2-Visibly Pushdown Automata*

Dario Carotenuto, Aniello Murano, and Adriano Peron

Università degli Studi di Napoli "Federico II", Via Cinthia, I-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 ([4, 9, 12]). In this verification method (for a survey see [5]), 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 [13, 2]). 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

* Work partially supported by MIUR FIRB Project no. RBAU1P5SS.

T. Harju, J. Karhumäki, and A. Lepistö (Eds.): DLT 2007, LNCS 4588, pp. 132–144, 2007.

in [10], where the specification of unbounded message buffers is considered. Since checking context-free properties on PDA is proved in general to be undecidable [7], 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) [1]. 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 [1]. 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 one triggering 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 we prove by a reduction from the halting problem over Minsky Machines. 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 stack 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 [3]. 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 [3], 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 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 we prove for 2-OVPA that intersection and emptiness can be performed in polynomial time while complementation in exponential time, and since inclusion for VPA is ExpTime-complete [1], 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 [7], as well as model checking multi-pushdown properties over MPDA.

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* [11], 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 acting 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, proofs are omitted and reported in the extended version[1].

2 Preliminaries

Let Σ be a finite alphabet partitioned into three pairwise disjoint sets Σ_c, Σ_r, and Σ_l standing respectively for *call*, *return*, and *local* alphabets. We denote

[1] `http://people.na.infn.it/~carotenuto/research/2vpaTechRep.pdf`.

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

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 $\widetilde{\Sigma}$ [1] is a tuple $M = (Q, Q_{in}, \Gamma, \bot, \delta, Q_F)$, where Q, Q_{in}, Q_F, and Γ 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 γ is pushed onto the stack and the control state changes to q'; $(q, a, \gamma, q') \in \delta_r$ a *pop transition*, where γ 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 σ is the stack content. A *run* $\rho = (q_0, \sigma_0) \dots (q_k, \sigma_k)$ of M on a word $w = a_1 \dots a_k$ is a sequence of configurations such that $q_0 \in Q_{in}$, $\sigma_0 = \bot$, and for every $i \in \{0, \dots, k\}$, 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 [3]. 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 Σ, Q, Q_{in}, Γ, 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 δ 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 σ^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 ([3]). *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 Σ. 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 Σ_{c_i} (resp., Σ_{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 Γ 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 δ 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 $\gamma_r, \gamma'_r \in \Gamma_\perp$, there is at most one transition of the form (q, a, q'), (q, a, q', γ), $(q, a, q', \gamma, \gamma')$, (q, a, γ_r, q'), $(q, a, \gamma_r, \gamma'_r, q')$, or $(q, a, \gamma_r, q', \gamma')$ belonging to δ.

Transitions in δ_l, δ_{c_i}, and δ_{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 γ and γ' 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 γ and γ' 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 γ is popped from the stack i and γ' pushed on the other.

A *configuration* of a 2-VPA M is a triple (q, σ^0, σ^1) where $q \in Q$ and $\sigma^0, \sigma^1 \in \Gamma^*.\perp$. For an input word $w = a_1 \dots a_k \in \Sigma^*$, a run of M on w is a sequence $\rho = (q_0, \sigma_0^0, \sigma_0^1) \dots (q_k, \sigma_k^0, \sigma_k^1)$ where $q_0 \in Q_{in}$, $\sigma_0^0 = \sigma_0^1 = \perp$, and for all $i \in \{0, \dots, k-1\}$, 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 $\gamma = \sigma_i^j = \sigma_{i+1}^j = \perp$, or $\gamma \neq \perp$ 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 $\gamma_k = \sigma_i^k = \sigma_{i+1}^k = \perp$, or $\gamma_k \neq \perp$ 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 $\gamma = \sigma_i^j = \sigma_{i+1}^j = \perp$, or $\gamma \neq \perp$ 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 ρ is *accepting* when it ends with a configuration containing a final state. A word w is *accepted* if there is an accepting run ρ 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 $M = (L_1 : I_1; L_2 : I_2; \dots; L_n : \texttt{halt})$ where $n \geq 1$, $L_1, \dots, L_n$ are pairwise different instruction labels, and $I_1, \dots, I_n$ are instructions of type *increment*, i.e., $C_m := C_m + 1$; `goto` L_j, or of type

test and decrement, i.e., `if` $C_m = 0$ `then goto` L_j `else` $C_m := C_m - 1$; `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 $\hookrightarrow \subseteq \mathit{Conf} \times \mathit{Conf}$ 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 [8].

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 $M = (L_1 : I_1; L_2 : I_2; \ldots; L_n : \mathtt{halt})$, 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, δ is defined as follows: if I_i is an increment instruction such as $C_m := C_m + 1$; `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 $(L_{s_1}, v_{s_1}^0, v_{s_1}^1) \ldots (L_{s_k}, v_{s_k}^0, v_{s_k}^1)$ 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 $(L_{s_1}, \sigma_{s_1}^0, \sigma_{s_1}^1) \ldots (L_{s_k}, \sigma_{s_k}^0, \sigma_{s_k}^1)$ of elements from $Q \times \Gamma^* . \bot \times \Gamma^* . \bot$ is a run of M', with $|\sigma_{s_i}^j| = v_{s_i}^j + 1$ for each $i \in \{1, \ldots, k\}$ and $j \in 0, 1$.

The above assertion implies that M halts iff M' accepts a word. □

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, and the sets Σ_c, Σ_r and Σ_{s_i}, with $i \in \{0, 1\}$, are empty.

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 Σ_r is empty and for all input word $w = a_1 \dots a_k \in \Sigma^*$ and run $\rho = (q_0, \sigma_0^0, \sigma_0^1) \dots (q, \sigma_k^0, \sigma_k^1)$ of M over w, for all $i \in \{1, \dots, n\}$, the following hold:*

Pop: $(q_i, a_i, \gamma, q_{i+1}) \in \delta_{r_1}$ *then* $\sigma_i^0 = \sigma_{i+1}^0 = \bot$ *and* $\sigma_{i+1}^1 = \gamma.\sigma_i^1$
Synch: $(q_i, a_i, \gamma, q_{i+1}, \hat{\gamma}) \in \delta_{s_1}$ *then* $\sigma_i^0 = \bot$ *and* $\sigma_{i+1}^0 = \hat{\gamma}.\bot$ *and* $\sigma_{i+1}^1 = \gamma.\sigma_i^1$.

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 ε-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 ε-moves: we only enable ε-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 δ_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 $\sigma_i^j = \sigma_{i+1}^j = \gamma^j.\sigma^j$, 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 ε-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 $M = (Q, Q_{in}, \Gamma, \bot, \delta, Q_F)$, 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), $(q_2, \varepsilon, \bot, B, q_3)$, (q_3, c, B, q_3), $(q_3, \varepsilon, \bot, \bot, q_F)\}$. 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

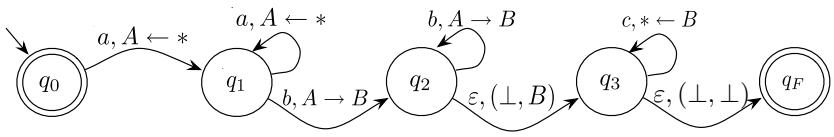

Fig. 2. A 2-OVPA$_\varepsilon$ accepting $L_1 = \{a^n b^n c^n | \exists n \in \mathbb{N}\}$

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 $s, B \leftarrow A$, 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 $a, * \leftarrow B$.

5 Expressiveness and Closure Properties

In this section, we compare 2-VPLs and 2-OVPLs with VPLs [1], deterministic and (nondeterministic) context-free languages (resp., DCFLs and CFLs) [6], and multi-pushdown languages [3] ($\mathcal{L}_{PD^n}$). Recall that the following chain of 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 they preserve union, intersection, complementation (and thus inclusion). These properties, along with the emptiness problem for 2-OVPA being 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 [1] 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 be a 2-VPA (resp., 2-OVPA) over $\widetilde{\Sigma}$. We define a deterministic 2-VPA (resp., 2-OVPA) M' over $\widetilde{\Sigma}$ such that $L(M) = L(M')$ behaving as sketched in the following example. We refer to the extended version for the detailed 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 $(S_0, R_0, c_1^0).\perp$, the second stack is $(S_1, R_1, c_1^1).\perp$, 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 on reading w. □

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 [1].

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 λ (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 λ 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 $\Sigma_r^0 \cap \Sigma_r^1$ 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 $\lambda \subseteq \delta_c^0 \times \delta_r^1 \cup \delta_r^0 \times \delta_c^1$, where δ^0 and δ^1 are the transition relations of M_0 and M_1, respectively.*

A run ρ on $w = a_1 \dots a_n \in (\Sigma^0 \cup \Sigma^1)^*$ for $M_0||_\lambda M_1$ is a pair of VPA runs on w, $\pi^0 = (q_0^0, \bot)(q_1^0, \sigma_1^0) \dots (q_n^0, \sigma_n^0)$ for M_0 and $\pi^1 = (q_0^1, \bot)(q_1^1, \sigma_1^1) \dots (q_n^1, \sigma_n^1)$ for M_1 such that, for all $k \in \{0, \dots, n-1\}$, where t_k^0 is the transition applied from (q_k^0, σ_k^0) to $(q_{k+1}^0, \sigma_{k+1}^0)$ in M_0, and t_k^1 is the transition applied from (q_k^1, σ_k^1) to $(q_{k+1}^1, \sigma_{k+1}^1)$ in M_1, such that if t_k^1 is a pop transition then σ_k^0 is empty and if $(t_k^0, t_k^1) \in \delta_c^0 \times \delta_r^1 \cup \delta_r^0 \times \delta_c^1$ then $(t_k^0, t_k^1) \in \lambda$. A run ρ is accepting if both π^0 and π^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

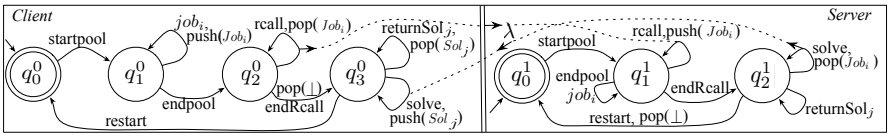

Fig. 3. An example of an S-VPA

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 λ relates each Job_i that the server has to pop, with its solution Sol_j that the client has to push, determining the computation.

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 [1] and multi-pushdown automata with two stacks [3]. 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

[1] Alur, R., Madhusudan, P.: Visibly pushdown languages. In: STOC'04, pp. 202–211. ACM Press, New York (2004)
[2] Bozzelli, L., Murano, A., Peron, A.: Pushdown module checking. In: Sutcliffe, G., Voronkov, A. (eds.) LPAR 2005. LNCS (LNAI), vol. 3835, pp. 504–518. Springer, Heidelberg (2005)
[3] Breveglieri, L., Cherubini, A., Citrini, C., Crespi-Reghizzi, S.: Multi-push-down languages and grammars. Int. J. Found. Comput. Sci. 7(3), 253–292 (1996)
[4] Clarke, E.M., Emerson, E.A.: Design and verification of synchronization skeletons using branching time temporal logic. In: Kozen, D. (ed.) Logics of Programs. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1982)
[5] Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (1999)
[6] Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (1979)
[7] Kupferman, O., Piterman, N., Vardi, M.: Pushdown specifications. In: Baaz, M., Voronkov, A. (eds.) LPAR 2002. LNCS (LNAI), vol. 2514, pp. 262–277. Springer, Heidelberg (2002)

[8] Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice-Hall, Englewood Cliffs (1967)
[9] Queille, J.P., Sifakis, J.: Specification and verification of concurrent programs in Cesar. In: Dezani-Ciancaglini, M., Montanari, U. (eds.) International Symposium on Programming. LNCS, vol. 137, pp. 337–351. Springer, Heidelberg (1981)
[10] Sistla, A., Clarke, E.M., Francez, N., Gurevich, Y.: Can message buffers be axiomatized in linear temporal logic. Information and Control 63(1-2), 88–112 (1984)
[11] Van Steen, M., Tanenbaum, A.S.: Tanenbaum. Distributed Systems: Principles and Paradigms. Prentice Hall, Englewood Cliffs (2002)
[12] Vardi, M.Y., Wolper, P.: Automata-theoretic techniques for modal logics of programs. J. of Computer and System Sciences 32(2), 182–221 (1986)
[13] Walukiewicz, I.: Pushdown processes: Games and Model Checking. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 62–74. Springer, Heidelberg (1996)