Università degli studi di Napoli
"Federico II"

# 2-Visibly Pushdown Automata

Dario Carotenuto, Aniello Murano, Adriano Peron

Preprint n. 53 (2006)

pubblicazioni
del

Dipartimento di Matematica e Applicazioni
"R. Caccioppoli"

Complesso di Monte S. Angelo - Edificio T
Via Cinthia - 80126 NAPOLI

# 2-Visibly Pushdown Automata

Dario Carotenuto, Aniello Murano, and Adriano Peron

Università degli Studi di Napoli "Federico II", Via Cinthia, 80126 - Napoli, Italy

**Abstract.** Visibly Pushdown Automata (*VPA*) are a special case of pushdown machines where the stack operations are driven by the input. In this paper, we consider *VPA with two stacks*, namely *2-VPA*. These automata introduce a useful model to effectively describe concurrent pushdown systems using a simple communication mechanism between stacks. We show that 2-VPA are strictly more expressive than VPA. Indeed, 2-VPA accept some context-sensitive languages that are not context-free and some context-free languages that are not accepted by any VPA. Nevertheless, the class of languages accepted by 2-VPA is closed under all boolean operations and determinizable in ExpTime, but does not preserve decidability of emptiness problem. By adding an ordering constraint on stacks (2-OVPA), decidability of emptiness can be recovered (preserving desirable closure properties) and solved in PTime. Using these properties along with the automata-theoretic approach, we prove that the model checking problem over 2-OVPA models against 2-OVPA specifications is ExpTime-complete.

## 1 Introduction

In the area of formal design verification, one of the most significant developments has been the discovery of the *model checking* technique, that automatically allows to verify on-going behaviors of reactive systems ([CE81, QS81, VW86]). In this verification method (for a survey see [CGP99]), one checks the correctness of a system with respect to a desired behavior by checking whether a mathematical model of the system satisfies a formal specification of this behavior.

Traditionally, model checking is applied to finite-state systems, typically modeled by labeled state-transition graphs. Recently, model checking has been extended to infinite-state sequential systems (e.g., see [Wal96, BMP05]). These are systems in which each state carries a finite, but unbounded, amount of information, e.g., a pushdown store. *Pushdown automata* (PDA) naturally model the control flow of sequential programs with nested and recursive procedure calls. Therefore, PDA are the proper model to tackle with program analysis, compiler optimization, and model checking questions that can be formulated as decision problems for PDA. While many analysis problems, such as identifying dead code and accesses to uninitialized variables, can be captured as regular requirements, many others require inspection of the stack or matching of calls and returns, and are non-regular context-free. More examples of useful non-regular properties are given in [SCFG84], where the specification of unbounded message buffers is considered. Since checking context-free properties on PDA is proved in general to be undecidable [KPV02], weaker models have been proposed to decide different kinds of non-regular properties. One of the most promising approaches is that of

*Visibly Pushdown Automata* (VPA) [AM04]. These are PDA where the push or pop actions on the stack are controlled externally by the input alphabet. Such a restriction on the use of the stack allows to enjoy all desirable closure properties and tractable decision problems, though retaining an expressiveness adequate to formulate program analysis questions (as summarized in Figure 1). Therefore, checking pushdown properties of pushdown models is feasible as long as the calls and returns are made visible. This visibility requirement seems quite natural while writing requirements about pre/post conditions or for inter-procedural flow properties. In particular, requirements that can be verified in this manner include all regular properties, and non-regular properties such as: partial correctness (if $P$ holds when a procedure is invoked, then, if the procedure returns, $P'$ holds upon return), total correctness (if $P$ holds when a procedure is invoked, then the procedure must return and $P'$ must hold at the return state), local properties (the computation within a procedure by skipping over calls to other procedures satisfies a regular property, for instance, every request is followed by a response), access control (a procedure $A$ can be invoked only if another procedure $B$ is in the current stack), and stack limits (whenever the stack size is bounded by a given constant, a property $A$ holds). Unfortunately, some natural context-free properties like "the number of calls to procedures $A$ and $B$ is the same" cannot be captured by any VPA [AM04]. Moreover, VPA cannot explicitly represent concurrency: for instance, properties of two threads running in parallel, each one exploiting its own pushdown store.

In this paper, we propose an extension of VPA in order to enrich with further expressiveness the model though maintaining some desirable closure properties and decidability results. We first consider VPA with an additional, input driven, pushdown store and we call the proposed model *2-Visibly Pushdown Automaton* (2-VPA). As in the VPA case, 2-VPA input symbols are partitioned in subclasses, each of them triggers a transition belonging to a specific class, i.e., push/pop/local transition, which also selects the operating stack, i.e., the first or the second or both. Moreover, visibility in 2-VPA affects the transfer of information from one stack to the other. 2-VPA turn out to be strictly more expressive than VPA and they also accept some context-sensitive languages that are not context-free. Unfortunately, this extension does not preserve decidability of the emptiness problem as it can be proved by a reduction from the halting problem over Minsky Machines [Min67]. In the automata-theoretic approach, to gain with a decidable model checking procedure, decidability of the emptiness problem is crucial. For this reason, we add to 2-VPA a suitable restriction on stack operations, namely we consider 2-VPA in which pop operations on the second stack are allowed only if the first is empty. We call such a variant *ordered* 2-VPA (2-OVPA). The ordering constraint is inspired from the class of *multi-pushdown automata* (MPDA), defined in [BCCR96]. These are pushdown automata exploiting an ordered collection of arbitrary number of pushdown stores in which a pop action on the $i$-th stack can occur only if all previous stacks are empty. In [BCCR96], it has been shown that the class of languages accepted by MPDA is strictly included into context-sensitive languages, it has the emptiness problem decidable, it is closed under union, but not under intersection and complement.

From an expressive point of view, 2-OVPA are a proper subclass of MPDA with two stacks ($PD^2$). Differently from $\mathrm{PD}^2$, exploiting visibility allows to recover in 2-OVPA closure under intersection and complement thus allowing to face the model checking problem following the automata-theoretic approach. In such an approach, to verify whether a system, modeled as a 2-OVPA $S$, satisfies a correctness requirement expressed by a 2-OVPA $P$, we check for emptiness the intersection between the language accepted by $S$ and the complement of the language accepted by $P$ (i.e., $L(S) \cap \overline{L(P)} = \emptyset$). Since intersection and complementation of 2-OVPA can be performed in polynomial and exponential time, respectively, and inclusion for VPA is ExpTime-complete [AM04], we get that model checking an 2-OVPA model against an 2-OVPA specification is ExpTime-complete. This is notable since checking context-free properties on PDA is proved to be undecidable [KPV02] and then also model checking multi-pushdown properties over MPDA is undecidable.

The extension we propose for VPA does not only affect expressiveness, but also gives us a way to naturally describe distributed pushdown systems behavior. In fact, we show that 2-OVPA capture the behavior of systems built on pairs of VPA running in a suitable synchronous way according to a distributed computing paradigm. To this purpose, we introduce a composition operator on VPA parameterized on a communication interface. Given a pair of VPA, this operator allows to build a *Synchronized System* of VPA (S-VPA), which behaves synchronously and in parallel. A communication between two synchronous VPA consists in a transfer of information from the top of the stack of one VPA to the top of stack of the other. If we interpret each one of the involved VPA as a process with its pushdown store (containing activation records of procedure calls, for instance), the enforced communication form can be seen as a *Remote Procedure Call* [ST02], widely exploited in the client-server paradigm of distributed computing. In our case, ordering of VPA modules can be interpreted as follows: we can see the former one acts as a client and the latter as a server. The client can always demand to the server the execution of a task and the server can return a result to the client whenever this is available (its stack is empty).

The properties of languages accepted by 2-VPA and 2-OVPA we obtain along the paper are summarized in Figure 1. Due to page limitations, some proofs are reported in Appendix.

| Languages | Closure Properties | | | Decision problems | |
|---|---|---|---|---|---|
| | $\cup$ | $\cap$ | Complement | Emptiness | Inclusion |
| Regular | Yes | Yes | Yes | Nlogspace | Pspace |
| CFL | Yes | No | No | Ptime | Undecidable |
| VPL | Yes | Yes | Yes | Ptime | ExpTime |
| $\mathcal{L}_{PD^2}$ | Yes | No | No | Ptime | Undecidable |
| 2-VPL | **Yes** | **Yes** | **Yes** | **Undecidable** | **Undecidable** |
| 2-OVPL | **Yes** | **Yes** | **Yes** | **Ptime** | **ExpTime** |

**Fig. 1.** A comparison between closure properties and decision problems.

## 2 Preliminaries

Let $\Sigma$ be a finite alphabet partitioned into three pairwise disjoint sets $\Sigma_c$, $\Sigma_r$, and $\Sigma_l$ standing respectively for *call*, *return*, and *local* alphabets. We denote the tuple $\widetilde{\Sigma} = \langle \Sigma_c, \Sigma_r, \Sigma_l \rangle$ a *visibly pushdown alphabet*. A *(nondeterministic) visibly pushdown automaton* (*VPA*) on finite words over $\widetilde{\Sigma}$ [AM04] is a tuple $M = (Q, Q_{in}, \Gamma, \bot, \delta, Q_F)$, where $Q$, $Q_{in}$, $Q_F$, and $\Gamma$ are respectively finite sets of *states*, *initial states*, *final states*, and *stack symbols*; $\bot \notin \Gamma$ is the stack *bottom* symbol and we use $\Gamma_\bot$ to denote $\Gamma \cup \{\bot\}$; and $\delta \subseteq \delta_c \cup \delta_r \cup \delta_l$, is the *transition relation* where $\delta_c = Q \times \Sigma_c \times Q \times \Gamma$, $\delta_r = Q \times \Sigma_r \times \Gamma_\bot \times Q$, and $\delta_l = Q \times \Sigma_l \times Q$. We call $(q, a, q', \gamma) \in \delta_c$ a *push transition*, where on reading $a$ the symbol $\gamma$ is pushed onto the stack and the control state changes to $q'$; $(q, a, \gamma, q') \in \delta_r$ a *pop transition*, where $\gamma$ is popped from the stack leading to the control state $q'$; and $(q, a, q') \in \delta_l$ a *local transition*, where the automaton on reading $a$ only changes its control to $q'$. A *configuration* for a VPA $M$ is a pair $(q, \sigma) \in Q \times (\Gamma^*.\bot)$ where $\sigma$ is the stack content. A *run* $\rho = (q_0, \sigma_0)...(q_k, \sigma_k)$ of $M$ on a word $w = a_1...a_k$ is a sequence of configurations such that $q_0 \in Q_{in}$, $\sigma_0 = \bot$, and for every $i \in \{0, ..., k\}$, one of the following holds: [**Push**]: $(q_i, a_i, q_{i+1}, \gamma) \in \delta_c$, and $\sigma_{i+1} = \gamma.\sigma_i$; [**Pop**]: $(q_i, a_i, \gamma, q_{i+1}) \in \delta_r$, and either $\gamma \in \Gamma$ and $\sigma_i = \gamma.\sigma_{i+1}$, or $\gamma = \sigma_i = \sigma_{i+1} = \bot$; or [**Local**]: $(q_i, a_i, q_{i+1}) \in \delta_l$ and $\sigma_{i+1} = \sigma_i$.

A run is *accepting* if its last configuration contains a final state. The language accepted by a VPA $M$ is the set of all words $w$ with an accepting run of $M$ on $w$, say it *L(M)*. A language of finite words $L \subseteq \Sigma^*$ is a *visibly pushdown language* (*VPL*) with respect to a pushdown alphabet $\widetilde{\Sigma}$, if there is a VPA $M$ such that $L = L(M)$. VPLs are a subclass of deterministic context-free languages, a superclass of regular languages, and are closed under intersection, union, complementation, concatenation, and Kleene-$^*$. Furthermore, the emptiness problem for a VPA $M$, i.e., deciding whether $L(M) \neq \emptyset$, is decidable with time complexity $O(n^3)$, where $n$ is the number of states in $M$.

In the literature, different extensions of classical pushdown automata with multiple stacks have been considered. Here, we recall *multiple-pushdown automata* as they were introduced in [BCCR96]. These machines are pushdown automata endowed with an ordered set of an arbitrary number of stacks and the constraint that pop operations occur sequentially and only operate on the first non-empty stack. Thus, push operations are never constrained and they can be performed independently on every stack. The formal definition follows.

A multi-pushdown automaton with $n \geq 1$ stacks ($PD^n$, for short) is a tuple $M = (\Sigma,\ Q,\ Q_{in},\ \Gamma,\ Z_0,\ \delta,\ Q_F)$, where $\Sigma$, $Q$, $Q_{in}$, $\Gamma$, and $Q_F$ are respectively finite sets of input symbols, states, initial states, stack symbols, and final states, $Z_0 \notin \Gamma$ is the *bottom stack symbol* and used to identify the initial non-empty stack, and $\delta$ is the transition relation defined as a partial function from $Q \times \Sigma \cup \{\varepsilon\} \times \Gamma$ to $2^{Q \times (\Gamma^*)^n}$. If $(q', \alpha_1, \ldots, \alpha_n) \in \delta(q, a, \gamma)$, on reading $a$ the automaton changes its control state from $q$ to $q'$, the stack symbol $\gamma \in \Gamma$ is popped from the first non-empty stack, and for each $i$ in $\{1, \ldots, n\}$, and $\alpha_i \in \Gamma^*$ is pushed on the $i$-th stack. A configuration of $M$ is a $n+2$-tuple $\langle q, x; \sigma^0, \ldots, \sigma^n \rangle$, where $q \in Q$, $x \in \Sigma^*$, $\sigma^0, \ldots, \sigma^n \in \Gamma^*$, and $\sigma^i$ is the content of the $i$-th stack. The above configuration is initial if $q = q_0$, $\sigma^0 = Z_0$, and all other stacks are empty, and it is final if $q \in F$. The transition relation $\vdash_M$ over configurations is defined in the following way: $\langle q, ax; \varepsilon, \ldots,\ \varepsilon, \gamma.\gamma_i, \ldots, \gamma_n \rangle \vdash_M \langle q', x; \alpha_1, \ldots, \alpha_{i-1}, \alpha_i\gamma_i, \ldots, \alpha_n\gamma_n \rangle$ if $(q', \alpha_1, \ldots,\ \alpha_n) \in \delta(q, a, \gamma)$. A word $w$ is accepted by a PD$^n$ $M$ *iff* $\langle q, w; Z_0, \varepsilon \ldots, \varepsilon \rangle \vdash_M^* \langle q_F, \varepsilon; \gamma_1, \ldots, \gamma_n \rangle$, where $\vdash_M^*$ is the kleene-closure of $\vdash_M$ and $q_F \in Q_F$. The language of a PD$^n$ $M$ is the set of words accepted by $M$. We denote the class of languages accepted by PD$^n$ as $\mathcal{L}_{PD^n}$. The following theorem summarizes the main results about PD$^n$.

**Theorem 1.** [**BCCR96**] *For every $n \geq 1$, we have that $\mathcal{L}_{PD^n}$ subsumes CFLs, it is strictly included in CSLs as well as in $\mathcal{L}_{PD^{n+1}}$. It is closed under union, concatenation and Kleene-$^*$. Moreover, it has a decidable emptiness problem and solvable in $O(|Q|^3)$, where $|Q|$ is the number of states of the automaton.*

## 3 Visibly Pushdown Automata with two Stacks

A *2-pushdown alphabet* is a pair of pushdown alphabets $\widetilde{\Sigma} = \langle \widetilde{\Sigma}^0, \widetilde{\Sigma}^1 \rangle$, where $\widetilde{\Sigma}^0 = \langle \Sigma_c^0, \Sigma_r^0, \Sigma_l^0 \rangle$ and $\widetilde{\Sigma}^1 = \langle \Sigma_c^1, \Sigma_r^1, \Sigma_l^1 \rangle$ are a possibly different partitioning of the same input alphabet $\Sigma$. The intuition is that the $\widetilde{\Sigma}^0$ drives the operations over the first stack and $\widetilde{\Sigma}^1$ those over the second. Symbols in $\widetilde{\Sigma}$ belonging to call, return or local partitions of both $\widetilde{\Sigma}^0$ and $\widetilde{\Sigma}^1$ are simply denoted by $\Sigma_c, \Sigma_r, \Sigma_l$, respectively. Furthermore, input symbols that drive a call operation on the first (resp., second) stack and a return on the second (resp., first) stack are called *synchronized communication* symbols and formally denoted as $\Sigma_{s_1} = \Sigma_c^0 \cap \Sigma_r^1$ (resp., $\Sigma_{s_0} = \Sigma_r^0 \cap \Sigma_c^1$). Finally, we denote with $\Sigma_{c_i}$ (resp., $\Sigma_{r_i}$) the set of call (resp., return) symbols for the stack $i$ and local for the other, with $i = 0, 1$. In the following, we use $\widetilde{\Sigma}$ to denote both a 2-pushdown alphabet and a (1-)pushdown alphabet, when the meaning is clear from the context.

**Definition 1 (2-Visibly Pushdown Automaton).** *A (nondeterministic) 2-Visibly Pushdown Automaton (2-VPA) on finite words over a 2-pushdown alphabet $\widetilde{\Sigma}$ is a tuple $M = (Q, Q_{in}, \Gamma, \bot, \delta, Q_F)$, where $Q, Q_{in}, Q_F$, and $\Gamma$ are respectively finite sets of states, initial states, final states and stack symbols, $\bot \notin \Gamma$ is the stack* bottom *symbol (with $\Gamma_\bot$ used to denote $\Gamma \cup \{\bot\}$), and $\delta$ is the transition relation defined as the union of the following sets, for $i \in \{0, 1\}$:*

- $\delta_{c_i} \subseteq (Q \times \Sigma_{c_i} \times Q \times \Gamma)$,
- $\delta_c \subseteq (Q \times \Sigma_c \times Q \times \Gamma \times \Gamma)$,
- $\delta_{s_i} \subseteq (Q \times \Sigma_{s_i} \times \Gamma_\bot \times Q \times \Gamma)$,
- $\delta_{r_i} \subseteq (Q \times \Sigma_{r_i} \times \Gamma_\bot \times Q)$,
- $\delta_r \subseteq (Q \times \Sigma_r \times \Gamma_\bot \times \Gamma_\bot \times Q)$,
- $\delta_l \subseteq Q \times \Sigma_l \times Q$.

We say that $M$ is deterministic if $Q_{in}$ is a singleton, and for every $q \in Q$, $a \in \Sigma$, and $\gamma \in \Gamma_\bot$, there is at most one transition of the form $(q, a, q')$, $(q, a, q', \gamma)$, $(q, a, q', \gamma, \gamma')$, $(q, a, \gamma, q')$, $(q, a, \gamma, \gamma', q')$, or $(q, a, \gamma, q', \gamma')$ belonging to $\delta$.

Transitions in $\delta_l$, $\delta_{c_i}$, and $\delta_{r_i}$ extend VPA's local, call, and return transitions to deal with two stacks, in a natural way. We call $(q, a, q', \gamma, \gamma') \in \delta_c$ a *double-call* transition where on reading $a$ the automaton changes its control state from $q$ to $q'$, and the symbols $\gamma$ and $\gamma'$ are pushed on the first and second stack, respectively; we call $(q, a, \gamma, \gamma', q') \in \delta_r$ a *double-pop* transition where on reading $a$ the automaton changes its control state from $q$ to $q'$, and the symbols $\gamma$ and $\gamma'$ are popped from the first and second stack, respectively; finally, we call $(q, a, \gamma, q', \gamma') \in \delta_{s_i}$, with $i \in \{0, 1\}$, a *synchronous (communication)* transition between stacks, where on reading $a$ the automaton changes its control state from $q$ to $q'$ and the symbol $\gamma$ is popped from the stack $i$ and $\gamma'$ pushed on the other.

A *configuration* of a 2-VPA $M$ is a triple $(q, \sigma^0, \sigma^1)$ where $q \in Q$ and $\sigma^0, \sigma^1 \in \Gamma^*.\bot$. For an input word $w = a_1 \dots a_k \in \Sigma^*$, a run of $M$ on $w$ is a sequence

$\rho = (q_0, \sigma_0^0, \sigma_0^1)...(q_k, \sigma_k^0, \sigma_k^1)$ where $q_0 \in Q_{in}$, $\sigma_0^0 = \sigma_0^1 = \bot$, and for all $i \in \{0, ..., k-1\}$, there are $j, j' \in \{0, 1\}$, $j \neq j'$, such that one of the following holds:

**Push**: $(q_i, a_i, q_{i+1}, \gamma) \in \delta_{c_j}$, then $\sigma_{i+1}^j = \gamma.\sigma_i^j$ and $\sigma_{i+1}^{j'} = \sigma_i^{j'}$;

**2Push**: $(q_i, a_i, q_{i+1}, \gamma, \gamma') \in \delta_c$ then $\sigma_{i+1}^j = \gamma.\sigma_i^j$ and $\sigma_{i+1}^{j'} = \gamma'.\sigma_i^{j'}$;

**Pop**: $(q_i, a_i, \gamma, q_{i+1}) \in \delta_{r_j}$, then either $\gamma = \sigma_i^j = \sigma_{i+1}^j = \bot$, or $\gamma \neq \bot$ and $\sigma_i^j = \gamma.\sigma_{i+1}^j$. In both cases $\sigma_{i+1}^{j'} = \sigma_i^{j'}$;

**2Pop**: $(q_i, a_i, \gamma_0, \gamma_1, q_{i+1}) \in \delta_r$ then, for $k \in \{0, 1\}$, either $\gamma_k = \sigma_i^k = \sigma_{i+1}^k = \bot$, or $\gamma_k \neq \bot$ and $\sigma_i^k = \gamma.\sigma_{i+1}^k$;

**Local**: $(q_i, a_i, q_{i+1}) \in \delta_l$ then $\sigma_{i+1}^0 = \sigma_i^0$ and $\sigma_{i+1}^1 = \sigma_i^1$;

**Synch**: $(q_i, a_i, \gamma, q_{i+1}, \hat{\gamma}) \in \delta_{s_j}$ then either $\gamma = \sigma_i^j = \sigma_{i+1}^j = \bot$, or $\gamma \neq \bot$ and $\sigma_i^j = \gamma.\sigma_{i+1}^j$. In both cases $\sigma_{i+1}^{j'} = \hat{\gamma}.\sigma_i^{j'}$.

From the above definition, we notice that communication between stacks is only allowed by applying a *synch.* transition. For a configuration $c$, we write $c \vdash_M c'$ meaning that $c'$ is obtained from $c$ by applying one of the rules above. We omit $M$ when it is clear from the context. A run $\rho$ is *accepting* when it ends with a configuration containing a final state. A word $w$ is *accepted* if there is an accepting run $\rho$ of $M$ on $w$. The language accepted by $M$, denoted by $L(M)$, is the set of all words accepted by $M$. A language $L \subseteq \Sigma^*$ is a 2-VPL with respect to $\widetilde{\Sigma}$ if there is a 2-VPA $M$ over $\widetilde{\Sigma}$ such that $L(M) = L$.

**Theorem 2.** *The emptiness problem for 2-VPA is undecidable.*

*Proof.* **[sketch]** We prove the result by showing a reduction from the halting problem of two counters Minsky machines. A Minsky machine with two counters $C_0$ and $C_1$ is a finite sequence $M = (L_1 : I_1; L_2 : I_2; \ldots; L_n : \texttt{halt})$ where $n \geq 1$, $L_1, \ldots, L_n$ are pairwise different instruction labels, and $I_1, \ldots, I_n$ are instructions of type *increment*, i.e., $C_m := C_m + 1;$ $\texttt{goto}\ L_j$, or of type *test and decrement*, i.e., $\texttt{if}\ C_m = 0\ \texttt{then goto}\ L_j\ \texttt{else}\ C_m := C_m - 1;\ \texttt{goto}\ L_k$, where $0 \leq m \leq 1$ and $1 \leq j, k \leq n$. A configuration of $M$ is a triple $(L_i, v_0, v_1)$ where $L_i$ is an instruction label, and $v_0, v_1 \in \mathbb{N}$ represent the values of the counters $C_0$ and $C_1$, respectively. Let *Conf* be the set of all configurations of $M$, the transition relation $\hookrightarrow \subseteq Conf \times Conf$ between configurations is defined in an obvious way, and $\hookrightarrow^*$ is the transitive and reflexive closure of $\hookrightarrow$. If $(L_1, 0, 0) \hookrightarrow \ldots \hookrightarrow (L_j, v_j^0, v_j^1)$ holds for a Minsky machine $M$, we say that $(L_1, 0, 0) \ldots (L_j, v_j^0, v_j^1)$ is an execution trace for $M$. The halting problem for $M$ is to decide whether there exist $v_0, v_1 \in \mathbb{N}$ such that $(L_1, 0, 0) \hookrightarrow^* (L_n, v_0, v_1)$. This problem is known to be undecidable [Min67].

We now prove that given a two counters Minsky machine $M$ there exists a 2-VPA $M'$ over $\widetilde{\Sigma}$ such that $L(M') \neq \emptyset$ iff $M$ eventually halts. Let $M = (L_1 : I_1; L_2 : I_2; \ldots; L_n : \texttt{halt})$, we define $M' = (Q, Q_{in}, \Gamma, \bot, \delta, Q_F)$ such that $Q = \{L_1, \ldots, L_n\}$, $Q_{in} = \{L_1\}$, $\Gamma = \{A\}$, where $A$ does not appear in $M$, $Q_F = \{L_n\}$, and $\widetilde{\Sigma}$ is the partitioned set of all instructions $I_i$, with $i = 1, \ldots, n$, such that $I_i \in \Sigma_{c_0}$ (resp., $I_i \in \Sigma_{c_1}$) if $I_i$ is an increment instruction of the counter $C_0$ (resp., $C_1$), or $I_i \in \Sigma_{r_0}$ (resp., $I_i \in \Sigma_{r_1}$) if $I_i$ is a test and decrement instruction over the counter $C_0$ (resp., $C_1$). Finally, $\delta$ is defined as follows: if $I_i$ is an increment instruction such as $C_m := C_m + 1;$ $\texttt{goto}\ L_j$, with $m \in \{0, 1\}$, then $(L_i, I_i, L_j, A) \in \delta_{c_m}$; otherwise, if $I_i$ is a test and decrement instruction such as `if` $C_m = 0$ `then goto` $L_j$ `else` $C_m := C_m - 1$; `goto` $L_k$, with $m \in \{0,1\}$ then $(L_i, I_i, \bot, L_j), (L_i, I_i, A, L_k) \in \delta_{r_m}$. It remains to prove that $M$ halts iff $M'$ accepts a word. It is easy to show by induction the following assertion:

Given a sequence of numbers $s = s_1 s_2 \ldots s_k$, with $s_i \in \{1, \ldots, n\}$ for all $i \in \{1, \ldots, k\}$, the sequence $(L_{s_1}, v^0_{s_1}, v^1_{s_1}) \ldots (L_{s_k}, v^0_{s_k}, v^1_{s_k})$ of elements from $\{L_1, \ldots L_n\} \times \mathbb{N} \times \mathbb{N}$ is an execution trace of $M$ if and only if the sequence $(L_{s_1}, \sigma^0_{s_1}, \sigma^1_{s_1}) \ldots (L_{s_k}, \sigma^0_{s_k}, \sigma^1_{s_k})$ of elements from $Q \times \Gamma^*.\bot \times \Gamma^*.\bot$ is a run of $M'$, with $|\sigma^j_{s_i}| = v^j_{s_i} + 1$ for each $i \in \{1, \ldots, k\}$ and $j \in 0, 1$.

This implies that $(L_{s_1}, v^0_{s_1}, v^1_{s_1}) \ldots (L_{s_k}, v^0_{s_k}, v^1_{s_k})$ is an halting execution trace of $M$ iff $(L_{s_1}, \sigma^0_{s_1}, \sigma^1_{s_1}) \ldots (L_{s_k}, \sigma^0_{s_k}, \sigma^1_{s_k})$ is an accepting run of $M'$ over $I_{s_1}, \ldots, I_{s_{k-1}}$, with $|\sigma^j_{s_i}| = v^j_{s_i} + 1$ for each $i \in \{1, \ldots, k\}$, since $L_{s_k} = L_n$ and $L_n$ is final for $M'$. □

It is interesting to notice that the reduction we consider in the proof of Theorem 2 also applies to the restricted model of VPA with 2 stacks where operations acting simultaneously on both stacks are avoided. This follows from the fact that two counters Minsky machine instructions only involves one counter at a time, which leaves empty the sets $\Sigma_c$, $\Sigma_r$ and $\Sigma_{s_i}$, with $i \in \{0, 1\}$.

## 4 Ordered Visibly Pushdown Automata with Two Stacks

In this section, we consider the subclass of 2-VPA which enforces the ordering constraints on using pushdown stores as defined for MPDA. In more detail, we consider a class of *ordered 2-VPA* (*2-OVPA*) as the class of 2-VPA in which a pop operation on the second stack can occur only if the first stack is empty. Thus, in such a model simultaneous pop operations are not allowed. The formal definition of 2-OVPA follows.

**Definition 2.** *A 2-OVPA $M$ over $\widetilde{\Sigma}$ is a 2-VPA such that $\Sigma_r$ is empty and for all input word $w = a_1 \ldots a_k \in \Sigma^*$ and run $\rho = (q_0, \sigma^0_0, \sigma^1_0) \ldots (q, \sigma^0_k, \sigma^1_k)$ of $M$ over $w$, for all $i \in \{1, \ldots, n\}$, the following hold:*
**Pop**: *$(q_i, a_i, \gamma, q_{i+1}) \in \delta_{r_1}$ then $\sigma^0_i = \sigma^0_{i+1} = \bot$ and $\sigma^1_{i+1} = \gamma.\sigma^1_i$*
**Synch**: *$(q_i, a_i, \gamma, q_{i+1}, \hat{\gamma}) \in \delta_{s_1}$ then $\sigma^0_i = \bot$ and $\sigma^0_{i+1} = \hat{\gamma}.\bot$ and $\sigma^1_{i+1} = \gamma.\sigma^1_i$.*

Directly from the fact that 2-OVPA are a subclass of MPDA and the fact that for MPDA the emptiness is solvable in cubic time, we get the following.

**Corollary 1.** *Given a 2-OVPA $M$, deciding whether $L(M) \neq \emptyset$ is solvable in $O(n^3)$, where $n$ is the number of states in $M$.*

While dealing with automata, one interesting question is whether the acceptance power increases while using $\varepsilon$-moves, i.e., transitions that allow to change the state without consuming any input. Here we investigate 2-VPA with the ability of performing a restricted form of $\varepsilon$-moves: we only enable $\varepsilon$-moves on reading the top of the stack symbols on a local action. More formally, the variant 2-VPA$_\varepsilon$ of 2-VPA we consider is obtained by replacing $\delta_l$ in Definition 1 with a subset of $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \times \Gamma \times Q$ and by substituting the **Local** rule in the definition of a run for 2-VPA with the following:
**Local**$_\varepsilon$: $a_i \in \Sigma_l \cup \{\varepsilon\}$ and there exists $(q_i, a_i, \gamma^0, \gamma^1, q_{i+1}) \in \delta$ such that $\sigma^j_i = \sigma^j_{i+1} = \gamma^j.\sigma^j$, for all $j \in \{0, 1\}$.

Since at each step, a 2-VPA$_\varepsilon$ can now choose whether to consume an input symbol or take an $\varepsilon$-move, we consider the run definition modified accordingly. In the following theorem, we show that 2-VPA and 2-VPA$_\varepsilon$, as well as 2-OVPA and 2-OVPA$_\varepsilon$, are expressively equivalent.

**Theorem 3.** *$L \in$2-VPL iff $L \in$ 2-VPL$_\varepsilon$ and $L \in$2-OVPL iff $L \in$2-OVPA$_\varepsilon$.*

We conclude the section with an example of a language accepted by a 2-OVPA$_\varepsilon$.

*Example 1.* Let $L_1 = \{a^n b^n c^n \mid \exists n \in \mathbb{N}\ \}$. We show a 2-OVPA$_\varepsilon$ $M$ accepting $L_1$. The alphabet $\widetilde{\Sigma}$ we use for $M$ is partitioned in $\Sigma_{c_0} = \{a\}$, $\Sigma_{s_0} = \{b\}$, and $\Sigma_{r_1} = \{c\}$ (i.e., all the other partition elements are empty). The automaton is the following $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$, with $Q = \{q_0, q_1, q_2, q_3, q_F\}$, $Q_{in} = \{q_0\}$, $Q_F = \{q_0, q_F\}$, $\Gamma = \{A, B\}$ and $\delta = \{(q_0, a, q_1, A)$, $(q_1, a, q_1, A)$, $(q_1, b, A, q_2, B)$, $(q_2, b, A, q_2, B)$, $(q_2, \varepsilon, \perp, B, q_3)$, $(q_3, c, B, q_3)$, $(q_3, \varepsilon, \perp, \perp, q_F)\}$. The 2-OVPA$_\varepsilon$ $M$ is depicted in Figure 2, where we adopt the following conventions to represent arcs: for a local transition such as $(q_i, a, A, B, q_j)$ we label the arc between $q_i$ and $q_j$ as $a, (A, B)$; for a synch transition such as $(q_i, a, A, q_j, B)$ we label the arc as $s, A \rightarrow B$, if $a \in \Sigma_{s_0}$, and as $s, A \leftarrow B$ , otherwise; moreover a push or pop transition is labeled like a synch transition but with one part missing. For example, a pop from the second stack $(q_i, a, B, q_j)$ is labeled as $a, * \leftarrow B$.

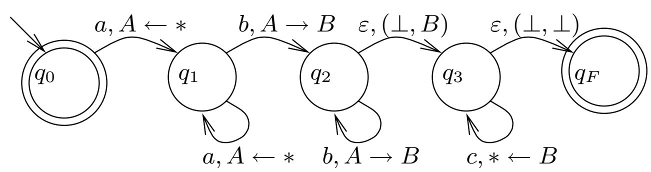


**Fig. 2.** A 2-OVPA$_\varepsilon$ accepting $L_1 = \{a^n b^n c^n | \exists n \in \mathbb{N}\}$

## 5 Expressiveness and Closure Properties

In this section, we compare 2-VPLs and 2-OVPLs with VPLs [AM04], deterministic and (nondeterministic) context-free languages (resp., DCFLs and CFLs) [HU79], and multi-pushdown languages [BCCR96] ($\mathcal{L}_{PD^n}$). Recall that the following chain of strict inclusions holds: VPLs $\subset$ DCFls $\subset$ CFls $\subset \mathcal{L}_{PD^2} \subset$ CSLs.

**Theorem 4.** *The following assertions hold:*
***a)*** *2-OVPLs $\subset$ 2-VPLs;* ***b)*** *VPLs $\subset$ 2-OVPLs;* ***c)*** *VPLs $\subset$ 2-VPLs;*
***d)*** *DCFLs $\setminus$ 2-VPLs $\neq \emptyset$;* ***e)*** *DCFLs $\setminus$ 2-OVPLs $\neq \emptyset$;*
***f)*** *2-VPLs $\cap$ CFLs $\setminus$ VPLs $\neq \emptyset$;* ***g)*** *2-OVPLs $\subset \mathcal{L}_{PD^2}$;* ***h)*** *2-OVPLs $\subset$ CSLs.*

Although 2-VPLs and 2-OVPLs are strictly more expressive than VPLs, we show that they preserve union, intersection, complementation (and thus inclusion). These properties, along with the property that the emptiness problem for 2-OVPA is solvable in **Ptime**, make 2-OVPA a powerful engine for system verification using the automata-theoretic approach. We recall that 2-VPA and MPDA do not support such an approach since MPDA does not enjoy closure under intersection and complementation, and for 2-VPA the emptiness problem is undecidable.

**Theorem 5. (Closure Properties)** *Let $L_1$ and $L_2$ be two 2-VPLs (resp., 2-OVPLs) with respect to the same $\widetilde{\Sigma}$. Then, $L_1 \cap L_2$, $L_1 \cup L_2$ are 2-VPLs (resp., 2-OVPLs) over $\widetilde{\Sigma}$. Also, $L_1 \cdot L_2$, and $L_1^*$ are 2-VPLs over $\widetilde{\Sigma}$. Furthermore, all the mentioned operations can be performed in polynomial-time.*

The closure of 2-VPA and 2-OVPA under complementation can be proved as an immediate consequence of determinization.

**Theorem 6 (Determinization).** *Given a 2-VPA (resp., 2-OVPA) $M$ over $\widetilde{\Sigma}$, there is a deterministic 2-VPA (resp., deterministic 2-OVPA) $M'$ over $\widetilde{\Sigma}$ such that $L(M) = L(M')$. Moreover, if $M$ has $n$ states, we can construct $M'$ with $O(2^{2n^2})$ states and $O(2^{n^2} \cdot |\Sigma|)$ stack symbols.*

*Proof.* [**sketch**] The proof we present is inspired from that given in [AM04] for VPA. There, the main idea is to do a subset construction, postponing handling push transitions. The push transitions are stored into the stack and simulated later, namely at the time of the matching pop transitions. The construction has two components: a set of *summary edges* $S$, that keeps track of what state transitions are possible from a push transition to the corresponding pop transition, and a set of *path edges* $R$, that keeps track of all possible state reached from an initial state. In our case, we have to handle two stacks and the communication mechanism. Therefore, we have to use two summary edges sets $S_0$ and $S_1$, and, in order to manage the communication transitions, we augment the structure of states adding information about the top of the stacks. Let $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$ be a 2-VPA (resp., 2-OVPA) over $\widetilde{\Sigma}$. We define a deterministic 2-VPA (resp., deterministic 2-OVPA) $M' = (Q', Q'_{in}, \Gamma', \delta', Q'_F)$ over $\widetilde{\Sigma}$ such that $L(M) = L(M')$. Let $Q' = \mathcal{P}(Q \times Q) \times \mathcal{P}(Q \times Q) \times \mathcal{P}(Q)$. $Id_Q$ is the set $\{(q, q) \mid q \in Q\}$, and $Q'_{in} = \{(Id_Q, Id_Q, Q_{in})\}$. The stack alphabet $\Gamma'$ is the set of elements $(S, R, a)$, where $(S, R) \in \mathcal{P}(Q \times Q) \times \mathcal{P}(Q)$ and $a \in (\bigcup_{i \in \{0,1\}} \Sigma_{c_i} \cup \Sigma_{s_i}) \cup \Sigma_s$. The set of final states is $Q'_F = \{(S_0, S_1, R) \mid \; R \cap Q_F \neq \emptyset\}$. We give an idea of $\delta'$ by means of the following example, referring to the Appendix for the formal definition. Let $w = w_1 c_1^0 w_2 c_1^1 w_3$ be an input word, where in $w_1$ each push, either into the first or into the second stack, is matched by a pop, but there may be unmatched pop transitions; $w_2$ and $w_3$ are words in which all push and pop transitions are matched for both stacks; $c_1^0$ and $c_1^1$ are push, the former for the first stack and the latter for the second. In $M'$, after reading $w$, the first stack is $(S_0, R_0, c_1^0).\bot$, the second stack is $(S_1, R_1, c_1^1).\bot$, and the control state is $(S''_0, S''_1, R'')$. $S_0$ contains all the pair of states $(q, q')$ such that the 2-VPA (resp., 2-OVPA) $M$ can go from $q$ with first stack empty to $q'$ with first stack empty on reading $w_1$. Analogously, $S_1$ contains all the pairs $(q, q')$ such that $M$ can go from $q$ with second stack empty to $q'$ with second stack empty on reading $w_1 c_1^0 w_2$. $R_0$ and $R_1$ are the sets of all states reachable by $M$ from an initial state on reading $w_1$ and $w_1 c_1^0 w_2$, respectively. $S''_0$ and $S''_1$ are the current summaries for the first and second stack, respectively, and $R''$ is the set of all states reachable by $M$ from an initial state on reading $w$. In this construction, we maintain as an invariant such a property of the stacks and control state. Now, after reading $w$, if $M$ reads a push symbol $a$ operating on the first stack, stacks and control state change as follows: the triple $(S''_0, R'', a)$ is pushed on the first stack, the second stack remains the same, and the new control state is $(Id_Q, S'_1, R')$ where $Id_Q$ is the initialization summary and $S_1'$ and $R'$ are updated (path and summary edges are extended) accordingly to all possible $\delta$ transitions. If a local transition symbol occurs, then only the control state is affected, which changes accordingly to $\delta$ transitions. If a pop symbol $a$ occurs after reading $w$, $M'$ pops $(S_0, R_0, c_1^0)$ from the first stack and it updates $S_0$ and $R_0$, using the current summary $S_0''$ along with a push transition on $c_1^0$ and a corresponding pop transition on $a$. If a synchronization symbol $a$ from the first to the second stack occurs, $M$ has to combine push and pop operations as above to update its stacks and control state. □

The closure under complementation for 2-VPLs and 2-OVPLs follows from Theorem 5.

**Corollary 2. (Closure under complementation)** *Let $L \in$ 2-VPLs (resp., 2-OVPLs) over $\widetilde{\Sigma}$, then $\Sigma^* \setminus L \in$ 2-VPLs (resp., 2-OVPLs) over $\widetilde{\Sigma}$.*

## 6 Model Checking and Synchronized Systems of VPA

A model checking procedure verifies the correctness of a system with respect to a desired behavior by checking whether a mathematical model of the system satisfies a formal specification of this behavior. Here, we consider the case whether both the model of the system and the formal specification of the required behavior are given by VPA with two stacks, say them $M$ and $P$, respectively. The automata-theoretic approach to model checking exploits the combination of closure properties and emptiness decidability: checking whether $M$ *satisfies* $P$ is reduced to check whether $L(M) \cap \overline{L(P)} = \emptyset$ (all the runs of the model $M$ satisfy the behavioral property represented by $P$).

Recall that the emptiness problem for 2-OVPA is solvable in cubic time (Corollary 1). Since determinization for 2-OVPA is in ExpTime (Theorem 6), and intersection can be done in polynomial-time (Theorem 5), we get an ExpTime algorithm to solve the model checking problem. The completeness follows from the fact that VPA model checking is ExpTime-complete [AM04].

**Theorem 7.** *The model checking problem for 2-OVPA is* ExpTime*-complete.*

In the remaining part of this section we show that 2-OVPA gives a natural way to describe distributed pushdown systems. In fact, we show that 2-OVPA capture the behavior of systems built on pairs of VPA working in a suitable synchronous way according to distributed computing paradigm. To this purpose, we introduce an operator of synchronous composition on VPA that allows to build a Synchronized System of VPA from a pair of VPA $M_0$ and $M_1$. The automata $M_0$ and $M_1$ run independently on the same input so that each input symbol can drive different transitions on the two, that is a local transition for the former and a push transition for the latter. Only communications between $M_0$ and $M_1$ have to be synchronized in accordance with a relation $\lambda$ (a parameter of the synchronous composition operator) that contains all the transitions that are push transitions for the one and pop transitions for the other. The idea is that $\lambda$ contains all the pairs of transitions on which the two VPA are allowed to communicate. The only constraint on the pushdown alphabets is that an input symbol can not trigger a pop transition on both VPA. Moreover, we have to prevent that $M_1$ can pop whenever $M_0$ has a non-empty stack, and thus every pop transition of $M_1$ is synchronized with $M_0$. Two VPA $M_0$ and $M_1$ over $\widetilde{\Sigma}^0$ and $\widetilde{\Sigma}^1$, respectively, are *synchronizable* if $\Sigma^0 = \Sigma^1$ and $\Sigma^0_r \cap \Sigma^1_r$ is empty.

**Definition 3 (Synchronized Systems of VPA).** *A Synchronized System of VPA (S-VPA) $M_0||_\lambda M_1$ is a pair of synchronizable VPA $M_0$ and $M_1$ over $\widetilde{\Sigma}^0$ and $\widetilde{\Sigma}^1$, respectively, together with a communication relation $\lambda \subseteq \delta^0_c \times \delta^1_r \cup \delta^0_r \times \delta^1_c$, where $\delta^0$ and $\delta^1$ are the transition relations of $M_0$ and $M_1$, respectively.*

A run $\rho$ on $w = a_1...a_n \in (\Sigma^0 \cup \Sigma^1)^*$ for $M_0||_\lambda M_1$ is a pair of VPA runs on $w$, $\pi^0 = (q^0_0, \bot)(q^0_1, \sigma^0_1)...(q^0_n, \sigma^0_n)$ for $M_0$ and $\pi^1 = (q^1_0, \bot)(q^1_1, \sigma^1_1)...(q^1_n, \sigma^1_n)$ for $M_1$ such that, for all $k \in \{0, \ldots, n-1\}$, , where $t^0_k$ is the transition applied from $(q^0_k, \sigma^0_k)$ to $(q^0_{k+1}, \sigma^0_{k+1})$ in $M_0$, and $t^1_k$ is the transition applied from $(q^1_k, \sigma^1_k)$ to $(q^1_{k+1}, \sigma^1_{k+1})$ in $M_1$, such that if $t^1_k$ is a pop transition then $\sigma^0_k$ is empty and if $(t^0_k, t^1_k) \in \delta^0_c \times \delta^1_r \cup \delta^0_r \times \delta^1_c$ then $(t^0_k, t^1_k) \in \lambda$. A run $\rho$ is accepting if both $\pi^0$ and $\pi^1$ are accepting and thus $w$ is accepted. $L(M_0||_\lambda M_1)$ is the set of words accepted by $M_0||_\lambda M_1$. From Definition 3, it follows that $L(M_0||_\lambda M_1) \subseteq L(M_0) \cap L(M_1)$. Next theorem states that 2-OVPA are more expressive than S-VPA.

**Theorem 8.** *Let $M_0||_\lambda M_1$ be a S-VPA over $\widetilde{\Sigma}^0$, $\widetilde{\Sigma}^1$, then $L(M_0||_\lambda M_1)$ is a 2-OVPL with respect to $\widetilde{\Sigma} = \langle \widetilde{\Sigma}^0, \widetilde{\Sigma}^1 \rangle$.*

We give an evidence of the power of the introduced S-VPA by means of an example of a system behaving in a context-sensitive way. Consider a client-server system of pushdown processes described by a pair of synchronized VPA (see Figure 3) behaving in the following way: first, the client collects in its pushdown store an ordered pool of jobs on reading a sequence of input $job_i \in JobSet$; after that, the client transfers ($rcall$) the whole ordered sequence of jobs to the server; then the server dispatches to the client a solution for each job ($solve$) in the same order the client has collected the jobs; moreover, the server waits a special commitment from the client ($returnSol_j$) after each dispatching, which is necessary to process next job; when the server runs out of pending jobs, the whole system can restart the computation ($restart$). Notice that the communication interface $\lambda$ relates each $Job_i$ that the server has to pop, with its solution $Sol_j$ that the client has to push, determining the computation.

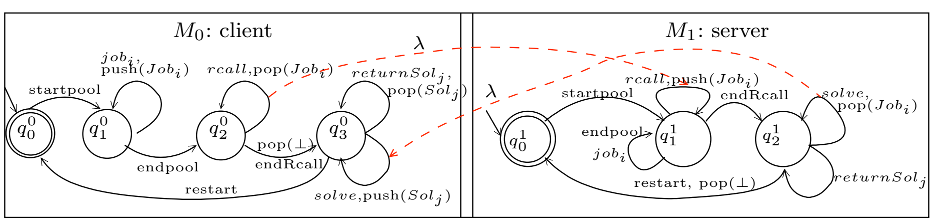


**Fig. 3.** An example of an S-VPA.

## 7 Conclusions

In this paper, we have investigated *ordered visibly pushdown automata with two stacks* (2-OVPA), obtained by merging the definitions of visibly pushdown automata [AM04] and multi-pushdown automata with two stacks [BCCR96]. We have shown that 2-OVPA are determinizable, closed under intersection and complementation, and have the emptiness problem decidable and solvable in polynomial time. Thus, we get that the inclusion problem is also decidable for 2-OVPA, and in particular, it is ExpTime-complete. It is worth noticing that dropping visibility or the ordering constraint from 2-OVPA makes inclusion undecidable. The properties satisfied by 2-OVPA, along with the fact that they accept some context-free languages that are not regular as well as some context-sensitive languages that are not context-free, make 2-OVPA a powerful model in system verification while using the automata-theoretic approach.

Finally, the model we propose can be also extended to deal with an arbitrary number $n$ of stacks ($n$-OVPA). We argue (it is left to further investigation) that $n$-OVPA still retain decidability and closure properties of 2-OVPA and that, from an expressivity viewpoint, $n$-OVPA define a strict hierarchy based on the number of pushdown stores.

## References


[AM04] R. Alur and P. Madhusudan. Visibly pushdown languages. In *STOC'04*, pages 202–211. ACM, 2004.

[BCCR96] L. Breveglieri, A. Cherubini, C. Citrini, and S. Crespi-Reghizzi. Multi-push-down languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996.

[BMP05] L. Bozzelli, A. Murano, and A. Peron. Pushdown module checking. In *LPAR'05*, LNCS 3835, pages 504–518, 2005.

[CE81] E.M. Clarke and E.A. Emerson. Design and verification of synchronization skeletons using branching time temporal logic. In *Proc. of Work. on Logic of Programs*, LNCS 131, pages 52–71, 1981.

[CGP99] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT Press, 1999.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 1979.

[KPV02] O. Kupferman, N. Piterman, and M. Vardi. Pushdown specifications. In *LPAR'02*, LNCS 2514, pages 262–277, 2002.

[Min67] Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.

[QS81] J.P. Queille and J. Sifakis. Specification and verification of concurrent programs in Cesar. In *Proceedings of the Fifth International Symposium on Programming*, LNCS 137, pages 337–351, 1981.

[SCFG84] A. Sistla, E.M. Clarke, N. Francez, and Y. Gurevich. Can message buffers be axiomatized in linear temporal logic. *Information and Control*, 63(1-2):88–112, 1984.

[ST02] Maarten Van Steen and Andrew S. Tanenbaum. *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2002.

[VW86] M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *J. of Computer and System Sciences*, 32(2):182–221, 1986.

[Wal96] I. Walukiewicz. Pushdown processes: Games and Model Checking. In *CAV'96*, LNCS 1102, pages 62–74. Springer-Verlag, 1996.

## 8 Appendix

**Proof of Theorem 2**.

*Proof.* Here we complete the proof of Theorem 2 proving the following assert:

Given a sequence of numbers $s = s_1 s_2 \dots s_k$, with $s_i \in \{1, \dots, n\}$ for all $i \in \{1, \dots, k\}$, the sequence $(L_{s_1}, v^0_{s_1}, v^1_{s_1}) \dots (L_{s_k}, v^0_{s_k}, v^1_{s_k})$ of elements from $\{L_1, \dots L_n\} \times \mathbb{N} \times \mathbb{N}$ is an execution trace of $M$ if and only if the sequence $(L_{s_1}, \sigma^0_{s_1}, \sigma^1_{s_1}) \dots (L_{s_k}, \sigma^0_{s_k}, \sigma^1_{s_k})$ of elements from $Q \times \Gamma^*.\bot \times \Gamma^*.\bot$ is a run of $M'$, with $|\sigma^j_{s_i}| = v^j_{s_i} + 1$ for each $i \in \{1, \dots, k\}$ and $j \in 0, 1$.

We prove the above assert by induction on the length of the sequence $s$. The base case for $|s| = 1$ is trivial since every trace of $M$ starts from $(L_1, v^0_1, v^1_1) = (L_1, 0, 0)$ and every run form $M'$ starts from $(L_1, \sigma^0_1, \sigma^1_1) = (L_1, \bot, \bot)$, and thus $|\sigma^j_1| = v^j_{s_1} + 1$, for $j \in \{0, 1\}$. Let us consider $|s| = k$ and prove the assert for $k+1$. By hypothesis we have that $(L_{s_1}, v^0_{s_1}, v^1_{s_1}) \dots (L_{s_k}, v^0_{s_k}, v^1_{s_k})$ is an execution trace of $M$ if and only if $(L_{s_1}, \sigma^0_{s_1}, \sigma^1_{s_1}) \dots (L_{s_k}, \sigma^0_{s_k}, \sigma^1_{s_k})$ is a run of $M'$, with $|\sigma^j_{s_i}| = v^j_{s_i} + 1$, for each $i \in \{1, \dots, k\}$. If $s_k = n$ then we can not extend the execution trace of $M$ and the run of $M'$, so the assert is trivially true. Otherwise, we have two sub-cases to consider depending on the kind of the instruction $I_{s_k}$. Assume first that $I_{s_k}$ is an increment instruction of the counter $C_0$ (resp., $C_1$), that is $I_{s_k} = c_0 := c_0 + 1;\ \texttt{goto}\ L_j$ (resp., $I_{s_k} = c_1 := c_1 + 1;\ \texttt{goto}\ L_j$), then the $k+1$-th element in the execution trace of $M$ is $(L_j, v^0_{s_k} + 1, v^1_{s_k})$ (resp., $(L_j, v^0_{s_k}, v^1_{s_k} + 1)$). In such a case, we have by definition that from the state $L_{s_k}$, the automaton $M'$ reads a the symbol $I_{s_k}$ using the transition $(L_{s_k}, I_{s_k}, L_j, A)$ belonging to $\delta_{c_0}$ (resp., $\delta_{c_1}$). Then, the next configuration in the run is $(L_j, \sigma^0_{s_{k+1}}, \sigma^1_{s_{k+1}})$, with $\sigma^0_{s_{k+1}} = A.\sigma^0_{s_k}$, $\sigma^1_{s_k} = \sigma^1_{s_{k+1}}$, and $|\sigma^0_{s_{k+1}}| = |\sigma^0_{s_k}| + 1 = v^0_{s_k} + 2 = v^0_{s_{k+1}} + 1$ (resp., $(L_j, \sigma^0_{s_{k+1}}, \sigma^1_{s_{k+1}})$, with $\sigma^1_{s_{k+1}} = A.\sigma^1_{s_k}$, $\sigma^0_{s_k} = \sigma^0_{s_{k+1}}$, and $|\sigma^1_{s_{k+1}}| = |\sigma^1_{s_k}| + 1 = v^1_{s_k} + 2 = v^1_{s_{k+1}} + 1$).

If $I_{s_k}$ is $\texttt{if}\ C_0 = 0\ \texttt{then goto}\ L_j\ \texttt{else}\ C_0 := C_0 - 1;\ \texttt{goto}\ L_m$ is a test-and-decrement instruction of the counter $c_0$ (resp., $c_1$), then the $k+1$-th element in the execution trace of $M$ is $(L_m, v^0_{s_k} - 1, v^1_{s_k})$ (resp., $(L_m, v^0_{s_k}, v^1_{s_k} - 1)$) if $v^0_{s_k} > 0$ (resp., $v^1_{s_k} > 0$), else the $k+1$-th element in the execution trace of $M$ is $(L_j, v^0_{s_k}, v^1_{s_k})$ (resp., $(L_j, v^0_{s_k}, v^1_{s_k})$) if $v^0_{s_k} = 0$ (resp., $v^1_{s_k} = 0$). In such a case we have by definition that from the state $L_{s_k}$, $M'$ reads $I_{s_k}$ using the transition $(L_{s_k}, I_{s_k}, A, L_m)$ if $\sigma^0_{s_k} \neq \bot$ (resp., $\sigma^1_{s_k} \neq \bot$), or the transition $(L_{s_k}, I_{s_k}, \bot, L_j)$ if $\sigma^0_{s_k} = \bot$ (resp., $\sigma^1_{s_k} = \bot$). In the first case, the next configuration in the run is $(L_m, \sigma^0_{s_{k+1}}, \sigma^1_{s_k})$, with $\sigma^0_{s_{k+1}} = A.\sigma^0_{s_k}$ (resp., $\sigma^1_{s_{k+1}} = A.\sigma^1_{s_k}$) and then $|\sigma^0_{s_k+1}| = |\sigma^0_{s_k}| - 1 = v^0_{s_k} = v^0_{s_{k+1}} + 1$ (resp., $|\sigma^1_{s_k+1}| = |\sigma^1_{s_k}| - 1 = v^1_{s_k} = v^1_{s_{k+1}} + 1$). In the latter case the transition is $(L_{s_k}, I_{s_k}, \bot, L_j)$ and so the next configuration in the run is $(L_j, \sigma^0_{s_{k+1}}, \sigma^1_{s_k+1})$, with $\sigma^0_{s_{k+1}} = \sigma^0_{s_k} = \bot$ (resp., $\sigma^1_{s_{k+1}} = \sigma^1_{s_k} = \bot$) and thus $|\sigma^0_{s_k+1}| = |\sigma^0_{s_k}| = |\bot| = 1 = v^0_{s_k} + 1 = v^0_{s_{k+1}} + 1$ (resp., $|\sigma^1_{s_k+1}| = |\sigma^1_{s_k}| = |\bot| = 1 = v^1_{s_k} + 1 = v^1_{s_{k+1}} + 1$). So, we have done with the assert. □

**Proof of Theorem 3**.

We present $\varepsilon$-Closure definition. Let $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$ be a 2-VPA$_\varepsilon$ (resp., 2-OVPA$_\varepsilon$) and $q \in Q$, with $\varepsilon$-Closure$(q, \gamma^0, \gamma^1)$ we denote the set of all $p \in Q$ such that are reachable from $q$ only by $\varepsilon$-moves of type $(q', \varepsilon, \gamma^0, \gamma^1, q'')$.

Moreover, we overload the $\varepsilon$-Closure definition by the following: let $Q$ a set of states, then $\varepsilon$-Closure$(Q, \gamma_0, \gamma_1) = \bigcup_{q \in Q} \varepsilon$-Closure$(q, \gamma_0, \gamma_1)$.

*Proof.* Let $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$ be a 2-VPA (resp., 2-OVPA). We show a 2-VPA$_\varepsilon$ (resp., 2-OVPA$_\varepsilon$) $M_\varepsilon$ such that $L(M) = L(M_\varepsilon)$. We consider $M_\varepsilon = (Q, Q_{in}, \Gamma, \delta_\varepsilon, Q_F)$ with $\delta_\varepsilon$ containing all push, pop and synch transitions of $\delta$. Moreover, $\delta_\varepsilon$ differs from $\delta$ on local transitions: for each $(q, a, p) \in \delta$ and for all $\gamma^0, \gamma^1 \in \Gamma$, then $(q, a, \gamma^0, \gamma^1, p) \in \delta_\varepsilon$. It is easy to prove that $M$ and $M_\varepsilon$ accept the same language.

Let now $M_\varepsilon = (Q, Q_{in_\varepsilon}, \Gamma, \delta_\varepsilon, Q_F)$ be a 2-VPA$_\varepsilon$ (resp., 2-OVPA$_\varepsilon$). We define a 2-VPA (resp., 2-OVPA) $M$ such that $L(M) = L(M_\varepsilon)$. We consider $M = (Q \times \Gamma \times \Gamma, Q_{in} \times \{\bot\} \times \{\bot\}, \Gamma \times \Gamma, \delta, Q_F \times \Gamma \times \Gamma)$, with $Q_{in} = \varepsilon$-Closure$(Q_{in_\varepsilon}, \bot, \bot)$ and $\delta$ is defined as follows:
**Push**: $a \in \Sigma_{c_0}$ and $(q, a, p, \gamma) \in \delta_\varepsilon$, then for all $\gamma^0, \gamma^1 \in \Gamma$ there are transitions $((q, \gamma^0, \gamma^1), a, (p', \gamma, \gamma^1), (\gamma, \gamma^0)) \in \delta$ with $p' \in \varepsilon$-Closure$(p, \gamma, \gamma^1)$;
**2Push**: $a \in \Sigma_c$ and $(q, a, p, \gamma, \gamma') \in \delta_\varepsilon$, then for all $\gamma^0, \gamma^1 \in \Gamma$ there are transitions $((q, \gamma^0, \gamma^1), a, (p', \gamma, \gamma'), (\gamma, \gamma^0), (\gamma', \gamma^1)) \in \delta$ with $p' \in \varepsilon$-Closure$(p, \gamma, \gamma')$;
**Pop**: $a \in \Sigma_{r_0}$ and $(q, a, \gamma, p) \in \delta_\varepsilon$, then for all $\gamma^0, \gamma^1 \in \Gamma$ there are transitions $((q, \gamma, \gamma^1), a, (\gamma, \gamma^0), (p', \gamma^0, \gamma^1)) \in \delta$ with $p' \in \varepsilon$-Closure$(p, \gamma^0, \gamma^1)$;
**2Pop**: $a \in \Sigma_r$ and $(q, a, \gamma, \gamma', p) \in \delta_\varepsilon$, then for all $\gamma^0, \gamma^1 \in \Gamma$ there are transitions $((q, \gamma, \gamma^1), a, (\gamma, \gamma^0), (\gamma', \gamma^1), (p', \gamma^0, \gamma^1)) \in \delta$ with $p' \in \varepsilon$-Closure$(p, \gamma^0, \gamma^1)$;
**Local**: $a \in \Sigma_l$ and $(q, a, \gamma^0, \gamma^1, p) \in \delta_\varepsilon$, then for all $\gamma^0, \gamma^1 \in \Gamma$ there are transitions $((q, \gamma^0, \gamma^1), a, (p', \gamma^0, \gamma^1)) \in \delta$ with $p' \in \varepsilon$-Closure$(q, \gamma^0, \gamma^1)$;
**Synch**: $a \in \Sigma_{s_0}$ and $(q, a, \gamma, p, \widehat{\gamma}) \in \delta_\varepsilon$, then for all $\gamma^0, \gamma^1 \in \Gamma$ there are transitions $((q, \gamma, \gamma^1), a, (\gamma, \gamma^0), (p', \gamma^0, \widehat{\gamma}), (\widehat{\gamma}, \gamma^1)) \in \delta$ with $p' \in \varepsilon$-Closure$(q, \gamma^0, \widehat{\gamma})$.

It is easy to prove that $M$ and $M_\varepsilon$ accept the same language. □

**Proof of Theorem 4.**

*Proof.* Assertion a) follows from the definition of 2-OVPA and the decidability of the emptiness problem for 2-OVPA, but not for 2-VPA.

To show assertion b), we can use $L_1 = \{a^n b^n c^n | \exists n \in \mathbb{N}\} \in$ 2-OVPLs (Example 1). The strict containment follows from the fact that, for all pushdown alphabets $\widetilde{\Sigma}$, we have $L_1 \notin$ VPLs [AM04].

Assertion c) follows immediately from assertions a) and b).

In order to prove the assertion d), we show that the DCFL $L_2 = \{w\#w^r | w \in \Sigma^*\}$ is not in 2-VPL. In particular, we prove that the sub-language $L = \{a^n\#a^n |n \in \mathbb{N}\}$ of $L_2$ is not a 2-VPL for any $\widetilde{\Sigma}$ (thus, $\Sigma = \{a, \#\}$). First, we consider that the input symbol # is only a marker that denote the starting of the second part of the input string, so it is not influential in the proof. Let $M$ be a 2-VPA over $\widetilde{\Sigma}$. We prove the assert showing that whatever partition of $\widetilde{\Sigma}$ we choose for $a$, $L(M) \neq L$. If $a \in \Sigma_l$, $M$ reduces to a finite automaton, and we know that $L$ is not a regular language. So $L$ is not a 2-VPL over $\widetilde{\Sigma}$ with $a \in \Sigma_l$. If $a \in \Sigma_c$, $M$ reduces to a VPA, because it uses two stacks like a single one, that can only push on reading $a$, so it cannot compare the number of $a$ before # with the number of $a$ after the marker symbol. The obtained automaton behaves as a zero reversal-bounded one-counter machine defined by Oscar H. Ibarra in 1978[1]

[1] O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. J. ACM, 25(1):116133, 1978.

that accept only regular languages. So L is not a 2-VPL over $\widetilde{\Sigma}$ with $a \in \Sigma_c$. The case of $a \in \Sigma_{c_i}$, with $i = 0$ or 1, is analogous. If $a \in \Sigma_{s_i}$, with $i = 0$ or 1, we have a 2-VPA $M$ acting as a VPA $M'$ with $a \in \Sigma_c$, where $M'$ simulates $M$ on the stack $j \neq i$, because the stack $i$ is always $\perp$. The above argument holds and $L$ is not a 2-VPL over $\widetilde{\Sigma}$ with $a \in \Sigma_{s_i}$, for $i = 0, 1$. So, we conclude that $L$ is not a 2-VPL over $\widetilde{\Sigma}$, for any $\widetilde{\Sigma}$.

Assertion e) is an immediate consequence of assertions a) and d).

To show assertion f) we use $L = \{w \in \Sigma^* \mid \ |w|_a = |w|_b\}$ of all strings having equal number of occurrences of $a$ and $b$. In [AM04] it is shown that $L$ is a CFL but not a VPL, for any pushdown alphabet. Here we show a 2-VPA $M$, over an alphabet $\widetilde{\Sigma}$, accepting $L$. Let $\Sigma$ be partitioned in $\Sigma_{s_0} = \{a\}$ and $\Sigma_{s_1} = \{b\}$, and $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$ where Q=$\{q_0, q_F\}$, $Q_{in} = \{q_0\}$, $Q_F = \{q_F\}$, $\Gamma = \{A, B, \overline{A}, \overline{B}\}$ and $\delta = \{(q_0, \varepsilon, \perp, \perp, q_F)$, $(q_0, a, \overline{B}, \perp, q_F)$, $(q_0, \varepsilon, \perp, \overline{A}, q_F)$, $(q_0, \varepsilon, \overline{B}, \overline{A}, q_F)$, $(q_0, a, \perp, q_0, A)$, $(q_0, a, \overline{B}, q_0, A)$, $(q_0, a, B, q_0, \overline{A})$, $(q_0, b, \perp, q_0, B)$, $(q_0, b, A, q_0, \overline{B})$, $(q_0, b, \overline{A}, q_0, B)\}$. We depict $M$ in Figure 4.

We now prove the assertion g) as follows. Let a 2-OVPA $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$ over $\widetilde{\Sigma}$, we now present a construction of a PD$^2$ automaton $M'$ such that $L(M) = L(M')$. Before starting with construction definition, w.l.o.g., we assume that stack symbols in $M$ are distinct for each stack. We use the subscript 0 (resp., 1) to denote a first (resp., second) stack symbol as in $\gamma_0$ (resp., $\gamma_1$). Let $M' = (Q', \Sigma, \Gamma, \delta', q_0, Q_F, Z_0)$ be defined as follows: $Q' = Q \cup q_0$, with $q_0 \notin Q$, $\delta'$ is defined as follows:
if $a \in \Sigma_l$ and $(q, a, q') \in \delta$, then for all $\gamma_0 \in \Gamma_\perp$ the function $\delta'$ has the transitions $(q, a, \gamma_0) \mapsto (q', \gamma_0, \varepsilon)$ and for all $\gamma_1 \in \Gamma$ the transitions $(q, a, \gamma_1) \mapsto (q', \varepsilon, \gamma_1)$,
if $a \in \Sigma_c$ and $(q, a, q', \gamma_0, \gamma_1) \in \delta$, then the function $\delta'$ has the transitions for all $\gamma_0' \in \Gamma_\perp$ $(q, a, \gamma_0') \mapsto (q', \gamma_0\gamma_0', \gamma_1)$, and for all $\gamma_1' \in \Gamma$ $(q, a, \gamma_1') \mapsto (q', \gamma_0, \gamma_1\gamma_1')$,
if $a \in \Sigma_{r_0}$ and $(q, a, \gamma_0, q') \in \delta$, then the function $\delta'$ has the transitions $(q, a, \gamma_0) \mapsto (q', \varepsilon, \varepsilon)$,
if $a \in \Sigma_{c_0}$ and $(q, a, q', \gamma_0) \in \delta$, then the function $\delta'$ has the transitions for all $\gamma_0' \in \Gamma$ $(q, a, \gamma_0) \mapsto (q', \varepsilon, \varepsilon)$, and for all $\gamma_1 \in \Gamma$ $(q, a, \gamma_1) \mapsto (q', \gamma_0, \gamma_1)$,
if $a \in \Sigma_{s_0}$ and $(q, a, \gamma_0, q', \gamma_1) \in \delta$, then the function $\delta'$ has the transitions $(q, a, \gamma_0) \mapsto (q', \varepsilon, \gamma_1)$, and symmetrically for the symbols in $\Sigma_{c_1}$,$\Sigma_{r_1}$, and $\Sigma_{s_1}$.
Furthermore, from $q_0$ the automaton $M'$ can only exploits a pop $\varepsilon$-move operating on the first stack and reaching a state in $Q_{in}$. MOreover, for each transition of $M$ involving $\perp$ symbol, we add to $\delta'$ the correspondent move with $\varepsilon$ used accordingly. Strict inclusion follows from the fact that 2-OVPLs, differently from $\mathcal{L}_{PD^2}$, are closed under intersection, as we later show in Theorem 5.

Finally, assertion h) immediately follows from assertion g) and the fact that $\mathcal{L}_{PD^2}$ are a proper subclass of CSLs (Theorem 1). □

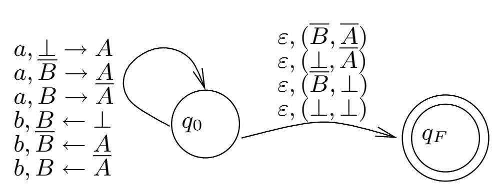


**Fig. 4.** A 2-VPA accepting $L = \{w \in \Sigma^* \mid \ |w|_a = |w|_b\}$

**Proof of Theorem 5.**

*Proof. Intersection.* Let $L_1$ and $L_2$ be two languages respectively accepted by 2-VPA (resp. 2-OVPA) $M_1$ and $M_2$. The 2-VPA (resp., 2-OVPA) $M$ accepting $L_1 \cap L_2$ is the synchronous product of $M_1$ and $M_2$. It is easy to see thet the synchronous product construction we now show preserves the ordering of stacks for 2-OVPA case, so in the following we speak indifferently of 2-VPA.

The resulting 2-VPA $M$ accepting $L_1 \cap L_2$ has defined in such a way that the sets of states, initial states, final states, and stack symbols are, respectively, the product of sets of states, initial states, final states, and stack symbols of $M_1$ and $M_2$. The automaton $M$ simulates on its first (resp., second) stack the first (resp., second) stacks of both $M_1$ and $M_2$. Each transition of $M$ is the synchronization (with respect to a common input symbol) of a transition of $M_1$ and a transition of $M_2$. Formally, let $M_1 = (Q^1, Q^1_{in}, \Gamma^1, \delta^1, Q^1_F)$ and $M_2 = (Q^2, Q^2_{in}, \Gamma^2, \delta^2, Q^2_F)$. We define $M = (Q^1 \times Q^2, Q^1_{in} \times Q^2_{in}, \Gamma^1 \times \Gamma^2, \delta, Q^1_F \times Q^2_F)$ where, for each $a \in \Sigma$, $i \in \{0, 1\}$, $\delta$ is defined as follows:
**Push**: $a \in \Sigma_{c_i}$, and there are $(q^1, a, p^1, \gamma^1) \in \delta^1$ and $(q^2, a, p^2, \gamma^2) \in \delta^2$, then $((q^1, q^2), a, (p^1, p^2), (\gamma^1, \gamma^2)) \in \delta$;
**2Push**: $a \in \Sigma_c$, and there are $(q^1, a, p^1, \gamma^1, {\gamma^1}') \in \delta^1$ and $(q^2, a, p^2, \gamma^2, {\gamma^2}') \in \delta^2$, then $((q^1, q^2), a, (p^1, p^2), (\gamma^1, \gamma^2), ({\gamma^1}', {\gamma^2}')) \in \delta$;
**Pop**: $a \in \Sigma_{r_i}$, and there are $(q^1, a, \gamma^1, p^1) \in \delta^1$ and $(q^2, a, \gamma^2, p^2) \in \delta^2$, then $((q^1, q^2), a, (\gamma^1, \gamma^2), (p^1, p^2)) \in \delta$;
**2Pop**: $a \in \Sigma_r$, and there are $(q^1, a, \gamma^1, {\gamma^1}', p^1) \in \delta^1$ and $(q^2, a, \gamma^2, {\gamma^2}', p^2) \in \delta^2$, then $((q^1, q^2), a, (\gamma^1, \gamma^2), ({\gamma^1}', {\gamma^2}'), (p^1, p^2)) \in \delta$;
**Local**: $a \in \Sigma_l$, and there are $(q^1, a, p^1) \in \delta^1$ and $(q^2, a, p^2) \in \delta^2$, then $((q^1, q^2), a, (p^1, p^2)) \in \delta$;
**Synch**: $a \in \Sigma_{s_i}$, and there are $(q^1, a, \gamma^1, p^1, \widehat{\gamma}^1) \in \delta^1$ and $(q^2, a, \gamma^2, p^2, \widehat{\gamma}^2) \in \delta^2$, then $((q^1, q^2), a, (\gamma^1, \gamma^2), (p^1, p^2), (\widehat{\gamma}^1, \widehat{\gamma}^2)) \in \delta$.

The correctness of the construction can be proved in a standard way.

*Union.* Let $L_1$ and $L_2$ be two languages respectively accepted by the 2-VPA (resp., 2-OVPA) $M_1$ and $M_2$. The 2-VPA (resp., 2-OVPA) $M$ accepting $L_1 \cup L_2$ is defined in such a way that non-deterministically behaves as $M_1$ or $M_2$ exploiting an $\varepsilon$-move.

*Composition.* Let $L_1$ and $L_2$ be two languages respectively accepted by the 2-VPA $M_1$ and $M_2$. The 2-VPA $M$ accepting $L_1 \cdot L_2$ acts in such a way that on reading an input word $w$, nondeterministically splits $w$ in two words $w_1$ and $w_2$, so that $M$ simulates $M_1$ on $w_1$ and $M_2$ on $w_2$. When $M$ starts to simulate $M_2$ it regards the current content of the stacks as they were empty. Formally, $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$ with $Q = Q^1 \uplus Q^2$ (we assume that $Q^1 \cap Q^2 = \emptyset$), if $Q^1_{in} \cap Q^1_F = \emptyset$ then $Q_{in} = Q^1_{in}$, otherwise $Q_{in} = Q^1_{in} \cup Q^2_{in}$, $\Gamma = \Gamma^1 \cup \Gamma^2$, $Q_F = Q^2_F$, and $\delta$ is defined as follows:
**Push**: if $(q, a, p, \gamma) \in \delta^1$, then $(q, a, p, \gamma) \in \delta$ and, if $p \in Q^1_F$ then for all $p' \in Q^2_{in}$ also $(q, a, p', \gamma) \in \delta$, if $(q, a, p, \gamma) \in \delta^2$, then $(q, a, p, \gamma) \in \delta$;
**2Push**: if $(q, a, p, \gamma, \gamma') \in \delta^1$, then $(q, a, p, \gamma, \gamma') \in \delta$ and, if $p \in Q^1_F$ then for all $p' \in Q^2_{in}$ also $(q, a, p', \gamma, \gamma') \in \delta$, if $(q, a, p, \gamma, \gamma') \in \delta^2$, then $(q, a, p, \gamma, \gamma') \in \delta$;
**Pop**: If $(q, a, \gamma, p) \in \delta^1$, then $(q, a, \gamma, p) \in \delta$, and if $p \in Q^1_F$, then for all $p' \in Q^2_{in}$ also $(q, a, \gamma, p') \in \delta$. For each $(q, a, \bot, p) \in \delta^2$, then for all $\gamma \in \Gamma^1$ also $(q, a, \gamma, p) \in \delta$, and if $(q, a, \gamma, p) \in \delta^2$, then it belongs also to $\delta$;

**2Pop**: If $(q, a, \gamma, \gamma', p) \in \delta^1$, then $(q, a, \gamma, \gamma', p) \in \delta$, and if $p \in Q^1_F$, then for all $p' \in Q^2_{in}$ also $(q, a, \gamma, \gamma', p') \in \delta$. For each $(q, a, \perp, \perp, p), (q, a, \perp, \gamma_1, p), (q, a, \gamma_0, \perp, p) \in \delta^2$, then for all $\gamma, \gamma' \in \Gamma^1$ also $(q, a, \gamma, \gamma', p)$, $(q, a, \gamma, \gamma_1, p)$, $(q, a, \gamma_0, \gamma', p) \in \delta$, respectively, and if $(q, a, \gamma, \gamma', p) \in \delta^2$, then it belongs also to $\delta$;
**Local**: If $(q, a, p) \in \delta^1$ then it belongs also to $\delta$, and if $p \in Q^1_F$ then for all $p' \in Q^2_{in}$ also $(q_i, a_i, {q_0}^2) \in \delta$, if $(q, a, p) \in \delta^2$, then it belongs also to $\delta$;
**Synch**: If $(q, a, \gamma, p, \widehat{\gamma}) \in \delta^1$, then it belongs also to $\delta$, and if $p \in Q^1_F$, then for all $p' \in Q^2_{in}$ also $(q, a, \gamma, p', \widehat{\gamma}) \in \delta$. If $(q, a, \gamma, p, \widehat{\gamma}) \in \delta^2$ then it belongs also to $\delta$, and if $\gamma = \perp$, then for all $\gamma^1 \in \Gamma^1$ also $(q, a, \gamma^1, p', \widehat{\gamma}) \in \delta$.

*Kleene-**. Let $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$ be a 2-VPA that accepts $L$. We build the automaton $M^*$ as follows. The main idea is similar to the VPA case, but we have to apply a duplication of states. $M^*$ simulates $M$ step by step, but when $M$ changes its state to a final state, $M^*$ can nondeterministically update its state to an initial state, and thus, restart $M$. After this switch, $M^*$ must treat the stack as if it is empty, and this requires labeling its state so that in a tagged state the top can be assumed to be $\perp$ ignoring the actual content of the stack. More precisely, $M^* = ((Q \uplus Q') \times (Q \uplus Q'), (Q_{in} \uplus Q'_{in}) \times (Q_{in} \uplus Q'_{in}), \Gamma \uplus \Gamma', \delta^*, (Q_F \uplus Q'_F) \times (Q_F \uplus Q'_F)$, and $\delta^*$ is defined as follows:
**Local**: Let $a \in \Sigma_l$ and $(q, a, p) \in \delta$, then $\delta^*$ contains the transitions $((q, q), a, (p, p))$, $((q', q), a, (p', p))$, $((q, q'), a, (p, p'))$, $((q', q'), a, (p', p'))$, and if $p \in Q_F$, then $\delta^*$ contains $((q, q), a, (r', r'))$, $((q', q'), a, (r', r'))$, $((q', q), a, (r', r'))$, $((q, q'), a, (r', r'))$, for each $r \in Q_{in}$;
**Push**: Let $a \in \Sigma_{c_0}$ and $(q, a, p, \gamma) \in \delta$, then $\delta^*$ contains the transitions $((q, q), a, (p, p), \gamma)$, $((q', q'), a, (p, p'), \gamma')$, $((q', q), a, (p, p), \gamma')$, $((q, q'), a, (p, p'), \gamma)$, and if $p \in Q_F$, then $\delta^*$ contains $((q, q), a, (r', r'), \gamma)$, $((q', q), a, (r', r'), \gamma)$, $((q, q'), a, (r', r'), \gamma)$, $((q', q'), a, (r', r'), \gamma)$, for each $r \in Q_{in}$;
**2Push**: this case is similar to the previous with two symbols to push;
**Pop**: Let $a \in \Sigma_{r_0}$ and $(q, a, \gamma, p) \in \delta$, then $\delta^*$ contains the transitions $((q, q), a, \gamma, (p, p))$, $((q, q'), a, \gamma, (p, p'))$, $((q, q'), a, \gamma', (p', p'))$, and if $p \in Q_F$, then $\delta^*$ contains $((q, q), a, \gamma, (r', r'))$, $((q, q'), a, \gamma, (r', r'))$ for each $r \in Q_{in}$.
If $(q, a, \perp, p) \in \delta$, then $\delta^*$ contains the transitions $((q', q), a, \gamma, (p', p))$, $((q', q'), a, \gamma, (p', p'))$ for each $\gamma \in \Gamma \uplus \Gamma'$, and if $p \in Q_F$, then $\delta^*$ contains also the transitions $((q', q), a, \gamma, (r', r'))$, $((q', q'), a, \gamma, (r', r'))$ for each $\gamma \in \Gamma \uplus \Gamma'$ and $r \in Q_{in}$;
**2Pop**: this case is similar to the previous with two symbols to pop;
**Synch**: Let $a \in \Sigma_{s_0}$ and $(q, a, \gamma, p, \widehat{\gamma}) \in \delta$, then $\delta^*$ contains the transitions $((q, q), a, \gamma, (p, p), \widehat{\gamma})$, $((q, q'), a, \gamma, (p, p), \widehat{\gamma}')$, $((q, q), a, \gamma', (p', p), \widehat{\gamma})$, and if $p \in Q_F$, then $\delta^*$ contains $((q, q), a, \gamma, (r', r'), \widehat{\gamma})$, $((q, q'), a, \gamma, (r', r'), \widehat{\gamma})$ for each $r \in Q_{in}$.
If $(q, a, \perp, p, \widehat{\gamma}) \in \delta$, $\delta^*$ contains the transitions $((q, q), a, \gamma, (p', p), \widehat{\gamma})$, $((q', q'), a, \gamma, (p', p), \widehat{\gamma}')$, for each $\gamma \in \Gamma \uplus \Gamma'$, and if $p \in Q_F$, then $\delta^*$ contains also the transitions $((q, q'), a, \gamma, (r', r'), \widehat{\gamma})$, $((q', q'), a, \gamma, (r', r'), \widehat{\gamma})$ for each $\gamma \in \Gamma \uplus \Gamma'$ and $r \in Q_{in}$;
and symmetrically for transitions of $M$ operating on the second stack. This construction is easily provable to be correct by means of standard mathematical tools. □

**Proof of Theorem 6**.

We now define the transition relation of the automaton presented in the sketch of the proof of Theorem 6. In order to simplify notation, we introduce extended states. Let $q$ be a state of a 2-VPA (resp., 2-OVPA) $M$, a triple $\overline{q} = (q, \gamma_0, \gamma_1)$ is an *extended state* of $M$, where $\gamma_0, \gamma_1$ are stack symbols of $M$. In the following, every state we use to define $M'$ is an extended state in which the pair of stack symbols corresponds to the top of stacks of $M$. In particular, when we have to compute the updating set $U$, we obtain a pair of extended states $(\overline{q}, \overline{q}')$ necessary to correctly update previous sets of path $(R)$ and summary $(S)$ edges. In the following, we denote with $q$ an arbitrary extended state of $M$ and we suppose that the transition relation $\delta$ of $M$ is changed accordingly (as we have shown in proof of Theorem 3). Moreover, for notational convenience, we present only the topmost stack symbol $\gamma$ in $\delta$ transition.

*Proof.* The transition relation $\delta'$ is defined as follows, with $i, j \in \{0, 1\}, i \neq j$:
**Local**: $a \in \Sigma_l$ and $((S_0, S_1, R), a, (S'_0, S'_1, R')) \in \delta'$ where
$S'_i = \{(q, q') \mid \exists q'' \in Q \,.\, (q'', a, q') \in \delta \wedge (q, q'') \in S_i\ \}$ and
$R' = \{q' \mid \exists q \in R \,.\, (q, a, q') \in \delta\}$;
**Push**: $a \in \Sigma_{c_i}$, and $((S_0, S_1, R), a, (S'_0, S'_1, R'), (S_i, R, a)) \in \delta'$ where $S'_j = \{(q, q') \mid \exists q'' \in Q \,.\, (q, q'') \in S_j \wedge (q'', a, q', \gamma) \in \delta\}$, and $S'_i = Id_Q$ and $R' = \{q' \mid \exists q \in R \,.\, (q, a, q', \gamma) \in \delta\}$;
**2Push**: $a \in \Sigma_c$, and $((S_0, S_1, R), a, (S'_0, S'_1, R'), (S_0, R, a), (S_1, R, a)) \in \delta'$ where $S'_0 = S'_1 = Id_Q$ and $R' = \{q' \mid \exists q \in R \,.\, (q, a, q', \gamma, \gamma') \in \delta\}$;
**Pop**: $a \in \Sigma_{r_i}$ and $((S_0, S_1, R), a, (S', R', a'), (S''_0, S''_1, R'')) \in \delta'$ where
if $a' \in \Sigma_{c_i}$ the set $U = \{(q, q') \mid \exists q_2, q_3 \in Q, \gamma \in \Gamma \,.\, (q, a', q_2, \gamma) \in \delta \wedge (q_2, q_3) \in S_i \wedge (q_3, a, \gamma, q') \in \delta\}$
else $a' \in \Sigma_{s_j}$ and the set $U = \{(q, q') \mid \exists q_2, q_3 \in Q, \gamma, \gamma' \in \Gamma \,.\, (q, a', \gamma', q_2, \gamma) \in \delta \wedge (q_2, q_3) \in S_i \wedge (q_3, a, \gamma, q') \in \delta\}$ and, in both cases,
$S''_i = \{(q, q') \mid \exists q'' \in Q \,.\, (q, q'') \in S' \wedge (q'', q') \in U\}$,
$S''_j = \{(q, q') \mid \exists q'' \in Q \,.\, (q, q'') \in S_j \wedge (q'', q') \in U\}$,
$R'' = \{q' \mid \exists q \in R' \,.\, (q, q') \in \delta\}$, and
$((S_0, S_1, R), a, \perp, (S'_0, S'_1, R')) \in \delta'$ where, for every $i \in \{0, 1\}$, $S'_i = \{(q, q') \mid \exists q'' \in Q \,.\, (q, q'') \in S_i \wedge (q'', a, \perp, q') \in \delta\}$ and
$R' = \{q' \mid \exists q \in R \,.\, (q, a, \perp, q') \in \delta\}$;
**2Pop**: $a \in \Sigma_{r_i}$ and $((S_0, S_1, R), a, (S'_0, R'_0, a'), (S'_1, R'_1, a''), (S''_0, S''_1, R'')) \in \delta'$ is obtained combining computation (as we have shown in pop case) of updating sets $U^0$ and $U^1$ for the first and the second stack, respectively. We define for all $i \in \{0, 1\}$
$S''_i = \{(q, q') \mid \exists q'' \in Q \,.\, (q, q'') \in S'_i \wedge (q'', q') \in U^i\}$ and $R'' = R^0 \cap R^1$, where $R^i$ is the set of updated path edges w.r.t. the stack $i$. Furthermore, we have to combine the other cases of bottom of stack reached on both stacks or only for one of them, so in the former case we define
$((S_0, S_1, R), a, \perp, \perp, (S'_0, S'_1, R')) \in \delta'$ where, for every $i \in \{0, 1\}$, $S'_i = \{(q, q') \mid \exists q'' \in Q \,.\, (q, q'') \in S_i \wedge (q'', a, \perp, \perp, q') \in \delta\}$ and
$R' = \{q' \mid \exists q \in R \,.\, (q, a, \perp, \perp, q') \in \delta\}$,
and the other sub-cases are easily derivable from these;
**Synch**: $a \in \Sigma_{s_i}$ and $((S_0, S_1, R), a, (S'_i, R', a'), (S''_0, S''_1, R''), (S_j, R, a)) \in \delta'$ where $S''_j = Id_Q$ and
if $a' \in \Sigma_{c_i}$ the set $U = \{(q, q') \mid \exists q_2, q_3 \in Q, \gamma \in \Gamma \,.\, (q, a', q_2, \gamma) \in \delta \wedge (q_2, q_3) \in S_i \wedge (q_3, a, \gamma, q') \in \delta\}$
else $a' \in \Sigma_{s_j}$ and the set $U = \{(q, q') \mid \exists q_2, q_3 \in Q, \gamma, \gamma' \in \Gamma \,.\, (q, a', \gamma', q_2, \gamma) \in \delta \wedge (q_2, q_3) \in S_i \wedge (q_3, a, \gamma, q') \in \delta\}$ and, in both cases,

$S_i'' = \{(q, q') \mid \exists q'' \in Q\ .\ \ (q, q'') \in S_i' \wedge (q'', q') \in U\}$ and
$R'' = \{q' \mid\ \exists q \in R'\ .\ \ (q, q') \in U\}$ and
$((S_0, S_1, R), a, \bot, (S_0', S_1', R'), (S_i, R, a)) \in \delta'$ where,
for every $i \in \{0, 1\}$, $S_i' = \{(q, q') \mid\ \exists q'' \in Q\ .\ (q, q'') \in S_i \wedge (q'', a, \bot, q', \gamma) \in \delta\}$ and
$R' = \{q' \mid\ \exists q \in R\ .\ \ (q, a, \bot, q', \gamma) \in \delta\}$. $\square$

**Proof of Theorem 8**.

*Proof.* Now we prove that given an S-VPA $M_0||_\lambda M_1$ where $M_0 = (Q^0, Q^0_{in}, \mathit{\Gamma}^0, \delta^0, Q^0_F)$ and $M_1 = (Q^1, Q^1_{in}, \mathit{\Gamma}^1, \delta^1, Q^1_F)$ are defined over $\widetilde{\Sigma}^0$ and $\widetilde{\Sigma}^1$, respectively, we define a 2-OVPA $M = (Q^0 \times Q^1, Q^0_{in} \times Q^1_{in}, \mathit{\Gamma}^0 \cup \mathit{\Gamma}^1, \delta, Q^0_F \times Q^1_F)$ over $\widetilde{\Sigma} = \langle \widetilde{\Sigma}^0, \widetilde{\Sigma}^1 \rangle$ such that $L(M) = L(M_0||_\lambda M_1)$, and where the transition function $\delta$ is defined as follows:

if $a \in \Sigma_{c_0}$, $(q, a, q', \gamma) \in \delta^0$, and $(p, a, p') \in \delta^1$, then $((q, p), a, (q', p'), \gamma) \in \delta$;
if $a \in \Sigma_{c_1}$ we have a symmetric case;
if $a \in \Sigma_{r_0}$, $(q, a, \gamma, q') \in \delta^0$, and $(p, a, p') \in \delta^1$, then $((p, q), a, \gamma, (p', q')) \in \delta$;
if $a \in \Sigma_{r_1}$ we have a symmetric case;
if $a \in \Sigma_c$, $(q, a, q', \gamma) \in \delta^0_c$, and $(p, a, p', \gamma') \in \delta^1_c$, then $((p, q), a, (p', q'), \gamma, \gamma') \in \delta$;
if $a \in \Sigma_{s_0}$, $(q, a, \gamma, q') \in \delta^0$, $(p, a, p', \gamma') \in \delta^1$, $((q, a, \gamma, q'), (p, a, p', \gamma')) \in \lambda$ then $((q, p), a, \gamma, (q', p'), \gamma') \in \delta$;
if $a \in \Sigma_{s_1}$, $(q, a, \gamma, q') \in \delta^1$, $(p, a, p', \gamma') \in \delta^0$, $((p, a, p', \gamma'), (q, a, \gamma, q')) \in \lambda$ then $((q, p), a, \gamma, (q', p'), \gamma') \in \delta$;
if $a \in \Sigma_l$, $(q, a, q') \in \delta^i$, $(p, a, p') \in \delta^j$, then $((q, p), a,\ (q', p')) \in \delta$.

We now prove the following assert:

For every word $w = a_1 \dots a_n \in \Sigma^*$, $M_0||_\lambda M_1$ has a run $\rho = (\pi_0, \pi_1)$ over $w$, with $\pi^0 = (q^0_0, \bot)\ (q^0_1, \sigma^0_1)\ ...\ (q^0_n, \sigma^0_n)$ for $M_0$ and $\pi^1 = (q^1_0, \bot)\ (q^1_1, \sigma^1_1)\ ...\ (q^1_n, \sigma^1_n)$ for $M_1$, if and only if the 2-OVPA automaton $M$ has a run $\pi = ((q^0_0, q^1_0), \bot, \bot)$ $((q^0_1, q^1_1), \sigma^0_1, \sigma^1_1)\ ...\ ((q^0_n, q^1_n), \sigma^0_n, \sigma^1_n)$ over $w$.

We now prove the assert by induction on the length of the input word $w$.

Base case: $w = \varepsilon$, then by construction $\pi^0 = (q^0_0, \bot)$ and $\pi^1 = (q^1_0, \bot)$ if and only if $\pi = ((q^0_0, q^1_0), \bot, \bot)$.

Induction case: the assert holds for every word of length $n$, we now prove for word of length $n + 1$. Let $w = a_1 \dots a_n a_{n+1}$, then, by inductive hypothesis, the assert holds for $w' = a_1 \dots a_n$, that is, on reading $w'$, $M_0$ and $M_1$ have runs $\pi^0 = (q^0_0, \bot)\ (q^0_1, \sigma^0_1)\ ...\ (q^0_n, \sigma^0_n)$ and $\pi^1 = (q^1_0, \bot)\ (q^1_1, \sigma^1_1)\ ...\ (q^1_n, \sigma^1_n)$, respectively, if and only if $M$ has a run $\pi = ((q^0_0, q^1_0), \bot, \bot)\ ((q^0_1, q^1_1), \sigma^0_1, \sigma^1_1)\ ...$ $((q^0_n, q^1_n),\ \sigma^0_n, \sigma^1_n)$ on reading $w$. Then we prove that the assert also holds on $w$ by cases as follows:

if $a_{i+1} \in \Sigma_l$, then by construction $(q, a, q') \in \delta^0_l$, $(p, a, p') \in \delta^1_l$, and $((q, p), a,$ $(q', p')) \in \delta_l$. So, the next configurations for $M_0, M_1$, and $M$ are the following $(q', \sigma^0_i), (p', \sigma^1_i)$, and $((q', p'), \sigma^0_i, \sigma^1_i)$, respectively;

if $a_{i+1} \in \Sigma_{c_0}$, then by construction $(q, a, q', \gamma) \in \delta^0_c$, $(p, a, p') \in \delta^1_l$, and $((q, p), a,\ (q', p'), \gamma) \in \delta_{c_0}$. So, the next configurations for $M_0, M_1$, and $M$ are the following $(q', \gamma.\sigma^0_i), (p', \sigma^1_i)$, and $((q', p'), \gamma.\sigma^0_i,\ \sigma^1_i)$, respectively;

symmetrically for $a_{i+1} \in \Sigma_{c_1}$;

$a_{i+1} \in \Sigma_{r_0}$, then by construction $(q, a, \ \gamma, q') \in \delta_r^0$, $(p, a, p') \in \delta_l^1$, and $((q, p), a, \gamma, (q', p')) \in \delta_{r_0}$. So, the next configurations for $M_0, M_1$, and $M$ are the following $(q', \sigma_{i+1}^0), (p', \sigma_i^1)$, and $((q', p'), \sigma_{i+1}^0, \sigma_i^1)$, respectively, where if $\gamma \neq \perp$ then $\gamma.\sigma_{i+1}^0 = \sigma_i^0$ else $\sigma_{i+1}^0 = \sigma_i^0 = \perp$;

symmetrically for $a_{i+1} \in \Sigma_{r_1}$, but with the added restriction that $\sigma_i^0 = \perp$;

$a_{i+1} \in \Sigma_c$, then by construction $(q, a, q', \gamma) \in \delta_c^0$, $(p, a, p', \gamma') \in \delta_l^1$, and $((q, p), a, (q', p'), \gamma, \gamma') \in \delta_c$. So, the next configurations for $M_0, M_1$, and $M$ are the following $(q', \gamma.\sigma_i^0), (p', \gamma'\sigma_i^1)$, and $((q', p'), \gamma.\sigma_i^0, \gamma'.\sigma_i^1)$, respectively;

$a_{i+1} \in \Sigma_{s_0}$, then by construction $(q, a, \gamma, q') \in \delta_r^0$, $(p, a, p', \gamma') \in \delta_c^1$, $((q, p), a, \gamma, (q', p'), \gamma') \in \delta_{s_0}$, and $((q, a, \gamma, q'), (p, a, p', \gamma')) \in \lambda$. So, the next configurations for $M_0, M_1$, and $M$ are the following $(q', \sigma_{i+1}^0), (p', \sigma_{i+1}^1)$, and $((q', p'), \sigma_{i+1}^0, \sigma_{i+1}^1)$, respectively, with $\sigma_{i+1}^1 = \gamma'.\sigma_i^1$, and if $\gamma \neq \perp$ then $\gamma.\sigma_{i+1}^0 = \sigma_i^0$ else $\sigma_{i+1}^0 = \sigma_i^0 = \perp$;

symmetrically for $a_{i+1} \in \Sigma_{s_1}$, but with the added restriction that $\sigma_i^0 = \perp$.

From the assert we deduce that $w \in L(M_0||_\lambda M_1)$ iff $w \in L(M)$. $\square$

Impaginato nel mese di novembre 2006
presso il Dipartimento di Matematica ed Applicazioni
“R. Caccioppoli”
dell’Università degli Studi di Napoli “Federico II”.