarrow{i} q'$ for some $i \in \Sigma_{\text{int}}$, then $\exists j.\ q, q' \in Q_j$;
4. if $q \xrightarrow{c/(q,c)} q'$ for some $c \in \Sigma^j_{\text{call}}$, then $q' = q_j$;
5. if $q' \xrightarrow{r/(q,c)} q''$ for some $c \in \Sigma_{\text{call}}$, then $\exists j.\ q, q'' \in Q_j$.

Intuitively, Q_0 is the base module (corresponding to the 'main' module of a program), and the transition relation is such that a call leads to a unique state in the module corresponding to the call (in models of programs, this state will be the initial control state of the function called), and upon return will return to the calling module. Such automata are no less expressive than VPAs: as Theorem 3 below shows, for any partition of call symbols, any well-matched VPL is accepted by some k-SEVPA.

We use the abbreviation k-SEVPA for such machines and explicitly denote them as $M = ((Q, q_0, \Gamma, \delta, Q_F), \{\Sigma^1_{\text{call}}, \ldots, \Sigma^k_{\text{call}}\}, \{Q_0, \ldots, Q_k\}, \{q_1, \ldots, q_k\})$. For example, for the first VPA in Figure 1, given the partition $\Sigma_{\text{call}} = \{\{c_1\}, \{c_2\}\}$, there is a partition of Q as $\{Q_0, Q_1, Q_2\}$, where $Q_0 = \{q_0, q_5\}$, $Q_1 = \{q_1, q_3\}$, and $Q_2 = \{q_2, q_4\}$ witnessing the fact that this is a 2-SEVPA. Similarly, for the second VPA, given the partition $\Sigma_{\text{call}} = \{\{c_1, c_2\}\}$, there is a partition of Q as $\{Q_0, Q_1\}$, where $Q_0 = \{q_0, q_5\}$ and $Q_1 = \{q_1, q_2, q_3, q_4\}$ establishing that it is a 1-SEVPA.

Remark 3. The automaton constructed in Theorem 1 is a 1-SEVPA. However, in general, this VPA will be much bigger than the smallest 1-SEVPA. The reason for this is similar to the reason why the finite automaton constructed for regular languages from the syntactic congruence is much larger than that obtained from the syntactic right congruence. Thus, in order to characterize the minimal k-SEVPAs, we need a new congruence that partitions words like the Myhill-Nerode right congruence does for regular languages.

In our construction of the minimal k-SEVPA for a language L, we will use $k + 1$ congruences. To model the case when the stack only has $\bot$, we will use $\sim_0$ (defined in Section 3.2 on strings with matched calls). For the case when the stack has additional symbols, we need k new congruences that make use of the fact that the states of the machine are partitioned into k modules identified by the call symbol. Given a k-SEVPA $M = (M', \{\Sigma^j_{\text{call}}\}_{j=1}^k, \{Q_j\}_{j=0}^k, \{q_j\}_{j=1}^k)$ accepting a language L over $\widehat{\Sigma}$, define the following congruences on well-matched strings: for every $j = 1, \ldots, k$,

$$w_1 \sim_j w_2 \text{ iff } \forall u, v \in \Sigma^* \forall c \in \Sigma^j_{\text{call}}.\ ucw_1v \in L \text{ iff } ucw_2v \in L$$

Since $\sim_j$'s will be used to define states when the stack has more than just $\perp$, when defining the equivalence we only need to consider contexts where there is an unmatched call. We are ready to present the main theorem of this section.

Theorem 3. *For any well-matched $\widehat{\Sigma}$-VPL L and any partition $\{\Sigma^j_{\text{call}}\}_{j=1}^k$ of Σ_{call}, there is a unique (upto isomorphism) minimum-state k-SEVPA for L with respect to this partition.*

Proof. We first show that given any partition $\{\Sigma_{\text{call}}\}_{j=1}^k$ of Σ_{call}, there is a k-SEVPA M that recognizes L. We construct M using the equivalences $\{\sim_j\}_{j=1}^k$ and $\sim_0$ (defined in Section 3.2 on strings with matched calls). We then show that this machine M is the unique minimum-state k-SEVPA that recognizes L. The construction of M relies on the observation that $\sim_0$ and $\sim_j$'s all have finite index if L is a VPL. From Theorems 1 and 2, we know that when L is a VPL, $\sim_0$ and $\approx$ are of finite index. Since $\approx$ is a refinement of $\sim_j$ for every j, it follows that all $\sim_j$'s are also of finite index. For $0 \leq j \leq k$, we use the notation $[u]_j$ to denote the equivalence class of $\sim_j$ containing u.

The formal construction of $M = ((Q, q_0, \Gamma, \delta, Q_F), \{\Sigma^j_{\text{call}}\}_{j=1}^k, \{Q_j\}_{j=0}^k, \{q_j\}_{j=1}^k)$ is: $Q_0 = \{[u]_0 \mid u \in MC(\widehat{\Sigma})\}$, and for every $j = 1, \ldots, k$, $Q_j = \{[w]_j \mid w \in WM(\widehat{\Sigma})\}$. For every $j \geq 0$, $q_j = [\epsilon]_j$, and $Q_F = \{[u]_0 \mid u \in L\}$. The transition function δ is given as follows.

- For every $i \in \Sigma_{\text{int}}$ and $j \geq 0$, $[u]_j \xrightarrow{i} [ui]_j$
- For every $c \in \Sigma^{j'}_{\text{call}}$ and $j \geq 0$, $[u]_j \xrightarrow{c/([u]_j, c)} [\epsilon]_{j'}$
- For every $r \in \Sigma_{\text{ret}}$ and $j, j' \geq 0$, $[w]_j \xrightarrow{r/([u]_{j'}, c)} [ucwr]_{j'}$
- For every $r \in \Sigma_{\text{ret}}$, $[u]_0 \xrightarrow{r/\perp} [ur]_0$

Q_F is well-defined because $\sim_0$ is an equivalence that saturates L. The transition function is consistent because $u_1 i \sim_j u_2 i$ whenever $u_1 \sim_j u_2$ and $i \in \Sigma_{\text{int}}$, and because when $u_1 \sim_{j'} u_2$ and $w_1 \sim_j w_2$, $u_1 c w_1 r \sim_{j'} u_2 c w_2 r$ for every $j > 0, j' \geq 0$ and $c \in \Sigma^j_{\text{call}}$, $r \in \Sigma_{\text{ret}}$. Thus, the above machine is well defined. Further observe that the following invariant is maintained during the execution: after reading a string u

- If $u \in MC(\widehat{\Sigma})$ then the state of M is $[u]_0$ and the stack is $\perp$.
- If $u = vc_1w_1 \ldots c_lw_l$, where $v \in MC(\widehat{\Sigma})$, each $c_j \in \Sigma^{m_j}_{\text{call}}$ and each $w_j \in WM(\widehat{\Sigma})$, then the state of M is $[w_l]_{m_l}$ and the stack is $([w_{l-1}]_{m_{l-1}}, c_l) \ldots ([w_1]_{m_1}, c_2)([v]_0, c_1)\perp$.

Hence, if a string reaches a final state, we are guaranteed that the stack only has $\perp$, and recognizes L. The formal proof of correctness is skipped.

Consider any k-SEVPA $M' = ((Q', q'_0, \Gamma', \delta', Q'_F), \{\Sigma^j_{\text{call}}\}_{j=1}^k, \{Q'_j\}_{j=0}^k, \{q'_j\}_{j=1}^k)$ recognizing L. We show that M is the unique minimum-state k-SEVPA by demonstrating a homomorphism from M' to M. In other words, we construct an onto function $f : \bigcup_{j \geq 0} Q'_j \to \bigcup_{j \geq 0} Q_j$ having the following properties.

1. $f(q'_j) = q_j$ for every $j \geq 0$
2. For any $i \in \Sigma_{\text{int}}$, if $p' \xrightarrow{i}_{M'} q'$ then $f(p') \xrightarrow{i}_{M} f(q')$
3. For any $c \in \Sigma_{\text{call}}$, if $p' \xrightarrow{c/(p',c)}_{M'} q'$ then $f(p') \xrightarrow{c/(f(p'),c)}_{M} f(q')$
4. For any $r \in \Sigma_{\text{ret}}$, if $p' \xrightarrow{r/(s',c)}_{M'} q'$ then $f(p') \xrightarrow{r/(f(s'),c)}_{M} f(q')$

Thus, we will be able to conclude that $|\cup Q'_j| \geq |\cup Q_j|$, and if $|\cup Q'_j| = |\cup Q_j|$ then f witnesses an isomorphism between M and M'.

The homomorphism f from M' to M is defined as follows:

$$f(q') = \begin{cases} [u]_0 \text{ if } \exists u \in MC(\widehat{\Sigma}).\ \delta'_\perp(q'_0, u) = q' \\ [w]_j \text{ if } \exists u \in \Sigma^*, c \in \Sigma^j_{\text{call}}, w \in WM(\widehat{\Sigma}).\ \delta'_\perp(q'_0, ucw) = q' \end{cases}$$

Note that $f(q'_j) = [\epsilon]_j$ for every $0 \leq j \leq k$. Observe that f maps states of Q'_j to the equivalence classes of $\sim_j$ for every $0 \leq j \leq k$. We need to show that f is well defined, i.e., f is indeed a function and does not map a state of M' to two different states of M. This follows from the following lemma.

Lemma 1. *If $u_1, u_2 \in MC(\widehat{\Sigma})$ are such that $\delta'_\perp(q'_0, u_1) = \delta'_\perp(q'_0, u_2)$, then $u_1 \sim_0 u_2$. In addition, for well-matched strings w_1 and w_2 and every $j = 1, \ldots, k$, if $\delta'_\perp(q'_j, w_1) = \delta'_\perp(q'_j, w_2)$ then $w_1 \sim_j w_2$.*

The proof of the above lemma is similar to the proofs in Theorems 1 and 2 where we show our congruences to have finite index. Thus, f is indeed a function. Further, f is clearly onto. Also, f preserves initial state and distinguished states q_j, by definition. It preserves the transitions of M' because $\sim_0$ and $\sim_j$'s are congruences. This completes the proof that there is a unique minimum-state k-SEVPA. □

While the above theorem shows that each (well-matched) VPL has a unique k-SEVPA with respect to a given partition of Σ_{call}, the constructed machine may be much bigger than the smallest VPA recognizing the language because in a k-SEVPA, each module is constrained to have a unique "entry" (an entry is the destination of a push-transition). The presence of multiple entries can greatly reduce the size of the VPA as the following proposition shows.

Proposition 2. *For positive integers m, n, there is a family of well-matched* VPL*s $L_{m,n}$ such that the smallest VPA recognizing $L_{m,n}$ has at most $O(nm)$ states, while the smallest 1-SEVPA recognizing $L_{m,n}$ has at least n^m states.*

As the following theorem states, there is an efficient algorithm to minimize k-SEVPAs. The algorithm is omitted due to lack of space, but can be found along with the proof of correctness and complexity analysis in [11].

Theorem 4. *Given a k-SEVPA M with respect to a partition $\{\Sigma^j_{\text{call}}\}_{j=1}^k$ of Σ_{call} accepting a well-matched language L, the unique minimum-state k-SEVPA with respect to $\{\Sigma^j_{\text{call}}\}_j$ that accepts L can be computed in time $O(n^3)$, where n is the size M.*

5 Conclusions

We presented a characterization of VPLs in terms of congruences on strings of finite index and gave constructions of canonical automata recognizing visibly pushdown languages. We showed that while VPLs in general do not have unique minimum-state deterministic recognizers, the class of well-matched VPLs do have unique minimal k-module single-entry deterministic visibly pushdown automata (k-SEVPAs) for any fixed partition of the call symbols.

Our constructions of visibly pushdown automata based on congruences can, in general, result in automata with exponentially more states than a smallest deterministic visibly pushdown automaton recognizing the language. A characterization and construction of visibly pushdown automata that are at most polynomial in the size of the smallest automaton recognizing a language is an interesting open problem.

We presented a minimization algorithm for k-SEVPAs that runs in time $O(n^3)$. The computational complexity of the problem of constructing the smallest k-SEVPA given any visibly pushdown automaton (not necessarily k-module) is open, and would be interesting to investigate.

Acknowledgements. We would like to thank a referee for strengthening the lower bound for Proposition 2.

References

1. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: Proceedings of STOC '04, ACM Press (2004) 202–211
2. Alur, R., Etessami, K., Madhusudan, P.: A temporal logic of nested calls and returns. In: Proceedings of TACAS '04. LNCS 2988, Springer (2004) 467–481
3. Ball, T., Rajamani, S.: Bebop: A symbolic model checker for boolean programs. In: SPIN 2000 Workshop on Model Checking of Software. LNCS 1885. Springer (2000) 113–130
4. Pitcher, C.: Visibly pushdown expression effects for XML stream processing. In: Programming Language Technologies for XML. (2005) 1–14
5. Murawski, A., Walukiewicz, I.: Third-order idealized algol with iteration is decidable. In: FOSSACS. 3441 (2005) 202–218
6. Löding, C., Madhusudan, P., Serre, O.: Visibly pushdown games. In: Proceedings of FSTTCS'04. LNCS (2004)
7. Nerode, A.: Linear automaton transformations. In: Proc. AMS. Volume 9. (1958) 541–544
8. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison Wesley (1979)
9. Hopcroft, J.E.: An $n \log n$ algorithm for minimizing the states in a finite automaton. In: The Theory of Machines and Computations. Acad. Press (1971) 189–196
10. Alur, R., Benedikt, M., Etessami, K., Godefroid, P., Reps, T., Yannkakis, M.: Analysis of recursive state machines. ACM Transactions on Programming Languages and Systems (to appear) (2005)
11. Alur, R., Kumar, V., Madhusudan, P., Viswanathan, M.: Congruences for visibly pushdown languages. Technical Report UIUCDCS-R-2005-2565, UIUC (2005)