

Visibly Pushdown Languages Are Closed under Prefix, Shuffle, and Hiding*

STEFAN D. BRUDA and MD. TAWHID BIN WAEZ

Department of Computer Science

Bishop's University

2600 College St

Sherbrooke, Quebec J1M 1Z7

CANADA

{bruda,mtbwaez}@cs.ubishops.ca <http://turing.ubishops.ca>

Abstract: Visibly pushdown languages are a subclass of context-free languages that is closed under all the useful operations, namely union, intersection, complementation, renaming, concatenation, and Kleene star. The existence of a fully compositional process algebra based on such languages require that these languages be also closed under slightly more esoteric operations, namely prefix, shuffle, and hiding. We prove here all these closure properties. We also give the semantics of visibly pushdown automata in terms of labelled transition systems. In effect, we prove the existence of a fully compositional process algebra based on visibly pushdown languages, and we also lay the foundations of such an algebra.

Key-words: Visibly pushdown languages, Visibly pushdown automata, Closure properties, Prefix, Hiding, Shuffle, Infinite-state process algebra

1 Introduction

The formal verification arena has been enhanced by the recent introduction of the class of visibly pushdown languages (VPL) [2], a subclass of context-free languages. VPL are particularly well suited for modelling software analysis, and they are also tractable and robust like the class of regular languages (and therefore they have almost all the prerequisites to support a fully compositional process algebra).

VPL are accepted by visibly pushdown automata (vPDA) whose stack behaviour is determined by the input symbols. A vPDA operates on a word over an alphabet that is partitioned into three disjoint sets of call, return, and local symbols. Any input symbol can change the control state but call and return symbols can also change the stack content. While reading a call symbol the automaton must push one symbol on the stack, and while reading a return symbol it must pop one symbol (unless the stack is empty).

VPL are closed under intersection, union, complementation, renaming, concatenation and Kleene star, just like regular languages. A number of decision problems such as universality, language equivalence and language inclusion, which are not decidable for context-free languages, become EXPTIME-complete

for VPL. VPL seem quite natural for verification of pre/post conditions or for inter-procedural flow properties. In particular, requirements that can be verified in this manner include all regular properties but also non-regular properties such as partial correctness, total correctness, local properties, access control, and stack limits.

From a conformance testing point of view, vPDA have been mostly the subject of studies related to logic based conformance testing (namely, model checking) [1, 2]. vPDA and its natural subclass visibly BPA (Basic Process Algebra) are also studied [6] (but to a lesser extent) in terms of behavioural equivalence such as bisimulation relations. There is to our knowledge no work on process algebra to represent complex, concurrent vPDA systems, even if the closure properties of VPL support such an approach almost completely. We note that a good compositional process algebra is the main tool in the virtually unstudied area of vPDA algebraic-based conformance testing.

One of the possible reason for the absence of a vPDA-based process algebra is that the known closure properties support such a development only partially. Indeed, suppose that a fully compositional vPDA-based process algebra (call it vPDAA for brevity) exist, and consider the operation of unrestricted interleaving \parallel existent in all the finite-state process algebras such as CCS [5] and CSP [4]. Such an operation

*This research was supported by the Natural Sciences and Engineering Research Council of Canada. Part of this research was also supported by Bishop's University.

is vital, as it models that part of the execution of concurrent processes that do not contain any communication or synchronization; obviously, at some point any two concurrent processes will contain such an execution, so our hypothetical vPDAA must contain such an interleaving operator. Consider now two processes P_1 and P_2 specified using vPDAA, whose traces form the languages L_1 and L_2 , respectively. It follows that both L_1 and L_2 are visibly pushdown languages, and since the unrestricted interleaving is an operator of vPDAA (an algebra), the language L of traces of $P_1 \parallel P_2$ must also be a visibly pushdown language. However, L is the shuffle of L_1 and L_2 . Therefore, a necessary condition for vPDAA to exist at all is that visibly pushdown languages be closed under shuffle.

A less critical but nonetheless useful operator in a process algebra is hiding. Such an operator is used to hide the internals of a process and expose only its interface to the environment. The same trace argument requires that VPL be closed under hiding for this operator to exist. Finally, the interrupt operator found in CSP allows for the arbitrary preemption of a process by another. Trace arguments similar to the above then require that VPL be closed under concatenation (already known) and also prefix (unknown) for interrupt to be possible at all.

We eliminate in this paper this last stumbling block toward a fully compositional vPDA-based process algebra, establishing the closure under shuffle, hiding, and prefix of visibly pushdown languages. In the process, we also establish a semantics for vPDA based on labelled transition systems, which is a building block in the development of the mentioned process algebra. Our results effectively prove the existence of a vPDA-based process algebra, and also support it by providing a natural semantical mechanism.

2 Preliminaries

The shuffle of two languages L_1 and L_2 over an alphabet Σ is defined as $L_1 \parallel L_2 = \{w_1v_1w_2v_2 \cdots w_mv_m : w_1w_2 \cdots w_m \in L_1, v_1v_2 \cdots v_m \in L_2 \text{ for all } w_i, v_i \in \Sigma^*\}$. Given a language L over an alphabet Σ and a set $A \subseteq \Sigma$, the result of hiding A in L is the set $L \setminus A$ that contains exactly all the strings from L but with all the occurrences of symbols in A erased. The prefix of a language L over Σ^* is the language $pre(L) = \{w : \exists u \in \Sigma^* : wu \in L\}$. We use the symbol ε to denote the empty word and only the empty word.

A visibly pushdown automaton (vPDA for short) [2] is

a tuple $M = (\Phi, \Phi_{in}, \tilde{\Sigma}, \Gamma, \Omega, \Phi_F)$, where Φ is a finite set of states, $\Phi_{in} \subseteq \Phi$ is a set of initial states, $\Phi_F \subseteq \Phi$ is the set of final states, Γ is the (finite) stack alphabet that contains a special bottom-of-stack symbol \perp , and Ω is the transition relation $\Omega \subseteq (\Phi \times \Gamma^*) \times \tilde{\Sigma} \times (\Phi \times \Gamma^*)$. In addition, $\tilde{\Sigma} = \{\Sigma_l \cup \Sigma_c \cup \Sigma_r\}$ is a finite set of visibly pushdown input symbols where Σ_l is the set of local symbols, Σ_c is the set of call symbols and Σ_r is the set of return symbols. $(\Sigma_l, \Sigma_c, \Sigma_r)$ is a partition over $\tilde{\Sigma}$ (meaning that $\tilde{\Sigma} = \Sigma_l \uplus \Sigma_c \uplus \Sigma_r$).

Every tuple $((P, \gamma), a, (Q, \delta)) \in \Omega$ (also written $(P, \gamma) \xrightarrow{a} (Q, \delta) \in \Omega$) must have the following form: if $a \in \Sigma_l \cup \{\varepsilon\}$ then $\gamma = \delta = \varepsilon$, else if $a \in \Sigma_c$ then $\gamma = \varepsilon$ and $\delta = a'$ (where a' is the stack symbol pushed for a), else if $a \in \Sigma_r$ then if $\gamma = \perp$ then $\gamma = \delta$ (hence vPDA allows unmatched return symbols) else $\gamma = a'$ and $\delta = \varepsilon$ (where a' is the stack symbol popped for a).

In other words, a local symbol is not allowed to modify the stack, while a call always pushes one symbol on the stack. Similarly, a return symbol always pops one symbol off the stack, except when the stack is already empty. Note in particular that empty transitions (that is, transitions that do not consume any input) are allowed but are not permitted to modify the stack [2].

Whenever we have a pair of symbols c and r such that $c \in \Sigma_c$, $r \in \Sigma_r$, and $(P, \varepsilon) \xrightarrow{c} (Q, a), (R, a) \xrightarrow{r} (S, \varepsilon) \in \Omega$, the two symbols are called *matched*. A call (or return) that has no matched return (or call) is called *unmatched*. Note that some call or return can be both matched and unmatched at the same time in a given vPDA.

The notion of run, acceptance, and language accepted by a vPDA are defined as usual: A run of a vPDA M on some string $w = a_1a_2 \dots a_k$ is a sequence $(q_0, \gamma_0)(q_{01}, \gamma_0) \cdots (q_{0m_0}, \gamma_0)(q_1, \gamma_1)(q_{11}, \gamma_1) \cdots (q_{1m_1}, \gamma_1)(q_2, \gamma_2) \cdots (q_k, \gamma_k)(q_{k1}, \gamma_k) \cdots (q_{km_k}, \gamma_k)$ such that $\gamma_0 = \perp$, $q_0 \in \Phi_{in}$, $(q_{j-1i}, \varepsilon) \xrightarrow{\varepsilon} (q_{ji}, \varepsilon) \in \Omega$ for all $1 \leq i \leq k$, $1 \leq j \leq m_i$, and $(q_{i-1m_{i-1}}, \gamma'_{i-1}) \xrightarrow{a_i} (q_i, \gamma'_i) \in \Omega$ for every $1 \leq i \leq k$ and for some prefixes γ'_{i-1} and γ'_i of γ_{i-1} and γ_i , respectively. Whenever $q_{km_k} \in \Phi_F$ the run is accepting; M accepts w iff there exists an accepting run of M on w . The language $L(M)$ accepted by M contains exactly all the strings w accepted by M .

It is possible for a call or a return which is both matched and unmatched in a vPDA to have only one characteristic in a particular string accepted by the vPDA (i.e., be either matched, or unmatched, but not

both in that string). A string $w \in L(M)$ is well-matched if it has no unmatched calls or returns.

A labelled transition system (LTS for short) [3] is a triple $(\Theta, \Sigma, \Delta, I)$, where Θ is a set of states, Σ is a finite set of actions (not containing the internal action τ), $I \in \Theta$ is the initial state, and Δ is the transition relation such that $\Delta \subseteq \Theta \times (\Sigma \cup \{\tau\}) \times \Theta$. If Δ is unambiguous and understood from the context, then we often use the following shorthands: $P \xrightarrow{a} Q$ whenever $(P, a, Q) \in \Delta$, $P \xrightarrow{a}$ whenever there exists a Q such that $P \xrightarrow{a} Q$, and $P \not\xrightarrow{a}$ whenever $P \xrightarrow{a}$ does not hold.

A run of an LTS M is a sequence $q_0 \tau q_{01} \tau \cdots \tau q_{0m_0} a_1 q_{11} \tau q_{11} \tau \cdots \tau q_{1m_1} a_2 q_2 \cdots a_k q_k \tau q_{k1} \tau \cdots \tau q_{km_k}$ such that $q_0 = I$, $q_{j-1i} \xrightarrow{\tau} q_{ji}$ for all $1 \leq i \leq k$, $1 \leq j \leq m_i$, and $q_{i-1m_{i-1}} \xrightarrow{a_i} q_i$ for all $1 \leq i \leq k$. The trace of this run is the sequence $a_1 a_2 \cdots a_k$. The run is maximal whenever there is no x such that $q_{km_k} \xrightarrow{x}$. The trace of a maximal run is called a complete trace. The language trace(M) [ctrace(M)] contains exactly all the traces [complete traces] of all the possible runs [maximal runs] of M .

3 The Semantics of vPDA

We can define the semantics of vPDA in terms of labelled transition systems in a natural way, as follows:

Definition 3.1 Given any vPDA $M = (\Phi, \Phi_{in}, \tilde{\Sigma}, \Gamma, \Omega, \Phi_F)$, the LTS $\llbracket M \rrbracket$ is defined as follows: $\llbracket M \rrbracket = ((\Phi \cup \{H, I\}) \times \Gamma^*, \tilde{\Sigma} \cup \{\tau, x\}, \Delta)$, where $I, H \notin \Phi$ and $x \notin \tilde{\Sigma} \cup \{\tau\}$. The transition relation of $\llbracket M \rrbracket$ is $\Delta \subseteq ((\Phi \cup \{I\}) \times \Gamma^*) \times (\tilde{\Sigma} \cup \{\tau, x\}) \times ((\Phi \cup \{H\}) \times \Gamma^*)$ and is defined as follows: $\Delta = \{((q, \gamma), a, (q', \gamma')) : ((q, \gamma), a, (q', \gamma')) \in \Omega\} \cup \{((I, \perp), \tau, (q, \perp)) : q \in \Phi_{in}\} \cup \{((q, \gamma), \tau, (H, \gamma)) : q \in \Phi_F\} \cup \{((q, \gamma), x, (q, \gamma)) : q \notin \Phi_F, \forall a \in \tilde{\Sigma} \cup \{\tau\} : (q, \gamma) \not\xrightarrow{a}\}$. ■

A state of $\llbracket M \rrbracket$ is labelled with a state of M as well as the stack content associated with that state of M in the given computation. We first include in Δ the transitions corresponding to the transition of the vPDA being modelled. $\llbracket M \rrbracket$ should be capable of starting from any state $\Phi_{in} \times \{\perp\}$; to create a unique initial state we introduce a brand new state (I, \perp) and we add to Δ the set of transitions that gets us nondeterministically to one of the initial states of the vPDA being modelled. Finally, we invent the final state H that has no outgoing transitions and is reachable from any final state of M via τ transitions. Non-final states

with no outgoing transitions gain a loop that performs an action never encountered in the original vPDA.

The following result establishes the LTS $\llbracket M \rrbracket$ thus constructed as the semantical model of the vPDA M :

Theorem 3.1 For any vPDA M it holds that $L(M) = \text{ctrace}(\llbracket M \rrbracket)$.

Proof. Let $M = (\Phi, \Phi_{in}, \tilde{\Sigma}, \Gamma, \Omega, \Phi_F)$.

Consider some $w = a_1 \dots a_k \in L(M)$ and let $(q_0, \perp)(q_{01}, \perp) \cdots (q_{0m_0}, \perp)(q_1, \gamma_1)(q_{11}, \gamma_1) \cdots (q_{1m_1}, \gamma_1)(q_2, \gamma_2) \cdots (q_k, \gamma_k)(q_{k1}, \gamma_k) \cdots (q_{km_k}, \gamma_k)$ be an accepting run of M on w . Then the run $\rho' = (I, \perp)\tau(q_0, \perp)\tau(q_{01}, \perp)\tau \cdots \tau(q_{0m_0}, \perp)a_1(q_1, \gamma_1)\tau(q_{11}, \gamma_1)\tau \cdots \tau(q_{1m_1}, \gamma_1)a_2(q_2, \gamma_2) \cdots a_k(q_k, \gamma_k)\tau(q_{k1}, \gamma_k)\tau \cdots \tau(q_{km_k}, \gamma_k)\tau(H, \gamma_k)$ exists in $\llbracket M \rrbracket$ by definition (indeed, q_0 is an initial state, hence the τ transition from (I, \perp) to (q_0, \perp) ; similarly, q_{km_k} is a final state, hence the transition from (q_{km_k}, γ_k) to (H, γ_k)). Moreover, ρ' is maximal (since no state (H, γ) has outgoing transitions) and therefore $w \in \text{ctrace}(\llbracket M \rrbracket)$. Thus, $L(M) \subseteq \text{ctrace}(\llbracket M \rrbracket)$.

Consider now some $w = a_1 \dots a_k \in \text{ctrace}(M)$. We then have a run $\rho' = (I, \perp)\tau(q_0, \perp)\tau(q_{01}, \perp)\tau \cdots \tau(q_{0m_0}, \perp)a_1(q_1, \gamma_1)\tau(q_{11}, \gamma_1)\tau \cdots \tau(q_{1m_1}, \gamma_1)a_2(q_2, \gamma_2) \cdots a_k(q_k, \gamma_k)\tau(q_{k1}, \gamma_k)\tau \cdots \tau(q_{km_k}, \gamma_k)\tau(H, \gamma_k)$ such that $q_0 \in \Phi_{in}$ and $q_{km_k} \in \Phi_F$. Indeed, the state (I, \perp) has only τ transitions outgoing toward states in $\Phi_{in} \times \{\perp\}$. In addition, exactly all the maximal runs of $\llbracket M \rrbracket$ end up in a state (H, γ) (every final state has a τ transition that leads to such a state, and no other state is the terminal state of a maximal run—those non-final states with no outgoing transitions in the original vPDA are given a loop in $\llbracket M \rrbracket$ in order to avoid such), so we must end any maximal run at (H, γ) . The preceding state (q_{km_k}, γ_k) is then (a) linked to (H, γ) by a τ transition (only these are available), and (b) has $q_{km_k} \in \Phi_F$ (only such states are linked directly to (H, γ)). Then $(q_0, \perp)(q_{01}, \perp) \cdots (q_{0m_0}, \perp)(q_1, \gamma_1)(q_{11}, \gamma_1) \cdots (q_{1m_1}, \gamma_1)(q_2, \gamma_2) \cdots (q_k, \gamma_k)(q_{k1}, \gamma_k) \cdots (q_{km_k}, \gamma_k)$ is an accepting run of M on w and thus $w \in L(M)$. Therefore $\text{ctrace}(\llbracket M \rrbracket) \subseteq L(M)$.

In all, $L(M) = \text{ctrace}(\llbracket M \rrbracket)$, as desired. ■

4 Closure Properties of VPL

It is already known that VPL are closed under union, intersection, complementation, renaming, concatenation, and Kleene star. In addition, we establish in this

section the promised closure under prefix, hiding, and shuffle.

We will use in what follows the given vPDA and also their associated LTS constructed according to Definition 3.1. We will then use the relation between the two constructs (vPDA and LTS) as given in Theorem 3.1.

Theorem 4.1 *VPL are closed under prefix.*

Proof. We note that vPDA accept by final state only, like finite automata do. Consider then a VPL L and the vPDA $M = (\Phi, \Phi_{in}, \tilde{\Sigma}, \Gamma, \Omega, \Phi_F)$ such that $L = L(M)$. Accepting $pre(L)$ requires a vPDA $M' = (\Phi', \Phi'_{in}, \tilde{\Sigma}, \Gamma, \Omega', \Phi'_F)$ that is able to stop and accept anywhere inside an accepting run of M . Stopping anywhere can be set up by just setting $\Phi' = \Phi$ and $\Phi'_F = \Phi$; however, this works in the finite automata case but could result here in some otherwise non-accepting runs to become accepting (since the stack also plays a role in the possible runs even if it does not participate in acceptance). The states participating in such runs must be identified first (and then excluded from the process of making all the states final).

To do this, we first identify the set N of states of $\llbracket M \rrbracket$ defined inductively as follows:

1. Any state (p, γ) such that $(p, \gamma) \xrightarrow{x}$ in is N .
2. Any state (p, γ) such that the only transitions of (p, γ) have the form $(p, \gamma) \xrightarrow{a} (q, \delta)$ for some a and some $(q, \delta) \in N$ in is N .

The states in N cannot participate in a successful run leading to a complete trace, so they cannot appear in any accepting run of M . We start modifying M as follows:

1. For every $(p, \gamma) \in N$ such that there is no $(p, \delta) \notin N$, we eliminate p from Φ and we also eliminate all the transitions in Ω involving p . Indeed, such a vPDA state can never participate in an accepting run.
2. For all the other $(p, \gamma) \in N$, the vPDA state p appears somewhere in an accepting run, so it cannot be eliminated. We then check all the incoming transitions $(q, \delta) \xrightarrow{a} (p, \gamma)$ with the associated vPDA transitions $(q, \delta') \xrightarrow{a} (p, \gamma') \in \Omega$. We then add to Ω the transition $(q, \delta') \xrightarrow{a} (p', \gamma')$,

where p' is a new name, we change all the occurrences of p in N to p' , and we modify Ω to reflect such a change. Finally, we add p' to the set Φ_N and also to the set Φ . The vPDA state p can participate in a successful run on another path of $\llbracket M \rrbracket$; we then rename the state on the current, unsuccessful path, while keeping the original name on the other (potentially successful) paths. The thus renamed state becomes once more unable to participate in an accepting run so we identify it accordingly.

The vPDA M thus modified clearly accepts the same language. However, the automaton now has an identified set of states $\Phi_N \subseteq \Phi$ that are exactly all the states that cannot appear in any accepting run. Indeed, we have identified those states in $\llbracket M \rrbracket$ that cannot be part of a maximal run, and we have renamed their vPDA states so as to distinguish them from those vPDA states that can also be part of a maximal run.

Now we can make all the states in M final to obtain M' that accepts $pre(L(M))$: specifically, we set $\Phi' = \Phi$, $\Phi'_{in} = \Phi_{in}$, $\Omega' = \Omega$, and $\Phi'_F = \Phi \setminus \Phi_N$. ■

Theorem 4.2 *VPL are closed under hiding.*

Proof. Consider a VPL L over $\tilde{\Sigma}$ and any set $A = A_c \uplus A_r \uplus A_l \subseteq \tilde{\Sigma}$. Let M be a vPDA that accepts L . We show how the symbols in A can be hidden one by one, so that in the end all the symbols from A can be hidden.

Hiding local symbols as well as hiding some call (return) together with all its matching returns (calls) can be accomplish by simply replacing in M the respective transitions by empty transitions that do not modify the stack (which yields a vPDA). Same goes for hiding unmatched calls and unmatched returns.

Consider now that we hide a call c but we do not hide its matching return r . Then every trace containing c and r in $\llbracket M \rrbracket$ will be transformed from $w_1 c w_2 r w_3$ (with w_2 well-matched) into $w_1 w_2 r w_3$. The last call unmatched in w_1 becomes however matched with r , and the matching (in w_3) of the other unmatched calls in w_1 “shift” one symbol to the right. This shifting is handled by a suitably modified $\llbracket M \rrbracket$, in which transitions are added so that the new, shifted matchings are allowed: Suppose the original path $w_1 c w_2 r w_3$ uses the transition $(P, \gamma) \xrightarrow{b} (Q, d\gamma)$ to handle a call from w_1 and $(R, d\delta) \xrightarrow{m} (S, \delta)$ to match b with a return m from w_3 , and suppose that a transition $(P', \gamma') \xrightarrow{b'}$ is used to handle the

symbol from w_1 that is the first symbols to the left of b in the original path that is either matched by a return from w_3 or is unmatched; then, the transition $(P, \gamma) \xrightarrow{b} (Q, d'\gamma)$ is added. Furthermore, one previously matched return r' in w_3 becomes unmatched, so whenever the transition $(P, a\perp) \xrightarrow{r'} (Q, \perp)$ is used in the original path, we add to the original definition a transition $(P, \perp) \xrightarrow{r'} (Q, \perp)$. If we perform this procedure for every possible trace $w_1cw_2rw_3$, then we obtain an LTS from which r has been eliminated (that is, hidden). Since we have added only transitions of the proper form, the resulting LTS clearly corresponds to a vPDA.

We perform a similar procedure (“shift” matchings, this time to the left) whenever we hide a return r but not its matching call c ; a call c' previously matched is now unmatched, but this situation does not need any new transition added to the original LTS, it being handled automatically (since vPDA accept by final state only).

Note that this is not an algorithmic procedure (e.g., we can have an infinite number of paths $w_1cw_2rw_3$), but does show closure under hiding which serves our purpose. ■

Theorem 4.3 *VPL are closed under shuffle.*

Proof. Consider two vPDA $M' = (\Phi', \Phi'_{in}, \tilde{\Sigma}, \Gamma', \Omega', \Phi'_F)$ and $M'' = (\Phi'', \Phi''_{in}, \tilde{\Sigma}, \Gamma'', \Omega'', \Phi''_F)$. We will construct the vPDA $M = (\Phi, \Phi_{in}, \tilde{\Sigma}, \Gamma, \Omega, \Phi_F)$ that accepts the shuffle of $L(M')$ and $L(M'')$. The construction performs an alternative simulation of M' and M'' and is constructed as follows:

We need to keep track of the states of both M' and M'' during any run of M , so we put $\Phi = \Phi' \times \Phi''$. M starts any of its runs from the start of both M' and M'' , so we put $\Phi_{in} = \Phi'_{in} \times \Phi''_{in}$. Similarly, at the end of the run M accepts the input iff both M and M' accept their corresponding inputs and thus $\Phi_F = \Phi'_F \times \Phi''_F$. The stack alphabet of M is $\Gamma = \Gamma' \cup \Gamma''$. The transition relation Ω is constructed as follows:

- We can shuffle any symbol with a local in an immediate fashion. If one of the symbols is a local, then it can arbitrarily appear earlier or later than the other symbol in the shuffle. That is, for every
 - pair of symbols x' and x'' such that either $x' \in \Sigma_l$ or $x'' \in \Sigma_l$, and

– set of rules

$$\begin{aligned} (P', \alpha') &\xrightarrow{x'} (Q', \beta') \in \Omega' \\ (P'', \alpha'') &\xrightarrow{x''} (Q'', \beta'') \in \Omega'' \end{aligned}$$

with suitable values for $\alpha', \beta', \alpha''$ and β'' ,

we add the following sets of rules: $\{((P', X), \alpha') \xrightarrow{x'} ((Q', X), \beta') : X \in \{P'', Q''\}\}$ and $\{((X, P''), \alpha'') \xrightarrow{x''} ((X, Q''), \beta'') : X \in \{P', Q'\}\}$.

- Let us shuffle any two call–return pairs in the two languages as if they were alone in the input. In the process the original matchings will change; indeed, if the original matchings are c' and r' in M' , and c'' and r'' in M'' , “cross-matchings” will be allowed in M between c' and r'' and between c'' and r' (for otherwise a shuffle is not possible). Formally, for every

- matching call c' and return r' in M' ,
- matching call c'' and return r'' in M'' , and
- set of rules

$$\begin{aligned} ((P', \perp) &\xrightarrow{c'} (Q', a)), \\ ((R', a) &\xrightarrow{r'} (S', \perp)) \in \Omega' \\ ((P'', \perp) &\xrightarrow{c''} (Q'', b)), \\ ((R'', b) &\xrightarrow{r''} (S'', \perp)) \in \Omega'' \end{aligned}$$

we add the following rules: $\{((P', X), \perp) \xrightarrow{c'} ((Q', X), a) : X \in \{P'', Q''\}\}$, $\{((X, P''), \perp) \xrightarrow{c''} ((X, Q''), b) : X \in \{P', Q'\}\}$, $\{((R', X), \alpha) \xrightarrow{r'} ((S', X), \perp) : X \in \{R'', S''\}, \alpha \in \{a, b\}\}$, and $\{((X, R''), a) \xrightarrow{r''} ((X, S''), \perp) : X \in \{R', S'\}, \alpha \in \{a, b\}\}$.

In effect, we allow the shuffling of the two pair of symbols in any combination: Whenever M is ready to accept c' it is also ready to accept c'' . If one of these two (e.g., c') was already accepted, then M is ready to accept the other symbol (e.g., c''), as well as the matching return of the already accepted input (e.g., r'). Whenever both calls have been accepted, either return is acceptable first. That matching call–return pairs do not exist in isolation but can be mingled with local symbols is taken care of by the previous case.

- We handle an unmatched call c as follows: Suppose we had a matched return for c ; if this were the case, we would be covered by the previous case. We do not have such a return, but we can however invent one (call it r) in the original vPDA (M' or M'') that contains the unmatched call. We then proceed with the construction outlined in the previous case. Once this is done, we hide $\{r\}$ in the resulting language (accepted by M). The call c becomes once more unmatched. Given Theorem 4.2, M continues to be a vPDA. An unmatched return is handled similarly: we invent a matching call for it in the original vPDA, we use the previous case to create the vPDA M and then we hide the just invented call.
- Nothing else is included in Ω , for indeed the cases above cover all the possibilities that can appear in a shuffle.

The correctness of the construction follows quite easily from the considerations expressed in each case of the construction (plus Theorem 4.2 since the construction uses hiding). ■

5 Conclusions

We have shown that VPL are closed under prefix, shuffle, and hiding. Together with the already known closure under union, intersection, complementation, renaming, concatenation, and Kleene star, we showed in effect that VPL have all the necessary closure properties in order for a VPL-based process algebra for infinite-state systems to be possible. We proved in effect the existence of such an algebra.

We also offered in the process support for the development of the algebra by establishing an LTS semantics for vPDA. LTS are the underlying semantic model for all the process algebras, so this is one significant step.

Continuing this step and actually developing a VPL-based process algebra is one of our active research interests. Finite-state algebras have proven useful for the specification and verification of hardware, communication protocols, and drivers. More complex application software cannot be readily modelled using finite-state mechanisms, as they contain a huge, impractical number of distinct finite states. We therefore believe that an infinite-state process algebra can dramatically open the domain of application software to specification and verification using for-

mal methods (and more specifically algebraic methods such as model-based testing).

Some important particularities of prefix, hiding, and shuffle are worth pointing out. Specifically, we note that the result of such an operator can yield a language that is completely different from the components, especially in terms of matching calls and returns. This might have considerable impact on system specification (using the corresponding algebraic operators) and is thus worth keeping in mind (though such a change in the language is likely to be justified in the trace semantics of the systems involved).

References:

- [1] R. ALUR, K. ETESSAMI, AND P. MADHUSUDAN, *A temporal logic of nested calls and returns*, in Proceedings of the 10th International Conference on Tools and Algorithms for the construction and Analysis of Systems (TACAS 04), vol. 2988 of Lecture Notes in Computer Science, Springer, 2004, pp. 467–481.
- [2] R. ALUR AND P. MADHUSUDAN, *Visibly pushdown languages*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC 04), ACM Press, 2004, pp. 202–211.
- [3] M. BROJ, B. JONSSON, J.-P. KATOEN, M. LEUCKER, AND A. PRETSCHNER, eds., *Model-Based Testing of Reactive Systems: Advanced lectures*, no. 3472 in Lecture Notes in Computer Science, Springer, 2005.
- [4] A. R. HOARE, *Communicating Sequential Processes*, Prentice-Hall, 1988.
- [5] A. J. R. G. MILNER, *A Calculus of Communicating Systems*, Springer, 1980.
- [6] J. SRBA, *Visibly pushdown automata: From language equivalence to simulation and bisimulation*, in Annual Conference on Computer Science Logic (CSL 06), vol. 4207 of Lecture Notes in Computer Science, Springer, 2006, pp. 89–103.